Quality Evaluation of Generative AI Systems

Processes, Metrics, Methods, and Frameworks for Industrial Software Engineering

Liang Yu

Department of Software Engineering
Blekinge Institute of Technology

Doctoral Dissertation Series no. 2026:01

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2026:01
ISSN 1653-2090
ISBN 978-91-7295-518-9

Quality Evaluation of Generative AI Systems

Processes, Metrics, Methods, and Frameworks for Industrial Software Engineering

Liang Yu

DOCTORAL DISSERTATION
for the degree of Doctor of Philosophy at Blekinge Institute of Technology to be publicly defended on 2026-01-29 at 13:00 in J1630, Blekinge Tekniska Högskola, 371 79 Karlskrona

Supervisors
Emil Alégroth, Blekinge Institute of Technology, Sweden
Panagiota Chatzipetrou, Örebro University, Sweden
Tony Gorschek, Blekinge Institute of Technology, Sweden

Faculty Opponent
Eric Knauss, University of Gothenburg, Sweden
Grading Committee
Tommi Mikkonen, University of Jyväskylä, Finland
Emelie Engström, Lund University, Sweden
Fabiano Dalpiaz, Utrecht University, Netherlands

Abstract

Generative Artificial Intelligence (GenAI) is being rapidly adopted in software engineering, introducing a paradigm shift toward human-AI co-creation. However, the non-deterministic, probabilistic, and often black-box nature of GenAI models presents challenges for traditional software quality assurance. Conventional verification and validation techniques are insufficient to handle outputs that are neither predictably correct nor incorrect, but rather stochastically plausible. This discrepancy creates an urgent need for practical processes, metrics, and new governance frameworks to evaluate and manage the quality of GenAI systems in industrial environments.

This thesis examines how industrial organizations adopt GenAI, identify metrics, and evaluate system qualities in alignment with ISO quality standards. Case studies were employed to explore real-world adoption processes, identify context-specific industrial metrics, and uncover practical insights within organizations. A snowballing literature review was conducted to systematically identify, categorize, and synthesize academic metrics for evaluating the output of GenAI systems. Finally, a controlled experiment was designed to quantitatively test the efficiency (e.g., E2E generation time) and effectiveness (e.g., accuracy) of GenAI agent choices.

The main contributions of this thesis are a synthesized actionable model and framework grounded in both industrial practice and quality standards. The first contribution is a four-stage adoption model, denoted as the **IMRM** model (**I**nnovate $\rightarrow$ considerations, **M**easure $\rightarrow$ metrics, **R**ealize $\rightarrow$ values, **M**anage $\rightarrow$ improvements) that integrates early-stage risk assessment (e.g., legal, security, and licensing) and quality evaluation throughout the GenAI adoption and usage.

The second contribution presents a detailed framework that connects risks and metrics to concrete decision support, justifying the business value (e.g., quality gates) and technical trade-offs of GenAI solutions. The third contribution provides a structured mapping of GenAI quality to ISO/IEC 25010, 25023, and 25059 characteristics, attempting to ground practical evaluation needs within a standardized vocabulary.

This thesis concludes that a structured quality evaluation process, which prioritizes risks and context, is a valuable approach intended to support building the business confidence required to leverage GenAI for efficient and effective software engineering in industry.

Keywords: Quality Evaluation, Metrics, Artificial Intelligence, AI, Generative AI, Empirical Software Engineering

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2026:01

Quality Evaluation of Generative AI Systems

Processes, Metrics, Methods, and Frameworks for Industrial Software Engineering

Liang Yu

Doctoral Dissertation in Software Engineering

Department of Software Engineering
Blekinge Institute of Technology
SWEDEN

Blekinge Institute of Technology
Department of Software Engineering

Blekinge Institute of Technology Doctoral Dissertation Series No. 2026:01
ISBN 978-91-7295-518-9
ISSN 1653-2090
urn:nbn:se:bth-28958

Printed in Sweden by Media-Tryck, Lund University, Lund 2025

"*A journey of a thousand miles begins with a single step*."

- Laozi, Dao De Jing

Acknowledgements

Words cannot express my gratitude to my supervisors, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek, at Blekinge Institute of Technology, for your invaluable knowledge, expertise, support, and patience.

I am also grateful to my managers, Maria Larsson, Martin Blom, Jon Påhls, and Parisa Yousefi, as well as my colleagues, past and present, at Ericsson. This endeavor would not have been possible without your support.

Lastly, I would like to thank my family, especially my dear wife, Xi Zhang, for her constant support and encouragement during this journey.

List of Studies

Study A

Liang Yu, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "Experience with Large Language Model Applications for Information Retrieval from Enterprise Proprietary Data," *Conference - Product-Focused Software Process Improvement*, 2024. DOI:10.1007/978-3-031-78386-9_7 (In print).

Study B

Liang Yu, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "Measuring the quality of generative AI systems: Mapping metrics to quality characteristics-Snowballing literature review," *Journal - Information and Software Technology*, 2025. DOI:10.1016/j.infsof.2025.107802 (In print).

Study C

Liang Yu, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "Evaluating the Quality of GenAI Applications in Software Engineering: A Multi-case Study," *Journal - Empirical Software Engineering*, 2025.
DOI:10.1007/s10664-025-10759-2 (In print).

Study D

Liang Yu. "Paradigm shift on Coding Productivity Using GenAI," *Conference - Evaluation and Assessment in Software Engineering*, 2025.
DOI:10.1145/3756681.3757081 (In print).

Study E

Liang Yu, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "A Framework for Evaluating GenAI Adoption and Use in Software Engineering," Submitted to *Journal - Transactions of Software Engineering*, 2025 (In submission).

Study F

Liang Yu, Jon Påhls, Emil Alégroth. "Evaluating the Sufficiency of Single-Agent LLM Systems for Algorithmic Problem Solving in Support and Operations," Submitted to *Journal - Journal of Systems and Software*, 2025 (In submission).

Other Papers not in this Thesis

- **Liang Yu**, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "Utilising CI environment for efficient and effective testing of NFRs," *Journal - Information and Software Technology 117*, 2020. DOI:10.1016/j.infsof.2019.106199 (In print).
- **Liang Yu**. "Ethical considerations in case studies," *Conference-20201st Workshop on Ethics in Requirements Engineering Research and Practice (REthics)*, 2020. DOI:10.1109/REthics51204.2020.00009 (In print).
- **Liang Yu**, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "Automated NFR testing in Continuous Integration Environments: A multi-case study of Nordic companies," *Journal - Empirical Software Engineering*, 2023. DOI:10.1007/s10664-023-10356-1 (In print).
- **Liang Yu**, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "A Roadmap for Using Continuous Integration Environments," *Journal - Communications of the ACM*, 2023. DOI:10.1145/3631519 (In print).

Funding

This research was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology.

Author's contribution to the papers

Liang Yu is the lead author of all the papers included in this thesis. He took primary responsibility for designing the studies, collecting and analyzing data, and reporting the findings in peer-reviewed publications.

Both Emil Alégroth and Panagiota Chatzipetrou participated as researchers on intermediate versions and the final draft of all the above papers. They provided valuable support regarding how to conduct research with suitable research methodologies and how to present the research findings. Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek reviewed and commented on intermediate versions and the final draft of the paper.

Abbreviations

AI	Artificial Intelligence.
AI4SE	Artificial Intelligence for Software Engineering.
CI	Continuous Integration.
DL	Deep Learning.
GenAI	Generative Artificial Intelligence.
GPT	Generative Pre-trained Transformer.
IDE	Integrated Development Environment.
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission.
LLM	Large Language Model.
LLMOps	Large Language Model Operations.
ML	Machine Learning.
NLP	Natural Language Processing.
RAG	Retrieval-Augmented Generation.
SE4AI	Software Engineering for Artificial Intelligence.
SQuaRE	Systems and Software Quality Requirements and Evaluation.
USSD	Unstructured Supplementary Service Data.

Table of Contents

List of Figures

List of Tables

1 Overview

1.1 Introduction

Generative Artificial Intelligence (GenAI) is a type of Artificial Intelligence (AI) that derives new content from a statistical model pre-trained on large amounts of data, e.g., text, code, imagery, audio, and video [1, 2]. When given a user input prompt, the GenAI model produces an output based on its training data, which, for Natural Language Processing (NLP) models, i.e., Large Language Model (LLM), means predicting the next most plausible token of the response [1].

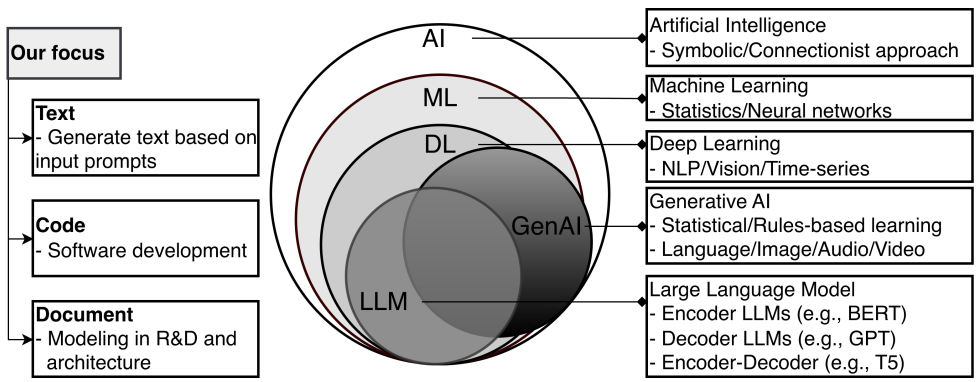

Figure 1.1: An overview of how GenAI differs from LLM, DL, ML, and AI. Our focus is on text, code, and document generation. NLP stands for natural language processing.

Figure 1.1 provides an overview of how GenAI relates to broader AI concepts. It illustrates that GenAI is a subset of Deep Learning (DL), which is a subset of Machine Learning (ML), which in turn is a subset of AI. LLM, a prominent type of GenAI, encompasses various architectures, including Encoder LLM (e.g., BERT model), Decoder LLM (e.g., Generative Pre-trained Transformer (GPT) models), and Encoder-Decoder LLM (e.g., T5 model) [3–5]. The focus of this thesis is on the adoption and use of GenAI systems in industrial contexts, specifically how to effectively and efficiently use GenAI to generate text, software code, and documents for R&D and architectural modeling.

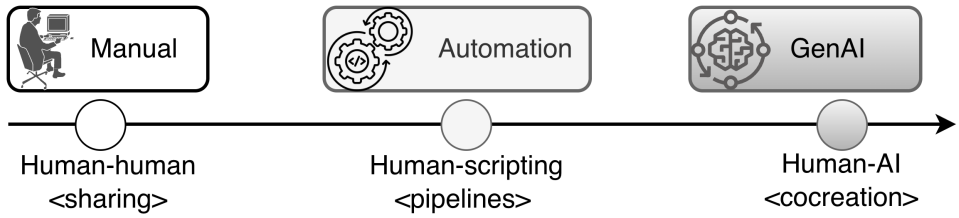

Figure 1.2: A paradigm shift from human to human knowledge sharing, human to scripting with automated pipelines, and human to AI through co-creation.

Software engineering practices have undergone fast shifts over time, as illustrated in Figure 1.2. Software development has evolved from manual, human-to-human information sharing processes to automation through human-scripting pipelines, such as Continuous Integration (CI). The advent of GenAI represents a further paradigm shift towards human-AI co-creation, where AI assists or partners in development tasks [6]. In this thesis, this practical integration of GenAI capabilities into an organization is referred to as *GenAI adoption and use*. The adoption and use encompass two perspectives: Artificial Intelligence for Software Engineering (AI4SE), where GenAI tools augment traditional software development activities (e.g., coding assistants) [7, 8], and Software Engineering for Artificial Intelligence (SE4AI), which involves practices for building, deploying, and maintaining reliable GenAI components as part of a software system [9]. While the majority of research has focused on AI4SE, SE4AI is less studied, with notable emerging examples in Large Language Model Operations (LLMOps) [10] and agentification [11].

However, this adoption and use of GenAI introduces new challenges to software development. A novel challenge stems from GenAI's non-deterministic nature [12, 13]. Unlike traditional software, GenAI models can produce different outputs even with the same input. This non-determinism, combined with the fact that generated content is often processed within an opaque, black-box model, makes their behavior unpredictable. This opacity presents a new challenge for traditional verification and debugging, as it is difficult to trace the internal reasoning or origin of outputs [14]. These challenges raise concerns about functionality issues, where the generated output may not meet required specifications or may contain subtle errors. Additionally, these same factors of non-determinism and opacity add new layers of complexity to evaluating non-functional quality characteristics. Assessing qualities such as reliability, security, and maintainability, which is already a non-trivial task for traditional software, becomes more complex for systems incorporating GenAI components [15]. Addressing these challenges requires new approaches for quality evaluation of both the adoption of AI-enabled tools (AI4SE) and the development of AI-based systems (SE4AI).

This thesis addresses the challenge of quality evaluation by proposing industrially applicable frameworks for enterprise organizations. The contributions are designed to support two scenarios. The first scenario, *strategic adoption and use*,

is addressed by the *IMRM model*, presented in Section 1.4.1. This model offers a high-level process for organizations, particularly medium- to large-sized enterprises, which need to manage the intake and rollout of GenAI capabilities. It is applicable when GenAI initiatives must be vetted for security, legal, and business risks before deployment. The second scenario is *tactical quality evaluation*, addressed by the *ISPDE framework*, presented in Section 1.4.5. This framework provides a detailed process for development teams and researchers. It is applicable when a specific GenAI system needs to be verified against prioritized quality characteristics, such as reliability or security, by using concrete metrics.

1.2 Background and related work

Generative AI (GenAI) is being rapidly adopted in software development companies [6, 16]. This adoption spans the entire software engineering lifecycle, from requirements analysis and documentation to code generation and testing [6, 17]. Industrial practices demonstrate that GenAI is utilized to build developer assistants, information retrieval systems, and data analysis tools [7].

However, this rapid adoption introduces challenges for quality assurance [13]. Due to the black-box nature and non-deterministic behavior of GenAI-based systems, their probabilistic outputs are difficult to verify [12, 14, 18]. These characteristics, including non-determinism (e.g., varying outputs in terms of syntactic and semantic correctness), prompt sensitivity (e.g., outputs of varying abstraction levels given contextual input), and the potential for 'hallucinations' or factually incorrect outputs (e.g., erroneous outputs derived from available training data), limit the applicability of conventional software quality assurance techniques [15, 19]. These challenges create an urgent need for new frameworks, metrics, and processes to evaluate and manage the quality of GenAI-based systems [20].

To reason about quality in a structured manner, this thesis is grounded in the established ISO/IEC 25000 series, also known as Systems and Software Quality Requirements and Evaluation (SQuaRE) [21, 22]. These standards provide unified and industry-applicable quality models. Specifically, ISO/IEC 25010 defines product-quality and product-in-use models [21]. These models are hierarchical, breaking down high-level product-quality characteristics into subcharacteristics, which are further defined by lower-level quality properties [21]. For example, the high-level characteristic *Security* is broken down into subcharacteristics such as *Confidentiality*, *Integrity*, and *Authenticity*, which in turn are measured by properties, such as *data encryption* and *authorization* [21]. Metrics used to measure these quality properties are defined and clarified with examples in ISO/IEC 25023 [22]. More recently, ISO/IEC 25059 was introduced to extend these quality models specifically for AI-enabled systems [23]. It differentiates between product quality (the system's inherent properties) and quality-in-use (the quality experienced by a user in a specific context). This

thesis focuses on a set of **Product quality** characteristics, including Functional Suitability, Reliability, Usability, and Security [21], and **Quality-in-Use** characteristics, such as Effectiveness, Efficiency, Satisfaction, and Freedom from risk [23]. Using these ISO/IEC standards provides a common foundation for defining, measuring, and discussing the multifaceted aspects of GenAI system quality.

To ensure clarity throughout this thesis, key terms are defined as follows:

- *Quality characteristic*: A property of a software system or product that can be measured to determine its quality, as defined by standards ISO/IEC 25023 [20, 22].
- *Metric*: A quantifiable measure used to assess how well a system performs in relation to a specific quality characteristic [20]. It translates an abstract quality into a concrete, measurable value.

To evaluate the quality of GenAI systems, the existing literature provides various metrics, such as BERTScore, FActScore, and CodeBLEU, to assess system outputs [20, 24–26]. However, a gap exists between these academic metrics and the context-specific evaluation practices used in industry. Practitioners often rely on lightweight, task-oriented checks rather than computationally intensive research metrics, as observed in the industrial case studies by Yu et al. [27]. Furthermore, the methods for applying these metrics vary, falling into three main categories: *Manual evaluation*, *Automated evaluation*, and *AI-assisted evaluation* [20]. This thesis demonstrates that practical quality evaluation is not a single-step verification practice. Instead, it is a continuous process integrated throughout the entire GenAI software adoption and/or development life-cycle. This aligns with the software quality assurance (QA) cycle, where a continuous QA process ensures that quality is engineered into the product to an acceptable level [20]. This process involves different stakeholders, including legal, security, development, and operations teams, who evaluate different quality aspects at various stages. Finally, evaluation can be used as an experimental method to compare technical alternatives (e.g., single-agent/multi-agent [28] systems for code generation) based on specific quality properties [21], such as code complexity ranges and acceptance rates [29].

1.3 Research problem and methodology

While organizations are rapidly adopting GenAI for software development tasks from code generation to document creation, they lack reliable approaches to evaluate and ensure the quality of these systems. Consequently, conventional quality assurance methods are insufficient to address the characteristics of GenAI systems [12–14].

This creates a critical gap: while GenAI offers enhanced productivity benefits for developers (AI4SE) [6] and enables product features that address previously un-

solvable challenges (SE4AI), organizations struggle to evaluate if their GenAI applications are working as expected, what metrics to use for such evaluation, how to manage application risk, and what technical approaches, architectures, or frameworks to choose from.

Without structured evaluation methods, organizations face several challenges:

1. *Quality uncertainty*: No systematic way to measure if GenAI outputs meet quality requirements.

2. *Risk exposure*: Inadequate processes to handle legal, security, and business risks from GenAI adoption and usage.

3. *Decision confusion*: Lack of evidence-based methods to compare technical alternatives.

This thesis addresses these challenges by investigating how organizations evaluate the quality throughout GenAI adoption and use, providing practical steps, metrics, and processes grounded in industrial practice as well as established quality standards.

1.3.1 Research questions

The objective of this thesis is to investigate and define effective processes, risk mitigation strategies, and metrics for evaluating the qualities of GenAI systems in industrial software engineering to provide structured decision support for their adoption and use. This research objective was broken down into four research questions, presented below, along with brief descriptions of how they were addressed. Table 1.1 presents a mapping between each research question (RQ) and the studies included in this thesis.

Table 1.1: A mapping of research questions to the included studies.

Study	Objective	RQ1	RQ2	RQ3	RQ4
A (Chapter 2)	Secure deployment and processes for GenAI use on internal documents.	X			
B (Chapter 3)	Metrics and standards to measure GenAI systems' output quality.			X	
C (Chapter 4)	Industrial use cases showing domain practices and metric application.		X	X	
D (Chapter 5)	Effects of coding assistants on workflow and outcome quality.				X
E (Chapter 6)	Adoption and evaluation framework with risk controls for GenAI.	X	X	X	
F (Chapter 7)	Evaluating the Sufficiency of Single-Agent for Algorithmic Problem Solving.				X

- **RQ1: What processes do teams use to support secure GenAI usage in industrial environments?**

  This research question aims to identify the concrete processes that facilitate the adoption of GenAI in practice, particularly in terms of security considerations. It focuses on engineering and governance enablers such as private or local hosting, access control, audit logging, and data handling rules. It distills operational practices that keep sensitive assets safe while maintaining delivery flow. Support is provided through a secure deployment case with defined

processes (Chapter 2) and an adoption framework that includes risk controls to guide the rollout (Chapter 6). The outcome is a reference approach and a staged framework that teams can adapt to their context.

- **RQ2: How do teams mitigate business risks stemming from GenAI usage in software development organizations?**

  This question examines how teams mitigate business risks that arise from the use of GenAI. It links risks (e.g., legal, licensing, privacy, and operational failure) to concrete evaluations, checks, and accountable owners. It structures these controls along the lifecycle with intake due diligence, pre-release verification, and live monitoring, and uses fit-for-purpose metrics where suitable. Support comes from domain practices and the application of controls in industrial settings (Chapter 4) and from an adoption and evaluation framework that defines gates and responsibilities (Chapter 6). The result is a practical governance guideline that reduces exposure while maintaining steady progress.

- **RQ3: What metrics can be used to measure GenAI-based system output quality for organizations?**

  Building on the risk mitigation needs identified in RQ2, this research question aims to concretely identify and catalog metrics for assessing the output quality of GenAI systems. It examines how organizations can measure quality using metrics grounded in specific software tasks that are applicable in practice. It defines methods for selecting, mapping, and applying these metrics, and shows how they are used in real-world projects. Support for this question was obtained through a snowballing literature review that identified metrics and mapped them to ISO/IEC 25023 using a forward/backward procedure and a six-step mapping protocol (Chapter 3), as well as an empirical industrial study that elicited and applied 28 metrics across industrial GenAI use cases (Chapter 4). Our findings were complemented by an ISO-aligned framework that positions where evaluations could occur throughout the GenAI usage process (Chapter 6).

- **RQ4: How can GenAI solution alternatives be evaluated with metrics to support adoption decisions?**

  This question explains how to evaluate technical alternatives using the metrics from RQ3 to support adoption decisions. It frames experiments with clear tasks, datasets, prompts, and hypotheses that tie each metric to a decision point such as acceptance rate, factuality, latency, or cost. Support includes controlled comparisons of alternative GenAI solutions that analyze workflow and outcome effects (Chapter 5) and experiments that assess the effectiveness and efficiency of Single-/Multi-Agent solutions on self-contained algorithmic

problems (Chapter 7). The method reports effect sizes, error modes, and resource use so that trade-offs are visible to stakeholders. The output is a decision brief that links measured gains or regressions to stakeholder decisions, thereby supporting both the business and technical aspects of a rollout plan.

1.3.2 Thesis research process

The included studies (Study A-F) in this thesis form an integrated research process that builds knowledge about GenAI quality evaluation. Figure 1.3 provides an overview of this research process, displaying the chronological progression of the included studies and their respective outcomes mapped to the RQs.

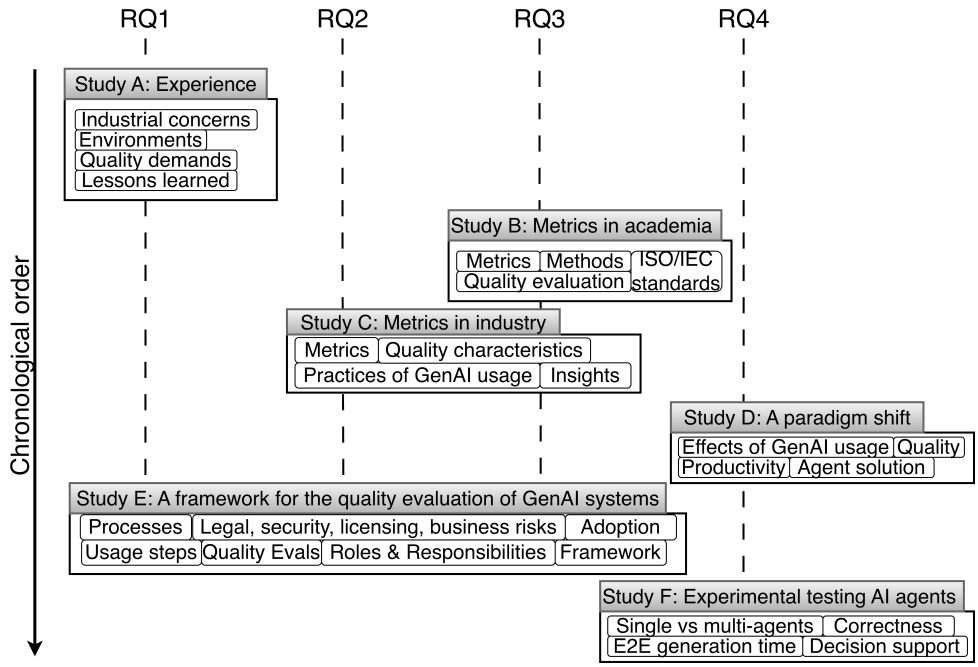

Figure 1.3: An overview of the research process and outcomes from the included studies, mapped to the thesis's research questions.

The integration logic of the studies is structured into four phases: foundational industry-data gathering, knowledge building, framework development, and validation. **Foundation phase** (Study A): Establishes the practical context by demonstrating secure GenAI deployment challenges and initial quality concerns in a real industrial setting. This study reveals that organizations need systematic evaluation approaches beyond ad-hoc manual checks. **Knowledge building phase** (Study B and C): Study B systematically identifies what metrics exist in academia, while Study C discovers what metrics are actually used in industry. Together, they reveal that academic metrics are complex for practical application, while industrial metrics lack

a systematic foundation. **Framework development phase** (Study E): Synthesizes insights from Studies A, B, and C to create a comprehensive evaluation framework that bridges academic rigor with industrial practicality. This study directly builds on the secure deployment patterns from Study A and the metric catalogs from Studies B and C. **Validation phase** (Study D and F): Test the framework's applicability through empirical evaluation. Study D validates the productivity claims and evaluation challenges identified in earlier studies, while Study F demonstrates how metric-based evaluation can support technical decision-making.

1.3.3 Research methodology

A research methodology provides a structured and scientific approach to investigate a research problem or question [30]. The primary goal of selecting a formal methodology is to ensure the research process is objective, rigorous, and replicable [30]. Adhering to a well-defined approach helps establish that the generated findings are credible and trustworthy [30]. In software engineering research, various methodologies are available, each possessing distinct strengths and limitations [31, 32]. A key responsibility for a researcher is to select and motivate the methodology that is most suitable for the specific research objectives and context. The suitability of this selection directly impacts the study's overall validity, which includes internal, external, construct, and conclusion validity [32]. An unsuitable methodological choice can, for example, limit the generalizability (external validity) or compromise the study's causal claims (internal validity) [31, 32].

The research presented in this thesis is based on the findings from the included studies (Studies A to F). Each of these studies employed specific research methodologies, selected to align with its distinct objectives. This combination of methodologies enables the triangulation of findings and balances the validity trade-offs [31, 32]. Figure 1.4 visualizes an overview of mapping the included studies to research methodologies.

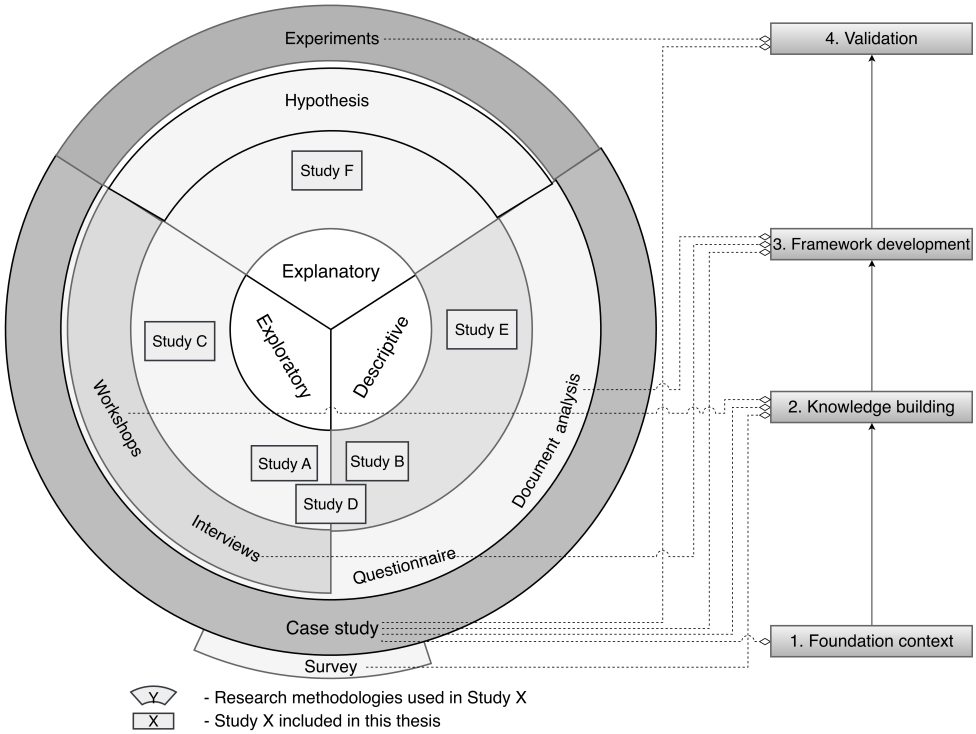

Figure 1.4: Visualization of research methodologies mapped to the studies included in this thesis.

As shown in Figure 1.4, this thesis is composed of studies that are exploratory, descriptive, and explanatory in nature [33]. *Exploratory* studies are utilized to gain an initial understanding of a problem, identify potential solutions, and formulate hypotheses [34]. *Descriptive* studies are used to describe the characteristics of a phenomenon or a specific group [35]. *Explanatory* studies are hypothesis-driven and are used to test relationships and causal claims [31]. Table 1.2 maps these study types and the specific research methodologies to the individual studies included in this thesis. Multiple research methodologies were employed to align with the aims of each included study. Case study research (Study A, D, and E) and a multi-case study (Study C) were used to capture in-depth, contextual, and process-related data [32]. A questionnaire-based survey (Study D) was used to quantify practitioner perceptions and usage patterns [36]. A snowballing literature review (Study B) was applied to consolidate existing metrics and definitions from a rapidly evolving field. A controlled experiment (Study F) was used to test causal effects between technical alternatives under fixed tasks and data [31].

Table 1.2: Research methodologies, study types, data collection techniques.

Study	Research methodology	Study type	Data collection technique
A (Chapter 2)	Case study	Exploratory	Workshops, interviews
B (Chapter 3)	Snowballing literature review	Descriptive	Forward/backward searches
C (Chapter 4)	Multi-case study	Exploratory	Workshops, interviews
D (Chapter 5)	Case study + Survey	Descriptive	Questionnaire, interviews
E (Chapter 6)	Case study	Descriptive	Document analysis, interviews
F (Chapter 7)	Controlled experiment	Explanatory	Fixed tasks, measured outcomes

Furthermore, the research process presented in this thesis follows a general progression from exploration to explanation. The process began with exploratory studies (Study A) to gain an initial understanding of the industrial problem space and identify key challenges [34]. This was followed by descriptive studies (Study B and C) to characterize the phenomenon, build a body of knowledge (e.g., cataloging metrics), and identify adoption patterns. Finally, the research transitioned to an explanatory phase, involving a controlled experiment (Study F) to test a specific hypothesis about technical alternatives and a case study (Study D) to verify the user experience of using GenAI tools, informed by the preceding descriptive work [31]. This progression is not a single, linear path, but an investigation into the complex phenomenon of GenAI quality. Different methodologies were required in parallel to build a comprehensive understanding before explanatory testing was feasible. The following paragraphs provide a detailed justification for the methodological choices of each individual study.

Study A: A case study was selected to examine secure enablement of GenAI within its organizational context, using workshops, interviews, and deployment artifacts as multiple sources of evidence [32]. This design fits the early and security-sensitive setting, where manipulation of variables or random assignment was infeasible. A survey would have yielded breadth but little process insight and weak traceability between accounts and artifacts. Action research [37] was not adopted because sustained intervention aimed at change across cycles was outside the scope and authority.

Study B: A snowballing literature review was chosen to consolidate metrics and align them with quality characteristics in the context of rapid publication growth. Snowballing is effective when terminology is fluid and indexing is incomplete, which matched the GenAI metric landscape at the time [38]. A traditional Systematic Literature Review (SLR) [39] was considered less suitable due to the challenge of defining stable search strings for a rapidly evolving field like GenAI metrics; such strings might miss relevant recent work or retrieve excessive noise. Furthermore, the primary goal was not necessarily exhaustive coverage, but rather the identification and synthesis of specific, impactful metrics currently being proposed and used. Snowballing provided an efficient method for identifying relevant publications through citation links, striking a balance between scope and feasibility in a dynamic research area [40]. The review thus prioritized traceable identification and coding of metrics.

Study C: A multi-case study was used to capture industrial practices and metric use across domains through workshops [32]. Action research would have been suitable if the intent were to change or improve the units under study through iterative interventions [37, 41]. Grounded theory would have been appropriate if the primary goal were to generate a new theory from data in a narrowly bounded area [42]. The topic was broad and spanned continuous integration and quality evaluation, which required multiple sources of evidence rather than a sustained intervention or pure theory building. The chosen design strikes a balance between discovery and description, while maintaining links between accounts and artifacts.

Study D: A case study with a questionnaire-based survey was conducted to describe adoption patterns, integration effort, and perceived productivity factors [32, 36]. A controlled experiment was not feasible in the live projects due to confounding factors, including tools, schedules, and policies, as well as a limited ability to randomize tasks. A diary or ethnographic study could have added temporal detail, but would have extended beyond the project window and raised confidentiality risks. The chosen mix provided both breadth (through surveys) and depth (via interviews) with manageable intrusiveness and traceability.

Study E: A case study was employed to identify GenAI adoption and use practices, and an evaluation framework was synthesized by integrating prior findings with established standards and governance practices, supported by document analysis and input from practitioners. Action research was considered but not selected because the researchers lacked the necessary authority to implement and control iterative change cycles within the organizations [41]. Grounded theory was not the aim, since the target artifact was a practical framework rather than a new theoretical construct [42]. A controlled field trial would have offered stronger causal claims, but it would have exceeded time and resource constraints. Scenario walk-throughs were therefore used to test clarity and feasibility before later trials.

Study F: A controlled experiment was selected to evaluate the sufficiency of Single-Agent LLM systems for algorithmic problem solving in Support and Operations, following guidance for explanatory studies [30]. A case study would have preserved context, but would not have isolated the agent orchestration from other factors. Repository mining would have increased external realism, but would not control model versions, prompts, or datasets. A field experiment was considered but deemed infeasible due to latency and cost constraints in production settings. The laboratory-style setup improved internal validity at the cost of a narrower task scope, which motivates follow-up studies in larger software tasks.

1.3.3.1 Validity in research

Each methodology has inherent characteristics that influence the validity of the study's findings, making it essential to carefully select the most suitable approach based on the research objectives [43]. Evaluating the validity of research in software engineering often involves considering four primary types, as outlined by Runeson and Höst [32].

- Internal validity concerns the appropriateness of the research design and methods for answering the research question and the accuracy of causal claims [44].
- External validity pertains to the extent to which findings can be generalized beyond the specific study sample and context [32].
- Construct validity refers to the degree to which the study measures what it intends to measure, including the operationalization of concepts [32].
- Conclusion validity addresses the accuracy of the conclusions drawn and whether they are supported by the collected data [32].

Different research methodologies inherently involve trade-offs among these types of validity. For instance, controlled experiments often exhibit strong internal validity due to rigorous control over variables. Still, they may face limitations regarding construct validity if the experimental setting does not adequately represent real-world conditions [31]. Conversely, case studies tend to offer strong construct validity because they are conducted in authentic contexts with relevant subjects, but they may have weaker internal validity due to the researcher's limited control over extraneous variables in industrial settings [32]. Understanding these trade-offs is essential for interpreting the results of each study included in this thesis.

1.3.3.2 Case studies

Case studies [32] involve in-depth examination of phenomena in real-world contexts, providing rich, context-specific data. While offering strong construct validity, they may have weaker internal validity due to limited control over variables in industrial settings [32].

To ensure validity and reliability, the included studies used multiple data sources, triangulation, and rigorous analysis through interviews, workshops, document analysis, surveys, and observations [32].

Interviews: Interviews [32] are commonly used for data collection in case study research and can be divided into three different types: structured, semi-structured, and unstructured interviews. Each type is conducted using an interview guide that contains step-by-step instructions for the interview, including the interview questions and research objectives.

Structured interviews [32] are frequently employed in research aimed at identifying specific attitudes, behaviors, or opinions. The primary advantages of structured interviews include their ability to generate precise and reliable data that is easily comparable across participants. However, they may not capture the full complexity of a participant's experience or allow for spontaneous discussion. Additionally, the predetermined questions may not address all relevant issues or accurately reflect the participant's perspective. Structured interview questions can take various forms, but multiple-choice or forced-choice questions are the most common types. Forced choice questions [32] can be analyzed with statistics but require a larger interview sample, which makes the method costly in terms of resources. Therefore, structured interviews were not used during the thesis work.

Unstructured interviews [32] are a type of qualitative research method that involves an open-ended conversation between the interviewer and the interviewee. Unlike structured interviews, unstructured interviews do not follow a predetermined set of questions. While an interview guide is used, it typically consists of a list of broad topics or themes to be explored, rather than a fixed question script. The interviewer begins with one of these broad topics and allows the conversation to flow naturally, probing the interviewee for more information or clarification as needed. The goal of an unstructured interview is to gain a deeper understanding of the interviewee's experiences, perspectives, and attitudes, and to explore topics that may not have been anticipated beforehand. Unstructured interviews [45] can provide rich, detailed data that is often difficult to obtain through other research methods. However, because the data is less structured and more subjective, it can be more challenging to analyze and generalize the findings.

Semi-structured interviews [32] are a type of interview method that combines the advantages of both structured and unstructured interviews. In semi-structured interviews [45], the interviewer prepares a list of open-ended questions in advance, but the researcher also has the flexibility to ask follow-up questions and probe deeper into the interviewee's responses.

They allow for more in-depth and nuanced responses than structured interviews, while also providing a more consistent and systematic approach than unstructured interviews [45]. Additionally, because the interviewer has some control over the interview structure, the data collected from semi-structured interviews can be more easily compared and analyzed [45] across multiple participants. Semi-structured interviews were employed in Study C due to the need for cross-comparison of data from four distinct software development firms.

Participant selection or sampling is an important consideration for interview-based research. While surveys often prioritize random sampling to support statistical generalization, qualitative interviews frequently use non-probability sampling techniques [32]. Techniques such as *purposive sampling* (selecting participants based on their specific knowledge) or *convenience sampling* (selecting participants based on availability) are common in case study research to ensure the data is information-

rich and relevant to the research questions. The choice of sampling strategy directly impacts the validity and generalizability of the interview findings.

Surveys: Surveys [34] are used to gather data from a broad range of populations, allowing for the collection of both quantitative and qualitative information, depending on the types of questions and response options employed.

To ensure the credibility and dependability of survey data, careful survey design is crucial [36]. This includes developing survey questions with great care, selecting appropriate sampling methods, determining the sample size, and accounting for potential sources of bias or error. Standard data analysis techniques utilized in survey research include descriptive statistics, correlation analysis, and regression analysis [36].

Given that Study B is a literature review encompassing a wide range of scholarly articles, survey methodology was employed to collect data for this purpose.

Workshops: Workshops [32] involve a group of people coming together for a designated period of time to engage in a guided discussion or activity related to the research topic. The workshop can be conducted to generate ideas, collect data, or facilitate collaboration and consensus-building among participants. Workshops can take on many forms, including brainstorming sessions, focus groups, or collaborative problem-solving sessions [45]. They can involve participants from diverse backgrounds and perspectives.

Workshops can serve as a valuable means for researchers to gather insights and data, especially when investigating intricate or multi-dimensional phenomena that necessitate inputs from various stakeholders or perspectives [32]. Furthermore, workshops can be helpful for researchers to interact with practitioners or policymakers, thereby ensuring the practicality and applicability of research findings.

In light of these benefits, Study C employed workshops to collect feedback from industrial participants and validate the study's findings.

Document analysis: Document analysis [32] is a methodology that aims to examine various types of documents to extract pertinent information related to the research question.

Document analysis can be used as either a primary or secondary data collection method, depending on the research question and the availability of relevant documents [45]. The process of document analysis encompasses several stages, including data collection, data organization, data coding, and data interpretation. Researchers need to carefully select appropriate documents, establish explicit inclusion and exclusion criteria, and devise a systematic approach for organizing and storing the data. The subsequent step is to code the data by identifying and categorizing key themes, patterns, and concepts. In Study C of this thesis, several peer-reviewed literature sources were examined to identify common themes and patterns.

Observations: Observation [32] is a technique that involves systematically observing and recording behaviors, events, and other phenomena to understand a particular phenomenon of interest. In this method, the researcher directly observes the

behavior or phenomenon in a natural setting or a controlled environment.

Observation can provide valuable insights into a wide range of phenomena, including human behavior, organizational processes, and natural phenomena. It can be used in combination with other research methods to provide a more comprehensive understanding of a phenomenon of interest [32]. However, observation also has limitations, including the potential for observer bias and the difficulty of generalizing findings to other settings or populations.

1.3.3.3 Experiments

Experimental research is a widely used research methodology in both the natural and social sciences, enabling researchers to test hypotheses and determine causal relationships [31]. In experimental research, researchers randomly assign participants to various groups, apply an intervention or treatment to one or more groups, and then measure the impact of the intervention on the dependent variable. Sampling is a critical design decision in experiments, because the way units are selected determines to which population the results can be generalized. Researchers should therefore define the target population and use suitable sampling strategies, to reduce bias in the findings.

Experimental research aims to establish a cause-and-effect relationship between variables, utilizing control groups, independent variable manipulation, and random assignment of participants [31]. The key strength of experimental research is its ability to establish causality, as well as its high degree of control over extraneous variables.

However, experimental research does have certain limitations [31]. For example, manipulating particular variables in human subjects can be difficult or unethical, and some phenomena cannot be examined in a laboratory setting.

1.3.3.4 Data analysis

Data analysis [31] is the process of transforming raw data into meaningful insights to answer research questions or inform decisions. Both quantitative and qualitative [43] data analysis were employed in this thesis work.

Quantitative data analysis [43] involves numerical data that can be analyzed using statistical techniques. Quantitative research involves collecting data through structured methods, such as experiments or surveys, and using statistical analysis to draw conclusions. Examples of quantitative data include the number of participants, their information, and responses. Quantitative data analysis in Study F focused on identifying patterns, correlations, and relationships between variables. Inter-rater reliability analysis [46] [47] in Study B was particularly helpful in ensuring that the inclusion and exclusion searching criteria were understandable and agreed upon by all involved researchers.

Qualitative data analysis [43], on the other hand, involves non-numerical data,

such as text, images, and audio recordings. Qualitative research involves collecting data through interviews, focus groups, or observations and analyzing it through content analysis or thematic analysis [48]. Qualitative data can provide detailed information about people's experiences, attitudes, and perceptions. Its analysis focuses on understanding themes, patterns, and trends in the data. During the thesis work, the thematic coding approach [48] was used for data analysis in Studies from A to E, as they relied on qualitative data. The approach was performed by four researchers aiming to mitigate coding bias.

1.3.4 Overview of publications

1.3.4.1 Study A: Industrial Experience with Large Language Model Applications

Study A, presented in Chapter 2, is titled "Experience with Large Language Model Applications for Information Retrieval from Enterprise Proprietary Data." The study presents a secure, sandboxed approach to deploy local LLMs for enterprise information retrieval and reports steps, results, and lessons learned from an industrial use case.

Research objective: The objective was to design and deploy a sandboxed LLM environment for proprietary data and to capture practical lessons from its real-world use.

Methodology: The study employed an exploratory *case study* in a FinTech company. A multi-stage process was used, starting with three *workshops* and thirty *semi-structured interviews* to elicit practitioner needs and security constraints. This was followed by the design and deployment of a Retrieval-Augmented Generation (RAG) solution, which was then evaluated by participants.

Results: The team identified a practical deployment flow that meets data isolation needs and enables local RAG information retrieval. Key findings from user evaluations showed that system utility was highly dependent on technical parameters, such as chunk size and the number of retrieved documents, which required context-specific tuning. The resulting solution included configurable retrieval parameters and user controls for chat history, allowing for a balance between context and precision. Participants emphasized the need for automated evaluation at scale, human-in-the-loop checkpoints, and explicit user disclaimers to manage output reliability and user trust. Approximately 65% of participants reported that the sandboxed application was helpful for retrieving product information in their daily work.

Contributions: The study presents a documented industrial deployment of a sandboxed RAG solution, including its implementation stack, parameterization guidance, and key lessons learned regarding failure modes (e.g., handling short prompts and trade-offs in chunking). It contributes a secure-by-design pattern (isolated VM, limited ports) and a process for setup, ingestion, operation, and evaluation that other

teams can adapt. The findings include practical evaluation demands, such as the need for human-in-the-loop checkpoints, and evidence of perceived value from practitioners. Limitations include the single-company scope and sandbox conditions, which motivate replication in other domains and full production settings.

1.3.4.2 Study B: Measuring the Quality of Generative AI Systems in academia

Study B, presented in Chapter 3, is titled "Measuring the Quality of GenAI Systems: Mapping Metrics to Quality Characteristics - Snowballing Literature Review." The study identifies quality evaluation metrics for GenAI system outputs and aligns them with ISO/IEC system quality characteristics.

Research objective: The objective was to find metrics that organizations can use to evaluate GenAI output quality and to map these metrics to ISO/IEC quality characteristics for practical use.

Methodology: A snowballing literature review combined a start-set of recent review publications with forward and backward searches across ACM, IEEE, Scopus, and ArXiv. Inclusion and exclusion criteria prioritized empirical and industry-relevant studies on text-based GenAI applications. Rigor and industrial relevance were scored using established rubrics, and inter-rater agreement was calculated with Fleiss' κ [49].

Results: The review screened 1583 records, filtered 69 papers, and included 43 studies after rigorous and relevant checks. Twenty-eight named metrics were consolidated and mapped to four ISO/IEC 25023 characteristics: *usability, reliability, functional suitability, and maintainability*. This finding provides a clear link between abstract quality goals and concrete, measurable metrics. The study also identified three main evaluation methods (Manual, Automated, and AI-assisted) and synthesized a three-step process for managing risky outputs (prepare datasets, compute scores, and monitor scores). An ISO-aligned five-step framework for metric selection and monitoring was also synthesized from the literature.

Contributions: The study provides a catalog of GenAI output metrics with an explicit mapping to ISO/IEC 25023. It also contributes a documented six-step mapping protocol and reliability evidence for study selection and coding. For industrial practitioners, the study provides a practical toolkit that links concrete metrics to quality goals, demonstrates evaluation methods with example datasets, and offers a repeatable process for detecting and tracking risky outputs over time. For researchers, subjectivity in metric-to-standard mappings was reduced through a predefined protocol and inter-rater checks. The study's limitations include its domain scope (text-focused) and the variance in how metrics are implemented.

1.3.4.3 Study C: Evaluating the Quality of GenAI Applications in industry

Study C, presented in Chapter 4, is titled "Evaluating the Quality of GenAI Applications in Software Engineering: A Multi-case Study." The study analyzes how industrial teams evaluate GenAI application quality across domains and which metrics and methods are applied in practice.

Research objective: The objective was to identify industrial GenAI use cases, document the metrics and evaluation methods in use, and surface challenges and opportunities for applying quality measurement at scale.

Methodology: A multi-case study was employed, covering three software companies with variation in domain and size. Data were collected through three workshops and twenty-three semi-structured interviews with thirty participants across a wide range of roles. Workshops used a structured template to elicit use cases, while interviews probed metric choices and challenges. The qualitative data were analyzed using thematic analysis to derive use-case classifications, metric catalogs, and practice patterns.

Results: Fourteen industrial use cases were identified and grouped into four application domains: document generation, data analysis and insights, customer service chatbots, and code generation. A set of 28 context-specific metrics currently used by practitioners was cataloged and linked to quality characteristics, including accuracy, correctness, and relevance. The study synthesized common workflows, such as Integrated Development Environment (IDE) code assistants, and highlighted key practitioner challenges, including the need for curated datasets and the computational cost of some research metrics.

Contributions: The study provides a set of industrial GenAI use cases and the metrics currently used to evaluate them. It contributes to a classification that distinguishes these context-specific industrial metrics from experiment-driven academic metrics. This classification offers guidance on the practical applicability, data requirements, and operational costs of different evaluation approaches. For industrial practitioners, the study provides actionable examples of metric usage in daily workflows and highlights practical blockers, such as dataset preparation and evaluation latency. Limitations include reliance on three companies and participant-reported practices.

1.3.4.4 Study D: Paradigm shift on Coding Productivity Using Generative AI

Study D, presented in Chapter 5, is titled "Paradigm Shift on Coding Productivity Using Generative AI." The study investigates the impact of coding assistants on user productivity and output quality in industrial projects.

Research objective: The objective was to identify factors that influence productivity with coding assistants and to distill adoption guidance.

Methodology: A multi-case study was conducted in two industrial projects within a large organization. The study employed a mixed-methods design, combining a *survey* to quantify usage patterns, satisfaction, and integration effort, with 17 *semi-structured interviews* to explore the qualitative factors behind these patterns. Data was analyzed using quantitative summaries for Likert-scale items and a stepwise thematic analysis for the qualitative interview data.

Results: The study found that while IDE integration required low effort, user satisfaction was complex; it tended to rise during the first few months of use but showed a negative correlation with a developer's own working experience, suggesting more senior developers were more critical of the outputs. Reported time savings averaged about 13% but were heavily concentrated on routine tasks like boilerplate code, refactoring, and explanations. Conversely, complex, domain-specific tasks or the generation of new feature code often require significant manual revisions, thereby negating productivity gains. Key challenges identified were the tools' limited context-awareness of the broader repository and a lack of support for project-specific design rules. Practices that improved outcomes included iterative prompt refinement and explicit specification of constraints. A need was identified for automated evaluation of generated code at scale to cover performance, security, and maintainability.

Contributions: Empirical findings are provided of adoption patterns, integration effort, and productivity factors for coding assistants in two industrial projects. Actionable findings specify where gains are likely (refactoring, explanation, and boilerplate) and where effort increases (new feature code and domain-intensive tasks). Adoption risks and friction points are closely tied to the need for automated quality checks and enhanced context integration. Limitations include a small sample size and a single-organization scope, which motivates replication and broader sampling.

1.3.4.5 Study E: A Framework for Evaluating GenAI Adoption and Use

Study E, presented in Chapter 6, is titled "A Framework for Evaluating GenAI Adoption and Use in Software Engineering" The study proposes a structured framework for introducing GenAI in software organizations, incorporating explicit quality evaluation and risk management.

Research objective: The objective was to identify industrial risk controls, measurable quality metrics, and decision gates during the GenAI adoption and usage.

Methodology: The study employed a *case study* methodology within two software development companies. A mixed-methods approach was used, combining 19 *semi-structured interviews* with *archival data analysis* of 15 documents and 184 internal web pages to capture adoption steps, quality concerns, and role responsibilities. These insights were synthesized to produce an evaluation framework. The framework's applicability was then verified through its implementation in a real-world industrial use case.

Results: The study empirically identified a three-phase adoption process practiced by industrial teams: *Ideation, Development, and Operation*. The findings detailed when and how quality evaluations (e.g., legal risk checks, output validation) occur in each phase. A key finding was the fragmented nature of quality ownership, which led to the identification of a need for a coordinating *GenAI Quality Lead* role to ensure accountability. The verification case (e.g., Unstructured Supplementary Service Data (USSD) XML generator) demonstrated the framework's feasibility, illustrating how its checkpoints and metrics support concrete decision-making throughout the lifecycle.

Contributions: The primary contribution is an evidence-based quality evaluation framework that aligns industrial GenAI adoption practices with ISO/IEC 25059. The study provides reusable artifacts, including checklists for legal and security intake, a role-responsibility matrix, and a monitoring plan template. It explicitly links secure enablement and risk mitigation (e.g., legal, security) to task-grounded quality evaluation. Limitations include the restricted organizational scope, motivating further trials in other domains.

1.3.4.6 Study F: Evaluating the Sufficiency of Single-Agent LLM Systems for Algorithmic Problem Solving

Study F, presented in Chapter 7, is titled "Evaluating the Sufficiency of Single-Agent LLM Systems for Algorithmic Problem Solving in Support and Operations." The study first quantifies how often small algorithmic patterns appear in industrial code, and then compares Single-Agent and Multi-Agent orchestration designs for solving corresponding LeetCode algorithmic problems, measuring acceptance, code structure, and cost (end-to-end generation time and token usage).

Research objective: The objective was to evaluate the effectiveness of Single-Agent LLM systems in supporting engineers in efficiently generating scripts of sufficient quality for self-contained algorithmic tasks in industrial contexts.

Methodology: A controlled experiment was conducted to evaluate the sufficiency of LLM agent systems in solving 150 LeetCode algorithmic problems (50 Easy, 50 Medium, 50 Hard) randomly sampled from three study plans among the top 25% most-liked LeetCode problems. Four frontier LLMs (GPT-5.1, Claude-4.5, Deepseek-chat-v3.1, and Gemini-2.5-pro) were evaluated in two setups: a Single-Agent direct-prompting system and a sequential four-agent chain (Problem Analyzer, Solution Designer, Code Executor, Solution Verifier). Metrics included LeetCode acceptance rate, cyclomatic complexity, lines of code, end-to-end generation time, and token usage, with three replications per model and setup and standard statistical tests applied for acceptance and code-quality comparisons.

Results: Across models and replications, both Single-Agent and Multi-Agent systems achieved high acceptance rates above 95%, and Fisher's exact tests showed no significant differences in acceptance between the two setups. Paired t-tests on

cyclomatic complexity and lines of code indicated only small, non-significant differences in code structure between Single-Agent and Multi-Agent solutions. In contrast, Multi-Agent orchestration increased latency and token usage by roughly four to six times on average.

Contributions: The study isolates AI agent orchestration as the treatment variable and provides controlled evidence that, for self-contained algorithmic scripting tasks, Single-Agent systems can match Multi-Agent systems in effectiveness and code quality while using fewer computational resources. It also empirically connects industrial repositories with LeetCode-style algorithmic problems by quantifying the prevalence of relevant algorithmic patterns in production code. The main practical implication is to recommend Single-Agent LLM solutions as the default for small algorithmic scripting tasks with clear oracles. Limitations include the algorithmic nature of the tasks, the focus on one organization's code base and four LLMs, and the fact that only one specific Multi-Agent architecture was evaluated.

1.4 Contributions, implications, and limitations

This thesis makes three research contributions (Section 1.4.1- 1.4.3) to the field of GenAI quality evaluation in software engineering. The contributions are synthesized from the findings presented in the included studies (Study A-F).

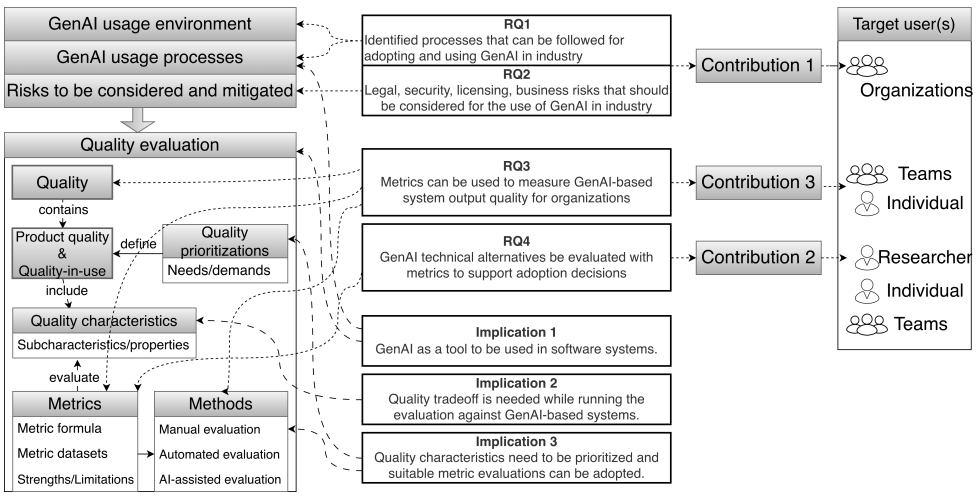

Figure 1.5: A mapping between the included study findings, thesis's research questions, contributions, and target users.

Figure 1.5 provides an overview that maps the findings from the included studies to the four research questions (RQ1-RQ4) and, subsequently, to the thesis's three main contributions. The figure illustrates how findings related to the *GenAI usage environment*, *usage processes*, and *risk mitigation* form the basis for addressing RQ1

and RQ2. Similarly, findings from the in-depth study of *Quality evaluation*, including its characteristics, metrics, and methods, are used to address RQ3. The evaluation of GenAI technical alternatives is addressed by RQ4. The figure further illustrates how these research questions are synthesized into the thesis's contributions. RQ1 and RQ2 are consolidated into **Contribution 1**, which focuses on organizational processes and risk management. RQ3, which covers metrics, is synthesized into **Contribution 3**, providing guidance for teams and individuals. RQ4, which addresses the evaluation of technical alternatives, forms **Contribution 2**, targeted toward researchers and development teams.

1.4.1 Contribution 1: A Model for GenAI Adoption and Use in Software Engineering

The first contribution presents a model that illustrates the process of adopting and utilizing GenAI in industrial settings.

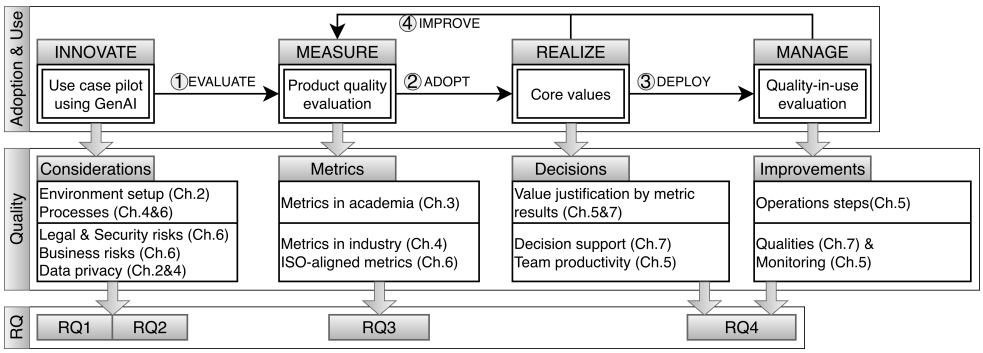

Figure 1.6: The **I**nnovate, **M**easure, **R**ealize, and **M**anage (**IMRM**) model (the top layer in this figure) for the adoption and use of GenAI in Software Engineering. The adoption and use are closely tied to tangible study results, including quality considerations, metrics, informed decisions, and resulting improvements. These findings are mapped to the thesis research questions (RQs).

As illustrated in Figure 1.6, the **IMRM** model comprises four stages: **I**nnovate, **M**easure, **R**ealize, and **M**anage [20]. The **Innovate** stage involves piloting use cases with GenAI, requiring initial considerations such as environment setup (Chapter 2) and risk assessment (Chapter 6). The **Measure** stage focuses on evaluating the product quality of the GenAI application using relevant metrics identified from both academic literature (Chapter 3) and industrial practice (Chapter 4), potentially aligned with ISO/IEC standards (Chapter 6). The **Realize** stage centers on adopting the GenAI solution based on core values justified by evaluation results (Chapters 5 and 7), supporting decisions related to technical choices and productivity impacts (Chapter 5) [7]. Finally, the **Manage** stage involves deploying the application and performing quality-in-use evaluations through ongoing monitoring and operational steps (Chapter 5) to drive improvements (Chapter 7) [7, 23]. This model connects the practical

steps of adoption and use to tangible considerations, metrics, decisions, and improvements derived from the studies included in this thesis [20]. The model maps these elements to the thesis research questions, illustrating how considerations such as environment setup (RQ1), risk management (RQ2), metrics (RQ3), and decision support (RQ4) fit into the overall adoption and use lifecycle (Figure 1.6) [20].

The IMRM model is designed for practical application. Teams and individuals can use this model as a structured guide for their daily work when adopting and using GenAI capabilities. It provides a clear, high-level process flow that helps ensure critical stages, such as initial risk considerations (Innovate) and metric-based evaluation (Measure), are consciously addressed. From the management perspective, it serves as a valuable checklist to structure, plan, and monitor the progress of GenAI initiatives from initial ideas to managed, operational systems.

This model's reliance on the 'Measure' and 'Realize' stages creates the necessity for Contribution 2, which provides details in connecting quality evaluation to specific decision support.

1.4.2 Contribution 2: An approach Connecting Quality Evaluation to Decision Support

The second contribution provides an overview of how quality considerations and metrics support decision-making when selecting GenAI technical solutions and identifying potential areas for improvement.

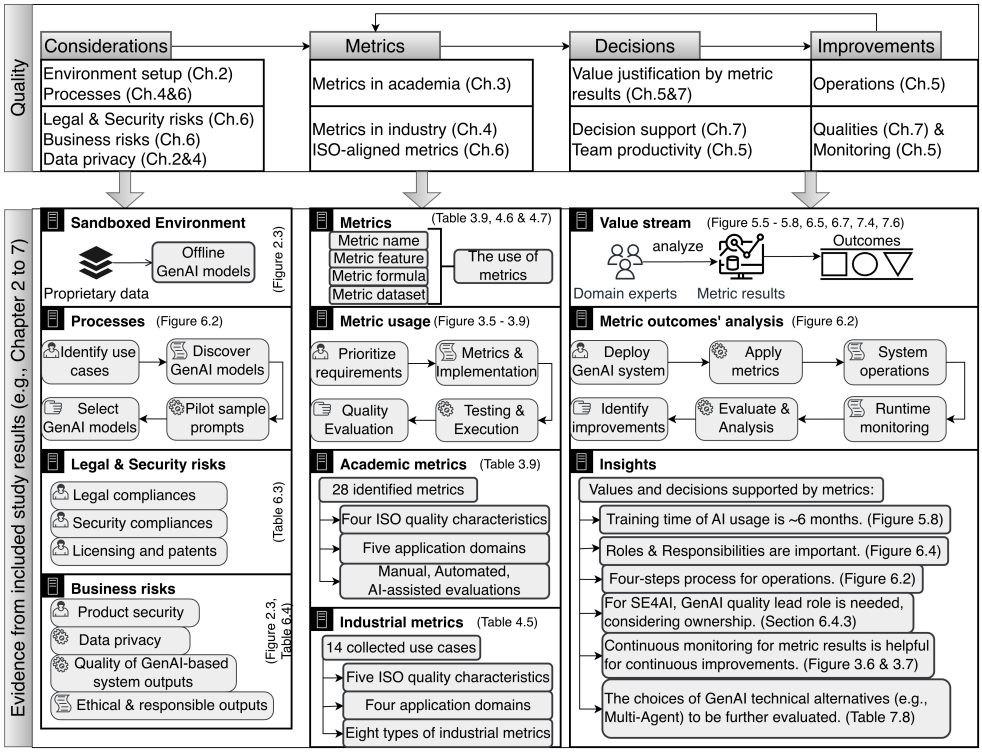

Figure 1.7: An overview of how quality considerations and metrics can be used to support decision-making of choosing GenAI technical solutions and identify potential quality improvements. 'Ch.X' stands for the Chapter ID in this thesis (e.g., Ch.2 means Chapter 2).

Figure 1.7 illustrates this connection [20]. Initial **Considerations** include secure environment setup (Chapter 2), defining adoption processes (Chapters 4 and 6), and assessing legal, security, and business risks (Chapter 6). These considerations inform the selection and application of **Metrics**, drawn from both academia (Chapter 3) and industry practice (Chapter 4), aligned with ISO/IEC standards where applicable (Chapter 6) [21, 23]. Metric usage involves prioritizing requirements, implementing measurement, and executing tests [20]. The results from metric evaluations feed into **Decisions**, such as justifying the value proposition of a GenAI solution (Chapters 5 and 7) or supporting choices between technical alternatives (e.g., RAG vs. Agent-based systems) (Chapter 7). Analysis of metric outcomes and domain expert input drives the identification of **Improvements**, which are implemented through operational adjustments (Chapter 5) and continuous monitoring (Chapter 5) [7]. This contribution highlights the practical *insights* derived from the included studies, such as the typical learning curve for GenAI tools (Figure 5.8), the importance of defined roles and responsibilities (Figure 6.4), the necessity of continuous monitoring for metric results (Figures 3.6 & 3.7), and the need for experimental evaluation of technical alternatives (Table 7.5).

However, for this decision-support to be effective, standardized and accessible

metrics are required. This establishes the need for a mapping of GenAI quality characteristics to metrics (See Contribution 3).

1.4.3 Contribution 3: Mapping GenAI Quality to ISO/IEC Standards and Metrics

The third contribution maps the concept of GenAI-based software quality to established ISO/IEC quality characteristics and connects these to concrete evaluation using metrics.

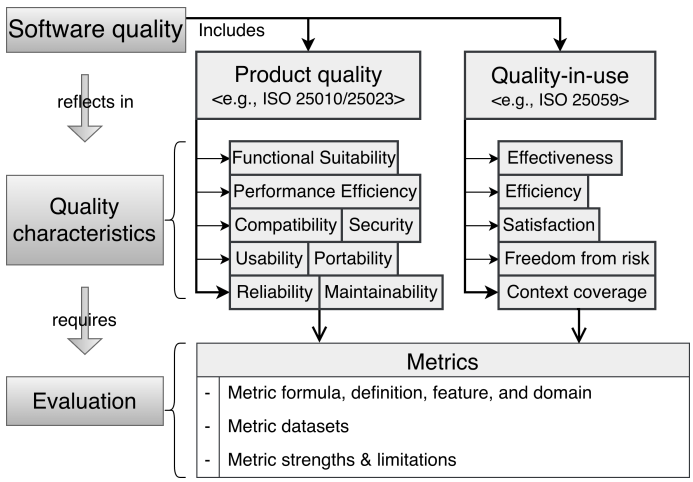

Figure 1.8: A mapping from GenAI-based software quality to ISO/IEC quality characteristics and quality evaluation using metrics.

As shown in Figure 1.8, **Software quality** for GenAI systems encompasses both **Product quality** (e.g., aligned with ISO/IEC 25010/25023) and **Quality-in-use** (e.g., aligned with ISO/IEC 25059) [21–23]. These high-level quality aspects are reflected in specific **Quality characteristics**, such as Functional Suitability, Reliability, Usability (for product quality), and Effectiveness, Satisfaction, Freedom from risk (for quality-in-use) [21, 23]. Evaluating these characteristics requires the use of **Metrics** [20]. This involves defining the metric formula, identifying relevant datasets, and understanding the metric's features, intended domain, strengths, and limitations (Chapters 3 and 4). This mapping provides a structured way for organizations to understand and measure the different facets of quality in their GenAI applications, grounded in internationally recognized standards.

1.4.4 Implication 1: GenAI as a Tool within Systems for Practitioners

The findings from this thesis position GenAI as a tool that can be adopted and used within software systems, as illustrated in Figure 1.9.

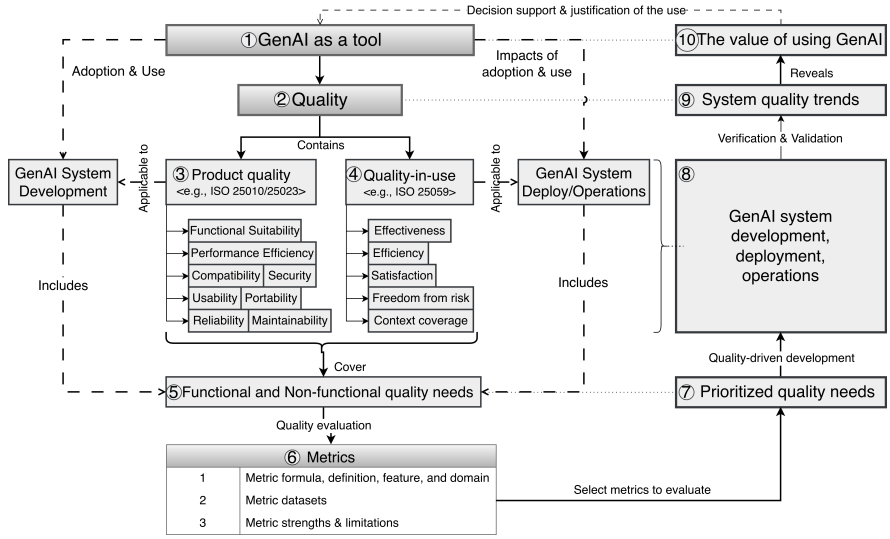

Figure 1.9: GenAI as a tool to be used in software systems. The GenAI-based system quality includes ISO/IEC product quality and quality-in-use across software development, deployment, and operations.

For practitioners, this tool-based perspective applies across *GenAI System Development* and *System Deployment and Operations*. The adoption and use of GenAI has direct impacts on *Quality*, which contains both *Product quality* and *Quality-in-use* [21, 23]. *Product quality*, aligned with standards like ISO/IEC 25010/25023, is primarily applicable during the development phase [21, 22]. This includes characteristics such as Functional Suitability, Performance Efficiency, Security, and Reliability [21]. *Quality-in-use*, aligned with ISO/IEC 25059, is applicable during deployment and operations [23]. This covers characteristics such as Effectiveness, Efficiency, Satisfaction, and Freedom from risk [23]. Both product quality and quality-in-use are intended to cover the *Functional and Non-functional quality needs* of the system. Through a real-world USSD use case verification (Chapter 6), quality trends for product quality and quality-in-use are measurable. This confirms that quality indicators can be tracked via monitoring dashboards to assess quality trends after deployment [27].

1.4.5 Implication 2: The Necessity of Quality for Practitioners

The second implication is that evaluating GenAI-based systems requires acknowledging and managing quality tradeoffs.

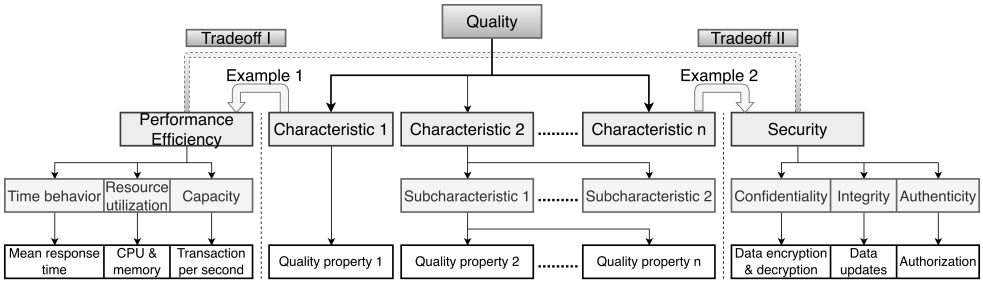

Figure 1.10: Quality contains high-level characteristics and sub characteristics, and each subcharacteristic consists of quality properties [21]. An example of 'Performance Efficiency' and 'Security' reveals the need for quality trade-offs from the evaluation perspective.

As illustrated in Figure 1.10, **Quality** is a multi-dimensional concept, composed of numerous high-level characteristics [21]. These characteristics, such as *Performance Efficiency* (Example 1) and *Security* (Example 2), can have competing goals. For instance, *Performance Efficiency* is broken down into subcharacteristics, such as time behavior and resource utilization, which are measured by properties like mean response time or CPU usage. In contrast, *Security* breaks down into subcharacteristics, such as confidentiality and integrity, which are measured by properties like data encryption or authorization. The figure indicates that tradeoffs exist between these characteristics, such as between *Performance Efficiency* and other characteristics (Tradeoff I), or between *Security* and others (Tradeoff II). A practical example is that maximizing *Performance Efficiency* by reducing response time might conflict with maximizing *Security*, as robust encryption and decryption processes (part of *Security*) inherently add computational overhead and increase response time. Therefore, practitioners can hardly enhance all quality characteristics simultaneously. They should make informed decisions, prioritizing certain characteristics over others based on the specific use case, risks, and business needs.

1.4.6 Implication 3: A Framework on Quality Evaluation for Researchers

The third implication is the need for a structured framework to guide the quality evaluation of GenAI-based systems, as shown in Figure 1.11.

This framework provides a five-step process [50], and is named as **ISPDE**, using the first letter of each step. The process begins with **Step 1: Identify quality characteristics** relevant to the system. It is followed by **Step 2: Select metrics**, where the appropriate evaluation methods are chosen based on metric types. Figure 1.11 illustrates that this selection is supported by an *evaluation scale* that aligns with the growing *quality evaluation demand*. This scale progresses from *Manual Evaluation* (high-context, nuanced understanding, small datasets) to *Automated Evaluation* (scalable, consistent results, medium/large datasets), and finally to *AI-assisted*

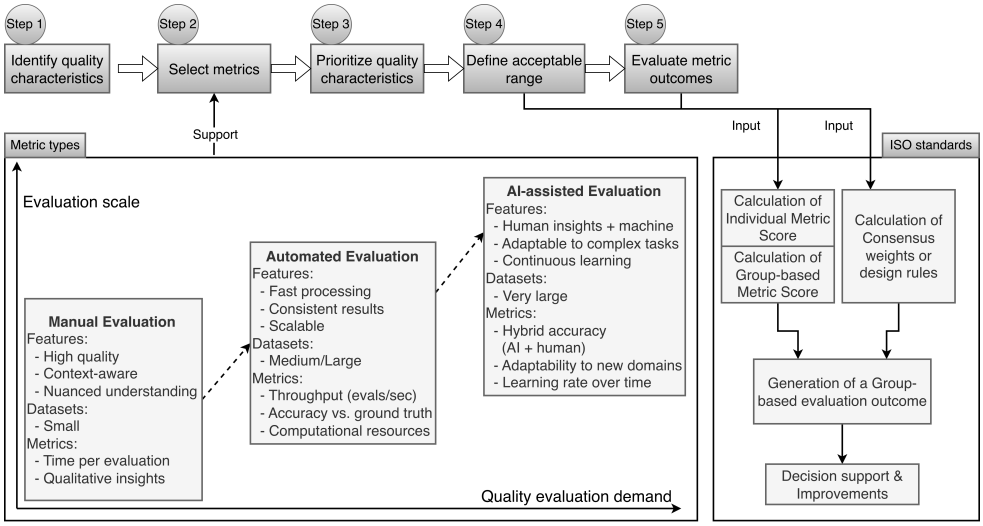

Figure 1.11: A five-step framework for researchers focused on the quality evaluation for GenAI-based systems.

Evaluation (hybrid accuracy, adaptable, large datasets). Next, in **Step 3: Prioritize quality characteristics**, practitioners must decide which qualities are most important for the specific context, linking back to Implication 2. In **Step 4: Define acceptable range**, acceptable performance thresholds for these prioritized metrics are set. Finally, in **Step 5: Evaluate metric outcomes**, the metric results are collected. These outcomes are used to calculate individual and group-based scores, which can be informed by inputs from *ISO/IEC standards* or consensus weights. The group-based evaluation outcome serves as *decision support* and improvements for GenAI systems. For the research community, the ISPDE framework highlights that quality characteristics should be prioritized, and suitable metric evaluations can be adopted.

It is important to clarify the relationship between the **IMRM** model (Contribution 1) and **ISPDE**. The IMRM model is a high-level process for GenAI adoption and use, synthesized from the empirical findings of this thesis (e.g., Study A-F). ISPDE is adapted from the work of Lavesson et al. [50] and detailed in the literature review (Study B). It serves as the detailed, tactical *implementation* of the *Measure* stage within the broader IMRM model. Its purpose is to provide a concrete, step-by-step method for researchers to select, prioritize, and apply metrics in alignment with ISO/IEC quality standards. Regarding models' verification, ISPDE is a framework derived from the literature. The IMRM model, synthesized from empirical observations of running processes and practices, has not yet been fully evaluated as a complete or unified model; such a holistic evaluation remains a subject for future work.

1.4.7 Limitations

Firstly, this thesis does not eliminate the non-deterministic nature inherent in GenAI models [12]. GenAI systems are probabilistic [13], and the research presented does not alter this fundamental characteristic. The evaluation frameworks (Study E) and metrics (Study B, C) are designed to evaluate the *outcomes* of this non-determinism, rather than removing the non-determinism itself. Furthermore, while these metrics can detect the presence of non-deterministic behaviors, they do not provide insights into the root causes of these behaviors or offer direct treatments to eliminate them. However, they do provide some measures for control and decision-making, such as determining whether a prototype meets the quality threshold required to proceed to the next stage of development.

Secondly, the research addresses the *symptoms* of GenAI problems rather than their root causes. For example, the metrics and processes (Study B, C, E) are designed to evaluate and identify issues such as inaccurate or low-quality outputs. However, they do not modify the underlying statistical models or the root causes of *why* these models produce such outputs. The focus is on quality assurance of the *application*, not on fundamentally changing the (black-box) GenAI models themselves.

Thirdly, it is essential to note that our model/framework does not apply in several organizational contexts. For example, *small organizations* (< 50 employees) may lack resources for the dedicated GenAI quality lead role and formal evaluation processes. *Highly regulated industries* (e.g., healthcare) may require additional compliance frameworks beyond our ISO-aligned approach. *Research-focused organizations* may need different evaluation criteria than the productivity-focused metrics we identified.

1.5 Discussions

This section discusses the findings of the thesis.

The relationship between risks and rewards. The research findings suggest a clear relationship between perceived risk, potential reward, and the maturity of GenAI adoption, as illustrated in Figure 1.12.

Organizations typically do not begin with high-impact, external-facing applications; instead, they follow a path that builds business confidence over time. This is supported by the findings in Study E, which identifies an *Ideation* phase where initial pilots are conducted. These initial use cases are often internal, such as building secure information retrieval tools (Study A) or developer assistants (Study D), which represent a low-risk, low-reward environment. The “risk” in this context is not only technical (e.g., hallucinations) but, as Study E emphasizes, is dominated by legal, security, and licensing concerns in the early stages. By managing these risks

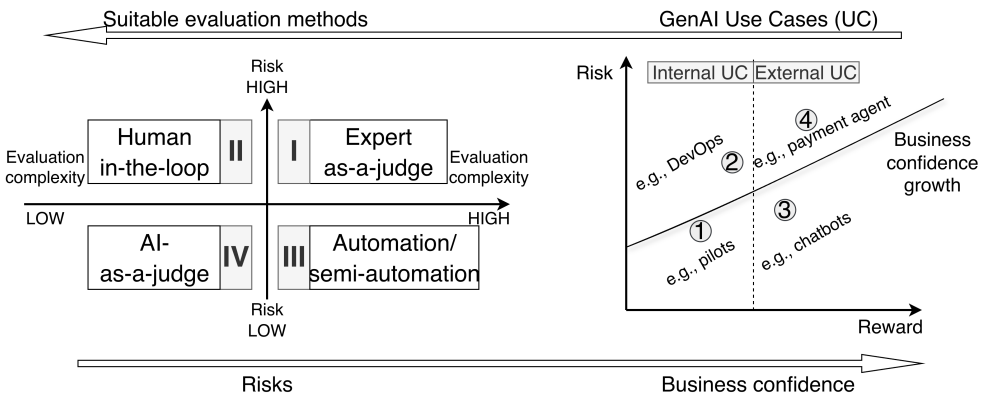

Figure 1.12: Based on GenAI use cases, using suitable evaluation methods to mitigate risks. Starting from small internal use cases to gain experience in handling different levels of risks and grow business confidence for external use cases.

in controlled, internal settings (Study A), organizations gain the necessary "business confidence" to pursue high-reward applications, such as external-facing chatbots or payment agents (see the right part of Figure 1.12). Therefore, the path from internal to external use cases is gated by an organization's ability to first evaluate and mitigate these foundational risks.

Selecting evaluation methods to deal with risk. The choice of evaluation method (Implication 3, Figure 1.11) is directly linked to the risk and complexity of the GenAI use case, as shown in the quadrants of Figure 1.12. For tasks with **high risk but low evaluation complexity** (Quadrant II), such as retrieving information from sensitive proprietary documents (Study A), a **Human-in-the-loop** approach is suitable. Here, the task is simple (reading an answer), but the risk of data leakage or factual error requires direct human oversight. For tasks with **high risk and high evaluation complexity** (Quadrant I), such as generating domain-specific code for a production system (Study D), an **Expert-as-a-judge** is required. This aligns with the findings in Study D, where senior developers with high validation capabilities were needed to assess complex code, and with the proposed *GenAI Quality Lead* role in Study E [7]. For tasks with **low risk but high evaluation complexity** (Quadrant III), such as the experimental testing of algorithmic code generation (Study F), **Automation/semi-automation** is the most effective method. In this scenario, the evaluation itself (running 133 problems, three replications, and calculating metrics) is complex; however, the risk is contained within the experiment, making automation an ideal solution. Finally, for tasks with **low risk and low evaluation complexity** (Quadrant IV), such as evaluating boilerplate code (Study D), an **AI-as-a-judge** or AI-assisted evaluation becomes a viable option [7, 20].

Quality evaluation contributes to business confidence We argue that quality evaluation is not merely a technical checkpoint, but a way for building **Business confidence** (Figure 1.12). This confidence enables an organization to transition from low-reward internal pilots to high-reward external products. Confidence begins with mitigating foundational risks, such as the legal and security concerns identified in Study E, by using sandboxed environments as demonstrated in Study A. Confidence grows by making tradeoffs visible and justifiable, as Implication 2 suggests. For example, the experiment in Study F provides the evidence to be *confident* in selecting a single-agent solution for algorithmic scripting tasks in Support or Operations. The metrics identified in Study B and C are the vocabulary used to articulate this confidence [20]. Without metrics for quality, reliability, and risk (Implication 3, Figure 1.11), business confidence remains subjective and fragile. Therefore, a mature quality evaluation process (Contribution 1, Figure 1.6) is the mechanism that transforms the *potential* of GenAI into tangible, trusted, and deployable business value.

1.6 Thesis summary

This thesis investigates the adoption and quality evaluation of GenAI usage in industrial software engineering settings. The research focuses on how organizations can ensure quality, manage risks, and make informed decisions when integrating GenAI technologies into their software systems. Employing a multi-methodological approach, the thesis combines insights from case studies (Studies A, C, D, E), a snowballing literature review (Study B), and a controlled experiment (Study F). The primary contributions include the IMRM model for GenAI adoption and use (Innovate, Measure, Realize, Manage), an approach that connects quality evaluation practices to decision support, as well as a mapping of GenAI quality to ISO/IEC standards and actionable metrics. Our implications highlight GenAI's role as a tool that necessitates careful quality management, the need to navigate trade-offs between competing quality characteristics, and the value of a structured evaluation framework (e.g., ISPDE). The research also identifies essential processes for secure GenAI usage, methods for mitigating business risks, and provides evidence-based guidance on selecting metrics and evaluation methods based on the quality evaluation demands. While acknowledging limitations, such as addressing symptoms rather than root causes of the output quality of GenAI systems, the thesis provides practitioners with practical metrics, methods, frameworks, and insights to support the responsible adoption and use of GenAI in software engineering.

Study A

Experience with Large Language Model Applications for Information Retrieval from Enterprise Proprietary Data

Abstract

Large Language Models (LLMs) offer promising capabilities for information retrieval and processing. However, the LLM deployment for querying proprietary enterprise data poses unique challenges, particularly for companies with strict data security policies.

This study shares our experience in setting up a secure LLM environment within a FinTech company and utilizing it for enterprise information retrieval while adhering to data privacy protocols. We conducted three workshops and 30 interviews with industrial engineers to gather data and requirements. The interviews further enriched the insights collected from the workshops.

We report the steps to deploy an LLM solution in an industrial sandboxed environment and lessons learned from the experience. These lessons contain LLM configuration (e.g., chunk_size and top_k settings), local document ingestion, and evaluating LLM outputs.

Our lessons learned serve as a practical guide for practitioners seeking to use private data with LLMs to achieve better usability, improve user experiences, or explore new business opportunities.

2.1 Introduction

Large language models (LLMs) have revolutionized tasks in healthcare and finance [51]. LLMs are artificial intelligence models trained on vast amounts of text data, capable of understanding and generating human-like text across various topics and tasks. Commercial LLMs like ChatGPT can understand the context and generate human-like text [52], with advanced features such as document summarization [53] and code generation [54].

However, while LLMs possess broad domain knowledge, directly applying ex-

isting commercial LLMs in industrial contexts is challenging, as companies operate on frequently updated private data that is not commonly available and, therefore, outside the LLMs' pre-trained data. This company-specific data needs to be processed by the LLMs to provide more contextual and domain-specific information retrieval for company users and make the LLMs applicable in specialized contexts — a capability that out-of-the-box LLMs generally lack without access to such data.

While fine-tuning [1] LLMs on private data is one potential solution, it raises security concerns. Recent research has shown risks of LLMs inadvertently exposing pre-trained data [55]. Companies seek to deploy local LLMs within their secure infrastructure to mitigate these risks and maintain complete data control. Existing studies, such as Zhao et al. [14] and Wu et al. [55], have shown that deploying LLMs in industrial settings introduces additional challenges. LLMs lack long-term memory capability, causing some acquired knowledge to gradually be "forgotten" over time and with continued training [19]. Furthermore, LLMs could produce incorrect or ambiguous answers to factual questions in open domains, especially when specific numerical values or dates are involved [14].

To address these issues, researchers have developed the retrieval-augmented generation (RAG) approach [56]. RAG combines information retrieval and LLM text generation, first retrieving relevant context data from up-to-date external sources (e.g., private data vectorized in a local vector database) before generating responses [56]. The mentioned "irrelevant or incorrect LLM response generation" can be mitigated by offering precise original quotes or references from enterprise private data.

Our study focuses on using local LLMs in a sandboxed environment for information retrieval based on private enterprise data. A sandboxed environment refers to a self-contained virtual space with a restricted network connection where LLM inferences run in isolation, preventing external sharing of sensitive data. Our research aims to address the following questions:

1. What is a suitable solution to set up a sandboxed LLM environment with private enterprise data?

2. What are the required steps to deploy and use a private sandboxed LLM environment?

3. What are lessons learned from deploying and using a sandboxed LLM environment for information retrieval?

This paper reports our experience in developing a suitable LLM solution at the architecture level for private enterprise data, deploying LLMs for information retrieval within an industrial sandboxed environment, and lessons learned. Empirical results show that the sandboxed LLM environment with the deployed LLMs is applicable and valuable in an industrial use case.

[1] https://platform.openai.com/docs/guides/fine-tuning

2.2 Related work

Prior studies demonstrate that LLMs, such as BERT, can comprehend user queries and document semantics for enhanced retrieval [3]. BERT and its variants have improved the understanding of user query context and semantics, leading to more accurate information retrieval.

The retrieval-augmented generation (RAG) approach [56] represents an advancement in addressing LLM limitations. By retrieving relevant context from external sources before generating responses, RAG can improve the factual accuracy and relevance of generated texts.

The deployment of LLMs in enterprise settings introduces challenges concerning privacy and data sensitivity [55]. Wu et al. [55] discuss the risks associated with data leakage and unauthorized access when using cloud-based LLMs. They propose several privacy-preserving techniques to mitigate these risks, such as differential privacy and federated learning.

Recent studies have explored the application of LLMs in various industrial domains. Researchers, such as Zhao et al. [14] and Wu et al. [55], highlight challenges LLMs face with constantly changing data in industrial settings. These studies highlight that it is crucial to employ robust privacy-preserving techniques in enterprise deployments. Continuous domain-specific embedding is also necessary to handle evolving data in industrial contexts.

Despite these valuable insights, there is limited research on the actual deployment process and the challenges encountered in real-world scenarios. We developed a solution and shared lessons learned, contributing to the growing body of knowledge on secure LLM applications in privacy-sensitive domains.

2.3 Context and approaches

We documented our experiences from a case in a FinTech company.

2.3.1 Case company

The case company provides financial services linked to a telecommunications system. It is a medium-sized enterprise with around 500 employees. The company is in rapid growth, with operations spanning four regions: the Middle East, Africa, Europe, and North America. The Research and Development (R&D) department is primarily based in Sweden and India. Our study focused on the Sweden branch, which employs approximately 150 engineers.

The product developed in the company is a mature business-to-business (B2B) financial platform that has successfully served customers for approximately two decades. This full-stack platform with both front-end and back-end applications offers a suite

of financial services and solutions to meet the diverse needs of businesses. The product's architecture has been transformed into micro-services running in Kubernetes clusters, which aims to improve teams' collaboration and productivity.

2.3.2 Participants

The participants in this study included 34 individuals from various roles, including software developers, managers, architects, security engineers, operations engineers, and document engineers. The number of participants in each role is between four and eight, and the working experience of participants ranges from three to 21 years. Table 2.1 summarizes the participants' roles and experiences.

Table 2.1: Participants from the case company

Index	Participant role	NO. of participants	Working experience (years)
1	Software developer	8	4 - 21
2	Manager	5	5 - 20
3	Software architect	7	7 - 18
4	Security engineer	5	3 - 15
5	Operation engineer	5	4 - 19
6	Document engineer	4	3 - 12

2.3.3 Challenges and security requirements

The rapid expansion of the case company's development teams and its architectural transformation have highlighted the importance of knowledge sharing among teams and team members, revealing two key **challenges**:

- *Documentation overload*: The vast number of files documenting product information makes keyword searches and manual document reviews time-consuming. This is particularly crucial when comparing different product versions to identify changes or differences, as creating a dependency map for specific feature implementations becomes resource-intensive.
- *Knowledge concentration*: Key knowledge and competencies are concentrated among a small group of experienced developers who are often occupied with high-priority tasks such as architecture-related work and customer trouble reports. This leaves them with little time for knowledge sharing, and the geographical distance of new teams hinders efficient knowledge transfer.

Given that the case company is in the financial business domain, two **security requirements** must be fulfilled:

1. *Data access control*: Measures must be in place to prevent unauthorized access to or leakage of the company's data. This ensures that sensitive information is not shared with third parties without authorization.

2. *Data isolation*: Mechanisms should be established to prevent the accidental duplication or storage of the company's private data within external systems or services, such as online LLM providers, without explicit consent. Maintaining data privacy and preventing unauthorized use or distribution of the company's proprietary information is essential. Study participants suggested implementing a restricted and isolated environment.

2.3.4 Data collection

We used both workshops [57] and semi-structured interviews [32] for data collection. The data collection process is summarized in Fig. 2.1.

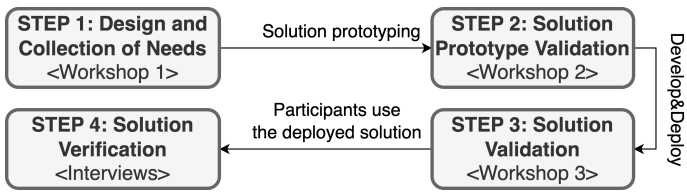

Figure 2.1: Data collection process overview

Workshops (**STEP 1**, **STEP 2**, and **STEP 3**) were used to collect the needs and experiences of using LLMs. Three workshops were managed, and each workshop lasted about four hours. Interviews (**STEP 4**) were conducted to gather complementary data and detailed insights into the LLM usage collected from the workshops. Thirty semi-structured interviews, including both online and face-to-face meetings, were performed. The duration of the interviews varied between 42 to 54 minutes. Due to unexpected conflicting priorities, some participants could not join the interview meetings as they had planned. In total, 30 interviews were successfully concluded.

The workshops and interviews were organized and conducted by the paper's first author, who is also employed as a software developer at the case company. The sessions were recorded to guarantee that no significant information was missed and to allow the researchers to review the discussions if necessary.

The workshop design followed three structured steps. In STEP 1, feedback on potential LLM solutions was gathered based on the participants' working experience and needs. STEP 2 focused on prototype verification, including aspects such as design, requirements, execution environment, and user interfaces of an LLM solution, which was developed based on the feedback from STEP 1. In STEP 3, the prototype was implemented and deployed in a sandbox environment. During this step, the LLM solution was introduced with a live demonstration, and participants were given the opportunity to test the deployed LLM. Their feedback was collected to refine the solution further.

The interview process included a preparatory phase that began about two weeks

before the interviews. Participants received reminders to try the sandbox solution, along with LLM test accounts and a manual guide. This ensured that they were familiar with the tools and comfortable with the LLM application before the interview.

During the interview, each participant was asked open-ended questions, available at `https://tinyurl.com/vvLLM`. Topics included current challenges and security concerns of LLM solutions, deployment strategies, effectiveness evaluation of LLM outputs, and lessons learned. Participants were encouraged to provide detailed responses, including specific examples of their experiences, reflections on using LLMs, and any additional comments or questions they had about the technology and its implementation.

2.3.5 Data analysis

We analyzed and synthesized the collected data to identify patterns and themes.

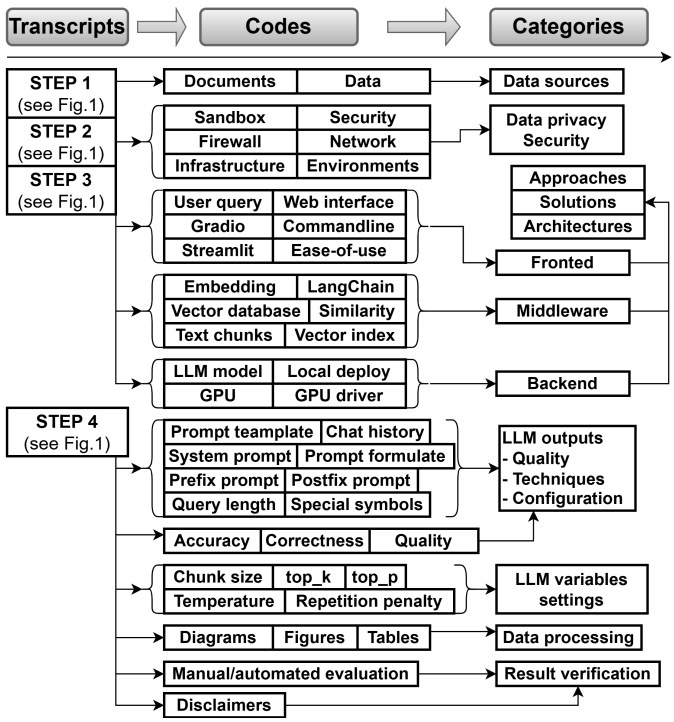

Figure 2.2: Data analysis process illustrating codes were derived from the workshop and interview transcriptions, and high-level themes were synthesized from the codes.

As shown in Fig. 2.2, our raw data came from the transcriptions of workshops and interviews conducted in STEP 1, 2, 3, and 4. These transcriptions were automatically generated using the transcription feature in Microsoft Teams.

Based on the transcribed texts, we applied semantic coding [48]. First, each transcript was read thoroughly to understand the context, and segments of text rep-

resenting pieces of information were highlighted and assigned an initial code. Next, these initial codes were reviewed and refined to ensure consistency and clarity, with similar codes merged and organized into a hierarchical structure. Finally, the codes were grouped based on commonalities to form broader themes [48]. For instance, all codes related to participants' insights gained from verifying the LLM solution were grouped into a theme called "Result verification."

2.4 Results

Our research questions are addressed in this section.

2.4.1 What is a suitable solution to set up a sandboxed LLM environment with private enterprise data?

The requirements summarized by the study participants, though specific to the case company's context, highlight important considerations for organizations looking to integrate LLMs into their systems. Three key areas were identified: a) the need for an easy-to-use web or command-line interface where users can type their prompts, select a target vector database, and submit the prompts; b) the use of vector databases to store private data locally for security purposes; c) the selection of preferred LLMs for the sample prompt, with the requirement that these models provide quotes or reference the original texts.

The core functionality of the LLM in this context is to retrieve relevant information from various sources, such as software product documents, application interfaces, development requirements, and operational information, based on user prompts. The participants' expectations for modular architecture and containerized deployment reflect a broader trend towards microservices and cloud-native architectures, which offer flexibility, scalability, and ease of integration with existing systems. These findings underscore the importance of designing LLM architectures and solutions that are abstract and modular enough to meet diverse organizational requirements. While specific details may vary, principles of ease of use, configurability, seamless integration, and effective information retrieval are widely applicable in industrial contexts.

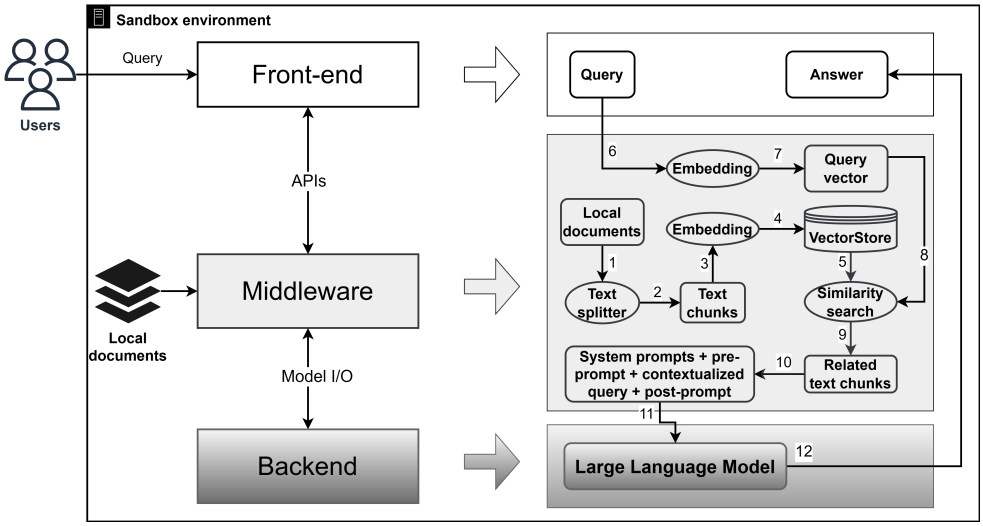

Figure 2.3: A proposed solution about the use of LLMs for information retrieval based on provided industrial data in a sandboxed environment.

Based on participants' feedback, a proposed solution was developed, as shown in Fig. 2.3. This solution comprises three modules:

1. **Front-End**: A web interface that allows users to submit queries and receive LLM outputs as answers.

2. **Middleware**: This layer processes data by parsing and splitting documents into text chunks, embedding these chunks into a vectorized format stored in a vector database. When users submit queries, the queries are transformed into query vectors used for similarity searches in the vector database. The identified text chunks are then combined with system prompts and pre-/post-query prompts for LLM inference.

3. **Backend**: It runs local large language models within an industrial sandbox environment to ensure data security.

Comparing this proposed solution with existing online LLM approaches reveals several advantages. First, it offers full data control, as data is managed locally rather than relying on external service providers. Second, it ensures data security by preventing data leaks onto the internet, making it more suitable for enterprises concerned about data privacy. Third, it provides transparent data processing, with the middleware layer operating as a white box that visualizes each step of data processing. In contrast, online LLM services operate as black boxes, limiting transparency and the explainability of outcomes.

2.4.1.1 Solution implementation

The proposed solution was implemented using Python and PyTorch libraries. Detailed code implementation has been published in GitHub: `https://github.com/Liang-Y-Yu/vvLLM`.

Our implemented LLM solution includes several key interfaces. Firstly, it requires downloadable models that are compatible with Python Torch, such as Llama. It supports Chroma and FAISS vector databases for efficient similarity searches and offers a configurable tokenization process for document splitting. The solution also allows for the customization of system prompts to cater to different application requirements, such as chat or instruction-based tasks.

Additionally, it supports the submission of prefix and postfix query prompts to guide the model's responses. The solution offers a range of adjustable LLM variables to enhance output, including chunk size, top_k (which controls the most probable tokens during text generation), top_p (which limits output to the smallest tokens exceeding a probability threshold), repetition_penalty, and sampling. These features collectively enhance the flexibility and configurability of the proposed LLM solution.

2.4.2 What are the required steps to deploy and use a private sandboxed LLM environment?

In total, six steps were employed to deploy and utilize a private sandboxed LLM environment, as detailed in the subsequent subsections.

Step 1: Prepare sandbox environment. A sandbox environment was used to ensure enterprise data privacy.

- **Data security**. LLM inferences are executed in a virtual machine with no internet connection and restricted port access. This setup allows users to query and pass information to LLMs without enterprise data going through third parties.

- **Industrial use case simulation**. In this sandboxed environment, the hardware includes a 16-core CPU, four Tesla T4 GPUs totaling 64GB, 128GB of memory, and a 500GB disk. The software comprises Ubuntu 20.04 LTS, Nvidia CUDA 12.3 GPU drivers, and Python 3.10. Security measures include restricted internet access and a firewall with only open ports 22, 443, and 80.

Step 2: Install required software packages. The required packages contain Gradio UI — a web interface; LangChain — a framework for processing source files; and Python Torch — a library for developing LLMs.

```
# Clone the implemented solution from GitHub
git clone https://github.com/Liang-Y-Yu/vvLLM
cd vvLLM
# Create a Python virtual environment inside the cloned folder
python3 -m venv venv
```

```
# Install required packages, including Python Torch, Langchain, Gradio UI, ChromaDB
./install_deps.sh
```

Step 3: Collect source files. The source files comprised product information documents, and PDF and XML formats were provided during system implementation.

Study participants reported that higher accuracy results were achieved while using the PDF files, based on their manual evaluation after trying all the provided file formats. As such, PDFs were selected as the primary source format for this research based on that feedback. All source files were manually uploaded to the sandbox environment through secure SSH connections. The files were stored in a dedicated "source_files/" folder for isolation and easy access.

Step 4: Start the LLM solution. The executable file below was used to run the implemented LLM solution in the sandbox environment.

```
# LLM system prompts
system_prompts="""
You are a helpful, respectful, and honest assistant. Please ensure that your responses are socially unbiased and positive in nature.
If a question does not make any sense or is not factually coherent, explain why instead of answering something that is not correct.
If you don't know the answer to a question, please don't share false information.
"""
# Query prefix prompts
query_prefix_prompt="""
Do not give me any information that is not mentioned in the PROVIDED CONTEXT. According to the information in the document sources provided within the context above,
"""
# Enable offline datasets
HF_DATASETS_OFFLINE=1 \
python generate.py \
  # A selected LLM model name
  --base_model=<LLM_MODEL_NAME> \
  # Vector databases
  --langchain_modes="['DB1','DB2']" \
  # Local enterprise private files
  --user_path=source_files --local_files_only=True \
  # Size of chunks while splitting documents
  --chunk_size 512  \
  # Add system and query prompts
  --system_prompt="${system_prompts}" --prompt_query="${query_prefix_prompt}$" \
  # LLM configuration
  --top_k_docs=5 --do_sample=True --top_p=0.95 --top_k=40 --repetition_penalty=1.1
```

Step 5: User prompts. Due to non-disclosure agreements signed within the company, we cannot provide the exact queries. Instead, we utilized publicly available questions and obtained the LLM responses:

```
Based on the provided context, the main security concerns are:
```

```
1. Privacy risks: It does not provide sufficient methods to preserve
    personal data according to GDPR, and it may share users' data with third
    -party entities without explicit user permissions.
---
Sources [Score | Link]:
0.9 | user_path/wu2023unveiling.pdf
Section 4.2. Privacy risks in ChatGPT It is not
Total Time: 1.7s
End Sources
---
```

These responses demonstrate that the model processes the provided context rather than relying on pre-trained data, adhering to the system requirement for data isolation. Relevant chunks from the source files are included with similarity scores, allowing users to review the assessed sections and understand the LLM's reasoning process.

However, users should exercise caution when validating the retrieved chunks, as their ability to determine the suitability of the source material may be limited by their knowledge and understanding. By directly quoting the sections, transparency and explainability are enhanced, but users are encouraged to critically evaluate the retrieved chunks and generated text, seeking additional expertise or clarification when needed. This example indicates that the solution conforms to specifications and helps mitigate company challenges.

Step 6: Post human evaluation. The final step involves the manual evaluation of LLM outputs, where human reviewers validate two key criteria. First, they assess the precision of the LLM's response, determining whether it directly answers the user's question based on the context provided. Second, they check the relevance of the quoted texts from source files to ensure they appropriately support the generated text. This process confirms that the model grounds its responses in the ingested proprietary data rather than fabricating information.

Consequently, approximately 65% of the participants reported that the sandboxed LLM application was helpful while retrieving product information for their daily development tasks. This feedback indicates that these six steps resulted in a practical and valuable process for supporting the needs of the case company.

2.4.3 What are lessons learned from deploying and using a sandboxed LLM environment for information retrieval?

Lesson learned 1

A local LLM deployment requires expertise in LLM mode type, size, and required infrastructure resources, and skills for data processing (e.g., splitting documents into chunks stored in the Vector database).

To effectively deploy local large language models (LLMs) for enterprise infor-

mation retrieval, certain key technical knowledge is essential. Engineers must understand the capabilities of existing LLMs for text generation and be familiar with open-source options such as GPT-2, Llama-2, and Mistral. For example, some participants found that GPT-J and GPT-Neo lacked the semantic search capabilities needed to effectively retrieve company information from internal documentation based on their hands-on experiences.

Another crucial aspect is understanding embeddings, which involves creating vector representations of texts that capture semantic meaning. For instance, an embedding model like BERT can identify semantic similarities between phrases such as "happy" and "glad."

Additionally, mastering vector databases is vital. Databases like Faiss and ChromaDB are used to add, store, and query high-dimensional vector data consisting of vectors with many parameters. It is important to know how to index and retrieve document vectors from these databases to build high-performance search indexes.

Lesson learned 2

LLM may produce "wrong" results for those prompts that contain only a few words or abbreviations.

We found that short prompts, such as an ID, an abbreviation, or a product name, do not effectively match chunks in a vector database based on word vector similarities. This poor matching can lead to inaccurate LLM inferences and sometimes generate completely irrelevant content. For instance, when we provided the LLM with an abbreviation like "MMM" found in the enterprise data, it generated unrelated content, making it unsuitable for the intended purpose.

To address this issue, we implemented actions: I) We increased the length of the query statement by adding descriptive words before "MMM"; II) We used special symbols, such as quotation marks and dollar signs, around the word, for example, "MMM", to indicate that the LLM that "MMM" should be matched precisely, not based on similarity.

Study participants confirmed that these actions mitigate issues with short prompts. While future LLMs may develop a more robust semantic understanding to handle short prompts better, users must provide sufficient context and formulate queries effectively. Even with advanced LLMs, ambiguous prompts without appropriate context can still lead to irrelevant outputs. However, users may not always know how to expand a prompt statement or remember to use symbols, resulting in poor LLM outputs.

Our experience suggests that developing an actionable handbook or user guide can be beneficial. This guide should educate LLM users on best practices for prompt formulation, including techniques for expanding prompts with relevant contextual information and using special symbols or formatting when necessary.

Lesson learned 3

The accuracy of LLM outputs is essential for users' trust. Providing well-defined prompts can improve output quality.

Our studied project is a financial product, so inaccurate outputs can seriously affect business settings. For example, inaccuracies can lead to misinformation if an LLM is used for tasks like retrieving system components' API data.

Study participants emphasized that even a 10% rate of "incorrect" LLM results could lead to a significant loss of customers' trust. Based on participant suggestions, we prepared a list of well-defined prompts based on customers' feedback as templates for the web interface of the LLM solution. This way, the chance of producing inaccurate LLM outputs due to prompt formulation can be reduced. Further research is required in this case to continuously improve user satisfaction with better prompt engineering and advanced information retrieval techniques.

Lesson learned 4

Chunk size and top_k settings should be configurable for different industrial cases since they impact the quality of LLM outputs.

As shown in Fig. 2.4, the data processing chain in the locally deployed LLM solution involves splitting the original document or data into chunks, retrieving these chunks through semantic similarity based on input prompts, and generating responses using the top_k chunks. The LLM selects text chunks with specific scores before using them for response generation.

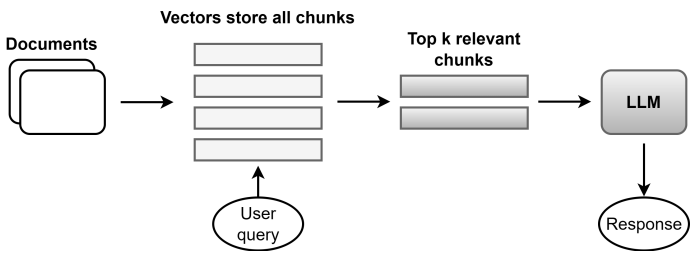

Figure 2.4: An overview of the data process chain in the locally deployed LLM solution

This process, illustrated as "Top k relevant chunks" in Fig. 2.4, may result in information loss, as not all data can be preserved in the LLM outputs. Important chunks from the original documents may be excluded due to the chunk size and top_k limits, causing critical information to be missed in the final responses.

In our tests, we evaluated over ten different combinations of chunk_size and top_k values. The results showed that the best user satisfaction was achieved with a chunk_size of 578 and a top_k value of 50, based on manual verification by study par-

ticipants using the deployed LLM solution. To enhance user experience, we provided web interfaces allowing users to configure chunk_size and top_k values in real-time according to their needs.

Our experience suggests practitioners should experiment with LLM inference to identify suitable chunk_size and top_k values for their specific business use cases. These settings can be discovered through a series of iterative trials.

Lesson learned 5

UML diagrams in source files should be parsed separately to mitigate information loss from LLM outputs.

In our case, almost all source documents contain a combination of text and images. Extracting only the text results in the loss of significant contextual information. Furthermore, directly converting images to vectors is ineffective for subsequent information retrieval using LLMs. To address this challenge, we used the Document Text Recognition (docTR[2]) library to parse the textual information, localizing and identifying each word within the provided documents.

It is important to note that many other Optical Character Recognition (OCR) libraries are available. However, we did not test these alternatives due to resource and time constraints. We recommend additional validation of these libraries to explore their effectiveness.

Lesson learned 6

The inclusion of chat history and user prompts can improve LLM outputs by improving its comprehension of the industrial context.

Another important aspect of developing a reliable LLM solution that can respond effectively to multiple instances of a single query involves considering the chat history. This consideration is essential for accommodating follow-up questions related to the context of the previous dialogue. The issue is addressed by incorporating the chat context in conjunction with the user prompts, as illustrated in Fig. 2.5.

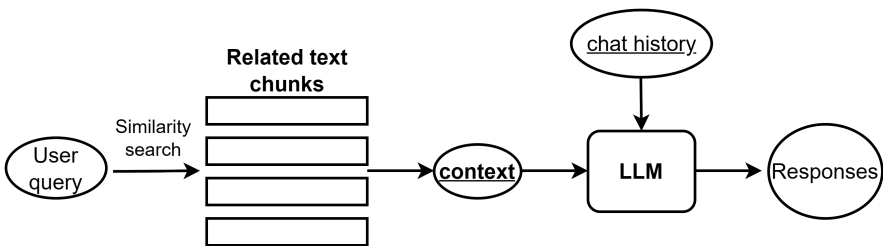

Figure 2.5: An illustration of combining chat history with contextualized user query and sending to LLM inference

[2]https://github.com/mindee/doctr

Additionally, we have implemented a radio button into the web interface for users who prefer not to include chat history. This allows users to turn off this feature when it is not required for specific use cases. When this feature is disabled, users' chat history will be excluded from LLM inferences, and users can choose to either store or delete their chat history.

Lesson learned 7

Automated evaluation for LLM outputs is needed, especially for a large number of user queries.

Manual evaluation of LLM outputs can be time-consuming and impractical, especially when the number of user queries is large. Based on participants' feedback, automated evaluation may cover the following areas:

- Factual accuracy [58]. The correctness of generated LLM outputs against a prompt requires domain-specific knowledge to verify the result.
- LLM output relevance [52]. How well LLM-generated outputs relate to the provided industrial data.
- Context relevance [58]. Context refers to the information in the provided data source files collected from industrial cases.

More empirical studies are expected to investigate automated evaluation in real industrial cases.

Lesson learned 8

Disclaimers are useful to remind users to self-evaluate LLM responses.

Disclaimers that remind users to verify the outputs of LLM are important, as relying solely on these outputs can be risky, particularly in financial business domains. The value of LLM outputs is inversely correlated with the receiver's ability to validate them.

Study participants emphasized the importance of displaying disclaimers alongside LLM responses to raise awareness of model limitations and uncertainties. However, prominent disclaimers alone may provide false reassurance if users do not rigorously validate the outputs against source data and subject matter expertise.

For instance, in one product information query about "how to transfer money from one account holder to another" within the system in the case company, the LLM conflated optional and mandatory steps based on an imprecise semantic understanding of source documents. Failure to catch such errors could result in costly customer support and operations.

To mitigate these threats, we used human-in-the-loop verification checkpoints in addition to disclaimers and logged all LLM outputs to enable root cause analysis of mistakes. These checkpoints are helpful when receivers lack the expertise to validate outputs.

In summary, while increasing user awareness through disclaimers is valuable, the industrial adoption of LLMs requires rigorous testing and validation. This is especially important for users with limited validation capabilities, for whom LLM outputs carry higher potential value and greater risk. Responsibility must be shared between models and human experts through carefully designed processes.

2.5 Limitations and future directions

While this work provides initial insights, it has some limitations that open up opportunities for future research. Our findings come from one specific company, limiting the study's generality. The private data, though real-world, might not fully show the variety and complexity of business data out there. Plus, sandbox testing is different from real production systems. Given these limitations, future studies could check if our approaches work in other companies and with other data types.

2.6 Conclusions

This paper presents our experience on step-by-step deployment for constructing a private LLM sandbox suitable for proprietary data, capturing real-world complexities.

We have reported eight key learned lessons, such as LLM configuration, infrastructure security, enterprise data processing, and output evaluation. The key takeaway is that I) LLM's parameters and prompts can improve the quality of its outputs; II) Automated verification of the outputs is needed to evaluate the accuracy of evolving private data continuously.

Importantly, we highlight the relationship between an LLM output receiver's validation capacity and the value derived from the output. *The higher the receiver's ability to validate outputs, the less incremental value LLM outputs provide. On the contrary, for receivers with limited validation capabilities, unverified outputs could hold higher potential value but also have risks without proper validation.*

Overall, our findings can serve as a valuable reference for future work in this area, helping enterprises to use LLMs effectively with their proprietary data in a secure environment.

Study B

Measuring the Quality of Generative AI Systems: Mapping Metrics to Quality Characteristics - Snowballing Literature Review

Abstract

Context: Generative Artificial Intelligence (GenAI) and the use of Large Language Models (LLMs) have revolutionized tasks that previously required significant human effort, which has attracted considerable interest from industry stakeholders. This growing interest has accelerated the integration of AI models into various industrial applications. However, the model integration introduces challenges to product quality, as conventional quality measuring methods may fail to assess GenAI systems. Consequently, evaluation techniques for GenAI systems need to be adapted and refined. Examining the current state and applicability of evaluation techniques for the GenAI system outputs is essential.

Objective: This study aims to explore the current metrics, methods, and processes for assessing the outputs of GenAI systems and the potential of risky outputs.

Method: We performed a snowballing literature review to identify metrics, evaluation methods, and evaluation processes from 43 selected papers.

Results: We identified 28 metrics and mapped these metrics to four quality characteristics defined by the ISO/IEC 25023 standard for software systems. Additionally, we discovered three types of evaluation methods to measure the quality of system outputs and a three-step process to assess faulty system outputs. Based on these insights, we suggested a five-step framework for measuring system quality while utilizing GenAI models.

Conclusion: Our findings present a mapping that visualizes candidate metrics to be selected for measuring quality characteristics of GenAI systems, accompanied by step-by-step processes to assist practitioners in conducting quality assessments.

3.1 Introduction

Generative Artificial Intelligence (GenAI) [59] has emerged as a transformative technology across various fields, applied for a variety of tasks ranging from natural language processing to artifact generation [60]. GenAI systems refer to software that can generate new content using AI models based on learned patterns from training data. The content includes text, code, images, audio, and video. These AI models cover large language models (LLMs), diffusion models, and generative adversarial networks [5]. However, in this study, when we reference GenAI, we explicitly mean systems based on LLMs [61].

The principal idea of GenAI systems is to utilize the capabilities of AI models to produce, manipulate, or analyze content in ways that mimic human-like behavior or understanding [60]. The integration of AI models in software products is increasingly deployed across diverse industrial domains. Key application areas include document generation [62], data analysis and insights [63], marketing and content creation [64], customer service chatbots [65], multilingual business communication [64], and code generation [66]. These domains represent critical use cases where GenAI outputs must meet specific quality standards.

However, existing evaluation methods often lack alignment with industrial quality requirements [67]. Notably, key risks include the quality and consistency of GenAI outputs across different system releases [68–70]. These risks arise from GenAI's inherent characteristics, such as non-deterministic behavior — where identical inputs may produce varying outputs, and the tendency to hallucinate [71, 72] — generating content not based on factual data. Such characteristics complicate efforts to ensure reliable and trustworthy outputs in GenAI applications.

Given these risks, existing studies [50, 73] suggest the use of metrics to measure the quality of AI-generated outputs. It is suggested that metrics are valuable for organizations to track and manage whether AI-generated outputs meet industry-specific quality requirements. Metrics in this study refer to quantifiable measures that assess the quality of the outputs produced by systems that incorporate AI rather than evaluating the underlying AI models themselves [74]. Examples of such metrics, such as BERTScore [24], COMET [75], and MoverScore [76], have been developed to assess aspects like correctness, accuracy, and relevance of GenAI system outputs.

Despite the availability of various metrics, existing literature lacks a standardized approach for categorizing these metrics in relation to quality characteristics specific to GenAI systems. Moreover, many metrics are introduced without clearly defining the quality aspects they measure, often focusing on terms like correctness, accuracy, or relevance without providing explicit mappings to recognized quality characteristics.

As such, this study examines existing metrics and methods used for evaluating GenAI system outputs. Through a snowballing literature review, we identify existing evaluation metrics and map them to quality characteristics defined by the ISO/IEC

25023 standard [22]. Details of the mapping are present in Section 3.3.4.1. This standard was chosen for the mapping due to its widespread recognition and applicability across various industries, offering a standardized framework for evaluating GenAI systems. Furthermore, we analyze how these metrics can be employed to assess risks associated with GenAI system outputs, such as inaccuracies or biases.

The main contributions of this study are:

- Mapping metrics to quality characteristics of generative AI systems.
- Evaluation methods showing metrics' applicability to assess the quality of outputs produced by generative AI systems.
- General processes for using metrics to assess risky outputs produced by generative AI systems.
- A descriptive framework explaining how to select evaluation methods to assess specific quality characteristics based on the GenAI system's requirements.

The rest of this paper is structured as follows: Section 3.2 introduces related work. Section 3.3 illustrates the snowballing literature review methodology. Section 3.4 presents the results of the literature review based on extracted data. Section 3.5 introduces a suggested framework synthesized from the study results. In Section 3.6, we discuss threats to the study's validity. Section 3.7 discusses the findings and implications of our research. The conclusions and ideas for future work are in Section 3.8.

3.2 Background and Related Work

This section presents background information and the study's related work.

3.2.1 ISO/IEC 25023 standard for system quality

ISO/IEC 25023 is a standard defining eight measurable quality characteristics for software systems: Functional Suitability, Reliability, Usability, Performance Efficiency, Maintainability, Compatibility, Portability, and Security. Unlike task-specific metrics, this standard provides an approach to evaluating system-level quality, which is critical for GenAI applications deployed in industrial settings. For example, Usability ensures outputs align with user needs (e.g., output accuracy), while Security addresses risks like sensitive data leakage in generated content. Our work adopts ISO/IEC 25023 due to its industry-wide applicability and emphasis on standardized quality evaluation.

3.2.2 Key definitions

In this paper, a 'metric' and a 'quality characteristic' are defined as follows:

- *Metric*: a quantifiable measure used to assess how well a system performs in relation to certain quality characteristics. It translates abstract quality aspects (e.g., usability, reliability) into concrete and measurable values to evaluate the system against predefined quality characteristics defined in standards (e.g., ISO/IEC 25023).

- *Quality characteristic*: a property of a software system or product that can be measured to determine its quality. We adopt the terminology and structure defined by ISO/IEC 25023 [22], where 'quality characteristic' refers to system quality aspects [50].

- *Risky output* is defined as output produced by GenAI systems that contains errors due to factors such as uncertainty propagation, non-replicability, contextual misalignment, logical inconsistencies, or data hallucinations [77].

3.2.3 Related work on GenAI system quality evaluation

Industrial applications that use GenAI models [78] [51] include text generation [25, 76, 79], document summarization [53, 63, 80], and code generation [66, 81, 82]. However, studies evaluating these applications have identified several challenges in LLM outputs, such as 1) accuracy [65], 2) relevance [66], 3) coherence [83], and 4) consistency [84]. These challenges underscore the need for quality evaluation methods for GenAI systems in industrial applications.

Traditional manual evaluation methods [85] rely on individuals with domain expertise to assess the quality of AI-generated content. These approaches are often resource-intensive and prone to biases and subjective judgments from evaluators [86]. Due to these limitations, automated evaluation methods [87] become more popular in many cases. While these methods improve the time efficiency and scalability of the evaluation, they also introduce a steep learning curve for evaluators due to system setup complexity.

Prior studies, such as [4], [88], [54], [89] and [71], have developed metrics for specific GenAI evaluation in a few domains, such as document generation (BERTScore [24], METEOR [90]), customer service chatbots (RUBER [91]), and code generation (PASS@k [82]). However, these metrics are often siloed by domain and lack alignment with standardized quality characteristics. For instance, while FActScore [25] evaluates factual accuracy (Reliability), no metrics explicitly address Security (e.g., detecting sensitive data in outputs). This fragmentation limits organizations' ability to evaluate GenAI systems. Our work mitigates this gap by identifying a detailed inventory of metrics and practical methods for evaluating GenAI system qualities.

3.3 Research Methodology

We conducted a snowballing literature review [38] [40] combining peer-reviewed and grey literature to account for the rapid evolution of GenAI.

As illustrated in Figure 3.1, our research process is structured into four key stages: Planning, Snowballing start-set, Snowballing forward & backward, and Data collection & analysis. The process began with defining research questions, followed by database searches and filtering to establish an initial set of studies. A snowballing method [38] was then adopted to expand the study set through forward and backward searching. The searching was iterated until no new papers were found. Based on the search results, inclusion and exclusion criteria were applied. Furthermore, Ivarsson and Gorschek's Rigor/Relevance filtering [92] was executed to select papers based on industrial relevance criteria. Finally, data extraction and synthesis were conducted to derive insights that address the research questions.

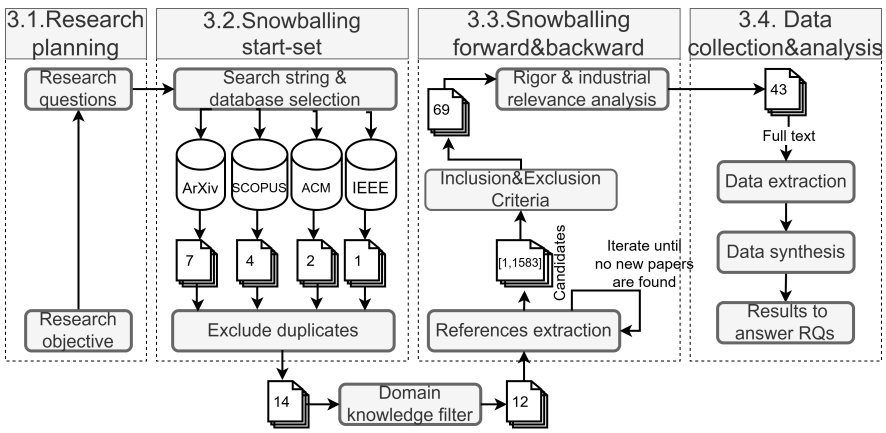

Figure 3.1: Research process overview.

3.3.1 Research planning

This section introduces our research objective and research questions.

3.3.1.1 Research objective

This study aims to identify existing metrics for evaluating the outputs of GenAI systems and map these metrics to the quality characteristics defined in ISO/IEC 25023, offering guidance on selecting metrics and approaches for assessing GenAI system quality.

3.3.1.2 Research questions

To achieve our research objective, four research questions have been formulated:

1. RQ1: What metrics can be used to evaluate the quality characteristics defined

by ISO/IEC 25023 for generative AI system outputs?

RQ1 seeks to identify existing metrics that are applicable for assessing the quality of generative AI system outputs and map these metrics to quality characteristics to provide a finer-grained view of quality. The mapping can either refine or complement the existing measures predefined by ISO/IEC 25023. This study provides an overview of the metrics and their mapping to quality characteristics, but does not include detailed formulas for each metric.

2. RQ2: What methods or approaches exist that could utilize the identified metrics to assess the quality of outputs produced by generative AI systems?

   RQ2 targets the existing evaluation methods that use metrics to assess the quality of outputs from generative AI systems. Answers for RQ2 could be useful for AI developers, researchers, and quality assurance professionals to select methods for ensuring that the outputs generated by GenAI systems meet their quality requirements.

3. RQ3: What processes can be used to apply metrics for evaluating the potential of risky outputs generated by generative AI systems?

   RQ3 aims to identify the processes for using metrics to evaluate the risky outputs generated by GenAI systems. Addressing inaccurate or harmful outputs is important for risk management teams. By establishing clear processes for using metrics, organizations can enhance the transparency and trustworthiness of GenAI systems.

3.3.2 Snowballing start-set

Following the guidelines set by Wohlin for snowballing [40], we initiated our literature search by identifying a start-set of recent surveys or literature review studies. Based on the start-set, we conducted backward snowballing to capture prior studies referenced in recent literature and forward snowballing [93] to identify newer studies that might not be cited in the existing literature. This combined approach ensures coverage in a rapidly evolving field.

Firstly, we created and refined keywords to formulate search strings through pilot testing. Since the study focuses on existing metrics with industrial applicability, in addition to covering metrics for known generative AI-based technology, the search string also included an emphasis on papers that report empirical or industrial results. The final search string is shown below:

```
((TITLE-ABS-KEY("large language model") OR TITLE-ABS-KEY(LLM)) AND (TITLE-
    ABS-KEY(metric*)) AND (TITLE-ABS-KEY("systematic literature review" OR
    survey)) AND TITLE-ABS-KEY(experience OR empirical OR industry*))
```

Secondly, we selected IEEE Xplore and ACM Digital Library as bibliographic databases, alongside Scopus and ArXiv. While Scopus, IEEE Xplore, and ACM

Digital Library primarily index peer-reviewed publications, ArXiv was included due to its relevance in capturing the recent research field of GenAI.

These databases were chosen for their coverage of computer science and engineering research, inclusion of peer-reviewed publications, and ability to capture recent developments in the rapidly evolving field of LLMs and AI. We ran the search string in the selected databases, resulting in 14 papers: seven in ArXiv, four in SCOPUS, two in ACM Digital Library, and one in IEEE Xplore. Finally, 12 papers were selected as the start-set for the snowballing when we read through the paper titles, keywords, and abstracts. Table 3.1 presents these papers, ordered by the number of citations.

Table 3.1: Snowballing start-set literature: grey literature on Arxiv was included. Note: the citation was collected from Google Scholar on 27th April 2024.

ID	Paper title	Year	Citation
P1	A Survey on Evaluation of Large Language Models	2024	497
P2	A Survey of Evaluation Metrics Used for NLG Systems	2023	198
P3	Evaluating large language models: A comprehensive survey	2023	67
P4	A Survey on Large Language Models Applications, Challenges, Limitations, and Practical Usage	2023	63
P5	Survey on factuality in large language models Knowledge, retrieval and domain-specificity	2023	61
P6	Trends in Integration of Knowledge and Large Language Models: A Survey and Taxonomy of Methods, Benchmarks, and Applications	2023	34
P7	Large Language Models for Forecasting and Anomaly Detection: A Systematic Literature Review	2024	22
P8	A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges	2024	16
P9	A Survey on Large Language Models for Software Engineering	2023	8
P10	Robustness, Security, Privacy, Explainability, Efficiency, and Usability of Large Language Models for Code	2024	4
P11	Machine learning for requirements engineering (ML4RE) A systematic literature review	2024	1
P12	Large Language Models for Blockchain Security: A Systematic Literature Review	2024	1

3.3.3 Snowballing forward and backward

The procedure to acquire papers for the literature review entailed both forward and backward snowballing of the starting set. Backward snowballing is achieved by looking at the references of each paper in the starting set to identify papers that can be likely candidates for the review. Forward snowballing is achieved by looking at papers that cited the reviewed paper to find more likely candidates for the review. For this work, the scraping of potential candidates was done exhaustively, using automated scripts (see Github repository) to include papers with a valid Digital Object Identifier (DOI) and manually for papers without a DOI. This scraping resulted in an initial set of 1,583 papers.

3.3.3.1 Inclusion and exclusion criteria

Our study aims to identify metrics that are actionable for practitioners evaluating the quality of GenAI systems. Thus, we focused on empirical studies, such as GenAI applications in Engineering, FinTech, and Healthcare, which actively integrate GenAI models requiring evaluation. Studies evaluating image, video, or multimedia-based GenAI outputs were excluded, as we target evaluation metrics. With more empirical data, the metrics have a higher chance of being verified practically, and challenges or barriers might be more domain-specific, which lab studies may overlook.

Table 3.2: Inclusion and exclusion criteria.

Inclusion criteria	Exclusion criteria
Empirical studies, case studies, workshops	Theoretical papers without empirical data
Engineering, FinTech, Healthcare	Image, video files
Metrics evaluate outputs produced by GenAI systems	Studies do not include metrics
Qualitative or quantitative results	No conclusive results
Studies from 2014 to 2024	Studies before 2014
English language	Non-English language

We manually applied inclusion and exclusion criteria, shown in Table 3.2, to the title, abstract, and the result of each paper. After applying the inclusion and exclusion criteria, 69 papers were identified.

3.3.3.2 Rigor and industry relevance evaluation

To ensure the practical applicability of the identified metrics in industrial settings, we employed Ivarsson and Gorschek's **Rigor and Industrial Relevance** framework [92] to evaluate the selected papers. Motivations for choosing the framework:

- *Focus*: The framework supports the identification of concepts (metrics in the context of this study) that are both methodologically reliable and practically valuable.
- *Emphasis on industrial relevance*: It helps address the potential disconnect between reported metrics and their real-world applicability.
- *Quantitative scoring*: It offers a quantitative scoring rubric for rigor and relevance (see Table 3.4), allowing for more objective comparisons between studies and concepts (i.e., metrics).
- *Granular evaluation*: It provides a detailed review of industrial relevance through criteria including subjects, context, scale, and research method [92].

Procedure how Ivarsson and Gorschek's framework was applied in this work:

1. The authors developed a rigor scoring rubric (see Table 3.3) based on Ivarsson and Gorschek's framework.

Table 3.3: Scoring rubric for evaluating rigor [92]

Aspect	Strong description (1)	Medium description (0.5)	Weak description (0)
Context described	The context is fully detailed, enabling comparison with other contexts.	The context is mentioned, but lack detail for full understanding.	No context information is provided.
Study design described	The study design is explained in detail for understanding, including variables, controls, and sampling.	The study design is mentioned briefly with minimal details.	There is no description of the study design.
Validity discussed	The validity is thoroughly explained, including detailed threats and measures to address them.	Validity is mentioned but not explained in detail.	No mention of validity or its threats is provided.

2. The authors ran Ivarsson and Gorschek's scoring rubric (see Table 3.4), focusing on practical applicability in industrial contexts.

Table 3.4: Scoring rubric [92] for evaluating industrial relevance

Aspect	Contribute to relevance (score 1)	Do not contribute to relevance (score 0)
Subjects	The subjects: - Represent the intended uses of metrics - Industrial professionals	The subjects: - Students - Researchers - Not mentioned
Context	Industrial settings: metrics for the efficiency and effectiveness of GenAI systems. It covers: - Methods/approaches for the system qualities assessment. - General risks, challenges, or limitations.	Laboratory settings: - Not a real usage situation - Only model benchmarks - Other
Scale	Realistic industrial scale size	Toy examples
Research method	Research methods (RM) include: - Case study - Field study, industry survey/questionnaire - Action research - Lesson learned	Excluded RM: - Conceptual analysis - Conceptual mathematical - Laboratory experiment - N/A

3. All authors refined the scoring rubrics.
4. The authors randomly selected 5% of selected papers (3 out of 69).
5. The first, second, and third authors used the defined scoring rubrics and analyzed both rigor and relevance independently.
6. Rigor evaluation:
    a) Read the full text of the selected paper, focusing on study contexts, design, and validity. Give the value of "1" for each of these three parts if they are detailed enough for full understanding or comparison; a value of "0.5" if they are partially described; and a value of "0" if there is no description.
    b) Calculate the rigor scores using the formulas below.

$$Rigor = C + S + V$$

    , where C refers to the context described, S is the study design described, and V stands for validity discussed. "Rigor" score ranges from zero to three, where a higher score indicates a more rigorous study.
7. Industrial relevance evaluation:
    a) Read the full text of the selected paper, focusing on study subjects, context, scale, and research method (see Table 3.4).
    b) Calculate Industrial Relevance scores using the formulas below.

$$Revelance = Subjects + Context + Scale + Researchmethod$$

    For each item in the formula, assign "1" if the paper contains sufficient information; otherwise, assign "0". The "Relevance" score ranges from zero to four, where a higher score indicates a better industry-relevant study.

8. The authors recorded the scoring results for the evaluation set (3 out of 69), as shown in Table 3.5, from the participating authors.

Table 3.5: A result of applying rigor and industrial relevance evaluation against the selected sample from the participated researchers.

ID	Paper title	Metric name	Include? (YES/NO)
Px	A discriminative metric for generation tasks with intrinsically diverse targets[S13]	DeltaBLEU	NO
Pm	Evaluating Large Language Models Trained on Code [S7]	PASS@k	YES
Pn	Evaluating the Factual Consistency of Abstractive Text Summarization[S39]	FactScore	YES

9. Inter-Rater reliability [46] analysis. We used the Fleiss' Kappa coefficient [49] by following the guidelines from Landis and Koch [94] to assess how strong the level of agreement was among the authors. The result was $(\kappa) = .57, p < 0.05$, which represents "good" strength agreement among our authors' opinions.

10. When the researchers reached an agreement, the first author conducted the rigor and relevance analysis on all papers identified through the snowballing search.

11. The authors stored the rigor and relevance analysis results in a spreadsheet file. An example is shown in Table 3.6.

Table 3.6: An example of the table storing the result of rigor and industrial relevance analysis.

ID	Paper	Metric name	Rigor: context	Rigor: study design	Rigor: validity	Relevance: subjects	Relevance: Context	Relevance: Scale	Relevance: RM
Pm	[S7]	PASS@k	1	1	1	0	0	0	0
Pn	[S39]	FactScore	1	1	0	1	1	0	1

12. The first author selected those papers with rigor scored "3" and industrial relevance scored "4", i.e. only papers that scored a "1" in all aspects of the model.

13. The authors reported the final result. Consequently, 43 papers were included and presented in Table 3.7. These included papers are now cited as [S1]–[S43] to differentiate them from traditional references.

Table 3.7: Selected papers

ID	Author(s)	Paper title	Year
[S1]	Kynkäänniemi et al.	Improved Precision and Recall Metric for Assessing Generative Models	2019
[S2]	Fabbri et al.	SummEval: Re-evaluating Summarization Evaluation	2021
[S3]	Huang et al.	AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation	2020
[S4]	Ghazarian et al.	Better automatic evaluation of open-domain dialogue systems with contextualized embeddings	2019
[S5]	Gruver et al.	Large language models are zero-shot time series forecasters	2024
[S6]	Mathur et al.	Tangled up in BLEU: Reevaluating the evaluation of automatic machine translation evaluation metrics	2020
[S7]	Chen et al.	Evaluating Large Language Models Trained on Code	2021
[S8]	Rei et al.	COMET: A neural framework for MT evaluation	2020
[S9]	Eddine et al.	FrugalScore: Learning Cheaper, Lighter and Faster Evaluation Metrics for Automatic Text Generation	2021
[S10]	Honovich et al.	TRUE: Re-evaluating Factual Consistency Evaluation	2022
[S11]	Sellam et al.	BLEURT: Learning Robust Metrics for Text Generation	2020
[S12]	Xu et al.	Optimizing statistical machine translation for text simplification	2016
[S13]	Galley et al.	deltaBLEU: A Discriminative Metric for Generation Tasks with Intrinsically Diverse Targets	2015
[S14]	Nguyen et al.	The effectiveness of feature attribution methods and its correlation with automatic evaluation scores	2024
[S15]	Mastropaolo et al.	Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization	2024
[S16]	Nallapati et al.	Abstractive text summarization using sequence-to-sequence rnns and beyond	2024
[S17]	Bakir et al.	Developing and Evaluating a Model-Based Metric for Legal Question Answering Systems	2023
[S18]	Pillutla et al.	MAUVE: Measuring the Gap Between Neural Text and Human Text using Divergence Frontiers	2021
[S19]	Kamalloo et al.	Evaluating open-domain question answering in the era of large language models	2023
[S20]	Zhang et al.	BERTScore: Evaluating Text Generation with BERT	2019
[S21]	Mathur et al.	Putting evaluation in context: Contextual embeddings improve machine translation evaluation	2019
[S22]	Chowdhery et al.	Palm: Scaling language modeling with pathways	2023
[S23]	Wu et al.	A comparative study of open-source large language models, gpt-4 and claude 2: Multiple-choice test taking in nephrology	2023
[S24]	Kang et al.	Identifying Inaccurate Descriptions in LLM-generated Code Comments via Test Execution	2024
[S25]	Stanojevic et al.	Beer: Better evaluation as ranking	2018
[S26]	Palotti et al.	TrecTools: an open-source Python library for Information Retrieval practitioners involved in TREC-like campaigns	2019
[S27]	Lardilleux et al.	CHARCUT: Human-targeted character-based MT evaluation with loose differences	2017
[S28]	Lo et al.	YiSi-a unified semantic MT quality evaluation and estimation metric for languages with different levels of available resources	2019
[S29]	Florian et al.	Exploring Precision and Recall to assess the quality and diversity of LLMs	2024
[S30]	Zhao et al.	MoverScore: Text Generation Evaluating with Contextualized Embeddings and Earth Mover Distance	2019
[S31]	Kryscinski et al.	Evaluating the Factual Consistency of Abstractive Text Summarization	2019
[S32]	Wang et al.	Asking and Answering Questions to Evaluate the Factual Consistency of Summaries	2020
[S33]	Le et al.	CodeRl: Mastering code generation through pretrained models and deep reinforcement learning	2022
[S34]	Graham et al.	Accurate Evaluation of Segment-level Machine Translation Metrics	2015
[S35]	Apidianaki et al.	METEOR-WSD: improved sense matching in MT evaluation	2015
[S36]	Toutanova et al.	A dataset and evaluation metrics for abstractive compression of sentences and short paragraphs	2016
[S37]	Johnson et al.	Assessing the Accuracy and Reliability of AI-Generated Medical Responses: An Evaluation of the Chat-GPT Model	2023
[S38]	Vaithilingam et al.	Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models	2022
[S39]	Min et al.	FActScore: Fine-grained atomic evaluation of factual precision in long form text generation	2023
[S40]	Chia et al.	INSTRUCTEVAL: Towards holistic evaluation of instruction-tuned large language models	2023
[S41]	Sharma et al.	An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation	2017
[S42]	Moramarco et al.	Human evaluation and correlation with automatic metrics in consultation note generation	2022
[S43]	Ren et al.	Codebleu: a method for automatic evaluation of code synthesis	2020

3.3.4 Data collection and analysis

Data collection involved two parallel processes: I) Metrics (e.g., names, definitions, sample datasets, and implementation codes) were extracted using predefined codes (Level 1-3 in Figure 3.2); II) Risks and challenges were identified through inductive coding of qualitative data.

At each level, codes were developed by identifying recurring themes, concepts, and findings within the chosen papers.

- Level 1: GenAI system quality evaluation, risks, or challenges of the evaluation. Four top-level codes were identified.

- Level 2: The Metric definitions, datasets, implementations, drawbacks or limitations, quality characteristics, known issues, concerns, and limitations. In

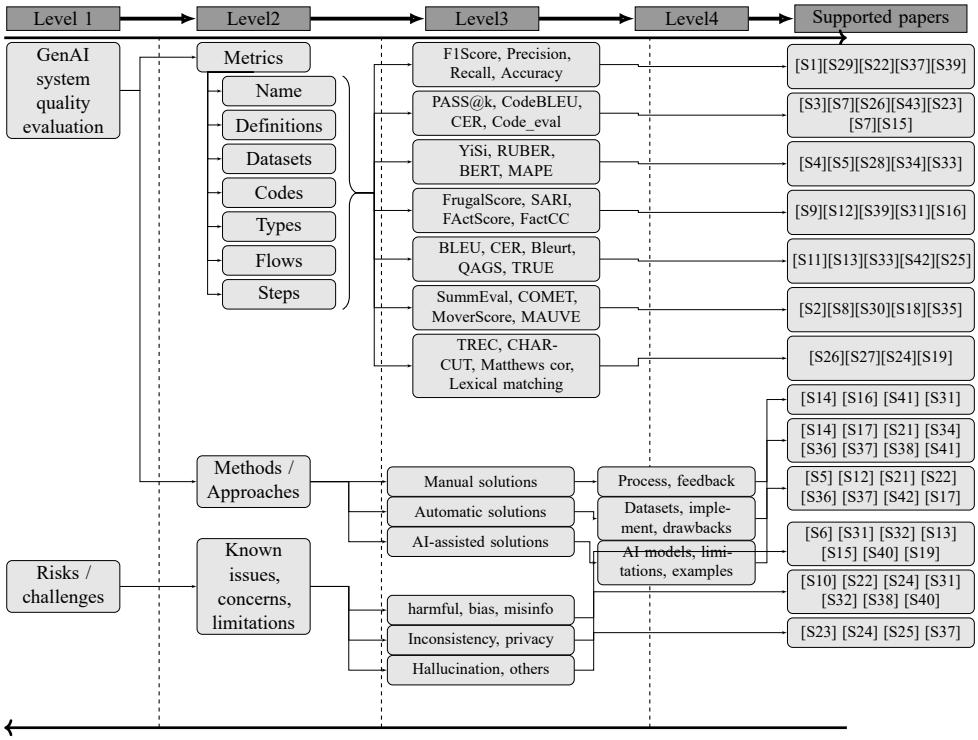

Figure 3.2: Coding process for extracting data. Metrics were extracted via structured coding (Levels 2–3 at the top); Thematic analysis was applied inductively to identify risks/challenges (Levels 1-3 at the bottom).

total, 25 codes were used.

- Level 3: Metrics, top-level quality characteristics, evaluation method types. 121 codes were identified and analyzed at this level.
- Level 4: Sub-attributes of quality characteristics. 31 codes were used at this level.
- Supported papers: The number of selected papers supports the extracted codes.

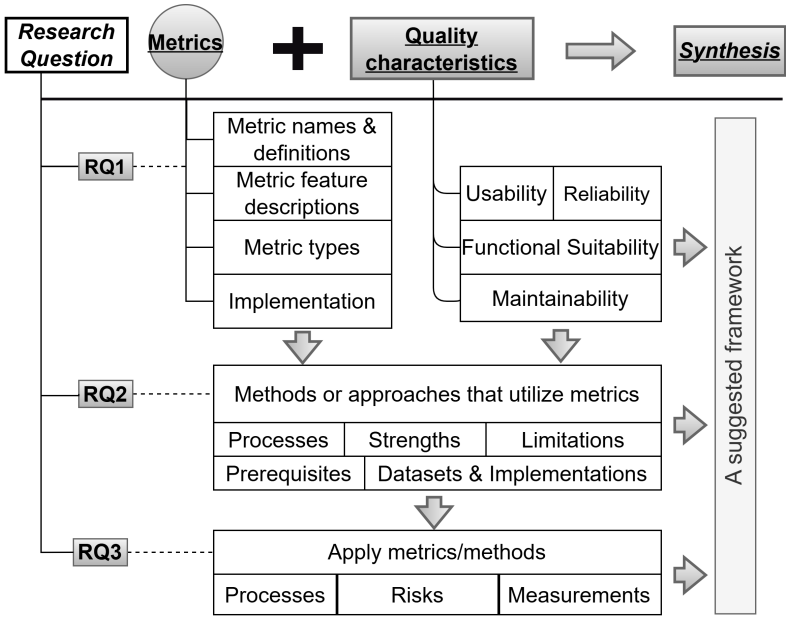

Figure 3.3: An overview of data synthesis based on collected codes.

An overview of data synthesis is presented in Figure 3.3. We followed the steps below for the thematic analysis.

1. Creation of initial codes: Initially, we generated a starting set of codes, including metrics and quality characteristics. This initial set was informed by our research questions and the recurring themes observed in the selected papers.

2. Familiarization with extracted codes: We began the analysis by thoroughly reading and examining all extracted codes. At this stage, the codes reflected various aspects like metric definitions, features, methods, quality characteristics, and risks.

3. Following familiarization, we reviewed and refined the initial codes. This iterative process allowed for continuous refinement and accuracy in coding.

4. Connect and combine codes: After refining the codes, we began connecting and combining them to identify potential themes. By grouping related codes, we formed broader themes that encapsulated the key findings from the papers.

5. Finally, we summarized the contributions of the study based on the identified themes of collected metrics, methods, and risks. These themes present evidence that the study's insights and conclusions are derived from, ensuring that the findings were well-grounded in the data.

Consequently, we suggested an expanded framework inspired by Lavesson et al.'s original five-step model. This framework extends the model's steps and combines our findings with metrics' datasets, implementation, and practical evaluation processes. Details about the framework are presented in Section 3.5.

3.3.4.1 Steps for mapping metrics to quality characteristics

We present the steps for mapping metrics to ISO/IEC 25023 [22] quality characteristics. This mapping is essential for two reasons: I) it allows us to refine the existing predefined measures in ISO/IEC 25023 if current metrics offer more precise or relevant assessments, especially for GenAI systems; II) it enables us to complement the existing measures in cases where additional metrics are needed to address the evaluation of GenAI system outputs.

Table 3.8: Steps for mapping the identified metrics to quality characteristics and subcharacteristics predefined by ISO/IEC 25023.

Steps	Description
Step 1	Read a metric feature description.
Step 2	Check the metric's measures and learn its formula, if available.
Step 3	Compare the metric's functionality with quality subcharacteristics definitions, identifying relevant subcharacteristics by evaluating how the metric assesses AI-generated outputs. For example, how relevant the outputs are.
Step 4	Map the metric to quality characteristics by examining how the link supports overall system qualities that have been predefined by ISO 25023.
Step 5	Exclude any mappings that are not supported by the reported use cases or evidence from the selected paper.
Step 6	Finalize the mapping of the metric to predefined quality characteristics through linked subcharacteristics.

The general process of our mapping is presented in Table 3.8. To better explain the application of this mapping process, we selected the "METEOR [90]" metric, which is frequently used in evaluating GenAI system outputs, to illustrate the steps in detail.

Step 1: Our mapping process begins with understanding a metric definition, and it is measured through a full-text reading of the selected papers. According to Banerjee et al.'s study [90], METEOR's main feature is to evaluate AI-generated texts' similarity while considering synonyms by comparing the generated output to human-written reference text.

Step 2: Examine the metric's measures and formula. In the example of METEOR, it measures $Precision = \frac{Relevant\ retrieved\ instances}{All\ retrieved\ instances}$ and $Recall = \frac{Relevant\ retrieved\ instances}{All\ relevant\ instances}$ of AI-generated texts with an additional penalty for incorrect word order. METEOR formula:

$$METEOR = \left(\frac{10 \times Precision * Recall}{Recall + 9 \times Precision}\right) * \left(1 - 0.5 \times \left(\frac{\#\text{chunks}}{\#\text{unigrams\_matched}}\right)^3\right)$$

If a system generates the sentence "The cat sat on the mat," and the reference is "A cat sat on the rug." METEOR compares the words and accounts for synonyms (mat

and rug) and stem forms (sat remains sat), resulting in a higher similarity score due to the semantic closeness.

Step 3: Compare metric functionality with ISO quality subcharacteristics' definitions. The METEOR metric's core function is to assess the semantic similarity of texts produced by GenAI systems, which indicates how suitable the system outputs are. The subcharacteristic "Appropriateness recognizability" in ISO/IEC 25023 stands out as a candidate mapping, according to its original definition: *Users have to be able to select a system/software product that is suitable for their intended use. The quality measures for appropriateness recognizability are used to assess the degree to which users can recognize whether a product or system is appropriate for their needs [22].*

Our interpretation of this subcharacteristic in the context of GenAI is: Appropriateness recognizability refers to how well the generated outputs align with their intended use, indicating whether engineers can determine if the output is suitable for their specific needs (e.g., document generation).

This interpretation explains why we mapped the METEOR metric to "Appropriateness recognizability" as an example. It is important to note that while the original definition focuses on systems or components, our interpretation centers on GenAI system outputs, considering the inherent black-box nature of GenAI models.

Step 4: Map the metric to quality characteristics. In ISO/IEC 25023, "Usability" is the top level quality characteristic for "Appropriateness recognizability." METEOR is mapped to Usability since the metric result indicates to which degree the system-generated text is understandable.

Step 5: Exclude any unsupported mappings. In Step 3, METEOR could map to the "Maturity" subcharacteristic, according to its original definition: *Maturity measures are used to assess the degree to which a system, product, or component meets the needs for reliability under normal operation [22]*. In the context of GenAI, we interpreted Maturity as reflecting the system's ability to consistently produce expected outputs under normal conditions, which aligns with the original definition. However, our review of the selected papers revealed insufficient direct evidence to support this link. Therefore, this mapping was excluded.

Step 6: Finalize the mapping of the metric to predefined quality characteristics through the associated subcharacteristics. In this example, METEOR is ultimately mapped to Usability through the subcharacteristic "Appropriateness recognizability."

Notably, although we have provided one example, the mapping process involved analyzing various other characteristics during the data analysis phase. By detailing each step, we aim to show transparency in the mapping process and elaborate the logic behind connecting a metric to specific quality characteristics. To minimize bias and potential misinterpretations, three authors actively participated in the development and analysis of this mapping through regular discussions and reviews.

3.3.5 Steps for identifying evaluation methods

The evaluation methods for answering RQ2 were derived by following the steps below.

1. **Step 1**: Extracting evaluation processes. While collecting metrics from the literature, we extracted descriptions of evaluation processes from the selected papers.

2. **Step 2**: Thematic coding. Level 3 codes, such as "human evaluation," "automated evaluation," and "AI-assisted evaluation" were created from phrases.

    - Human evaluation: Final assessment is human-driven, even if automated tools are used in data preparation.
    - Automated evaluation: Algorithm-driven scoring without human involvement.
    - AI-Assisted evaluation: Using AI models to augment human judgment or automate scoring.

3. **Step 3**: Linking codes to metrics and paper references. Metrics and papers were grouped based on their primary evaluation methods. For example, papers using METEOR without domain experts' involvement were coded as automated evaluation. Studies employing automated metrics using AI models were coded as AI-assisted evaluation.

4. **Step 4**: Validation. Three authors of this study cross-validated the coding. Discrepancies (e.g., ambiguous cases) were resolved through multiple group discussions.

3.3.6 Synthesizing processes for risky output evaluation

The processes for RQ3 about risky output evaluation were synthesized through:

1. **Collecting code** - Coding recurring patterns (e.g., risk mitigation steps, dataset validation) from the selected papers. The themes include Level 3 codes (see Figure 3.2 including bias or harmful content.

2. **Common process** - Grouping similar processes, for example: "steps validate system outputs against reference datasets" —> Dataset preparation; "tracking system quality thresholds" —> Monitoring.

3. **Synthesis of practices** - Analyzing papers that implicitly described steps without formalizing workflows. For example, FActScore [25] was used for fact-checking but did not outline a process.

4. **Reviews** - Reviewing coded themes iteratively into a generalized process by three authors, ensuring alignment with the best practices of system outputs' evaluation.

3.4 Results

This section presents study results to answer research questions.

3.4.1 RQ1: What metrics can be used to evaluate the quality characteristics defined by ISO/IEC 25023 for generative AI system outputs?

We have identified 28 distinct metrics, as shown in Table 3.9. We observed that metrics like METEOR are commonly used to evaluate text similarity — how closely AI-generated text aligns with natural language and meaning. Additionally, Lexical matching, CER, and BERTScore assess low-level relevance, such as word, character, and token similarity. Metrics like FactScore and TRUE focus on evaluating the factual accuracy of generated text. These metrics reflect growing concerns about both linguistic and factual accuracy in AI-generated outputs.

Table 3.9: Identified metrics, feature description, supporting papers, and target usage domains. *(X) - Number of supporting papers for a metric. (Embeddings) - Metric can process embeddings as contextualized data.*

Metric name	Feature description	Supported paper(s)	Usage domain(s)
BLEU	Measures n-gram overlap between texts	[S3] [S33] [S42] [S16] [S13] [S5] [S4] [S6] [S7] [S10] [S9] [S25] (12)	Text similarity Text matching
METEOR	Evaluates precision and recall in text matching	[S35] [S34] [S4] [S25] [S14] [S10] [S8] [S30] (8)	Text similarity Synonymy matching
F1Score	Special mean of precision and recall	[S1] [S4] [S22] [S29] [S33](5)	Text segmentation Classification
BERTScore	Compares embeddings for semantic similarity	[S20] [S4] [S21] [S9] [S2] (5)	Token similarity
Precision	Ratio of correct positive predictions	[S29] [S29] [S22] (3)	Data retrieval
Recall	Ratio of correctly identified true positives	[S29] [S22] [S29] (3)	
Bleurt	Evaluates text fluency and semantic similarity	[S11] [S21] (2)	Semantic similarity
Matthews correlation	Assesses classification accuracy for imbalanced data	[S24] [S42] (2)	Classification Linear correlation
Pearson correlation	Measures linear correlation between variables	[S14] [S42] (2)	
Code_eval	Tests code correctness	[S7] [S33] (2)	Code correctness
YiSi	Evaluates translation quality using multilingual embeddings	[S28] [S31] (2)	Semantic similarity (Embeddings)
Lexical matching	Matches exact words between texts	[S19] [S7] (2)	Word similarity
COMET	Evaluates translation quality with contextual embeddings	[S8](1)	Semantic similarity (Embeddings)
MAUVE	Assesses how close AI-generated text is to human writing	[S18](1)	Text analysis Human-like texts
RUBER	Measures the similarity between generated texts and references	[S4](1)	Text similarity (Embeddings)
MAPE	Measures accuracy of predictions in regression tasks	[S5](1)	Numerical accuracy Char accuracy
CER	Calculates character-level error rate	[S23] (1)	
TREC	Evaluates the quality of QA outputs with multiple datasets	[S26](1)	Text accuracy Text relevance
PASS@k	Checks how often code passes tests within k attempts	[S3](1)	Code correctness
FrugalScore	Evaluates semantic similarity	[S9](1)	Semantic similarity
SummEval	Assesses summary quality using multiple evaluation models	[S2](1)	Text summarization
TRUE	Measures QA accuracy using truthfulness and reasoning	[S10](1)	Factual consistency
MoverScore	Measures semantic similarity with word embeddings	[S30](1)	Semantic similarity (Embeddings)
FActScore	Evaluates factual consistency in generated text	[S39](1)	Factual consistency
QAGS	Checks for factual correctness in summaries	[S32](1)	Text summarization
CHARCUT	Measures character-level changes between texts	[S27](1)	Char matching
SARI	Evaluates quality of text simplifications	[S12](1)	Text accuracy
CodeBLEU	Assesses code generation with structure and syntax checks	[S43](1)	Code syntax matching

In the field of code generation, metrics such as Code_eval and CodeBLEU are infrequently used in the literature, possibly indicating that the development of robust evaluation for this domain is still ongoing.

Additionally, metrics such as COMET and YiSi, which utilize contextualized embeddings, aim to evaluate GenAI systems integrated with enterprise proprietary data stored in vectorized databases. These embedding-based metrics are particularly valuable for use cases where pre-trained LLMs may not meet the specific requirements imposed by private data.

3.4.1.1 Mapping metrics to ISO quality characteristics

To gain a finer-grained view of how the identified metrics contribute to the overall quality of generative AI systems, we mapped them to ISO/IEC 25023 quality characteristics, as visualized in Figure 3.4.

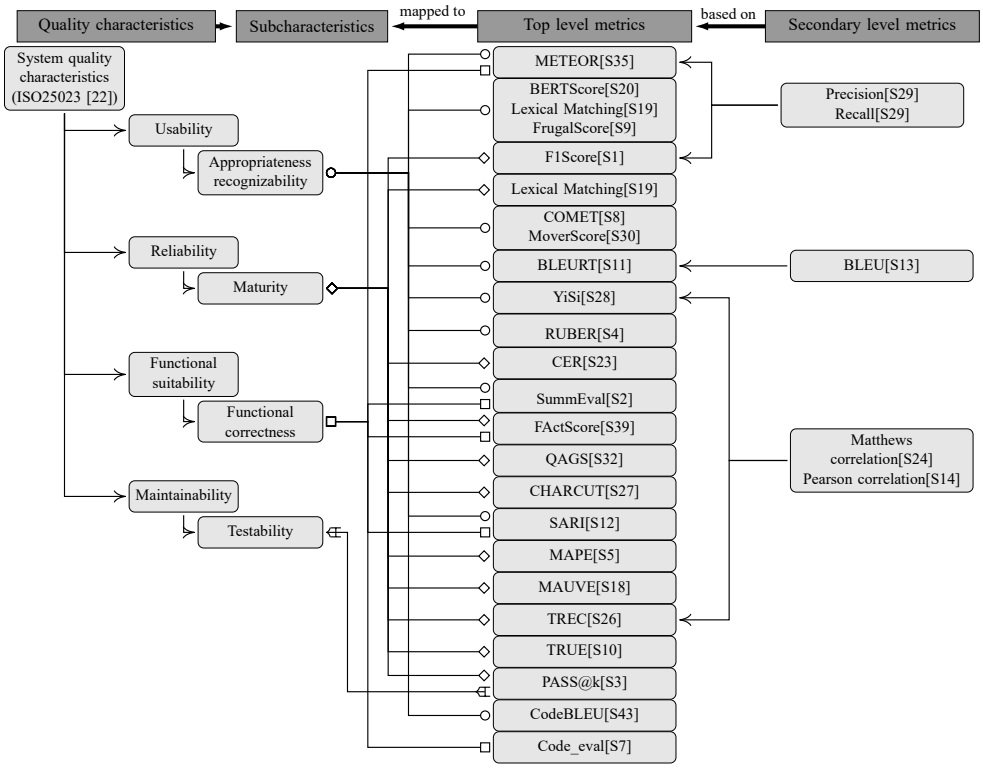

Figure 3.4: Mapping the identified metrics to ISO/IEC 25023 quality characteristics.

As shown in Figure 3.4, the identified metrics correspond to four main characteristics: usability, reliability, functional suitability, and maintainability. These mappings are built by using the process described in Section 3.3.4.1. Metrics (e.g., METEOR and F1Score) are fundamentally based on Precision and Recall, and BLEURT is derived from BLEU. To maintain focus and avoid redundancy, only top-level metrics are included in our mapping.

Building upon Hadi et al.'s categorization [64] and the reported cases in the selected papers, we identified six key GenAI application domains: document generation, data analysis and insights, marketing and content creation, customer service chatbots, multilingual business communication, and code generation. These domains represent areas where LLMs have demonstrated significant business impacts, particularly in enterprise contexts [52]. Table 3.10 presents how the metrics map to quality characteristics within these application domains.

Table 3.10: Metrics mapped to quality characteristics and GenAI application domains.

Quality characteristics	Sub-characteristics	GenAI application domains						SUM
		Document generation	*Data analysis and insights generation*	*Marketing and content creation*	*Customer service ChatBots*	*Multilingual business communication*	*Code generation*	
Usability	*Appropriateness recognizability*	METEOR BLEURT BERTScore MoverScore FrugalScore YiSi COMET SARI	SummEval METEOR BLEURT BERTScore MoverScore YiSi	METEOR BLEURT BERTScore MoverScore	BLEURT RUBER	RUBER	CodeBLEU	**11**
Reliability	*Maturity*	FActScore QAGS CHARCUT Lexical matching	TRUE F1Score	QAGS MAPE	F1Score TREC	CER MAUVE F1Score	PASS@k	**11**
Functional suitability	*Functional correctness*	METEOR SummEval FActScore	SummEval	SARI	-	-	Code_eval	**5**
Maintain-ability	*Testability*	-	-	-	-	-	PASS@k	**1**
SUM (The number of metrics)		**13**	**8**	**7**	**4**	**4**	**3**	**-**

Usability: The identified metrics like METEOR, BLEURT, BERTScore, MoverScore, FrugalScore, YiSi, COMET, SARI, and RUBER are mapped to "Appropriateness recognizability," as they assess how closely the generated outputs align with intended use cases across domains (e.g., document generation or content creation). For code generation tasks, CodeBLEU evaluates the appropriateness of AI-generated code.

Reliability: Metrics, such as FActScore, QAGS, CHARCUT, Lexical matching, TRUE, and MAPE, are mapped to "Maturity," reflecting the GenAI system's ability for content creation to produce accurate outputs under normal conditions. Metrics like F1Score and TREC provide precision measures, critical for ensuring factual consistency in customer service chatbot applications. Metrics, such as CER and MAUVE, offer measurements to ensure accurate human-like texts for multilingual business communication, and PASS@k measures the GenAI system's ability to consistently produce accurate code solutions.

Functional suitability: Metrics, such as SummEval, SARI, FActScore, METEOR, and Code_eval, map to "Functional correctness," as they assess how accurately the GenAI system outputs meet task-specific requirements, such as summarization or text simplification, indicating the system's ability to deliver correct results for the intended use.

Maintainability: PASS@k is mapped to "Testability," reflecting the ease with which the system's output can be verified. This is particularly useful for code generation, where the metric indicates how many trials are needed to produce accurate code, thus providing an indicator of how effectively the GenAI system can meet predefined criteria.

Key observations:

- Our mapping reveals that Usability and Reliability are the most frequently addressed quality characteristics in the identified GenAI application domains. This emphasizes the critical aspects for the GenAI system to generate outputs that are both appropriate and consistent.

- Functional suitability is primarily addressed in domains where the correctness of the output is critical, such as document generation.
- Maintainability, while less frequently mapped, plays a vital role in ensuring that generated code can be efficiently tested and validated, highlighting the importance of evaluating long-term system qualities for code generation tasks.
- Certain metrics address multiple ISO characteristics depending on their application context. For example, PASS@k evaluates both code maturity (Reliability) and testability (Maintainability). While this overlap reflects the metric's versatility, practitioners should interpret results based on their specific quality priorities.

While our study mapped metrics to four ISO/IEC 25023 quality characteristics, gaps remain in addressing the other characteristics, for example, Performance Efficiency and Security, particularly. These characteristics are critical for industrial adoption but lack dedicated metrics in the current literature.

In conclusion, our findings — from a list of available metrics, through their mapping against ISO quality characteristics, to their alignment with the reported application domains — offer a matrix view on the use of metrics to assess generative AI system qualities. The key takeaway is the importance of selecting metrics that are closely aligned with specific system qualities and relevant to each GenAI application domain.

3.4.2 RQ2: What methods or approaches exist that could utilize the identified metrics to assess the quality of outputs produced by generative AI systems?

Analysis of the identified metrics shows that they can be used in different methods. In this section, we have presented a synthesis of these methods and connected them to the different metrics—Results from RQ1. Organizations might apply these metrics directly without realizing they are inherently tied to a particular method. This flexibility means that while metrics can be used independently, understanding the methods behind them provides insights into the processes, strengths, and limitations of the chosen evaluation approaches.

By following the steps in Section 3.3.5, three evaluation methods, such as human, automated, and AI-assisted, were identified from the literature. For instance, three metrics in Table 3.11 described human evaluators validating system outputs, while 21 metrics in Table 3.12 utilized scripts for automated scoring. Five AI-assisted metrics in Table 3.13 were reported using model-based evaluators. Details of these evaluation methods are presented separately in Sections 3.4.2.1, 3.4.2.2, and 3.4.2.3.

3.4.2.1 Human evaluation method

Human evaluation involves the manual review and assessment of outputs produced by GenAI systems, typically conducted by domain experts or end-users. This method ensures that outputs are not only technically accurate but also contextually appropriate for industry-specific applications.

Evaluating outputs with domain experts offers a deeper understanding of industry contexts and facilitates detailed error analysis. For example, while GenAI systems may produce code that compiles without errors, human evaluators can assess whether the code adheres to best practices and company standards and integrates with existing codebases. This evaluation ensures that functional outputs also align with design patterns critical for team collaboration.

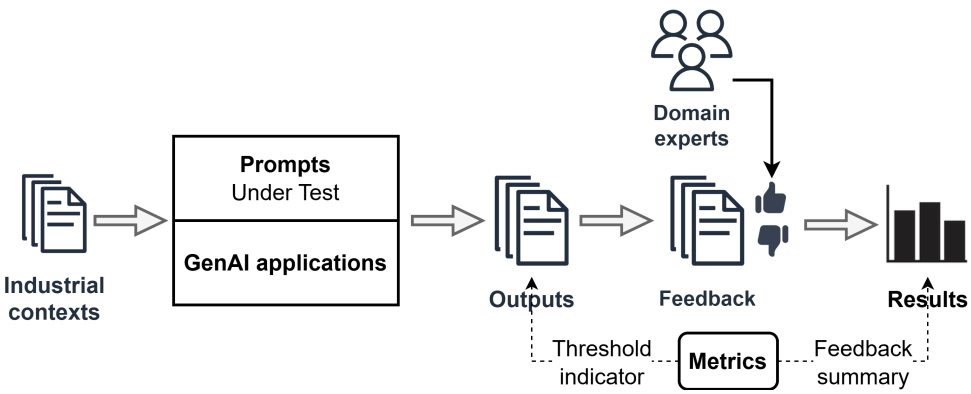

Figure 3.5: A suggested human evaluation process using metrics.

The suggested evaluation process, illustrated in Figure 3.5, includes the following steps:

1. Industrial contexts: The process begins with enterprise data inputs, which can be documents or requirements with context information.

2. Prompts: The data inputs are then fed into a GenAI application with prompts under test.

3. Outputs: Responses or results are generated by GenAI applications based on the input prompts and user queries.

4. Feedback: The outputs are then passed to human experts with domain knowledge or competencies. These experts review the outputs and provide feedback. Metrics could be scores about the' similarity of outputs or the percentage of expert judgment.

5. Results: Human feedback contributes to the generation of final results, which may include quantitative data towards the feedback, such as the percentage of agreement among feedback providers.

Table 3.11: Metrics from RQ1 that can support human evaluation.

Metric from RQ1	Evaluation	Example
SARI	Outputs with citations	Measuring text similarity extracted from documents
PASS@k	Code generation review	Assessing human reviews for generated codes
SummEval	Review report	Evaluating reviews' summary

Table 3.11 shows that metrics can play a supportive role by providing a quantitative indicator that human evaluators can use to guide their assessments. For instance, metrics such as SARI can help reduce the cognitive load on human evaluators by filtering outputs that meet a threshold score of 0.8 in semantic similarity and providing cited source data. This allows evaluators to focus on a smaller set of high-quality results. These metrics are classified under human evaluation because the final assessment relies on domain experts' judgment.

While human evaluation remains essential, it is not immune to errors, as evaluators could miss or misinterpret risky outputs. By automating the filtering process with metrics, we can streamline the evaluation and provide an objective layer of validation, thereby reducing the likelihood of human bias. This approach ensures that human judgments are supported by data, further enhancing the final assessment.

For practitioners, human evaluation requires evaluators to possess domain-specific knowledge. **For those evaluators with limited validation capabilities, unverified outputs could introduce risks without proper validation**. Additionally, scaling human evaluation for large datasets is challenging, highlighting the necessity for automated methods, which we explore in the next section.

3.4.2.2 Automated evaluation method

Automated evaluation involves using algorithms, scripts, and metrics to assess the quality of AI-generated outputs and enables scalable and efficient evaluation across large datasets.

Automated evaluation methods provide consistent and repeatable assessments with predefined standards [87]. These methods are typically used for identifying patterns and trends in AI-generated outputs, making them suitable for evaluating large volumes of data where human evaluation may be impractical.

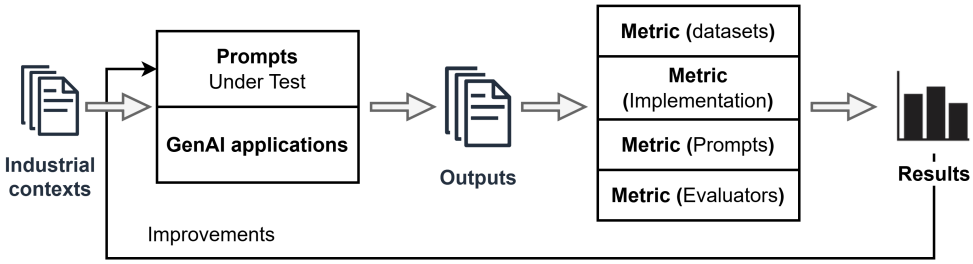

Figure 3.6: An identified evaluation process in automatic manners.

The evaluation process, as illustrated in Figure 3.6, involves several key components and steps:

1. Industrial contexts input: The process begins with the identification and input of relevant industrial contexts. These contexts define the requirements, constraints, and specific needs that the GenAI applications must meet.

2. GenAI applications and prompts under test: GenAI applications are then tested using various prompts that align with the industrial contexts. These prompts are designed to simulate real-world scenarios.

3. Generating outputs: The application processes the prompts and generates outputs. These outputs are the direct results of the LLM's response to the provided prompts within the context of the defined industrial needs.

4. Application of metrics: The outputs are then evaluated using metrics.

    a) Metric (datasets): The outputs are compared against relevant datasets to validate their correctness, accuracy, and relevance. These datasets provide a reference against which the LLM applications' output is measured.

    b) Metric (Implementation): This refers to the practical application of the metrics. For example, this might involve using Python scripts to automatically calculate metric scores, ensuring that the evaluation process is both executable and reproducible.

    c) Metric (Prompts): Prompts used to assess GenAI applications.

    d) Metric (Evaluators): Evaluators (e.g., algorithms/scripts) involve a qualitative and quantitative assessment to ensure that the outputs meet the defined quality standards.

5. Results analysis: The results from these evaluations are then compiled and analyzed to determine how well the GenAI application has performed in the context of the industrial tasks. While metrics provide a quantitative assessment, an expert or human evaluator is important in interpreting and monitoring the metric data.

6. Feedback Loop for Improvements: The insights gained from the results analysis are fed back into the process, informing improvements to the GenAI applications, the design of prompts, and the overall evaluation process.

Table 3.12: Metrics from RQ1 with prerequisites, datasets, implementations, and drawbacks/limitations for automated evaluation.

Metric from RQ1	Prerequisites	Datasets & Implementation	Drawbacks/Limitations
BLEU	- Reference translations. - Candidate translations.	- WMT14. - NLTK BLEU.	- Limit to n-gram matches. - Overlook the semantic meaning. - Sensitive to minor variations
METEOR	- Reference translations. - Candidate translations.	- WMT19. - Meteor Github	- Depend on high-quality references. - Penalize creative paraphrasing.
F1Score, Precision, Recall	- Ground truth labels. - Predicted labels	- Huggingface. - Scikit-learn Github.	- Misleading for imbalanced datasets. - Does not account for true negatives.
Matthews correlation	- Ground truth labels. - Predicted labels.	- Kaggle data. - Github.	- Sensitive to class imbalance. - Require a confusion matrix.
Pearson correlation	- Paired numerical data points.	- Kaggle data. - SciPy Github.	- Assume linear relationships. - Sensitive to outliers. - Does not imply causation.
Code_eval	- Reference code. - Generated code. - Test cases.	- HumanEval. - Codex Github.	- Focus on test case pass rate. - Require extensive test cases. - Limit to functional correctness.
YiSi	- Source texts. - Reference translations. - Candidate translations.	- WMT16. - YiSi Github	- Requires multilingual embeddings. - Sensitive to translation quality. - Complex to implement.
Lexical Matching	- Reference texts. - Candidate texts.	- Kaggle NLP. - NLTK Github.	- Rely on exact matches. - Overly simplistic for complex texts.
MAPE	- Predicted values. - Actual values.	- Kaggle data. - Sklearn Github.	- Sensitive to small errors. - Biased by outliers. - Less effective for zero or near-zero values.
CER	- Reference texts. - Hypothesis texts.	- Openslr Huggingface. - CER Github.	- Focus on character-level errors. - Overly penalize minor typos. - Not suitable for longer texts.
TREC	- Data samples.	- Trectools data. - TREC Github.	- Focused on information retrieval.
RUBER	- Data samples.	- datasets. - RUBER Github.	- Focused on dialog systems.
PASS@k	- Problem descriptions. - Generated code samples. - Unit tests.	- HumanEval. - PASS@k Github.	- Dependent on test case quality. - Does not assess code efficiency. - Limited to functional correctness.
BERTScore	- Reference texts. - Hypothesis texts.	- Huggingface data. - BERTScore Github.	- Maximum sentence length is 512.
FrugalScore	- Small set of human-annotated translations.	- Kaggle NLP. - FrugalScore Github.	- Potential bias from a small dataset. - Scaling issues with large datasets. - Limited semantic range capture.

Continued on next page

Table 3.12 – *Continued from previous page*

Metric from RQ1	Prerequisites	Datasets & Implementation	Drawbacks/Limitations
SummEval	- Reference summaries. - Candidate summaries. - Multiple pre-trained evaluation models.	- CNN/Daily mail. - SummEval Github.	- Dependent on multiple models. - Complexity in aggregation. - High computational demand.
FActScore	- Source documents. - Generated summaries.	- Wiki data. - FActScore Github.	- Focused on factual consistency only. - Sensitive to source quality. - Require fact extraction.
CHARCUT	- Reference texts. - Hypothesis texts.	- WMT17. - CHARCUT Github.	- Focused on character-level edits. - Not suitable for complex texts.
SARI	- Original texts. - Simplified reference texts.	- WikiLarge. - SARI Github.	- Require high-quality references. - Penalize creative simplifications.
QAGS	- Generate questions. - Answer questions. - Compare answers.	- NewsQA. - QAGS Github.	- Prepare QA data.
CodeBLEU	- Reference code. - Generated code.	- CodeXGLUE. - CodeBLEU Github.	- Complex scoring system. - Focused on code structure. - Require high-quality reference code.

Table 3.12 provides metric implementations, along with sample data and GitHub repository links. When implementing a metric, practitioners may need to interact with the code, possibly adapt it for their particular scenarios, and verify that their datasets conform to the metric's prerequisites.

While automated evaluation provides scalability and consistency, there are inherent limitations:

- Contextual Limitations: Metrics like CER are highly context-dependent and may not fully capture the content they are evaluating. For example, CER focuses on character-level accuracy, which is useful for technical content but might miss broader errors in context or logic.

- Dependence on Reference Data: Many metrics, including CER, rely heavily on high-quality reference data. The results of these metrics depend on the availability of accurate and comprehensive reference datasets. In cases where such data is limited or biased, the metrics might provide inaccurate or less reliable evaluations.

- Inability to Handle Ambiguity: Metrics like SARI are not well-suited to handling ambiguous prompts where multiple correct answers may exist. These metrics are often rigid in their assessment criteria, which can lead to penalizing outputs that are correct but expressed differently than the reference data.

Automated evaluation offers a scalable approach for evaluating GenAI system outputs, but practitioners must be aware of its limitations.

3.4.2.3 AI-assisted evaluation method

AI-assisted evaluation involves using AI models to automate the initial assessment of system outputs, process results, and suggest refinements, thereby enhancing and adapting the overall evaluation process.

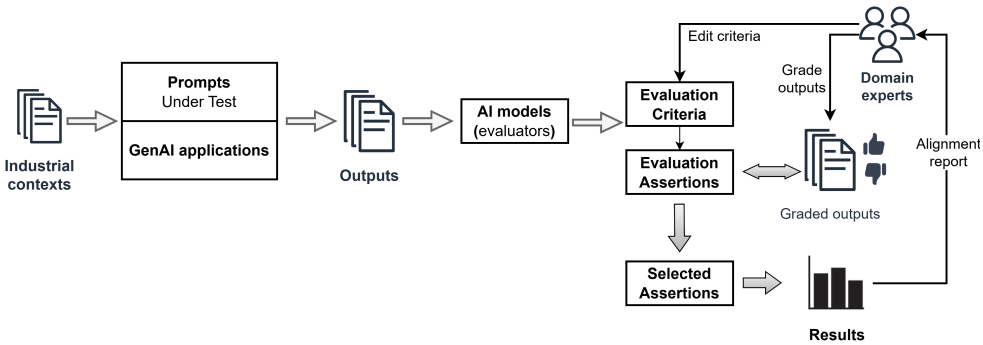

Figure 3.7: A synthesized AI-assisted evaluation process based on literature. Domain experts iterate through the process to edit evaluation criteria and system output grading.

As illustrated in Figure 3.7, the AI-assisted evaluation process offers an adaptive assessment that refines evaluation criteria over time to maintain the quality of GenAI system outputs. The process involves the following key steps:

1. Industrial contexts: Input industrial contextualized information or documents to the GenAI system.
2. Prompts Under Test and GenAI applications: Industrial contexts are processed through the prompts in GenAI applications.
3. Generating Outputs: The GenAI applications generate outputs based on the input data and prompts.
4. AI Evaluators: AI model(s) trained for evaluation, assess the outputs.
5. Evaluation Criteria: The evaluator LLMs use predefined criteria to judge the outputs.
6. Evaluation Assertions: Based on the criteria, the evaluator models generate specific assertions about the quality of the outputs.
7. Graded Outputs: The original outputs are graded based on these assertions.
8. Domain Expert Involvement: Domain experts review the graded outputs and can refine the evaluation criteria and grading process.
9. Results and Alignment Report: The process produces evaluation results and an alignment report, which can be used to improve the system.

To implement AI-assisted evaluation, selecting appropriate metrics and AI models is important. Table 3.13 presents a detailed overview of metrics, automation steps, required AI models, available datasets, and implementation examples.

Table 3.13: Metrics from RQ1 with required AI models, datasets, implementations, and examples that can be used for AI-assisted evaluation.

Metric from RQ1	AI assist	Datasets & Implementation	Example	Drawbacks / Limitations

Bleurt	BLEURT	- WMT21.
		- Bleurt Github
			`from bleurt import score`
			`references = ["The cat sits on the mat."]`
			`candidates = ["There is a cat on the mat."]`
			`scores = score.score(references=references, candidates=candidates)`
			`print(f"BLEURT Score: {scores[0]:.3f}")`
				- Requires extensive fine-tuning.
				- Sensitive to domain shifts.
				- High computation cost.

MAUVE	GPT2	- GPT-2 data.
		- MAUVE Github.
			`from examples import load_gpt2_dataset`
			`import mauve`
			`p_text = load_gpt2_dataset('data/amazon.valid.jsonl', num_examples=100) # human`
			`q_text = load_gpt2_dataset('data/amazon-xl-1542M.valid.jsonl', num_examples=100) # machine`
			`out = mauve.compute_mauve(p_text=p_text, q_text=q_text, device_id=0, max_text_length=256, verbose=False)`
			`print(out.mauve) # prints 0.9917`
				- Relies on large datasets.
				- May not capture all nuances of language quality.
				- Requires high computational resources.

MoverScore	BERT Embeddings	- WMT19.
		- MoverScore Github.
			`from moverscore_v2 import sentence_score`
			`refs = ['The dog bit the man.', 'The dog had bit the man.']`
			`sys = 'The dog bit the man.'`
			`moverscore = sentence_score(sys, refs)`
			`print(f"Mover Score: {moverscore[0]:.3f}")`
				- Dependent on high-quality word embeddings.
				- Limited by the quality of the reference text.

Continued on next page

Table 3.13 – *Continued from previous page*

Metric	AI assist	Datasets & Implementation	Example	Drawbacks / Limitations
TRUE	T5	- Annotation sentences. - TRUE Github.	`python true/src/ meta_evaluation_scores.py --input_path=data/"{ INPUT_FILE}" \ --metrics_scores_columns="{ METRICS}" \ --output_path="{OUTPUT_PATH}"`	- Focused on factual consistency. - Require specific data types.
COMET	COMET	- WMT22. - COMET Github	`from comet import download_model, load_from_checkpoint model_path = download_model(" wmt20-comet-da") model = load_from_checkpoint(model_path) data = [{"src": "The cat sits on the pad.", "mt": " There is a cat on the pad ." "ref": "A cat is sitting on the pad."}] seg_scores, sys_score = model .predict(data, batch_size =8, gpus=1) print(f"COMET Score: { seg_scores[0]:.3f}")`	- Requires complex model fine-tuning. - Sensitive to the quality of input encodings. - High resource demands for processing.

Table 3.13 highlights that metrics can utilize pre-trained AI models like GPT-2, showcasing the versatility of LLMs in evaluation tasks [95].

AI-assisted evaluation methods offer several advantages over automated metrics:

- Contextualized embeddings: AI-assisted metrics use pre-trained language models to create context-aware representations of text. This way can capture the subtleties of meaning and intent, rather than relying solely on exact word matches. However, this approach requires access to context-dependent data, which can be challenging.

- Human-aligned learning: Many of these metrics are fine-tuned on datasets of human ratings, allowing them to better align with human perceptions of quality.

- Semantic comprehension: These metrics excel at capturing semantic similarities, going beyond surface-level text features. By understanding the underlying meaning of a text, AI-assisted metrics provide more accurate and meaningful assessments of generated outputs.

Despite their advantages, AI-assisted evaluation methods face certain limitations:

- Dependence on metric dataset quality: These metrics rely heavily on high-quality, contextually relevant datasets for evaluation. Acquiring such datasets can be challenging in financial analysis, where data is often sensitive and difficult to obtain.

- Complexity in implementation: AI-assisted metric implementation may require extra technical expertise compared to automated metrics. The need to integrate and deploy AI evaluation models, such as COMET, can be a barrier for teams.

- Interpretability challenges: The outputs from AI-assisted evaluation may be less transparent, potentially making it difficult for evaluators to understand the reasoning behind a given score in the evaluation process.

- Computational intensity: The advanced processing capabilities of AI-assisted metrics often come at the cost of higher computational requirements. Running these metrics can be resource-intensive, making them less suitable for scenarios where computational resources are limited.

3.4.2.4 Summary of human, automated, and AI-assisted evaluation

Human evaluation, despite being resource-intensive and subject to individual biases, provides detailed and context-aware assessments for evaluating GenAI system outputs. When integrated with automated and AI-assisted evaluation techniques, it enhances the robustness and depth of the quality assurance process.

For the simplification of analytical clarity, we classify evaluation methods into three categories: human, automated, and AI-assisted. In real-world cases, the degree of automation in evaluation can vary. Feldt et al. [96] highlight their view in a taxonomy on the ways AI is applied in software engineering, distinguishing between levels such as automation support, decision augmentation, and full automation.

For example, in AI-assisted evaluation, domain experts often iteratively refine prompts while relying on AI-generated assessments as input. This interaction aligns more closely with the "collaboration" levels described in Feldt et al.'s taxonomy, rather than full automation. Similarly, some automated methods embed indirect human assumptions through dataset design or threshold calibration, blurring categorical boundaries.

While our findings help guide the practical usage of evaluation methods, we acknowledge that evaluation methods exist along a continuum of automation rather than as strictly evaluation types. As evaluation tools mature and human-AI collaboration increases, future work should consider extending the current classification of quality evaluation methods for GenAI systems.

3.4.3 RQ3: What processes can be used to apply metrics for evaluating the potential of risky outputs generated by generative AI systems?

GenAI systems can generate outputs that are misleading, biased, and inaccurate. Through this study, we identified six types of risky outputs, as shown in Table 3.14: harmful content, bias and discrimination, misinformation, inconsistency, privacy violations, and copyright infringement.

Metrics can be used to measure a specific risk. For instance, the MAUVE metric can be utilized to assess harmful content; FactScore or TRUE metrics can be employed to check the misinformation. These measurements can flag potentially risky outputs before they cause harm in real-world applications.

Table 3.14: Risky output dimensions, example metrics, and measurements

Risk dimensions	Example Metrics	Measurements
Harmful Content	MAUVE	Measures the degree of toxicity in outputs, helping identify potentially harmful content.
Bias and Discrimination	BERTScore, BLEURT, COMET	Analyzes outputs for demographic or ideological biases using semantic similarity and contextual understanding.
Misinformation	FActScore, QAGS, TRUE	Compares outputs against trusted sources to detect false information.
Inconsistency	F1Score, Precision, Recall	Measures consistency across multiple system outputs for the same input.
Privacy Violations	Lexical Matching, TRUE	Uses adversarial testing and named entity recognition to identify potential disclosure of sensitive information.
Copyright Infringement	BLEU, METEOR, BERTScore, MoverScore	Compares outputs to reference texts to detect potential plagiarism.

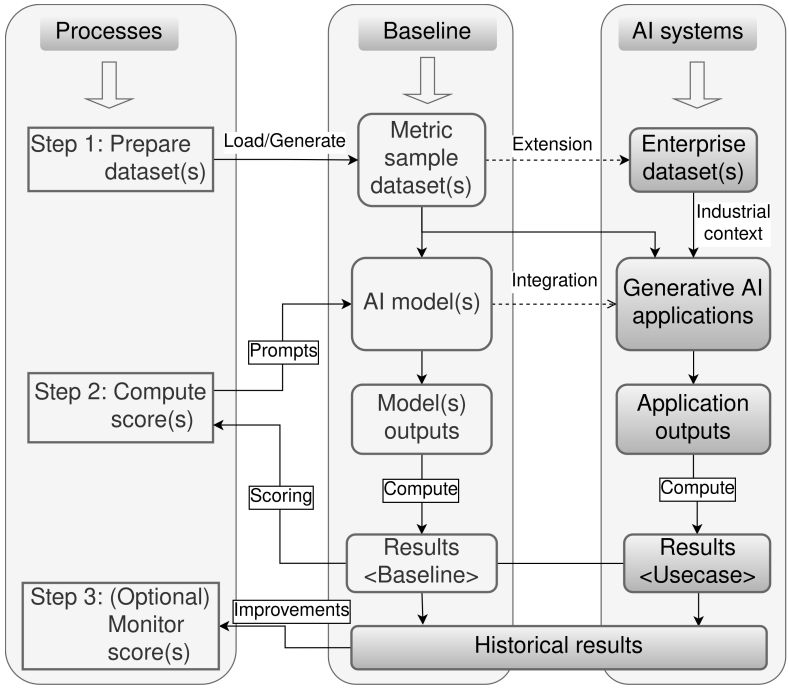

Figure 3.8: A three-step process to apply metrics for evaluating the risky outputs produced by generative AI systems.

Following our predefined steps (presented in Section 3.3.6), we synthesized a three-step process for risky output evaluation. As illustrated in Figure 3.8, the process contains three steps: Step 1 - prepare datasets, Step 2 - compute scores, and Step 3 - monitor scores.

Step 1: prepare datasets: This step involves gathering datasets that represent the GenAI system's intended application domain. Practitioners should incorporate sample datasets that reflect diverse contexts and can be extended to develop enterprise-specific datasets, ensuring evaluations are relevant to their business scenarios.

In **Step 2: compute scores**, using the prepared datasets, outputs are generated from the GenAI models and evaluated against predefined metric prompts. This scoring quantifies key aspects such as accuracy, safety, and bias. Baseline scores are established from standalone AI models, while scores from GenAI applications provide performance insights in specific use cases. In industrial contexts, proprietary datasets can be challenging to develop; in such cases, sample datasets may be utilized for evaluations.

Step 3: monitor scores (Optional): Monitoring involves continuously tracking metric scores over time to detect variations in the quality of GenAI outputs. By comparing results to the established baseline, organizations can identify trends and potential areas needing improvement. This step is particularly valuable for long-term

quality assurance and allows for prompt detection and mitigation of emerging risks.

To illustrate the processes, an example using the BERTScore [24] metric to assess gender biases in AI-generated outputs is provided. Steps are:

1. *Step 1: Prepare datasets*. Datasets should include prompts that represent a wide range of demographic groups and contexts. One can utilize an existing dataset or create a new one. For example, a dataset about profession, gender, and roles might include sentences like "The programmer completed the code," "The athlete secured the victory," or "The doctor instructed the nurse."

2. *Step 2: Compute Scores*. The prepared datasets, along with prompts, are inputted into the AI model (such as GPT), and the outputs produced by the model are collected. The BERTScore metric calculates scores by comparing the AI-generated outputs against a set of unbiased reference outputs. For example, if the prompt is "The doctor said to the nurse," and the AI consistently generates "he" for the doctor and "she" for the nurse, it may suggest a gender bias.

3. *Step 3: Monitor Scores*. Track BERTScore over time to detect deviations indicating potential biases. If biases are identified, refine the model by adjusting datasets or prompts and recompute the scores to assess improvements.

   While this step is optional, it is essential to periodically re-evaluate the GenAI system outputs displayed in trend visualizations with updated datasets and prompts. Upon detecting biases, users can refine GenAI systems by infusing their datasets or system prompts with more contextual information. Subsequently, recalculating the scores will indicate if the adjustments have mitigated the bias.

Key considerations for metric selection: Given the limitations reported in RQ2 (Section 3.4.2.2 and 3.4.2.3), metric selection should be guided by:

1. Relevance to use cases: Choose metrics that align with the specific quality characteristics essential for the GenAI application (e.g., BERTScore for coherence, FActScore for factual accuracy).

2. Dataset quality and availability: Ensure datasets are representative and free from biases to maintain evaluation integrity.

3. Combined evaluation methods: Integrate automated metrics with human or AI-assisted evaluations for a more thorough assessment.

4. Scalability and resource constraints: Consider the computational resources required, especially for large datasets or complex models.

3.5 A suggested framework for evaluating generative AI systems

Our study results show that a GenAI system evaluation may require assessing a set of risky output dimensions to continuously measure and manage quality characteristics in an industrial application. Such a framework for this purpose is needed based on our findings in this study.

Therefore, as shown in Figure 3.9, we suggest a framework that supports practitioners with multi-methods, metrics, and processes for quality attribute evaluation of GenAI systems. This framework is inspired by the research work of Lavesson et al. [50] on an application-oriented-validation-evaluation method. Their method contains five sequential and general steps to address the evaluation of AI components based on quality requirements prior to the inclusion of the components in software systems/products. However, detailed information about existing evaluation metrics or methods is missing. We extend their method by filling in the missed information.

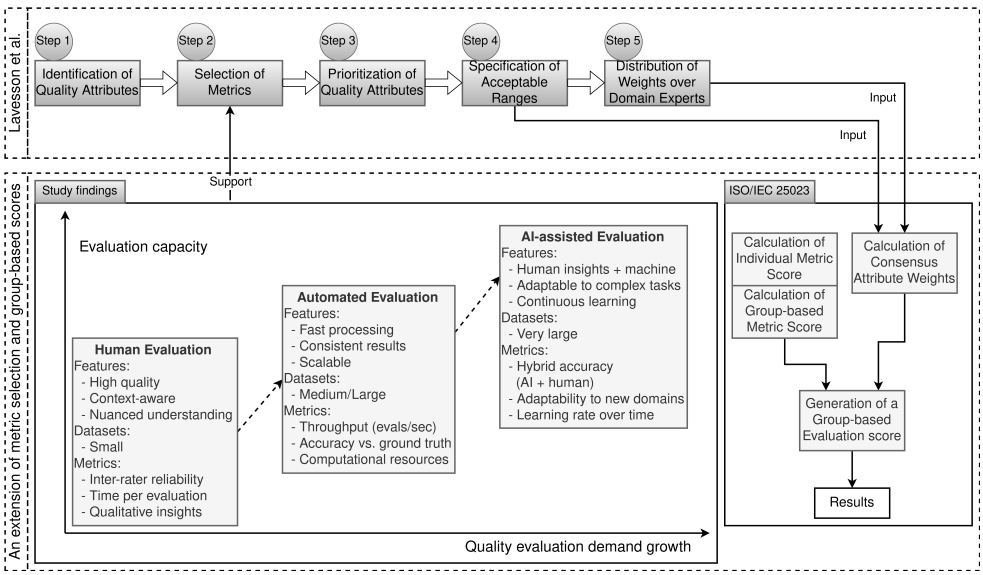

Figure 3.9: A framework for quality evaluation of GenAI systems.

As shown in Figure 3.9, the proposed framework consists of five steps. Steps 1 to 5 are initially from the study of Lavesson et al. [50].

Step 1: Identification of quality characteristics. Product designers or architects should provide an initial list of quality characteristics. When these are unclear, the ISO/IEC 25023 standards can assist in defining them based on our experience.

Step 2: Selection of metrics. Our study findings show that metrics are associated with evaluation methods or approaches. In practice, the choice of evaluation method depends on the specific industrial requirements in Step 1. Human evalua-

tion is an initial approach for assessing GenAI system outputs on a limited set of user queries and small datasets. As the scale of evaluation increases or the need for continuous quality monitoring grows, automated evaluation becomes a more viable option. AI-assisted evaluation offers a compelling solution for mission-critical applications (e.g., healthcare) requiring strict accuracy with factual sources. It enhances automated evaluations with deeper contextual insights and better alignment with human expectations.

Moreover, the progression from human to automated to AI-assisted evaluation methods showcases a steadily increasing capacity in assessing GenAI systems. Here, capacity refers to the evaluation method's ability to capture context, semantic nuance, and domain-specific criteria, not its throughput or scalability. While automated methods generally scale better with input volume, they may lack the interpretive depth required in complex or safety-critical domains. Each method has its place in the evaluation toolkit, and the choice among them can be guided by the process defined in Lavesson et al. [50].

Step 3: Prioritization of quality requirements. Consider a real-world application of supervised learning where multiple quality characteristics are recognized. It is unlikely that these attributes hold equal significance. According to Lavesson et al., individual metric weights provided by domain experts worked in this step. As mentioned, each identified metric is associated with an ISO quality attribute, which can be treated as quality attribute weights.

Step 4: Specification of acceptable ranges. We inherit this step from the work of Lavesson et al. [50]. It means that each domain expert — i ($i = 1, 2, 3, ..., k$) is allowed to associate with each selected metric — j ($j = 1, 2, 3, ..., n$) an acceptable range — r_{ij} ($r_{ij} = \left[b_{ij}^{l}, b_{ij}^{u}\right]$), where the former denotes the least desired acceptable score, and latter denotes the desired score.

Step 5: Distribution of weights over domain experts. Domain experts are permitted to rate their peers, thus indicating the extent of influence they are willing to accept from the group. The detailed decision process is based on the study by Lavesson et al. [50].

Furthermore, to streamline the definition and calculation of metric formulas, we utilize ISO/IEC 25023, which provides a clear definition and formula for quality measures. Practitioners may follow the established ISO calculations or tailor these calculations to obtain their evaluation scores. An example of maturity measures within the ISO standards is described in Table 3.15.

Table 3.15: Maturity measures [22] in ISO/IEC 25023.

ID	Name	Description	Measurement function
RMa-2-G	Mean time between failure (MTBF)	What is the MTBF during the system/ software operation?	X = A/B, where A=number of failures detected during observation time, B=duration of observation.

In short, the steps in this proposed framework have been validated by several

empirical case studies, such as [97], [39], [98]. Consequently, this framework serves as a supportive guideline to assist engineers when they have a need to introduce quality assessment and monitoring of GenAI systems within the industry.

3.6 Validity threats

Following Ampatzoglou et al. [99], we analyze threats using their framework for literature review studies.

3.6.1 Study selection validity

A threat is potential selection bias in the snowballing start-set (Table 3.1) due to reliance on recent literature reviews, which might exclude older papers. Forward/backward snowballing could exclude some related studies that are not indexed in bibliographic databases. To mitigate this, we combined snowballing with review of the ArXiv repository to capture grey literature and documented inclusion/exclusion criteria transparently (Table 3.2).

3.6.2 Data validity

A threat is a subjectivity in mapping metrics to ISO characteristics (e.g., METEOR to "Appropriateness Recognizability"). We reduced this via a six-step protocol (Table 3.8) with three authors cross-validating mappings. Another threat is inconsistency in rigor/relevance scoring during paper filtering (Section 3.3.3.2). We calculated Fleiss' $\kappa = 0.57$ to quantify inter-rater reliability and resolved discrepancies through group discussions. The oversimplification of evaluation methods (human/automated/AI-assisted) was mitigated by grounding classifications in inductive coding (Figure 3.2).

3.6.3 Research validity

A threat is domain specificity: findings for text-based GenAI systems (e.g., document generation) may not generalize to multimedia domains like image/video generation. We explicitly scoped the study to text-based GenAI applications (Section 3.1). Reliance on ISO/IEC 25023, which lacks GenAI-specific traits (e.g., non-determinism), was mitigated by proposing framework extensions (Section 3.5). Variance in metric implementation (e.g., Bleurt) was mitigated by providing implementation repositories and examples (Tables 3.13). Small sample rigor/relevance assessments (3/69 papers) were offset by transparent reporting and inter-rater agreement analysis.

3.7 Discussions

3.7.1 Datasets required for automated metrics

The result of automated evaluation metrics is influenced by the quality and relevance of the datasets used. High-quality, domain-specific datasets enhance the accuracy of metrics like BERTScore, which relies on reference data for assessment. However, challenges arise when datasets are outdated, biased, or incomplete, potentially leading to misleading results. Metrics like SARI, for example, may penalize valid semantic variations that differ from reference texts, overlooking the richness of meaning.

This dependency on high-quality datasets is problematic in domains like Healthcare, where such datasets are not easy to acquire. Although automated metrics offer scalable assessments, their limitations highlight the need to complement them with AI-assisted techniques to gauge the qualities of datasets, for example, AI synthetic data.

3.7.2 Cost versus quality

Implementing evaluation metrics can be resource-intensive in the industry. Organizations should balance the cost of these metrics with the potential benefits while using them. Advanced metrics like MoverScore and COMET provide deeper insights but come with increased computational demands. As Zhao et al. [76] suggest, conducting cost-benefit analyses is crucial for determining which metrics align best with an organization's budget and quality objectives.

Lavesson et al. [50] further propose an analytic hierarchy process that engages domain experts in prioritizing quality characteristics through pairwise comparisons. This pragmatic approach aids organizations in optimizing metric selection according to their specific needs and constraints.

3.7.3 Selecting appropriate metrics to evaluate system output quality

The selection of an appropriate set of metrics is crucial for evaluating GenAI system output quality. Lavesson et al. [50] emphasize the importance of involving domain experts in this process, offering an evaluation method designed for industrial applications. A combination of human, automated, and AI-assisted evaluation methods can yield a more in-depth assessment, covering a wider range of quality characteristics.

Eddine et al. [80] support this multi-method approach, advocating for a combination of metrics to capture diverse aspects of output quality. This suggests that practitioners should not rely on a single evaluation method but instead employ a blended approach to achieve more impact and nuanced assessments.

3.7.4 Risky outputs exist in GenAI industrial applications

Despite the potential of using metrics to mitigate risks, a significant challenge hindering GenAI adoption in the industry is its unpredictable nature. Minor changes in input prompts, such as punctuation or abbreviations, can lead to vastly different outputs, which can negatively affect user experience and system reliability.

Although dynamic data sampling could help address this unpredictability, current metrics typically rely on static datasets, as indicated by our study findings. Therefore, further research is needed to develop more dynamic evaluation methods that can accommodate the inherent variability of GenAI-generated outputs, ensuring they meet industry quality standards.

3.7.5 The potential for undiscovered risky outputs and the need for new metrics

As GenAI technology continues to evolve, new forms of risky outputs are likely to emerge from novel applications, unforeseen interactions with other systems, or evolving ethical considerations. For example, risky outputs may occur when interacting with other GenAI systems or complex infrastructures.

The possibility of such undiscovered risks highlights the need for new metrics and evaluation methods. Future research should focus on developing metrics that assess not only GenAI system outputs but also their impacts, involving experts from fields such as ethics and cybersecurity.

3.7.6 Limitations of snowballing with a small start-set

While our snowballing enabled a focused synthesis of literature on metrics for GenAI systems, it introduces limitations compared to traditional systematic literature reviews. Specifically, the reliance on 12 SLRs as a start-set — selected for their thematic relevance and citation impact — may narrow the selection of the start-set. Jalali and Wohlin's comparative analysis [40] on the research topic 'Agile practices in global software engineering' underscores that snowballing and database searches result in different paper sets, though converging on similar conclusions. However, no direct comparative analysis exists between snowballing and SLR outcomes in the domain of GenAI metrics, limiting our ability to assess potential coverage gaps. Future studies could evaluate the differences between these methods empirically in synthesizing GenAI metrics.

3.8 Conclusions

This study focused on exploring and identifying metrics for evaluating the quality of outputs generated by GenAI systems. Through a snowballing literature review, we identified 28 metrics mapped to four ISO/IEC 25023 quality characteristics: usability, reliability, functional suitability, and maintainability. Our findings suggest that these metrics provide valuable insights for the quality evaluation of GenAI systems, although our conclusions are derived from qualitative synthesis rather than quantitative measurements.

In our analysis of human, automated, and AI-assisted evaluation methods, we observed that a combined approach might offer a broader evaluation based on recurring themes across the reviewed studies. Specifically, our review indicates that: I) Human evaluation, while context-aware, may be influenced by individual biases; II) Automated methods deliver scalability but depend heavily on the quality of the underlying datasets; III) AI-assisted methods can enhance adaptability, but they may introduce additional complexity in terms of deployment and computational requirements. We emphasize that the methods' robustness and deployment complexity emerged from qualitative interpretations of the literature and were not measured directly.

Building on these insights, we proposed a framework, guided by Lavesson et al.'s method, to assist industrial engineers in selecting appropriate quality evaluation methods for GenAI system outputs according to their quality requirements.

In conclusion, this study offers primarily practitioners, but also researchers, a set of metrics, evaluation methods, and processes to improve the quality assessment of GenAI system outputs. Continuous refinement and adaptation of these evaluation methods will be essential as GenAI applications become increasingly integrated into industrial contexts.

Study C

Evaluating the Quality of GenAI Applications in Software Engineering: A Multi-case Study

Abstract

Context: Generative AI (GenAI) is increasingly adopted in software development for tasks such as document generation, data analysis, and code generation. However, evaluating the quality of GenAI applications becomes challenging, as traditional quality measurements may not be fully applicable.

Objective: In this study, we explore how practitioners evaluate the quality of GenAI applications and investigate quality evaluation techniques.

Method: We conducted a multi-case study in three industrial projects from software development companies. We examined four GenAI application domains: document generation, data analysis and insight generation, customer service, and code generation. Data were collected through three workshops and 23 semi-structured interviews with industrial practitioners.

Results: We identified fourteen GenAI use cases and 28 metrics currently used to evaluate the quality of GenAI applications' outputs. We synthesized the identified metrics' usage patterns and challenges based on the collected data.

Conclusions: This study presents practical insights into using metrics to measure GenAI-based system qualities in real industrial settings. Our findings indicate that practitioners use custom-built and context-specific metrics; combining these with academic metrics can strengthen GenAI system quality evaluation.

4.1 Introduction

The rapid adoption of Generative Artificial Intelligence (GenAI) in the industry has revolutionized tasks such as chatbot, information retrieval, code review/generation, and anomaly detection [71, 72]. The industrial interest in utilizing GenAI reflects a desire to streamline and improve the processes of tasks traditionally requiring significant human effort [100]. For instance, GenAI-powered *chatbot* services have been widely integrated into industrial web interfaces to handle user queries [29]. Similarly,

GenAI models support *information retrieval* from large documents [101], assist in *code review* by highlighting issues such as security vulnerabilities [72, 102], and identify *financial anomalies* in source files within minutes [71].

However, evaluating the quality of these GenAI applications remains a challenge. In this paper, we use the term *GenAI applications* to refer to systems that support software development activities by integrating generative AI models for tasks such as document generation, trouble report analysis, and code generation. Our focus lies in understanding how GenAI technologies are adopted in ongoing software development projects and how the quality of these applications is evaluated in industrial practices. Traditional software quality assurance approaches are mostly designed for deterministic systems with transparent logic and traceable outputs. In contrast, GenAI systems produce probabilistic, non-deterministic responses [103]. They often function as 'black boxes', making conventional evaluation techniques inadequate [104]. This shift in quality assurance calls for new or adapted evaluation approaches that better reflect the unique characteristics of GenAI outputs [105].

In response, prior research has proposed a variety of experiment-driven metrics to assess GenAI systems. These metrics, including accuracy, relevance, and consistency, aim to quantify output quality based on reference-based or task-based scoring [106]. In this study, we refer to such metrics as *experiment-driven metrics*, developed within academic research contexts for controlled evaluation purposes. However, it remains unclear whether these metrics can be effectively applied in industrial environments, given constraints related to scale, domain specificity, and the lack of labeled ground-truth data. GenAI systems deployed in practice often require adaptation to domain-specific language or tasks [106]. For example, a pre-trained model may perform poorly on legal or financial documents, prompting organizations to adopt custom evaluation criteria tailored to their operational goals. Consequently, evaluating GenAI system outputs often depends on use-case-specific quality requirements that standard research metrics may not adequately capture.

To investigate industrial practices, we studied four GenAI application domains in software development companies: document generation, data analysis and insight generation, customer service chatbots, and code generation. These domains were identified through engagement with industrial engineers and are also highlighted in recent literature on GenAI adoption in industry, such as Esposito et al. [107] and Kenthapadi et al. [15]. Feldt et al.'s AI-SEAL taxonomy [96] further categorizes how GenAI is integrated into software development, including human-in-the-loop, automated, and AI-assisted workflows, which align with the evaluation patterns observed in our study.

To understand how quality is evaluated in real-world GenAI applications, we conducted a multi-case study across three software development companies. We gathered insights through workshops and semi-structured interviews with engineers directly involved in GenAI implementation and evaluation. Our goal was to identify the quality metrics used, the methods of applying them, and the challenges encoun-

tered during evaluation.

The main contributions of this study are:

- An overview of metrics currently used in the industry for GenAI quality measurement.
- Insights into the practical use of these metrics within specific industrial contexts and suggestions for utilizing experiment-driven metrics.
- Challenges when using metrics to evaluate the quality of GenAI applications.

Our findings reveal that quality evaluation in industry is highly context-specific. Practitioners rely on lightweight, task-oriented checks that reflect their domain needs and project constraints. While experiment-driven metrics from research are rarely used directly, several are seen as promising when appropriately adapted. We also identify distinct evaluation challenges, which we translate into actionable recommendations for practitioners seeking to assess GenAI systems' quality.

The rest of this paper is structured as follows: Section 4.2 introduces related work. Section 4.3 illustrates the research methodology. Section 4.4 presents the study results based on extracted data. Section 4.6 discusses threats to the study's validity. Section 4.7 discusses the findings of our research. The conclusions and future work are in Section 4.8.

4.2 Related work

4.2.1 GenAI in Software Engineering

Generative AI (GenAI) technology has been used for requirement analysis, documentation, bug triaging, code generation, and testing [17] in Software Engineering (SE). Bommasani et al. [72] and Zhang et al. [102] reviewed these areas and noted GenAI's automation potential across several development stages. Kenthapadi et al. [15] reported practical challenges observed in industrial adoption, such as domain alignment and model hallucination. Esposito et al. [107] explored the use of GenAI to support various phases of software engineering, including requirements-to-architecture and architecture-to-code transformations. These findings support the relevance of the domains analyzed in our study. Feldt et al. [96] introduced the AI-SEAL taxonomy, which classifies AI applications in software engineering based on integration depth and human involvement. Their taxonomy reinforces our evaluation framework and highlights current industrial patterns in GenAI adoption.

While these studies demonstrate the growing relevance of GenAI in SE, few address how GenAI system qualities are evaluated in practice, especially under the non-deterministic and probabilistic nature of GenAI outputs.

4.2.2 Evaluation metrics for GenAI in Software Engineering

In software engineering, ISO/IEC 25010 and ISO 25023 [22] define quality characteristics, such as functional suitability, reliability, and maintainability. Research work (e.g., [108] [109] [26] [25]) has employed metrics [110] like accuracy, relevance, and consistency to score GenAI-generated outputs [106]. Most of these metrics are tested in controlled experiment settings and lack validation across industrial domains. The remainder of this section reviews metric usage in four application domains.

4.2.2.1 Document generation domain

Document generation tasks, such as summarizing requirements, drafting design documents, and generating implementation proposals, are common areas of GenAI application in software engineering [72, 107]. In academic settings, such tasks are often evaluated using metrics adapted from natural language generation, including BLEU [108], ROUGE [109], and METEOR [90]. These metrics quantify text-level similarity between generated and reference texts, but may not adequately capture semantic correctness or task-specific value. Recent metrics, such as BERTScore [24] and FActScore [25], aim to assess semantic alignment and factual consistency. These metrics rely on source data citations, which require intensive time and effort to be manually prepared in industrial settings. Despite these academic metrics, little is known about their actual usage in GenAI-enabled software projects.

4.2.2.2 Data analysis and insight generation domain

GenAI can be helpful in log interpretation, incident correlation, trend identification, and insight extraction from historical defect reports [15, 102]. However, evaluating the relevance of such insights is particularly difficult in industrial contexts, because a relevant output depends on domain-specific expectations and often lacks predefined reference data for comparison. Academic methods often use F1-score [110] to evaluate the relevance. When GenAI ranks outputs (e.g., most likely causes or top anomalies), *Precision* is often used [29]. Some approaches frame the task as supervised classification, using labeled categories [17]. These metrics rely on structured validation datasets; therefore, manual review, user feedback, or correctness checks are needed in these scenarios.

4.2.2.3 Customer service chatbots domain

GenAI chatbots can handle support queries and ticket triage in IT service domains [15, 101]. Quality assessment in this domain contains *response quality* and *user experience* dimensions. Technical criteria include correctness and relevance. Evaluation methods in research use both automatic and human-involved approaches. Automatic metrics include BLEU [108] and ROUGE [109] for response generation, as well as more chatbot-specific measures such as F1-score for *dialogue success rate* [102]. Recent works evaluate *factual consistency* using FActScore [25] or similar metrics.

Human-centered evaluations often use rating scales to assess helpfulness [106]. In industrial settings, evaluations involve user satisfaction scores and support logs.

4.2.2.4 Code generation domain

GenAI coding assistants support code completion, refactoring, documentation, and automated test generation [72, 102]. Evaluating generated code involves assessing both functionality and maintainability. Research employs PASS@k [82] (execution success of generated code), CodeBLEU [26], and *Exact Match* [29] to assess the quality of GenAI-generated code. Static analysis metrics such as *cyclomatic complexity*, *code duplication*, and *lint violations* can also be employed, especially when code is not executable. However, these metrics may miss system-level concerns like explainability, transparency, or trust.

4.3 Research methodology

We conducted a multi-case study inspired by Runeson et al.'s guidelines [32]. The case study methodology was chosen because it allows for an in-depth exploration of use cases and deeper insights into the quality measurement practices, metrics used, and challenges practitioners face for GenAI-supported software development. An overview of research processes is shown in Fig. 4.1.

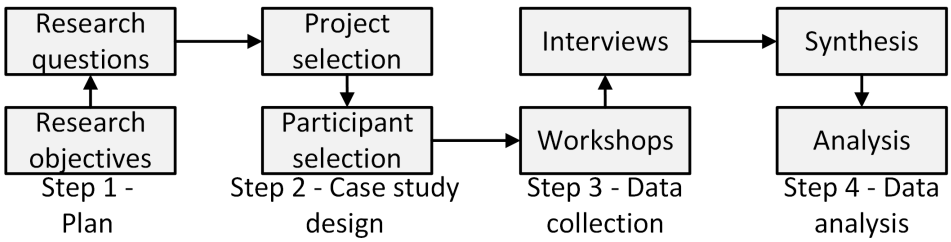

Figure 4.1: Overview of research processes.

4.3.1 Step 1 - Plan

In this phase, we formulated research objectives and research questions.

4.3.1.1 Research objectives

This study aims to I) understand how GenAI is currently being applied in industrial software development settings; II) explore how practitioners evaluate the quality of outputs produced by GenAI-enabled systems; III) identify challenges and contextual factors that influence the use of metrics in these environments.

4.3.1.2 Research questions

To achieve the research objectives, we formulated the following research questions:

- RQ1: What metrics are used to evaluate the quality of GenAI applications?
- RQ2: What types of evaluation methods are used to measure the identified metrics for GenAI applications?
- RQ3: What challenges are encountered in applying metrics to evaluate the quality of GenAI applications?

4.3.2 Step 2 - Case study design

This section presents the contexts of the industrial projects and the study participants' selection.

4.3.2.1 Project selection

Industrial cases were sampled using purposive sampling [111] from our existing industrial network, specifically companies that were, at the time of the study, engaged in developing GenAI applications. This purposive sampling ensured that the selected organizations – serving as the units of analysis – had ongoing experience with GenAI in software development. Our focus was on organizations that had already integrated GenAI capabilities into real-world projects, so that we could investigate how system qualities were evaluated in practice, rather than in experimental or theoretical contexts [111]. Furthermore, the study is descriptive, aiming to describe the use cases that the purposively sampled organizations currently have for GenAI.

Project A operates in the FinTech domain, focusing on mobile digital payments, while Projects B and C are in the telecommunication domain, providing operational support services and network management.

Table 4.1 presents the context information of the selected industrial projects. According to the guidelines set by Petersen et al. [112], each context is described by its business domain, product type, maturity, and organizations.

Table 4.1: Context information [112] of the selected industrial projects.

Context aspect	Context element	Project A	Project B	Project C
Project	Business domain	FinTech	Telecom billing	Telecom network
	Product type	Finance services	Operation support services	Network management
	Maturity of product	Mature product	Mature product	Mature product
Organization	number of engineers	>500	>1000	>200
	Distributed development	Yes (global)	Yes (global)	No (nationally)
GenAI	Usage of GenAI	Yes	Yes	Yes
Quality attributes	ISO 25023	Security Usability Reliability	Functional suitability Usability Reliability Maintainability	Usability Reliability

As can be seen in Table 4.1, there are several differences in business domains, team structures, and quality focus areas across the selected projects. This selection thereby offers a foundation for exploring varied approaches to GenAI quality evaluation. Additionally, this diversity enhances the generalizability of our findings, providing an understanding of the applicability of different metrics in various industrial contexts. However, we stress that the results are still captured from a limited sample, albeit diverse, which may limit our findings, as further discussed in Section 4.6.

4.3.2.2 Participant selection

We used purposive sampling [31] to recruit participants through industrial contacts at the case companies. The selected participants were actively involved in GenAI-related development, integration, or evaluation tasks. The same group of participants was involved in both the workshops and follow-up interviews, enabling continuity across data collection phases and allowing us to deepen and validate the findings through multiple engagement formats.

As shown in Table 4.2, 30 participants were chosen. Participant roles included software developers, managers, software architects, security engineers, operations engineers, and document engineers. Their working experience ranges from 4 to 25 years.

Table 4.2: Participants from the selected industrial projects A, B, and C.

ID	Participant role	NO. of participants	Working experience (years)	Project
1	Software developer	6	5 - 20	A (2), B (2), C (2)
2	Manager	5	5 - 25	A (1), B (2), C (2)
3	Software architect	5	7 - 23	A (2), B (2), C (1)
4	Security engineer	5	5 - 17	A (1), B (3), C (1)
5	Operation engineer	5	4 - 15	A (2), B (1), C (2)
6	Document engineer	4	5 - 10	A (1), B (1), C (2)

For the workshops, the focus was on a broad understanding of how GenAI can

be specified, integrated, and tested in industrial software applications. Participants provided input on the use cases and their contexts during these collaborative sessions. The workshops were complemented with interviews that aimed to gather more detailed insights into the metrics used for quality evaluation in the identified use cases.

To mitigate threats to external validity [113], we applied purposive sampling to select three software companies with variation in industry domain, company size, and maturity of GenAI adoption. This approach allowed us to explore a range of GenAI application scenarios while ensuring relevance to practical settings. Construct validity was also considered during this phase by defining key concepts (e.g., "GenAI use case," "quality evaluation") internally among the research team and validating them through a pilot interaction with one participating company.

4.3.3 Step 3 - Data collection

This section presents the workshop and interview processes of the data collection.

4.3.3.1 Workshop design

All authors participated in designing the workshop, which involved three distinct phases developed through four rounds of meetings and refinements (see Fig. 4.2). In total, three workshops were executed separately in Projects A, B, and C to address the specific contexts of each project.

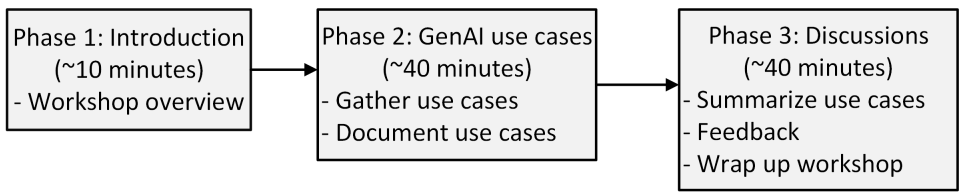

Figure 4.2: Overview of the workshop process.

Phase 1: Introduction (approx. 10 minutes). The workshop began with an introduction, during which we explained the purpose and objectives of the study. Explicitly, the participants were told that the study focused on gathering experiences from use cases where GenAI was applied, explored, or tested. Hence, speculative or future use cases, which had not yet been applied in the projects, were not in scope for this study. This phase ensured participants understood the workshop's goals and their role in contributing relevant insights.

Table 4.3: A template for documenting GenAI use cases [114]. An example inspires participants to fill in their use case details.

Template for a LLM use case	
Use case name	E.g., Software upgrade path assistant
Use case description	E.g., A LLM chatbot assists business analysts to generate report about product feature/interface changes between releases.
Target user(s)	E.g., Business Analyst (BA), Solution Architect (SA)
Challenge(s)/Issue(s) - resolved by this use case	E.g., Business analysts/Solution analysts often struggle to maintain a clear understanding of product changes, particularly when there are frequent releases. Consequently, software upgrades can be quite challenging.
Preconditions- the state before performing/running this use case	E.g., Multiple product releases exist, and an upgrade is anticipated.
Postconditions- the state when the use case finish	E.g., A document is generated with correct product changes in terms of functionalities and interfaces using a LLM chatbot.

Phase 2: Use case specification (approx. 40 minutes). Participants worked individually or in small groups to identify GenAI use cases from their own experiences using a structured template (see Table 4.3). Each individual or group filled out the template to document their use cases uniformly, capturing details such as the use case name, description, target users, challenges addressed, preconditions, and expected outcomes. The workflow for this phase:

1. Participants wrote down their use cases individually or collaboratively.

2. After documenting, participants placed their templates (or post-it notes summarizing their templates) on a shared whiteboard.

3. A round-robin discussion followed, during which participants presented their use cases one by one. This allowed the group to collectively review, clarify, and enhance each use case.

The workshops in Phase 2 were facilitated using guided prompts that asked participants to describe ongoing GenAI use cases, associated quality concerns, and any evaluation strategies currently applied. Workshops were conducted using shared online collaboration boards and real-time notes to document participant input. Two researchers independently reviewed the workshop notes and synthesized recurring patterns, including task types, quality attributes (e.g., accuracy, factuality), and metric use or absence.

The synthesized outputs from each company were then cross-compared to identify recurring themes across cases. These themes, particularly regarding domain-specific evaluation methods/techniques and perceived metric gaps, were used to design the semi-structured interview protocol for Phase 3. Interview questions were constructed to validate workshop findings, explore rationales behind metric use, and capture practices not raised in the workshops. This structure ensured continuity between study phases and allowed deeper investigation of observed practices.

Phase 3: Discussion (approx. 40 minutes). In this phase, the first author presented the preliminary grouping of use cases and invited participants to provide feedback on the clustering. The group reviewed the clusters and engaged in open discussions to refine the groupings, clarify specific use cases, and address questions.

The discussions were broad in scope, covering user interactions with GenAI applications (e.g., achieving specific tasks), developer-centric perspectives such as technical insights, and additional viewpoints like testing, specification, and managerial considerations. This multi-focus ensured that the workshops captured experiences from diverse stakeholders, including developers, managers, end users, and testers, while also addressing the broader context of GenAI applications. This phase facilitated a deeper understanding of the use cases, reduced potential misunderstandings, and allowed participants to contribute diverse perspectives from their professional roles.

Following the discussions, the first author summarized the key insights shared by participants and reviewed the clustering. The clusters that garnered the most feedback and discussion were highlighted. It is important to note that this observation reflects the use cases and application areas that participants found most relevant or engaging during the workshop. However, these clusters do not indicate objective importance or represent the full breadth of possible applications. The clustering and prioritization reflect the perspectives of the workshop's participants based on their current roles and experiences, which could vary if the context or participant sample were changed/expanded. Thus, presenting a potential threat to both the internal and external validity of the study's results, as further discussed in Section 4.6.

Finally, the workshop concluded by summarizing the key results and providing the participants with contact information to provide additional information.

4.3.3.2 Interview design

Following the workshops, we conducted semi-structured interviews with participants to gain further insights into quality measurement practices, the metrics used, and the challenges encountered in applying them.

The semi-structured interview guide was developed by the first three authors. Its design was based on established guidelines for qualitative research [32], and it was based on insights that emerged from our Phase 2 workshops. To improve construct validity, we conducted a pilot interview and refined the interview protocol for clarity and alignment with the study's objectives. We also targeted multiple roles (e.g., developers, architects, managers, operations, and document engineers) to gather diverse perspectives on GenAI quality evaluation.

The guide consisted of thematic sections covering I) the participant's role and involvement in GenAI adoption; II) current practices for evaluating GenAI outputs; III) challenges in metric usage. The complete interview guide is available on Figshare.

Each interview was divided into two phases: introduction and core questions.

Phase 1: Introduction (approx. 5-10 minutes). In the introduction, the interviewer outlined the purpose of the interview and clarified its goals. While the workshops aimed to gather and structure collective knowledge about use cases, the interviews were designed to acquire individual experiences, focusing on detailed explanations of metrics, evaluation methods, and challenges. This complementary focus was explained to participants. Participants were reminded of the confidentiality of their responses to encourage free and objective sharing of their experiences.

Phase 2: Core questions (approx. 30-40 minutes). The core of the interview consisted of open-ended questions that were structured into four key areas:

- Metrics and methods: Participants were asked to describe the specific metrics used to assess GenAI quality and the methods employed to measure each metric.
- Measurement techniques: Questions focused on how participants approached quality measurement, distinguishing between automated, human, and AI-assisted methods.
- Contextual factors: Participants provided details on how organizational and project-specific factors influenced their quality measurement processes.
- Challenges: Participants discussed any challenges encountered in evaluating GenAI quality, including technical and non-technical aspects.

In the end, we conducted 23 interviews across the three selected projects. The interviews were conducted in a mixed format, with onsite and online meetings, depending on participants' availability and location. Each interview lasted between 43 and 55 minutes, allowing sufficient time to explore each topic without burdening the participants.

4.3.4 Step 4 - Data analysis

We conducted a thematic analysis [48] of the collected data to identify patterns and themes related to GenAI use cases, metrics, and challenges. An overview of the analysis process is presented in Figure 4.3.

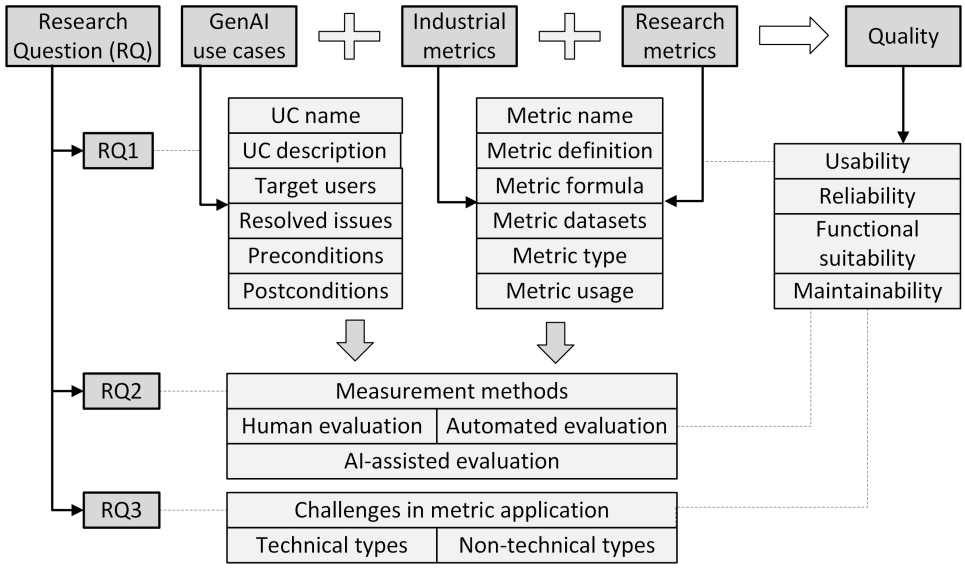

Figure 4.3: Data synthesis based on collected data from the selected industrial projects.

This analysis was performed according to the following steps:

1. *Create initial codes*: A starting set of codes was generated based on recurring concepts in the workshop notes and interview transcripts. The codes included terms related to the collected use cases (e.g., "trouble report analysis"), metrics (e.g., "accuracy"), and quality measurements (e.g., "automated evaluation"). The initial set was guided by our research questions and relevant phrases extracted directly from the data.

2. *Review extracted codes*: The codes were iteratively reviewed and refined to ensure that the codes reflected specific aspects of the data and avoided ambiguity. For example, a code initially labeled "accuracy" was refined to distinguish between "accuracy of document retrieval" and "accuracy of code review."

3. *Connect and combine codes*: Related codes were clustered to form broader categories. For instance, codes related to "accuracy of outputs," "correctness of outputs," and "relevance to context" were grouped into a broader theme —"usability quality characteristics." Similarly, "difficulties with the use of metrics" and "human evaluation challenges" were combined under "challenges in metric application."

4. *Derive themes*: These categories were further synthesized into overarching themes that aligned with the study's research questions. Thematic constructs such as "evaluation practices" and "metric usage" were developed to encapsulate key findings.

To strengthen internal validity and reduce researcher bias, data coding was performed collaboratively by three authors in iterative rounds. Disagreements were resolved through discussion, and key interpretations were reviewed in joint analysis sessions. Preliminary findings were also validated with participants through workshop feedback and cross-checking, supporting the interpretation's reliability.

4.3.4.1 Experiment-driven metrics and mapping process

To incorporate research-based metrics into our analysis, we drew from a systematic snowballing literature review conducted in our previous research (Yu et al. [20]). The review study followed Wohlin et al. [38] guidelines and identified peer-reviewed metrics used to evaluate AI-generated outputs in software engineering contexts.

For clarity, 'experiment-driven metric' was defined as those identified in academic literature and developed for theoretical or experimental purposes. As illustrated in Table 4.4, metrics were classified as industrial if applied in any studied projects that were simple to compute and interpret within specific contexts. Experiment-driven metrics were often more complex, requiring standardized datasets, embeddings, or advanced tools for their computation.

Table 4.4: A classification of industrial and experiment-driven metrics.

Classification aspect	Context-specific metric	Experiment-driven metric
Application context	Industrial settings: - System-specific quality evaluation - Development processes and artifacts (e.g., bug triage accuracy)	Research settings: - Lab-specific cases - Outside specific context (e.g., semantic similarity)
Data type	Domain-specific documents and codebases	Standardized/Sample data Synthetic data
Computation	Rule-based, interpretable metrics (e.g., accuracy of task assignment)	Embedding or lab-based scoring (e.g., FActScore)

The mapping was independently performed by the first two authors, who manually evaluated the contextual relevance of each experiment-driven metric to the GenAI use cases and associated quality characteristics. The results were then compared and refined collaboratively. Disagreements between authors were resolved through discussions, guided by a re-examination of the metric definitions and the industrial context. The final mapping was reviewed and confirmed by all authors to ensure consistency and validity. The mapping process is:

1. Relevance screening: Metrics identified from the literature were screened for relevance to the quality characteristics (e.g., Accuracy, Correctness, Relevance).

2. Contextual fit: Each metric was assessed for its applicability to specific use cases based on its original intended purpose and the type of output it evaluates (e.g., BERTScore for semantic similarity in document summarization, CodeBLEU for evaluating code generation).

3. Domain-expert feedback: The mapping was reviewed by workshop participants, who provided feedback on the practical feasibility of adopting these

metrics in their projects. For example, participants suggested that some metrics, such as FActScore, could complement industrial practices by offering a fine-grained evaluation of factual consistency.

4.4 Results

This section presents our findings to answer the study's research questions.

4.4.1 Results of RQ1 -- What metrics are used to evaluate the quality of GenAI applications?

Firstly, we collected 14 industrial GenAI use cases presented in Table 4.5. These use cases are reported directly as they emerged from the workshops and interviews. Each row documents an existing use case, including its name, description, target users, issues to be resolved, preconditions for execution, and expected outcomes as postconditions. For instance, UC1 highlights how intelligent chatbots can retrieve relevant information from customer product documents. Similarly, UC3 demonstrates how GenAI applications integrated with GitLab can streamline code review processes by examining merge requests.

Table 4.5: Overview of collected industrial GenAI use cases. UC stands for Use Case.

ID	Use case name	Description	Target users	Issues to be resolved	Preconditions to run the use case	Postconditions of the use case
UC1	Intelligent ChatBot	Information retrieval against customer product documents.	Developers Architects Customers	Poor results of keyword searching against many documents.	Customer product information (CPI) documents are ready.	GenAI APPs retrieve the most relevant information from documents' embeddings based on user queries.
UC2	CodingBuddy	Coding assistants integrated in IDE	Developers	Developer productivity	Code repository are ready. IDE is used.	GenAI APPs suggest codes while developers are coding in IDEs.
UC3	GitlabReview	GenAI APP reviews Gitlab merge requests	Developers	Lead time for waiting merge request approvals	Integration to Gitlab and code repositories	GenAI APPs present, rank, and summarize code reviews.
UC4	Log classification	GenAI classifies log files.	Architects Developers	Grouping all reasons for failed functional testing jobs.	Log files are collected from failed functional tests.	GenAI APPs process log files and classify failures into groups.
UC5	Repo analyzer	GenAI assesses Gitlab repositories	Developers Architects	Root-cause analysis in a repository is challenging	Major issues have been reported or observed in a repo	GenAI APP analyzes codes, texts, files, and folders to suggest improvements.
UC6	DeveloperChatBot	Code generation for developer productivity.	Developers	Barriers of knowledge sharing	A knowledge base is required	LLM applications suggest relevant codes based on developers' queries.
UC7	Trouble report analysis	Analyze trouble reports to identify root-causes.	Developers Operations	Root-cause analysis of bugs/errors	Trouble reports contain detail logs of the run-time system	LLM applications suggest the most relevant root causes within the system.
UC8	Document ChatBot	Information retrieval against provided documents.	Developers Architects Testers Managers Customers	Poor results of keyword searching against many documents. Human effort of keyword searches.	The documents are accessible	LLM applications retrieve relevant and expected information from documents' embeddings based on user queries.
UC9	Development copilot	Coding assistants	Developers	Developer productivity	Existing codebases are accessible	LLM applications generate codes that are relevant to the provided contexts.
UC10	Document multi-agents	Document generation, summarization, and refinements.	Developers Architects Testers	Time and effort for document handling	GenAI multi-agent models	LLM agents process documents and generate expected outputs.
UC11	Intelligent code reviewer	GenAI APP reviews code changes in a codebase.	Developers	Leadtime for a proper code reviews	Extracting code changes and code context data are required	LLM APP presents, ranks, and summarizes code reviews.
UC12	AI for anomaly detection	AI-powered anomaly detection for CI logs	DevOps Developers	Failures in CI logs mix all types of errors.	Log files are accessible and captured.	Source information or original texts are cited in LLM generations.
UC13	Test case generation	Generate test cases from provided requirements	Developers Testers	Manual testing is time consuming. Maintaining tests is challenging	Requirements are prepared with detail information.	The data extracted from requirements are knowledge base are input for LLMs as contextualized embeddings to generate test cases.
UC14	Trouble report management	AI APP predicts links between TRs and components of a system	Managers Developers Architects	Lead-time is $\sim$1.5 week while distributing TRs to the ``right'' Dev teams	TR contains sufficient text description and logs from the run-time systems.	LLM APP suggests ranked components of a system linking to Dev teams.

Secondly, we extracted 28 metrics from the selected industrial projects by interviewing study participants. These metrics, including their associated datasets, definitions, and related use cases (UCs), are present in Table 4.6.

Each metric is categorized based on its evaluation focus, such as accuracy (Acc.), correctness (Corr.), alignment (Align.), relevance (Rel.), transparency (Tran.), completeness (Comp.), security (Sec.), and factuality (Fac.).

The table highlights the specific datasets used for each metric (e.g., requirement documents, codebases, system logs). For instance, 'Acc.A' measures accuracy using requirement documents and is linked to UC3 (GitLab reviews), while 'Rel.A' focuses on relevance using user manuals in UC10 (document multi-agents). This second finding illustrates the diversity of metrics and their applications that connect to the identified use cases.

Table 4.6: Collected metrics, datasets, definitions, and formulas from study participants. Note: Each metric is shown only in the use case(s) where it was explicitly reported by participants. While some metrics may conceptually apply to multiple use cases, we did not generalize beyond the empirical data.

Index	Metric ID	Datasets	Quality characteristics	UC ID
1	Acc.A	Requirement docs Source codes	Usability	UC3
2	Acc.B	Requirement docs	Usability	UC10
3	Acc.C	Codebases	Usability	UC11
4	Acc.D	Product component documents	Usability	UC14
5	Acc.E	System logs	Usability	UC7
6	Acc.F	CI build logs	Usability	UC12
7	Acc.G	Code repositories	Usability	UC2
8	Acc.H	Code repositories	Usability	UC9
9	Corr.A	SQL schema	Functional Suitability	UC3
10	Corr.B	Code repositories	Functional Suitability	UC5
11	Corr.C	Programming docs	Functional Suitability	UC6
12	Corr.D	Technical docs	Functional Suitability	UC8
13	Corr.E	Requirement docs	Functional Suitability	UC13
14	Align.A	Functional test logs	Functional Suitability	UC4
15	Align.B	CI logs	Functional Suitability	UC12
16	Align.C	Developer knowledge base	Reliability	UC8
17	Cons.A	Trouble report logs	Reliability	UC4
18	Rel.A	User manuals	Functional Suitability	UC10
19	Rel.B	Gitlab merge requests	Functional Suitability	UC11
20	Rel.C	Trouble reports	Reliability	UC7
21	Rel.D	Coding documents	Maintainability	UC2
22	Rel.E	Developer queries	Maintainability	UC9
23	Rel.F	Test cases	Maintainability	UC13
24	Tran.A	Trouble report logs	Usability	UC14
25	Comp.A	Code reviews	Reliability	UC7
26	Sec.A	System audit logs	Security	UC1
27	Fac.A	Product manuals	Usability	UC1
28	Fac.B	Product documents	Usability	UC6

Thirdly, we grouped the collected use cases and metrics into LLM application domains. Table 4.7 presents four primary application domains: *Document Generation*, *Data Analysis and Insights Generation*, *Customer Service ChatBot*, and *Code Generation*.

For each use case (e.g., UC3, UC10), the table provides the metrics currently used in the projects and suggests experiment-driven metrics that could enhance qual-

ity measurement. For instance, in the Document Generation domain, metrics like Accuracy (Acc.) and Correctness (Corr.) are applied, while research metrics such as METEOR and FActScore are proposed for further evaluation. Similarly, in Code Generation, metrics like Accuracy (Acc.) are complemented by suggestions like CodeBLEU and PASS@k. The table also shows the associated project names, linking the use cases to their industrial contexts.

Table 4.7: LLM application domain view of industrial use cases and metrics --- the one that has been used or can be used.

LLM application domain	Use case ID	Metrics (used)	Research metrics (can be used)	Project name
Document generation	UC3	Acc. (Accuracy of reading XMLs, APIs, and requirements)	METEOR[90]	Project C
		Corr. (Correctness of API calls)	FActScore[25]	
	UC4	Align. (Alignment between LLM outputs and human expectations)	F1Score[110] SARI[115]	
		Cons. (Consistency of the LLM outputs towards the same or similar prompts)	FrugalScore[80]	
	UC10	Acc. (Accuracy of text summarization)	BERTScore[24]	Project A
		Rel. (Relevance of retrieved information from source documents)	YISI[116]	
	UC11	Acc. (Accuracy of code review comments produced by LLM-based systems)	COMET[75]	Project B
		Rel. (Relevance of generated code review reports)	CodeBLEU[26]	
	UC14	Tran. (Transparency of trouble report (TR) generation)	MoverScore[76]	Project B
		Acc. (Accuracy of distributing trouble reports to teams/individuals)	SARI[115]	
Data analysis and insights generation	UC5	Corr. (Correctness of generated insights)	SummEval[117] QAGS[118]	Project C
		Comp. (Completeness of the outputs generated by LLM-based systems)	BLEURT[119]	
	UC7	Rel. (Relevance of the generated TR analysis)	F1Score[110]	Project A
		Acc. (Accuracy of the generated TR analysis)	TRUE[84]	
	UC12	Align.(Alignment between the identified anomalies and human expectation)	N/A	Project B
		Acc. (Accuracy of the identified anomalies)	METEOR[90]	
Customer service ChatBot	UC1	Sec.(Security of system vulnerabilities)	N/A	Project C
		Fac. (Factuality of outputs produced by the ChatBot)	TRUE[84] FActScore[25]	
	UC6	Corr. (Correctness of generated texts and citations)	BLEURT[119]	Project B
		Align. (Alignment of generated texts and source information)	N/A	
	UC8	Corr. (Correctness of chat content towards document embeddings)	MoverScore[76] COMET[75]	Project A
		Fac.(Factuality of generated chat content)	FActScore[25]	
Code generation	UC2 UC9	Acc. (Accuracy of the generated codes)	PASS@k[82]	Project A
		Rel. (Relevance of generated codes)	CodeBLEU[26]	
	UC13	Corr. (Correctness of generated test cases)	Code_eval[29]	Project B
		Rel. (Relevance of generated tests)	PASS@k[82]	

This classification highlights the alignment between practical applications and academic insights, offering a structured view of GenAI's role in industrial quality evaluation.

4.4.2 Results of RQ2 -- What types of evaluation methods are used to measure the identified metrics for GenAI applications?

This section begins by detailing the usage of the metrics collected in RQ1, organized by domain as discussed in sub-sections 4.4.2.1 to 4.4.2.4. Following this, we describe the evaluation methods in sub-section 4.4.2.5, providing an overview of the approaches used in different contexts.

4.4.2.1 Document generation domain

The Document Generation domain includes industrial GenAI applications designed to produce, summarize, or refine documents based on user input or pre-existing data. We identified five use cases (UC3, UC4, UC10, UC11, and UC14) within this domain, focusing on different quality characteristics such as Usability, Reliability, and Functional Suitability. These use cases include code review, log classification, document analysis, and trouble report management.

Table 4.8: Metrics used in document generation domain.

Quality characteristic	Metric	Collected industrial use case
Usability	Acc. (Accuracy)	Create/improve documents (UC10) Code review comments (UC3,UC11) Distributed trouble reports (UC14)
	Tran. (Transparency)	Managing trouble report tickets (UC14)
Reliability	Cons. (Consistency)	Classified log groups (UC4)
Functional Suitability	Align. (Alignment) Rel. (Relevance) Corr.(Correctness)	Log classification (UC4) Retrieved data from docs or codes (UC10,UC11) Code generation (UC3)

Metrics that have been used in the document generation domain: Table 4.8 summarizes the metrics used for each quality characteristic. The metrics applied across the identified use cases include accuracy, alignment, consistency, relevance, and transparency. For instance, accuracy is critical in document summarization (UC10) to capture essential details, while alignment is essential in classification tasks (UC4) to ensure that outputs align with human expectations.

Practitioners' insights: Study participants shared their insights into how GenAI applications improved their workflows and addressed specific pain points in the Document Generation domain. A common document generation workflow is illustrated in Fig. 4.4 based on study participants' inputs. The workflow involves stages from document processing to GenAI-based generation, with most use cases relying on Retrieval-Augmented Generation (RAG) using proprietary enterprise data. The identified context-specific metrics focus on evaluating the outputs of GenAI applications and often depend on domain experts' judgments.

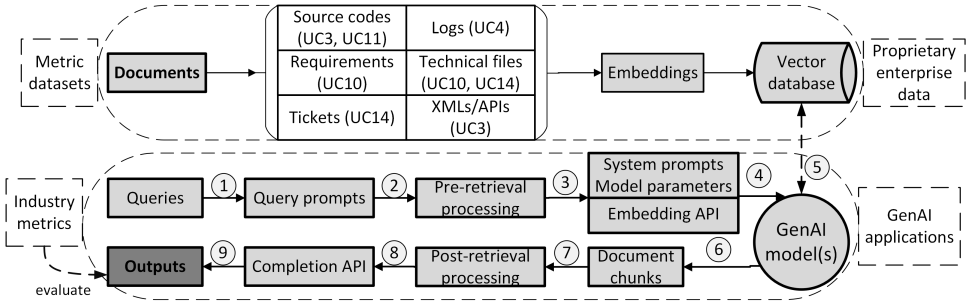

Figure 4.4: Document generation general workflow based on study participants' inputs.

For **code reviews (UC3, UC11)**, several participants highlighted the impact of GenAI on code review processes. One senior developer noted how GenAI solutions reduced the time spent on reviewing complex SQL-related code changes, which often required domain-specific expertise: "Before adopting GenAI, reviewing SQL code was tedious and error-prone, especially when team members lacked expertise in database optimization." Another feedback was, "With GenAI, the tool now flags potential issues, such as unnecessary joins or non-parameterized queries, which helps us address potential vulnerabilities and performance bottlenecks." This insight reflects how GenAI enhances productivity by acting as an assistant, particularly in areas outside developers' immediate expertise. However, participants reported that "the outputs of GenAI are not always optimal and often require human refinement to ensure completeness and accuracy." For example, when the top-ranked code suggestions fail to retrieve the most relevant or contextually accurate recommendations, critical issues may be overlooked.

Regarding **Log classification and troubleshooting (UC4, UC14)**, a lead software architect explained: "We generate thousands of log entries daily from automated tests, and identifying root causes used to be like searching for a needle in an ocean." As stated by a developer, "GenAI's automated classification groups similar errors, such as 'flaky test issues' or 'network failures,' allowing us to prioritize critical problems for fact-based decisions." However, challenges remain, particularly when logs contain ambiguous terms or abbreviations that GenAI struggles to interpret accurately. This sometimes leads to "wrong" classifications, requiring manual re-evaluation by domain experts.

In **document summarization and creation (UC10)** tasks, GenAI was employed to mitigate the burden of processing lengthy technical documents. A project manager shared: "Reviewing hundreds of pages of requirements or design documents was time-consuming and mentally exhausting." GenAI produces structured summaries, which can assist engineers to quickly grasp the main content and focus on strategic decisions. However, some participants noted limitations, such as missing critical content when the GenAI model fails to identify the most relevant texts or keywords from the source documents, often due to incorrect ranking of retrieved data chunks.

Across these use cases, participants reported common themes while using GenAI for document generation. One recurring issue was "non-satisfaction" with outputs, where GenAI provided responses that were either too vague or overly detailed or failed to meet the required level of specificity. Another one was the system's dependency on the quality and completeness of proprietary enterprise data. Missing data or poorly formatted input often led to errors, requiring manual intervention to correct or refine system outputs.

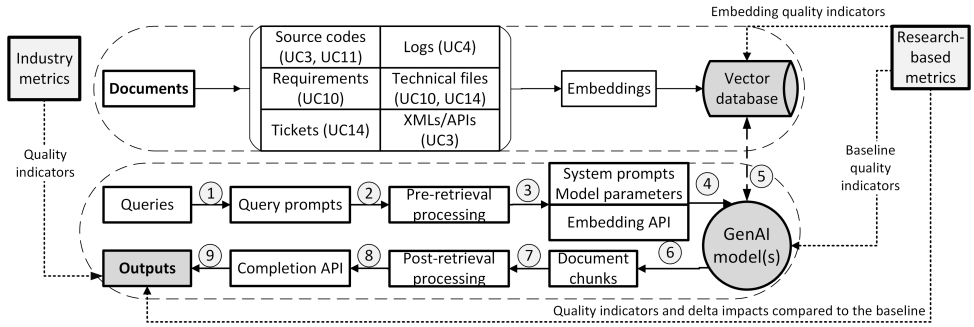

Figure 4.5: Differences between industrial used metrics and experiment-driven metrics.

Experiment-driven metrics in this domain: Metrics, such as METEOR for semantic accuracy or F1Score for classification accuracy, often utilize standardized datasets and embeddings to evaluate theoretical improvements. As shown in Fig. 4.5, the metrics operate through a sequential process: they interact with datasets (e.g., embeddings), evaluate outputs generated by GenAI models, and compare these outputs against predefined baselines to assess their actual quality impact. Based on our discussions with participants, experiment-driven metrics can offer:

- A deeper understanding of GenAI application quality: Depending on the specific needs of the evaluation, experiment-driven metrics can provide more detailed insights into certain quality attributes (e.g., factuality or semantic alignment).
- A more automated approach for quality evaluation: These metrics often come with scripts and tools, which can streamline evaluations in controlled environments, reducing reliance on subjective judgments. However, the extent to which automation is feasible depends on the availability of the datasets and the alignment of the metrics with industrial contexts.

4.4.2.2 Data analysis and insight generation domain

The *"Data analysis and insight generation"* refers to a thematic grouping of the collected use cases where GenAI applications are employed to analyze data and produce

actionable insights. These insights enable teams to make informed decisions based on patterns, trends, and contextual information extracted from complex datasets.

Table 4.9: Metrics used in data analysis and insight generation domain.

Quality characteristic	Metric	Identified industrial use case
Functional Suitability	Corr. (Correctness)	Analyzing GitLab Repositories (UC5)
Reliability	Comp. (Completeness)	Root-Cause Analysis (UC7)
Functional Suitability	Rel. (Relevance)	Detected anomalies (UC12)
Usability	Acc. (Accuracy)	Detecting Anomalies in CI Logs (UC12)

We identified three use cases in this domain (UC5, UC7, and UC12), which focus on improving productivity and accuracy in tasks such as code repository analysis, root-cause analysis, and anomaly detection. Table 4.9 summarizes the metrics that have been used to evaluate the accuracy, completeness, relevance, and correctness of the generated insights. For example, Completeness is crucial in root-cause analysis (UC7) to ensure all relevant patterns are identified, while Accuracy is key for anomaly detection (UC12) to catch issues impacting CI builds' stability.

Practitioners' insights: Study participants shared their insights on how GenAI applications have reduced the manual effort in the data analysis and insight generation domain. A general workflow is shown in Fig. 4.6 based on the collected data from participants. The process begins with uploading data from several sources, including codebases, logs, and tickets. The data undergoes querying and is processed with system prompts and model parameters before entering the GenAI models. The outputs generated by the GenAI model are then evaluated using the identified metrics.

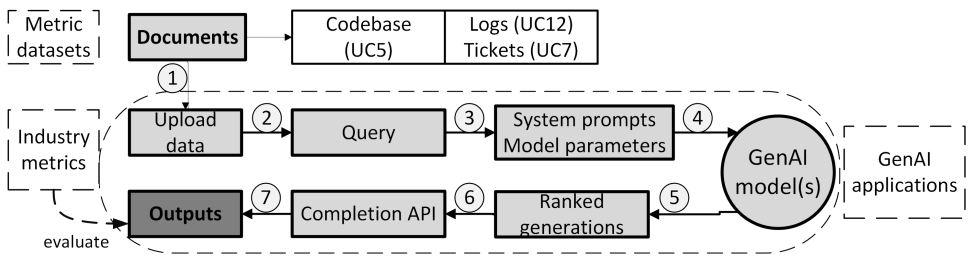

Figure 4.6: A generic workflow for data analysis and insight generation using GenAI.

In **UC5** (analyzing gitLab repositories), a software architect highlighted how GenAI has transformed the process of understanding large codebases. "Analyzing dependencies and identifying technical debt was once a labor-intensive and error-prone task," the architect explained. "With GenAI, we can quickly identify redundant modules and outdated dependencies, which could otherwise lead to security vulnerabilities." However, participants noted a few challenges, such as cases where a GenAI application provided incorrect suggestions due to its inability to fully understand the highly customized codebases.

In **UC7** (root-cause analysis for trouble reports), an operations manager shared his experience of using a GenAI application to streamline the analysis of trouble reports. "Reviewing and diagnosing issues across complex, multi-component systems was challenging. GenAI's ability to identify patterns and pinpoint root causes can reduce our troubleshooting time, especially for issues that span across different modules." Despite this, a limitation frequently mentioned was that GenAI struggled with issues involving interdependencies between modules. The generated analysis was either incomplete or too generalized, necessitating deeper manual investigation.

For **UC12** (anomaly detection in CI logs), a DevOps engineer reported that "manually identifying critical errors in the CI environment was not easy. With GenAI models, we can catch issues earlier in the process, which not only reduces the time needed to fix critical build errors but also enhances CI jobs' stability." Meanwhile, several developers pointed out that GenAI's anomaly detection sometimes produced false positives, requiring teams to correct them manually.

Challenges in industrial adoption: While these experiment-driven metrics offer a more granular evaluation, their adoption in industry often requires specialized knowledge, particularly in setting up the metrics and integrating them into existing workflows. Metrics like QAGS and BLEURT rely on embedding models, which may not be familiar to many industrial teams. While interpreting the results is generally straightforward, the primary challenge lies in configuring these metrics, specifically in preparing the required datasets. This knowledge gap can limit their immediate applicability without prior training or expert guidance.

4.4.2.3 Customer service chatbot domain

This domain employs GenAI applications to enhance customer interaction by providing more context-aware responses. GenAI chatbots can handle a wide range of customer queries, from answering product-related questions to technical queries. The primary goal is to streamline customer support processes, reduce wait times, and improve customer satisfaction.

We identified three use cases within this domain: **UC1**, **UC6**, and **UC8**, focusing on two main quality characteristics: usability and reliability.

Table 4.10: Metrics that have been used in the customer service chatbot domain.

Quality characteristic	Metric	Identified Use Case
Usability	Corr. (Correctness)	Product support chatbot (UC1)
	Fac. (Factuality)	Developer chatbot for coding assistant (UC6)
Security	Sec. (Security)	Product operation (e.g., delete commands) (UC1)
Reliability	Align. (Alignment)	ChatBot for query knowledge base (UC8)

Context-specific metric: Table 4.10 shows the metrics that have been utilized for evaluating Usability, Security, and Reliability of a software system. For example, Factuality is critical for the developer ChatBot (UC6) to ensure that coding support

provided by GenAI is technically sound, while Security is crucial for the Intelligent ChatBot (UC1) to safeguard sensitive information during user interactions.

Practitioners' insights: Study participants described a common workflow while employing GenAI for chatbot services. The workflow is illustrated in Fig. 4.7. The main differences compared to the other application domains are: I) the chatbot's role specification is required; II) contextualized information through embeddings or prompts is needed.

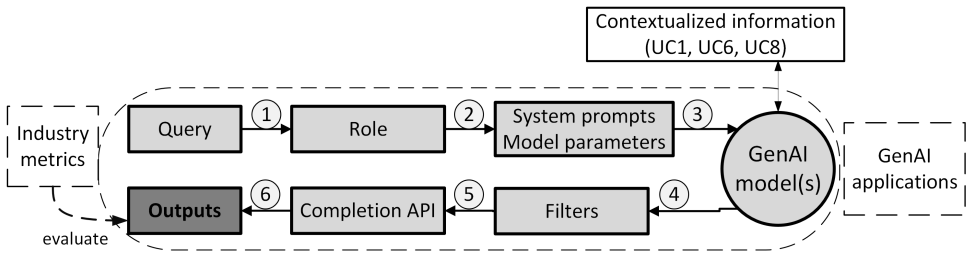

Figure 4.7: A common workflow for customer service chatbots.

In **UC1** (chatbot for product information support), a product support manager noted that the GenAI chatbot reduced the volume of routine queries handled by operations teams. As the manager explained, "Since implementing the GenAI-powered chatbot, we have been able to handle a larger number of queries securely, reducing the workload on our teams and improving response times." This benefit was echoed in **UC6**, where a software engineer observed that the GenAI chatbot enabled developers to access relevant code snippets and best practices directly within their workflow, accelerating troubleshooting and implementation. Similarly, in **UC8**, a technical writer highlighted the chatbot's ability to retrieve information from large document collections, which improved user satisfaction and reduced search times. In short, these use cases illustrate how GenAI chatbots can streamline operations, enhance productivity, and make information more accessible.

However, study participants reported several limitations that hindered the full potential of GenAI-based chatbots. A major challenge across all use cases was maintaining the accuracy and factuality of GenAI system outputs. For example, in UC1 and UC8, incorrect information retrieval or outdated content led to user frustration and increased follow-up queries, indicating a need for regular data curation and model updates. In UC6, the Developer ChatBot occasionally provided code snippets incompatible with the other dependencies, requiring developers to rework existing code.

Challenges in industrial adoption: It is important to note that many metrics, such as FActScore and TRUE, require computational resources that may hinder real-time

responsiveness, which is critical for applications like customer service chatbots or some decision-making systems. The complexity of implementing metrics like MoverScore, which relies on extra AI models, poses challenges for industrial teams due to the increased costs, required additional computational resources (e.g., GPUs).

4.4.2.4 Code generation domain

The Code Generation domain uses GenAI applications to assist developers in generating code snippets, fixing bugs, and providing code suggestions. We found three use cases (UC2, UC9, and UC13) in this domain, and each UC evaluated different system quality characteristics, such as Usability, Reliability, and Maintainability.

Table 4.11: Metrics used in the code generation domain.

Quality characteristic	Metric	Identified Use Case
Usability	Acc. (Accuracy)	Coding Assistants in IDE (UC2,9)
Reliability	Corr. (Correctness)	Contextualized test case generation (UC13)
Maintainability	Rel.(Relevance)	Code Generation (UC2,9,13)

Metrics: Table 4.11 presents the quality characteristics, used metrics, and the identified use cases. For example, Acc. (Accuracy) is essential for GenAI-based coding assistants to ensure the generated code is accurate, while Corr. (Correctness) ensures reliability in contextually generated code for complex systems.

Practitioners' insights: The inputs from study participants enabled us to outline a general workflow illustrating how GenAI models were utilized across the identified use cases. As shown in Fig. 4.8, Integrated Development Environment (IDE) APIs served as the primary interfaces through which engineers interacted with GenAI models for code generation. While the APIs provide the necessary integration points, the scalability of these solutions primarily depends on the performance of the APIs, the underlying models queried through them, and the hardware infrastructure supporting those models. Factors such as response times, concurrency handling, and the computational resources available to the models are critical in maintaining the quality of generated code and ensuring support for a large number of connected developers.

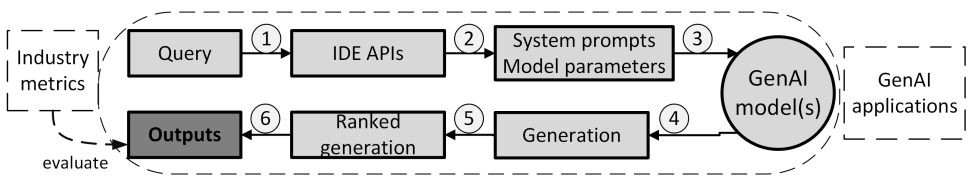

Figure 4.8: A generic workflow for GenAI code generation based on participants' inputs.

An observation among developers was the ability of GenAI tools to help them 'stay in the zone' while coding. As one developer in UC2 mentioned: "Having an

AI coding tool directly in the IDE means I do not have to break my ongoing work to search for code solutions online." This ability to provide quick, contextual suggestions kept developers focused, accelerating task completion by integrating help directly where they work.

GenAI's context-awareness was valuable when generating code that needed to fit into complex systems. A lead developer in UC9 praised Codeium's ability [1] for producing code that aligns with the existing source files: "The contextualized code suggestions mean we're not just getting code, but code that fits into our existing class/function." This alignment may reduce the time spent on rework, making GenAI tools useful for implementing new features within established codebases. However, study participants reported that they occasionally fell short in handling more complex tasks, requiring developers to step in and make manual adjustments.

In testing, testers noted that manually creating test cases for each scenario was often a bottleneck. As one participant observed, "With GenAI, we generate test suites in a fraction of the time, which allows us to focus more on exploratory testing." This improved test coverage for simple scenarios, but GenAI often missed critical edge cases that required a deeper understanding of specific requirements. As a result, human oversight is required to ensure that critical test scenarios are covered.

Challenges in adopting experiment-driven metrics: No major blockers have been reported that prevent the use of these experiment-driven metrics in industrial projects. However, one potential reason is that industrial engineers may simply be unaware of these metrics, indicating a need for closer collaboration between academia and industry. Additionally, industrial teams may require training to incorporate new metrics.

4.4.2.5 Types of evaluation methods

Our investigation of the collected industrial GenAI use cases revealed that different metrics require different levels of domain-experts' involvement and automation, depending on the complexity of the task and industrial contexts. Table 4.12 summarizes three types of evaluation methods, such as human, automated, and AI-assisted evaluations, to group the identified context-specific metrics.

Human evaluations are often necessary for tasks that require subjective judgment, contextual understanding, or expertise that cannot be easily codified into rules. However, the evaluation can be time-consuming and inconsistent across evaluators. On the other hand, **automated evaluations** enable faster, scalable assessments that can handle large volumes of data, but may lack the ability to understand complex scenarios. The emergence of **AI-assisted evaluations** bridges this gap by using AI models to assist or complement human and automated evaluations. These models can provide analyses that are more detailed and contextualized compared to simpler metrics or traditional human evaluations, such as contextual relevance scoring.

[1] https://codeium.com

Regarding the metrics in the identified evaluation methods, **Accuracy** (Acc.) and **Correctness** (Corr.) show a balanced reliance on both *human* and *automated* evaluation, indicating that while automated checks can handle various validations, manual review is still essential for verifying the outputs.

Table 4.12: Evaluation types for the identified metrics.

Metric	Manual Eval.	Automated Eval.	AI-Assisted Eval.
Acc.	✓(manual review)	✓(automated checks)	✗
Corr.	✓(manual verification)	✓(automated validation)	✗
Align.	✓(expert assessment)	✓(semantic analysis)	✓(AI models)
Comp.	✓(manual check)	✓(automated checks)	✗
Rel.	✓(subjective review)	✗	✓(AI scoring)
Fac.	✓(fact-checking)	✗	✓(AI verification)
Sec.	✓(security review)	✓(vulnerability scan)	✗
Tran.	✓(clarity review)	✗	✗

Relevance (Rel.) and **Factuality** (Fac.) metrics primarily benefit from *AI-assisted* evaluation alongside human oversight. For instance, AI models can provide contextual relevance scores or fact-checking support, but final decisions often need to be reviewed by humans to ensure reliability.

Conversely, **Security** (Sec.) emphasizes the importance of *automated* evaluation, such as vulnerability scanning, while still requiring manual security reviews to catch context-specific issues that automated tools might miss. However, **Transparency** (Tran.) relies solely on human evaluation, reflecting that clarity and readability are mainly judged by human reviewers, given their subjective nature.

In summary, human, automated, and AI-assisted evaluation approaches can be employed while using industrial and experiment-driven metrics to measure the quality of GenAI-based systems. The practical insights and challenges highlighted in this study should be considered to ensure successful implementation in industrial applications.

4.4.3 Results of RQ3 -- What challenges are encountered in applying metrics to evaluate the quality of GenAI applications?

As stated by participants from the selected projects, with the increasing deployment of GenAI, scalable evaluation methods are critical. Challenges such as automated output verification and managing context can hinder scalability. As stated by a developer in Project C, *"Manual checking is okay at first, but it can hardly scale when we deal with thousands of prompts per day."*

Managing context refers to ensuring that the evaluation metrics account for the specific requirements and dependencies of the assessed use cases. For example, in code generation, the relevance of generated code snippets depends on their compatibility with existing frameworks or system architecture. A lead developer reported:

"A code snippet might work in isolation, but it would be useless if it does not fit our current technology stack."

Another issue raised was the interpretability of metrics. As mentioned by an R&D manager, *"the metric says 0.83 accuracy, but what does that mean for my team or product? We cannot act on that."* Participants also highlighted a lack of awareness or access to advanced research metrics. For instance, a senior architect shared: *"I was not even aware of some of these evaluation tools until your interview. We mostly rely on internal manual verification."*

Table 4.13: Challenges related to quality evaluation with metrics. MC refers to metric-related challenges.

ID	Challenge name	Description
MC1	Chunk Size for Retrieval	Choosing appropriate chunk sizes can impact the quality of GenAI outputs. Smaller chunks may improve precision but lose context, while larger chunks retain context but may dilute relevance. Balancing these factors is a technical challenge noted by Regenwetter et al. [120]
MC2	Human Output Verification	Manual review is essential for assessing nuanced aspects like relevance and factuality, but it can be time-consuming, inconsistent, and difficult to scale, as highlighted in Kenthapadi et al. [15]
MC3	Output Consistency	Achieving consistent responses for similar prompts is challenging. Minor variations in input phrasing can lead to different outputs, complicating the assessment of GenAI system reliability. This is a well-documented issue in GenAI systems [121]
MC4	Context Embeddings	Determining which context data to embed and how to do it effectively is crucial for improving GenAI outputs. Ineffective embeddings can lead to incorrect or irrelevant responses, as explored in Regenwetter et al. [120]
MC5	Metric Interpretability	Even when metrics are effectively implemented, their scores must be clear and interpretable for stakeholders. Ensuring metrics provide actionable insights rather than just numerical values is essential.

As illustrated in Table 4.13, five main challenges have been identified. These challenges align with prior research findings, which have explored issues like chunk size optimization and context embeddings in GenAI systems. For example, Regenwetter et al.[120] discuss the trade-offs in chunk sizes for retrieval, while Jiang et al.[121] highlight the difficulty in achieving output consistency. However, MC5 (Metric Interpretability) emerged as a distinct concern in the industrial evaluations conducted in this study, emphasizing the importance of actionable metrics tailored to practical use.

4.5 Implications for practice

Based on the findings from our multi-case study, we highlight applicable quality aspects, recommend metrics, and suggest steps to apply them in four identified SE domains. We intend to help practitioners identify entry points for metric adoption and build progressively toward more structured quality evaluation practices for GenAI applications.

4.5.1 Document generation domain

Document generation can be used for summarizing requirements, generating implementation proposals, or producing compliance reports in software development. The

quality of outputs in this domain is often judged by practitioners based on accuracy, completeness, and factual correctness, but a human-in-the-loop is needed. Based on our findings, we suggest the following metrics and practices, as outlined in Table 4.14.

Table 4.14: Suggested metrics and practices for document generation.

Quality aspects	Metric	Steps to apply in practice
Accuracy	**Task-specific correctness rate** (rule-based)	Step 1: Define required content (e.g., top 3 changes). Step 2: Count captured vs. missed or hallucinated items. Step 3: Use checklists or templates for structured review.
Relevance	**BERTScore** (embedding-based semantic similarity)	Step 1: Compare GenAI output with cited examples. Step 2: Set threshold ranges based on task type. Step 3: Apply in batch mode for multiple summaries.
Factuality	**FActScore** (embedding and entity-level matching)	Step 1: Apply to high-stakes content (e.g., compliance documents). Step 2: Automate comparison with trusted source documents if feasible. Step 3: Add manual review for edge cases.
Alignment with expectations	**Reviewer scorecard** (human checklist or rating)	Step 1: Ask stakeholders to rate clarity, coverage, and usefulness. Step 2: Use Likert-scale or task-based scoring forms. Step 3: Collect feedback to improve prompting and review flow.

Practitioners are encouraged to embed these checks into their current workflows. Lightweight rule-based metrics can serve as first-pass filters. Embedding-based metrics, such as BERTScore or FActScore, can be used selectively when precision, traceability, and quality assurance are critical.

4.5.2 Data analysis and insight generation

For analyzing software artifacts (e.g., logs, error reports, and historical issues), GenAI can be used to generate actionable insights or identify anomalies. Key quality characteristics for these tasks include correctness of insights, relevance, traceability to source data, and completeness. The following metrics and practices, as listed in Table 4.15, are recommended based on our findings.

Table 4.15: Suggested metrics and practices for data analysis and insight generation.

Quality aspects	Metric	Steps to apply in practice
Correctness	**Issue classification accuracy** (rule-based or supervised)	Step 1: Define labels for known categories (e.g., bug, enhancement). Step 2: Calculate accuracy or F1-score over labeled test sets. Step 3: Use a confusion matrix to diagnose misclassifications.
Relevance	**Precision** (ranking-based relevance metric)	Step 1: Use feedback from domain-experts to label top-k outputs as relevant or not. Step 2: Compute relevance proportion within top-k (e.g., top 3 results). Step 3: Optimize GenAI prompt or parameters for top-k precision.
Traceability	**Reference coverage rate** (rule-based)	Step 1: Check if generated insights include references to correct artifacts (e.g., logs). Step 2: Count correct vs. missing trace links. Step 3: Apply rule-based extraction for audit traceability.
Completeness	**Insight completeness score** (review-based)	Step 1: Define a checklist of expected insight components. Step 2: Manually score the presence or absence of components. Step 3: Use sample-based audits for scaling to larger datasets.

Practitioners should start with human validation of GenAI-generated insights and then integrate rule-based metrics or feedback scoring to gradually scale evaluation.

4.5.3 Customer service

GenAI-based support agents can assist users with troubleshooting and technical questions. Quality evaluation in this domain accounts for correctness, factual reliability, and information safety. Based on our findings, we recommend the following metrics and practices, as shown in Table 4.16.

Table 4.16: Suggested metrics and practices for customer service.

Quality aspects	Metric	Steps to apply in practice
Correctness	**Response accuracy** (manual or rule-based)	Step 1: Define expected correct answers for standard queries. Step 2: Evaluate response correctness against known responses. Step 3: Track accuracy across different interaction types.
Factuality	**FActScore** (embedding & entity-level matching)	Step 1: Use for validating GenAI answers against verified knowledge bases. Step 2: Apply selectively for high-impact or regulated topics. Step 3: Log low-scoring responses for audit and improvements.
Role alignment	**Role classifier score** (supervised or LLM-based)	Step 1: Use classifiers to assess sentiment and role. Step 2: Align role to context (e.g., empathetic for error messages). Step 3: Integrate with QA workflows for flagged conversations.
Information safety	**Sensitive content detection** (rule-based or LLM-based)	Step 1: Use regex or LLM to detect leakage of private or harmful information. Step 2: Evaluate model responses against policy checklists. Step 3: Block or alert on risky responses in real time.

GenAI chatbots require quality controls that extend beyond functional correctness to include safety, trustworthiness, and user experience. It is important for practitioners to filter sensitive content and flag hallucinations using multiple question-answering metrics.

4.5.4 Code generation

GenAI has been used to support developers in writing, refactoring, reviewing, and documenting code. In this domain, evaluation focuses on accuracy, relevance to intent, functional validity, and maintainability. Based on our findings, we recommend the following metrics and practices, as presented in Table 4.17.

Table 4.17: Suggested metrics and practices for code generation.

Quality aspects	Metric	Steps to apply in practice
Accuracy	**Unit test pass rate** (execution-based)	Step 1: Run the generated code against predefined unit tests. Step 2: Count total passed vs. failed cases. Step 3: Use as a gating metric for GenAI output acceptance.
Relevance	**Intent alignment score** (prompt-task match)	Step 1: Define intended functionality from prompts. Step 2: Review if the generated code aligns with the functional intent. Step 3: Use developer feedback to refine the scoring scale.
Functionality	**CodeBLEU** (embedding-based syntax & semantics)	Step 1: Compare the generated code with the reference implementation. Step 2: Use CodeBLEU to evaluate syntax and structure. Step 3: Combine with static analysis for a holistic view.
Maintainability	**Cyclomatic complexity** (static analysis)	Step 1: Measure the complexity of the generated code using analysis tools (e.g., PMD). Step 2: Set internal complexity thresholds by team/project. Step 3: Flag overly complex output for refactoring.

For GenAI-assisted coding, evaluation can be embedded directly into developers' Integrated Development Environments (IDEs). Embedding-based metrics, such as CodeBLEU, can complement human reviews.

4.6 Validity threats

We structure our discussion of validity threats following the guidelines by Runeson and Höst's [32] four categories: internal, external, construct, and reliability validity. Mitigation strategies applied during the study design and execution are described in Section 4.3.

4.6.1 Internal validity

One potential threat is the distribution of participants across the selected projects. Participants from some roles, such as operation engineers or document engineers, may concentrate more on one or two projects. This uneven distribution could lead to some projects having more or fewer practices and challenges documented compared to others, potentially biasing the results.

To mitigate this potential threat, the data analysis [111] accounted for variations in project contexts, ensuring that insights from projects with more participants were cross-verified against those with fewer participants. Representativeness was determined based on the diversity of roles and the relevance of the GenAI use cases within each project context, rather than simply the frequency of GenAI application.

4.6.2 External validity

The study focuses on three specific organizations, which cannot represent the full range of industries or geographical regions that use GenAI. Although it may be challenging to include all industries, selecting a diverse set of companies can enhance representativeness. Additionally, the metrics identified and discussed in this study reflect practices observed in these specific organizations and may not generalize to all industrial contexts. Future research is expected to address this limitation by broadening the scope of case studies to include a more diverse range of industries and application domains.

It is important to note that this study does not evaluate GenAI models themselves (e.g., architecture, licensing, cost-performance trade-offs, or infrastructure dependencies). Such model-level factors are crucial for system-level decisions but are beyond the scope of this work. Our focus remained on evaluating the output and quality characteristics of GenAI applications in practice, as experienced by industrial practitioners. Benchmarking GenAI models has been extensively studied elsewhere and complements but does not overlap with our objectives.

4.6.3 Construct validity

There may be differences in how certain metrics (e.g., accuracy, relevance) are defined and understood in research versus industrial contexts. This could lead to misun-

derstandings in data interpretation. We mitigated this threat by clarifying definitions during workshops and interviews, ensuring that all participants had a shared understanding of key concepts.

The use of self-reported data through interviews and workshops could introduce bias. Participants might overstate successes or underreport challenges. To address this, we assured participants of anonymity and encouraged honest and open discussions.

4.6.4 Conclusion validity

Given the exploratory nature of a multi-case study, there's a risk of over-interpreting qualitative findings. We ensured that our conclusions were grounded in the data by providing direct participant quotes and following a systematic coding process.

As this study is based on qualitative data, the findings may lack statistical generalizability. We aimed to provide insights into the experiences of practitioners. We recommend that future research include quantitative methods to complement and validate these findings.

4.7 Discussions

4.7.1 How much accuracy is enough?

While existing metrics provide ways to quantify the accuracy of AI-generated outputs, our findings suggest that achieving satisfactory levels of accuracy remains a challenge in many industrial applications. Participants often expressed concerns about the impact of inaccuracies on trust and user satisfaction. One participant highlighted that "a 10% inaccuracy in AI-generated outputs could lead to a permanent loss of customer trust." While this statement highlights the critical role of precision in maintaining user confidence, it also underscores the importance of metrics in providing insights into system correctness. Metrics with high precision enable practitioners to understand and quantify the limitations of GenAI system outputs, guiding iterative improvements to the system. Without such metrics, identifying and addressing gaps in system correctness would largely rely on guesswork.

Another participant offered a broader perspective, suggesting that "Instead of focusing solely on accuracy, we should consider the tangible improvements GenAI tools bring to our workflow." This view supports the importance of metrics beyond measuring accuracy—they serve as tools to evaluate incremental improvements and track the evolving performance of GenAI systems. Metrics provide the necessary data to establish thresholds for acceptable performance, which are critical for balancing improvement against the cost of inaccuracies. For example, identifying an acceptable accuracy threshold through metrics can help stakeholders decide when

the system's outputs are sufficient for deployment.

For less accuracy-sensitive domains, metrics can play an equally important role. Measuring improvements over time, such as tracking trends in accuracy gains or reductions in error rates, allows teams to monitor progress and make informed decisions. "What threshold of improvement or accuracy gain will satisfy end users?" becomes a question that metrics can help answer by providing more objective data.

4.7.2 Challenges of contextual understanding

One of our identified barriers to GenAI adoption in software engineering is the mismatch between output quality and contextual expectations. Our findings highlight a growing concern about whether the generated content aligns with task-specific and organizational context. For practitioners, it means that teams cannot simply "drop in" a GenAI model and expect useful results. Instead, GenAI output evaluation needs to incorporate local standards — such as terminology, structure, or procedural conventions — which are not captured by many metrics. Our findings confirm the challenges raised in Kenthapadi et al. [15] about GenAI's lack of grounding in enterprise-specific knowledge.

Practitioners spend manual effort to adapt GenAI outputs to their domain, such as embedding company-specific terminology into prompts, fine-tuning the information retrieval process, or involving human reviewers to validate the results. However, current evaluation tools and metrics do not fully support these needs. To address this gap, future metrics should cover: *context awareness* — able to validate whether an output respects domain constraints, and *configurable* — allowing teams to encode domain-specific rules.

For metric adaptation, our findings uncover that evaluation scores are useful when adjusted to domain expectations, which requires domain-expert feedback and custom thresholds. Practitioners should treat contextual evaluation not as a static task, but as an iterative process.

4.7.3 Bridging research and practice in metric usage

Our study shows a difference between experiment-driven and context-specific metrics. In academic research, experiment-driven metrics like BLEU, ROUGE, BERTScore, or FActScore are commonly used. These metrics offer evaluations of aspects like semantic similarity and factual accuracy. However, they rely on online datasets and reference outputs that are often unavailable or impractical in industrial settings.

On the other hand, practitioners in industry tend to use context-specific metrics grounded in artifacts used in their development, such as test cases, bug reports, or requirement documents. Using these artifacts, the metrics are easier to compute, interpret, and act upon, especially in domains with time or resource constraints.

Despite this divergence, the two groups of metrics should not be seen as mutually exclusive. Several metrics in academia, including FActScore or BERTScore, could be adapted to industrial use. For example, these metrics could be adopted to assess factuality in generated documentation, while rule-based systems or expert review layers capture domain-specific correctness. Moreover, industry teams may benefit from understanding the methodological rigor of academic metrics, especially for tasks like summarization, log interpretation, or code review, where context-aware accuracy is difficult to measure. Future research may explore how these two metric categories can be translated or hybridized for use in enterprise-grade GenAI systems.

4.8 Conclusions

This study presents 28 metrics, three evaluation approaches, and five challenges for the quality measurement of Generative AI (GenAI) applications from 14 industrial GenAI use cases spanning domains such as document generation, data analysis, customer service chatbots, and code generation.

Study results highlight the importance of a multi-faceted approach to quality measurement for GenAI applications. For example, a combination of human, automated, and AI-assisted evaluation while using metrics. Metrics such as accuracy, relevance, alignment, correctness, transparency, completeness, security, and consistency are helpful for evaluating GenAI system outputs. However, traditional metrics alone are insufficient. Companies need to incorporate multiple/combined metrics that measure system qualities in specific industrial contexts. For instance, developers can leverage these metrics to answer critical questions, like "Are we making enough progress?" or "Is the application 'good enough' to be deployed?" Managers can gain a better understanding of an application's quality during the development cycle, enabling informed decisions about deployment readiness.

Moreover, some metrics in research studies may not fully align with the specific requirements of industry applications. Industry practitioners prioritize metrics that are interpretable, actionable, and adaptable to domain-specific contexts. The identified challenges, such as metric interpretability and the use of context embeddings, reinforce the practitioners' preferences.

In short, as GenAI has been integrated into industrial processes, practical quality measurement is needed. By combining both academic and context-specific metrics, organizations can improve the quality measurement of GenAI applications that reflect theoretical rigor and align with real-world applications' demands.

For future work, building upon this study, a quality model could be developed by exploring broader industrial contexts. Such a model would synthesize quality characteristics, metrics, and software development domains to provide guidance on what to measure, when to measure, and how to achieve high quality in GenAI-supported development.

Study D

Paradigm shift on Coding Productivity Using Generative AI

Abstract

Generative AI (GenAI) applications are transforming software engineering by enabling automated code co-creation. However, empirical evidence on GenAI's productivity effects in industrial settings remains limited. This paper investigates the adoption of GenAI coding assistants (e.g., Codeium, Amazon Q) within telecommunications and FinTech domains. Through surveys and interviews with industrial domain experts, we identify primary productivity-influencing factors, including task complexity, coding skills, domain knowledge, and GenAI integration. Our findings indicate that GenAI tools enhance productivity in routine coding tasks (e.g., refactoring and Javadoc generation) but face challenges in complex, domain-specific activities due to limited context-awareness of codebases and insufficient support for customized design rules. We highlight new paradigms for coding transfer, emphasizing iterative prompt refinement, an immersive development environment, and automated code evaluation as essential for effective GenAI usage.

5.1 Introduction

Software development practices have progressed from manual coding toward increasingly automated and intelligent workflows [122], as shown in Figure 5.1.

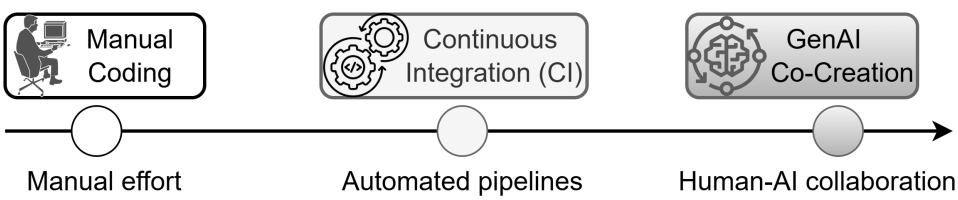

Figure 5.1: Progress in software development practices

A key milestone in this progress was the adoption of Continuous Integration (CI) [123]. CI automates verification and validation pipelines and offers fast feed-

back to reduce integration overhead and accelerate the coding cycle. Building on CI pipelines for efficiency, *Generative Artificial Intelligence* (GenAI) [59] has emerged as a next step, leveraging Large Language Models (LLMs) [51] to generate both text [66] and code [81]. Whereas CI improves code quality by rapidly detecting human errors, GenAI promises to expand automation into *co-creation*, enabling developers to offload specific coding tasks to AI assistants (e.g., Codeium, Amazon Q).

Despite these technological advances, there is limited empirical evidence [122] about how GenAI coding assistants affect broader aspects of *coding productivity*, such as problem-solving process, learning experiences, and overall code quality [124]. To address this gap, we investigate the impact of GenAI coding assistants in industrial contexts, analyzing how they reshape development practices and what lessons emerge for practitioners while adopting AI-supported code generation. We assume that users experience productivity gains with GenAI compared to traditional coding work without GenAI, and domain-specific knowledge in using GenAI is related to the productivity impacts.

The *main contributions* of this study are: I) Empirical data on GenAI coding tools' impact on coding productivity; II) Practical lessons while using GenAI coding assistants.

The remainder of this paper is structured as follows. Section 5.2 introduces background information and related work. Section 5.3 outlines the research methodology. Section 5.4 reports study findings. Finally, Section 5.5 concludes the study results and future research.

5.2 Related work

While automated code generation has been explored for decades, recent approaches utilizing LLM-based models have opened new opportunities. Omidvar et al. [125] studied LLM-based code migration, highlighting the importance of a human-AI partnership in complex refactoring tasks. Similarly, Heng et al. [126] showed that LLM-driven code generation could support learning new frameworks (e.g., Flutter), indicating potential productivity gains and educational benefits.

Other studies focus on the empirical impact of AI-based code assistants in professional environments. Gonçalves et al. [127] conducted an empirical assessment of GitHub Copilot, revealing gains in coding speed and noting concerns regarding code correctness. Corso et al. [128] extended this line of work by measuring the adequacy of AI-based code assistants in method-generation tasks, underscoring the crucial need to manage AI-generated technical debt. Despite these contributions, the literature lacks evidence concerning qualitative dimensions of *productivity*, such as industrial engineers' quality perceptions and design rules sharing across teams. Our work addresses this gap by investigating industrial settings, offering insights into how

GenAI tools reshape modern software development.

5.2.1 Definitions

Productivity: Inspired by Gonçalves et al. [127], we define productivity as a combination of *user satisfaction, coding tasks, acquired skills/knowledge from using tools, and quality perceptions of tools' outputs* rather than merely lines of code completed or numbers of bugs reduced. Here, coding tasks refer to activities such as implementing features, refactoring code, debugging, writing tests, code reviews, and documentation.

5.3 Research Methodology

We conducted a multi-case study following Runeson et al.'s guidelines [32]. Survey and semi-structured interview methods were used to extract in-depth insights into the uses of GenAI coding assistants.

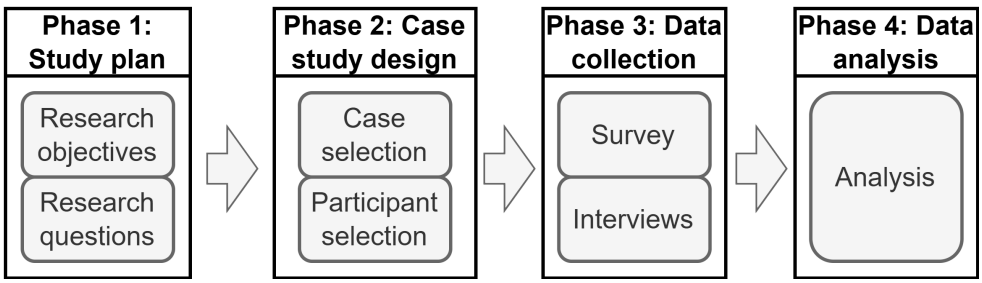

Figure 5.2: Research process overview

An overview of the research process is shown in Figure 5.2. The process includes four phases: Study plan, Case study design, Data collection, and Data analysis. We present details of these phrases from Section 5.3.1 to Section 5.3.4.

5.3.1 Phase 1: Study plan

In this phase, we formulated research objectives and questions.

5.3.1.1 Research objectives

We aim to investigate insights about coding productivity by exploring the practices of using GenAI coding assistant tools in industrial settings.

5.3.1.2 Research questions

We formulated two research questions:

- **RQ1:** What factors impact practitioners' productivity when using GenAI coding assistants?
- **RQ2:** What lessons emerge from practitioners while adopting GenAI coding assistants?

5.3.2 Phase 2: Case study design

This section presents the case and participant selections.

5.3.2.1 Case selection

Ericsson is our case company in this study. It leads the way in using GenAI coding assistants in the telecommunications domain for software development teams.

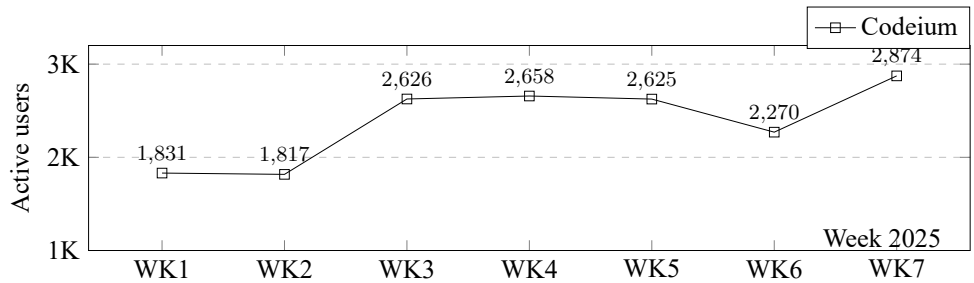

Figure 5.3: Active users per week for Codeium

Coding assistants in the case company include Codeium, Amazon Q, Tabnine, and GitHub Copilot. Recent statistics gathered by the first author, who works as an embedded developer within the company, indicate that Codeium has emerged as the most widely adopted tool. As illustrated in Figure 5.3, the number of Codeium users grew by 57% at the start of 2025. Amazon Q ranks second, with approximately 200 active users, while Tabnine and GitHub Copilot currently have minimal or no usage. A likely explanation for the low adoption of the latter tools is security concerns, such as a lack of support for on-premise deployment or risks that may unintentionally expose proprietary code to third parties.

Table 5.1: Context information [112] of the selected projects

Context		**Project A**	**Project B**
Project	Business domain	FinTech	Telecom
	Product type	Finance services	Network provisioning
	Maturity	Mature product	Mature product
Organization	number of engineers	>500	>200
Development	GenAI usage	Yes	Yes

We selected two projects, denoted as Project A and Project B, using purposive sampling [111] from our existing industrial network. At the time of the study, these

two projects used GenAI coding assistants. Table 5.1 presents the contexts of the selected projects. We started our research on the selected project in this study. We plan to replicate the research on the other projects shortly.

Project A offers financial services (e.g., transactions, loans, and payments) that have successfully served customers for approximately two decades. Project B is in the telecommunications domain, providing support services for network management. Both projects are rapidly growing, and the need to use GenAI to scale their business grows. Our study focused on the product development department in Sweden, which employs approximately 200 engineers.

5.3.2.2 Participant selection

Participants were selected using convenience sampling [32], determined by the individuals' availability and willingness to participate in the study.

Table 5.2: Participants from the selected Project A and B

Job role	NO. of participants	Work experience (years)	Project
Software developer	9	3 - 20	A (5), B (4)
Software architect	5	7 - 26	A (3), B (2)
DevOps engineer	4	5 - 14	A (2), B (2)

As shown in Table 5.2, 18 participants were chosen. All participants have experience in using either Codeium or Amazon Q. Participant roles included software developers, software architects, and DevOps engineers. Their working experience ranges from three to 26 years.

5.3.3 Data collection

We developed a survey with a questionnaire to gather feedback on using GenAI coding assistant tools. The survey included: I) demographics, roles, and years of experience; II) tool usage (e.g., frequency and duration of use); III) Likert-scale items, e.g., perceptions of satisfaction and tool integration.

Following the survey, we conducted 17 interviews with participants to gain further practical insights. The interviews were conducted in a mixed format, with onsite and online meetings, depending on participants' availability. Each interview lasted between 33 and 45 minutes. All authors participated in designing the survey and interviews through four rounds of meetings and refinements.

5.3.3.1 Data Availability

The survey results and extracted data are available at Figshare.

5.3.3.2 *Measuring productivity*

In line with the definition presented in Section 5.2.1, we asked participants to rate their satisfaction with AI-generated suggestions on a five-point Likert scale (1=very low, 5=very high). We gathered estimates of time saved by comparing coding effort with and without GenAI for everyday tasks (e.g., writing tests and code refactoring). We examined interview transcripts for examples of newly acquired techniques or clarifications of existing knowledge, representing learning gains. Finally, to assess the quality of GenAI tools' outputs, we asked how often participants needed to revise AI-generated code, using a five-point scale and follow-up interview probes for detailed explanations.

5.3.4 Data analysis

We first computed frequencies and means for the Likert-scale data (e.g., satisfaction, integration effort levels), enabling us to gauge overall trends. Next, we carried out a step-by-step thematic analysis [48] on the collected qualitative data.

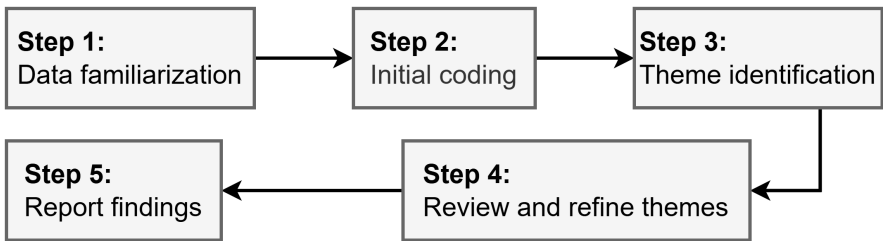

Figure 5.4: Data analysis steps

Figure 5.4 illustrates the data analysis steps of our study. The first author performed the data analysis, and all authors reviewed and discussed the results.

Step 1: Data familiarization. We read both survey open-ended responses and interview transcripts sentence by sentence to note initial impressions.

Step 2: Initial coding. We inductively labeled codes in the sentences or phrases created in Step 1. Codes refer to words or terms that could represent potential main subjects, for example, "user satisfaction", "productivity", and "domain knowledge."

Step 3: Theme identification. We grouped similar codes into higher-level themes. For example, "types of tasks," "human oversight," "IDE integration," and "Codebase context".

Step 4: Review and refine themes. Firstly, we reviewed the relationship between themes, i.e., whether higher-level themes were grouping lower-level themes appropriately. Secondly, we updated duplicate themes and made the required changes in the names of the themes. Thirdly, we merged and refined themes where necessary, ensuring their consistency.

Step 5: Report findings. We utilized our refined themes to identify recurring

usage patterns of GenAI coding assistants, which informed our main findings.

5.4 Results

This section presents our findings to answer the research questions.

5.4.1 Results of RQ1: What factors impact practitioners' productivity when using AI coding assistants?

According to study participants, the duration of using GenAI coding assistants ranged from one month to over six months. Based on the collected data, we have synthesized five factors that affect practitioners' productivity, as shown in Table 5.3. Our findings for these factors are presented from Section 5.4.1.1 to Section 5.4.1.5.

Table 5.3: Factors that impact practitioners' productivity

Factor name	Description
Task types	Task types refer to code generation, refactoring, reviews, explanation and test generation.
Coding skills	Skills of writing and debugging code.
Domain knowledge	Expertise in a specific field or programming experience.
IDE integration	Connecting a tool with an integrated development environment (IDE).
Tool familiarity	Learning/training on how to use a specific software.

5.4.1.1 Task types

Different coding tasks result in different productivity perceptions. These tasks include code generation, refactoring, review, and test creation. 60% of participants cited reduced effort for code refactoring (e.g., functions and variables updates) and review (e.g., code explanation) tasks. For example, several developers noted: *"It generates repetitive blocks of code and similar variable definitions easily and quickly. It avoids the pain of typing again."*

However, the capacity of GenAI coding tools for complex coding tasks (e.g., code/test generation for Gradle modules in JAVA projects) was limited. Participants stated: *"If you ask the tool to make changes to the whole file, it will often start to rewrite code unrelated to the task you asked it to solve."*

Regarding the quality of AI-generated code, we evaluated the mean primary adjustment levels across the identified task types. As shown in Figure 5.5, "Code/Test generation" has the highest level at about 3.67, suggesting that creating or generating code involves the most significant revisions. As participants criticized: *"Code generation of these tools is not good enough compared to OpenAI models,"* which indicates concerns that not all AI models are equally proficient. Both "Write tests" and "Fix bugs" show similar levels at around 3.4. "Refactor/Review code" has a moderate level of 2.5, indicating fewer major changes. For example, several participants

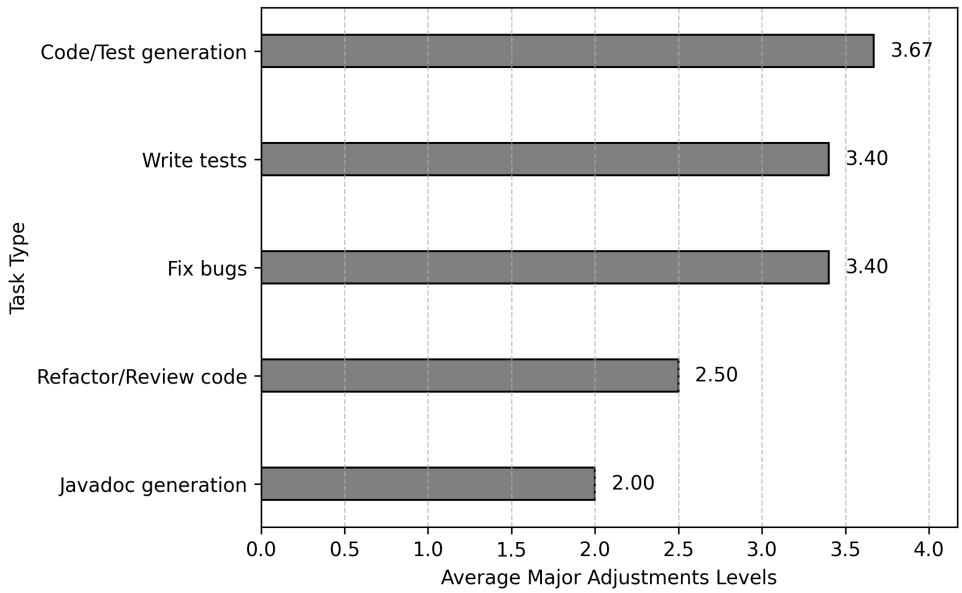

Figure 5.5: Average major adjustments required for different types of tasks based on AI-generated outputs

reviewed the quick AI-generated code suggestions: *"You always need to examine what it suggests."* "Javadoc generation" stands lowest at 2, indicating minimal required adjustments.

5.4.1.2 Coding skills

Participants commented that with more coding experience gained, less productivity was perceived while using AI coding assistants. One possible reason could be that with more coding experience, users know exactly how to complete a task without retrieving or acquiring external information. Since processing information from AI coding assistant tools consumes time. Like a participant complained: *"Lack of insight into what code the plugin has in its context, what I should expect, and when I should not waste my time trying to get the chat to understand."*

However, users with rich coding experience may neglect recent/advanced knowledge they may not be familiar with. As participants commented: *"The solution suggested by AI tools might not be one that you would have considered before, and then you can learn new ways of doing things."*

5.4.1.3 Domain knowledge

Domain-experts with a better understanding of contextualized codebase information are more capable of judging whether AI-generated or human-developed solutions.

As shown in Figure 5.6, the regression trend indicates that the higher the user's ability to validate AI-generated code, the lower the perceived satisfaction. On the

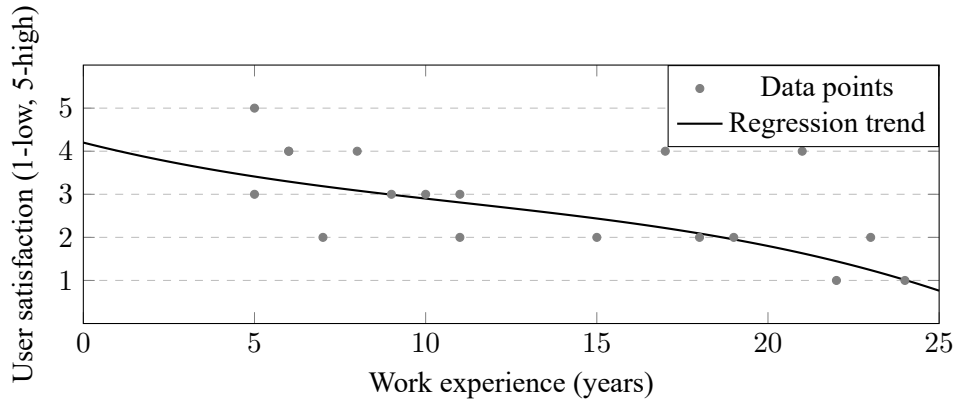

Figure 5.6: The relation between user satisfaction and working experience with regression trend

contrary, for users with limited knowledge of how to validate AI-generated code, AI tools provide higher potential value but also introduce risks without proper validation.

Additionally, common issues mentioned included over-aggressive suggestions and mismatches between the suggested code and the user's actual expectations. Participants highlighted their frustration: *"Wrong auto-completion code suggestions"* and *"The tool does not have a full understanding of my code repository."*

5.4.1.4 IDE integration

As shared by participants, a simple user interface (UI) or quick enabling/disabling of code suggestions was seen as a productivity booster.

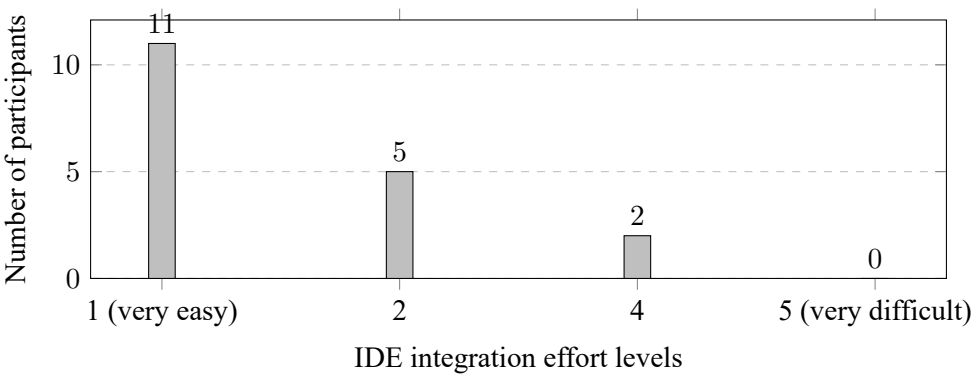

Figure 5.7: The effort levels while integrating GenAI coding tools into integrated development environments (IDEs)

Figure 5.7 presents the distribution of effort levels reported by participants while integrating GenAI coding assistant tools into their IDEs. The *x*-axis represents different levels of integration effort, ranging from Level 1 (very easy) to Level 5 (very

difficult), while the *y*-axis shows the number of participants at each level.

Study results show that most participants (11 out of 18) found the integration process very easy (Level 1), suggesting that GenAI coding assistant tools are straightforward to integrate into existing development environments. Additionally, five participants reported an effort as Level 2, while two participants rated the effort as Level 4. Notably, no participants reported the highest difficulty (Level 5 – very difficult), further reinforcing that the integration process is not highly complex.

These findings suggest that GenAI coding assistants require minimal setup effort for most users, potentially reducing adoption barriers and enabling smoother workflows within IDEs.

5.4.1.5 Tool familiarity

Study results indicate that as users utilized the AI coding assistant tools more frequently, they gained better results and learned to *bounce ideas* effectively.

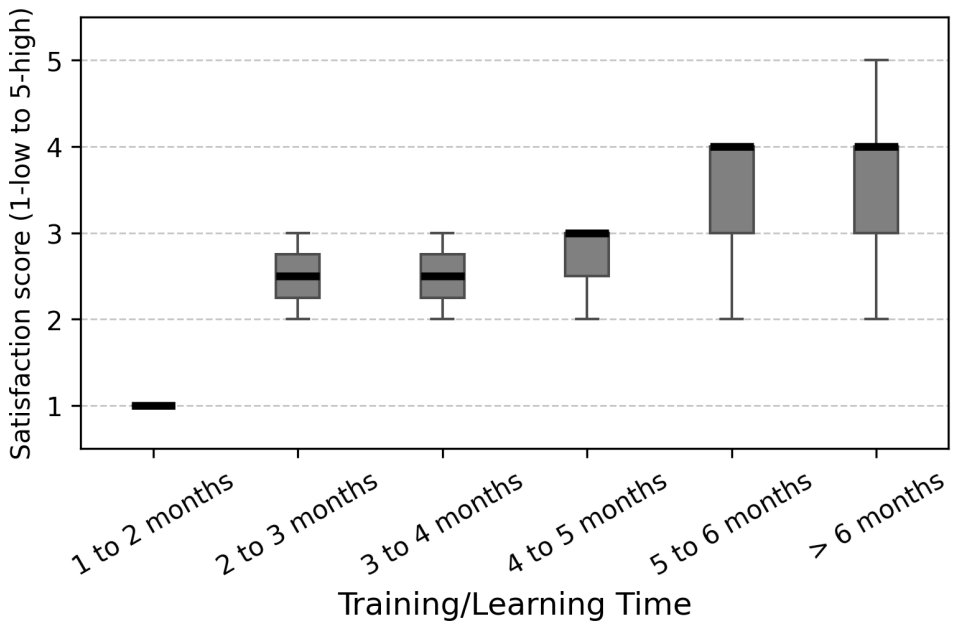

Figure 5.8: Median satisfaction scores based on the usage/training time of coding assistants

Figure 5.8 shows median satisfaction scores for using or learning AI coding assistants across different periods. The figure reveals that users with limited experience (1 to 2 months) report the lowest satisfaction, whereas satisfaction generally increases with extended usage. Users with five to six months of experience exhibit a higher median satisfaction score of around 4.

For those exceeding six months, satisfaction remains high but exhibits variability, suggesting that prolonged use leads to mixed experiences. This finding indicates

that familiarity with the tool improves satisfaction within the first six months. However, new challenges or limitations emerge as users become more accustomed to specific coding assistant tools, potentially reducing satisfaction.

5.4.1.6 Summary

In short, our findings show that domain knowledge and the GenAI tool's training interact with how productivity is realized. Partial productivity gains, such as time savings of about 13%, have been confirmed by study participants. However, these gains are situational, primarily seen in boilerplate generation rather than complex implementation tasks, such as creating a new Gradle module in a codebase.

5.4.2 Results of RQ2: What lessons emerge from practitioners while adopting GenAI coding assistants?

Lesson learned 1

The type of coding tasks matters when measuring users' productivity.

Study participants reported that GenAI coding assistants excel at generating boilerplate code. For instance, a DevOps developer commented: *"It auto-suggests repeating blocks of code right away, which saves me a lot of tedious typing."*

However, more complex tasks, particularly those requiring specialized domain knowledge or deeper architectural considerations, rarely showed comparable benefits. One architect with fifteen years of experience in Project B explained: *"If it is a completely new functionality that needs deep understanding of our Gradle plugins, it takes longer to guide the AI than to just do it myself."* Two possible reasons were mentioned for this shortfall: I) The GenAI's suggestions often lacked context about custom frameworks (e.g., Junit5-based function test framework) or legacy code structures (e.g., monolith components); II) repeated re-prompting was needed to correct inaccuracies, undermining any time savings.

In addition, a developer found that, when exploring an unfamiliar payments API, *"GenAI assistants (e.g., Codeium) actually had newer OpenAPI examples, so I only had to make a few small tweaks."* This successful case underscores how GenAI's broader training data can sometimes yield direct gains for complex tasks, provided the API in question is well-represented in its knowledge base.

Lesson learned 2

AI coding assistants allow users to focus more on development tasks without leaving their IDEs.

Participants praised how Codeium and Amazon Q integrate seamlessly into their

development environments, minimizing the need to switch between web searches. Several developers repeated: *"I no longer have to open a browser and check StackOverflow. The assistant suggests relevant snippets in my IDE."* Participants stated that this single environment helps them maintain concentration, particularly when working with multiple files in large repositories. Although some participants mentioned limitations with the GenAI's contextual accuracy, most agreed that staying in the IDE reduces distraction and promotes an immersive development environment.

Lesson learned 3

Context-related suggestions are challenging while using AI coding assistants.

While AI coding assistants offer support for routine tasks, they often struggle with providing context-aware suggestions, which is a confirmation of Nam et al.'s [129] findings. As participants noted: *"The tool sometimes failed to grasp the nuances of my project's specific framework, leading to suggestions that required significant manual adjustment."* Similarly, a participant complained that *"I often found that the generated code was too generic, missing the unique context of our specialized codebase."*

Even when partial code context is provided through IDE extensions, the models sometimes overlook important files, dependencies, or architectural conventions beyond their immediate context window. This leads to generic recommendations that developers must extensively rework, which reduces the initial efficiency gains of using GenAI coding assistants.

Lesson learned 4

Continuous and iterative refinement of prompts may result in more accurate suggestions.

Participants discovered that prompt phrasing and incremental clarifications influenced the accuracy of AI-generated code. A software architect remarked: *"The first suggestion was often incomplete, but after I clarified the requirement step by step, the tool finally got it right."* Similarly, a developer shared: *"I used to give a long prompt once, and the output was so-so. Now, I break the task into smaller pieces and guide the AI iteratively. That works much better."*

By narrowing the scope or specifying constraints (e.g., target frameworks, data types), practitioners could make the GenAI tools produce code closer to their expectations. However, frequent re-prompting can become time-consuming if the user must repeatedly correct the GenAI assistant's misunderstandings, particularly in complex domains. This shows the importance of balancing iterative guidance against potential overhead.

Lesson learned 5

Customized design rules are not supported by GenAI coding assistants.

Study participants complained that neither Codeium nor Amazon Q offers robust support for customized design rules that can be configured in IDEs. As a result, the AI-generated code reflected general coding patterns rather than the project's unique standards.

Design guidelines are often organization-specific and involve layered constraints (e.g., architecture, security, performance). Current GenAI models are not tuned to interpret and apply such interdependent rules out of the box. Consequently, practitioners had to perform manual adjustments to align their custom rule sets, thus reducing the overall benefit of AI-generated code.

Lesson learned 6

Quality evaluation of AI-generated code is required.

While study participants saw the convenience of AI-generated suggestions, they expressed concerns about verifying code performance, security, and maintainability at scale. As an architect commented: *"We can do code reviews, but as soon as the AI starts generating large chunks of logic, manual oversight isn't enough. We need automated checks."* Given this, scalable and automated quality evaluation frameworks are needed for AI-generated code. Future work in this area is expected.

5.4.3 Limitations

Potential limitations include the small sample size, which may not generalize across diverse software domains. Further, some participants may have used multiple GenAI tools (e.g., ChatGPT) outside our scope. To mitigate bias, we anonymized responses and encouraged honest feedback.

5.5 Conclusions

This study investigated how practitioners in two industrial projects adopted GenAI coding assistants (Codeium and Amazon Q) and examined the resulting impact on their productivity. Study results revealed that while GenAI tools accelerate everyday tasks such as code refactoring and Javadoc generation, they offer less consistent benefits for complex or domain-intensive work.

We learned that repeated re-prompting, limited domain-specific knowledge, and the lack of integrated design rules create barriers to time savings. Conversely, we found that iterative refinement of prompts can improve output quality, but this pro-

cess requires additional effort and oversight to be effective. Our findings highlight that the productivity impact of GenAI is not uniform. Domain knowledge, skills, and tooling integration influence the values of GenAI coding assistants in practice.

Future work includes replicating our multi-case approach in additional domains and expanding the sample size to complement and enrich our findings. Further research on contextualized codebases and customized design rules would increase the usage of GenAI.

Study E

A Framework for Evaluating GenAI Adoption and Use in Software Engineering

Abstract

Generative Artificial Intelligence (GenAI) is increasingly adopted in software development for faster ideations, code assistance, and automation, from early exploration to operational deployment. However, organizations are uncertain about how to evaluate product quality and quality-in-use when using GenAI across these stages. Standards, such as ISO/IEC 25059, distinguish between software product quality (e.g., usability) and quality-in-use (e.g., satisfaction). However, there is a lack of empirical studies on applicability, usefulness, and evaluation methods in industrial practice. These insights are important for the trend of growing GenAI adoption.

In this study, we investigate GenAI adoption and use, in particular, how quality is evaluated for adoption decisions, monitoring, and assessment of product quality. We conducted 19 semi-structured interviews, supported by archival data analysis for triangulation (15 documents and 184 web pages), in two companies belonging to a large organization, to understand adoption steps, quality concerns, evaluation practices, and role responsibilities.

Our findings describe a three-phase adoption process – Ideation, Development, and Operation – highlighting where and how quality evaluations occur, the criteria used, and the distribution of responsibilities. We synthesize these insights into an evidence-based quality evaluation framework.

Through a GenAI for Software Engineering (AI4SE) use case, we conclude that our framework demonstrates its applicability in industrial practice for guiding structured quality evaluation. Furthermore, the results highlight a need for a coordinating role – the GenAI Quality Lead – to improve accountability and traceability in AI-based systems. This contributes to Software Engineering for AI (SE4AI) by clarifying responsibilities when developing GenAI-based systems.

6.1 Introduction

Generative Artificial Intelligence (GenAI) [1] with Large Language Models (LLMs) [130] has been rapidly adopted in software engineering. This adoption covers two complementary perspectives: Software Engineering for AI (SE4AI), which provides practices for building reliable AI-enabled systems, and AI for Software Engineering (AI4SE), where GenAI assists traditional development tasks [131]. From requirements analysis to code generation [7], GenAI is transforming how software is developed and maintained [132]. Industrial GenAI adoption typically starts with prompt-based experimentation and iterative refinement, gradually maturing into production-ready implementations. This adoption introduces risks for software quality [12].

The main challenge is to define what to evaluate, when, and how — with adoption checkpoints and monitoring — for product quality and quality-in-use. Unlike traditional deterministic software [18], GenAI-based software produces probabilistic outputs that can vary across runs, even with identical inputs [12]. The generated content can be plausible but factually incorrect, biased, or potentially harmful, raising concerns around accuracy, security, and compliance [12, 133]. These characteristics – non-determinism, prompt and data sensitivity, and behavioral drift – limit the applicability of traditional software quality assurance techniques, which assume fixed logic and predictable outputs.

ISO/IEC 25059 extends the established ISO/IEC 25010 quality model to AI-enabled software [134], distinguishing between product quality (e.g., functional suitability, reliability, security) and quality-in-use (e.g., effectiveness, efficiency, freedom from risk) [135]. However, to the best of our knowledge, there is limited research on how these qualities are interpreted and evaluated when GenAI is used in industrial software development. In particular, it is unclear which quality aspects practitioners prioritise, when and how evaluations occur during the adoption, and who holds responsibility for them [136].

To address these questions, we investigated how engineers adopt GenAI, how quality evaluation is performed, and how responsibilities are distributed in two software development companies. We conducted a case study within two software development companies, following the guidelines set by Runeson et al [32]. A mixed-methods approach was used, combining interviews with analysis of internal web pages and documents, which enabled us to capture both documented processes and practices. Using the study findings, we designed an evaluation framework aligned with ISO/IEC 25059, and then validated the framework on a real-world GenAI use case.

Our contributions are:

- Empirical evidence on how GenAI is adopted and used in industrial software development.
- Insights on quality evaluation covering quality characteristics in ISO/IEC 25059,

including evaluation aspects, metrics, and criteria used.

- A cross-functional role, the GenAI Quality Lead, was proposed to coordinate quality-related responsibilities across the GenAI adoption.
- An evidence-based evaluation framework that aligns industry workflows with ISO/IEC 25059.
  - Validation on a real-world GenAI use case that shows applicability and decision support across adoption checkpoints.

Through investigating quality evaluation in industrial practices, our study helps organizations pursue responsible GenAI adoption and apply ISO standards in daily engineering work.

The remainder of this study is structured as follows. Section 6.2 presents background information and related work on GenAI in software engineering and quality evaluation approaches. Section 6.3 describes our research methodology, including research questions, case company details, and data collection and analysis methods. Section 6.4 presents the proposed quality evaluation framework, and Section 6.5 illustrates the framework verification through a use case implementation. Section 6.6 addresses threats to validity. Section 6.7 discusses implications, including the potential emergence of new roles such as AI Quality Lead. Section 6.8 concludes the paper with a summary of contributions and future research directions.

6.2 Background and Related work

6.2.1 Background

GenAI models [130], such as GPT-4o, have been used in software engineering activities, including code generation [137], documentation [138], and requirements drafting [139]. In industry, the adoption of GenAI is not always tied to a complete "system" from the beginning [16]. It often begins with integrating a model into a development task or piloting an idea with prompts. This adoption view helps teams understand how quality considerations arise along GenAI adoption [136].

ISO/IEC 25059 [23] extends the ISO/IEC 25010 [21] quality model to AI-enabled software, that is, software integrating AI models or services whose outputs influence system runtime behaviors. It structures quality into two complementary categories:

- Product quality: characteristics include functional suitability, performance efficiency, reliability, security, maintainability, and portability [23].
- Quality-in-use: characteristics cover effectiveness, efficiency, satisfaction, freedom from risk, and context coverage [23].

Both categories are important to GenAI adoption and use. For example, 'functional suitability' concerns whether outputs satisfy specified functional requirements for the intended tasks, whereas 'freedom from risk' includes legal, ethical, and security considerations.

6.2.2 Related work

6.2.2.1 *Quality evaluation methods for GenAI*

Several methods for evaluating the quality of AI-based software exist. Unlike this work, Zhang et al. [131] introduced a structured quality model addressing aspects such as data quality, model robustness, and integration maturity. Riccio et al. [134] developed testing strategies targeting correctness and validation challenges in AI-driven systems. Similarly, Breck et al. [140] proposed the ML Test Score, a checklist-based framework for assessing operational readiness and reducing hidden technical debt.

More recently, researchers have begun to explore evaluation techniques tailored to generative tasks such as code generation. Ribeiro et al. [141] proposed CheckList, a behavioral testing framework for NLP systems that identifies failures in specific linguistic capabilities, such as negation handling and coreference resolution. Wang et al. [137] reviewed evaluation metrics for code generation, including functional correctness, code readability, and runtime performance. These approaches advance technical evaluation and can be applied in later development and product evaluation. Some of them offer guidance on system-level prototypes. However, their practical applicability, scalability, and suitability for industrial adoption remain uncertain due to limited validation in real-world contexts.

6.2.2.2 *Empirical studies of GenAI in software engineering*

Empirical research has examined how GenAI is applied in real-world software development contexts. Barke et al. [142] and Donvir et al. [136] investigated developer interactions with GenAI-assisted tools, reporting both productivity gains and new categories of errors introduced by model-generated code. Other studies have demonstrated GenAI's applicability in tasks such as test case generation [12], requirements analysis [143], and documentation [138].

These studies also raise concerns regarding quality and trust. Donvir et al. [136] observed that GenAI can produce subtle, hard-to-detect defects despite speeding up ideation and coding. Such defects are hard to detect because the generated code compiles and can pass limited tests while masking logic errors and missing checks. Non-determinism and prompt sensitivity for open-ended tasks make failures difficult to reproduce and verify. Aleti et al. [12] highlighted the difficulty of verifying correctness and security in GenAI-generated outputs. Park et al. [135] emphasised the importance of prompt engineering and developer awareness for achieving high-

quality outputs.

6.2.2.3 *Research gap*

Existing literature focused on GenAI adoption does not provide a usage-oriented view of how quality is evaluated in industry. Our study addresses this gap by documenting the process of GenAI adoption in software development and mapping observed evaluation practices to ISO/IEC 25059.

6.3 Research methodology

We conducted a case study following Runeson et al.'s guidelines [32]. Archival data and semi-structured interviews were used to collect in-depth insights into how quality is evaluated throughout GenAI adoption in software development. An overview of the research process is shown in Figure 6.1.

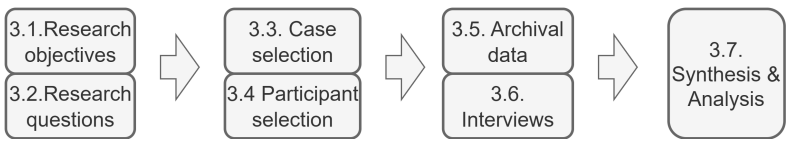

Figure 6.1: Research process overview.

6.3.1 Research objectives

Our study objective is to investigate how industrial engineers adopt GenAI in software development. We focus on capturing the exact processes followed by practitioners. In parallel, we examine how software quality is evaluated throughout the use of GenAI to identify recurring practices or patterns. For example, the practices include which quality aspects are considered important, when and how evaluations are conducted, and who is responsible for carrying them out.

6.3.2 Research questions

To reach the research objectives, we formulate the following research questions:

- **RQ1:** What processes do industrial engineers follow when adopting GenAI in software development?

  This question breaks down adoption into phases and practical steps, covering model selection, system integration, and operational use. It provides the baseline for understanding the surrounding activities where quality concerns may emerge.

- **RQ2:** What quality aspects are evaluated across GenAI adoption and use in software development?

  This question focuses on the types of quality concerns that practitioners prioritize throughout the adoption and use, such as legal risk, security, or data handling.

- **RQ3:** Who is responsible for evaluating quality throughout GenAI adoption?

  Given the identified evaluation aspects, we examine which roles participate in quality-related work. It connects identified quality concerns to organizational ownership.

- **RQ4:** How is quality evaluation conducted throughout GenAI adoption?

  We investigate existing approaches to carry out the evaluations, which reveal how quality is judged in practice.

6.3.3 Case selection

We selected two software development companies, denoted as Company A and Company B, using purposive sampling [31] from our industrial network. Both belong to the same corporate group but operate in different domains (FinTech and Telecom). We anonymized the companies under non-disclosure agreements. When this study began, few external partners had established processes for using GenAI in software development. The selected companies offered ongoing AI initiatives and access to internal artifacts, enabling a focused in-depth case study. We discuss implications for generalizability in Section 6.6.

Table 6.1: Context information [112] of the selected companies.

Context		Company A	Company B
Product	Business domain	FinTech	Telecom
	Product type	Finance services	Hardware devices, software products, and support services
	Maturity	Mature product	Mature product
Organization	Number of employees	>500	>2000
	Corporate group	Same corporate group	Same corporate group
GenAI applications	number of AI initiatives	9	263

Table 6.1 presents the contexts of the selected companies. Company A produces a financial system (e.g., transactions, loans, and payments) that has successfully served customers for approximately two decades. Company B is in the telecommunications domain, providing support services for network management. Both companies operate globally and expand rapidly. Our data collection focused on the Swedish and Indian development sites. Both have mature software engineering practices and have set organizational goals to adopt GenAI in products. At the time of the study, 272 AI initiatives in total were recorded. Most of the recorded initiatives were ongoing, and some had been completed.

With many parallel AI initiatives, ad hoc practices lead to non-comparable results, duplicated effort, and limited visibility into risk and drift across teams. Shared adoption patterns and processes, such as common checkpoints (e.g., verification of data privacy), a minimal metric start-set (e.g., metrics on output validity/cost), clear role ownership (e.g., product owners), and traceable records (e.g., model version and prompt templates), enable cross-team monitoring and incident review. This supports consistent decisions and reduces rework in legal and security reviews.

6.3.4 Participant selection

Participants for this study were selected using purposive sampling [31] following a multi-step process. Firstly, we identified active GenAI initiatives within each company using internal documentation (see Table 6.1 for initiative counts). Secondly, the first author was embedded with development teams in both companies, which provided us with sufficient understanding of the organization. Thirdly, this embedded access gave visibility into AI initiatives (teams, internal documentation, and demo sessions), which we used to identify participants across roles and responsibilities. We prioritized practitioners with hands-on GenAI responsibilities and direct involvement in quality evaluation activities. During interviews, we collected detailed information about participants' specific roles, geographic locations, and deeper insights into their GenAI experiences. As a result, shown in Table 6.2, 19 participants were chosen.

Table 6.2: Participants selected from Company A and Company B

ID	Participant role	NO. of participants	Working experience (years)	Company
1	Software developer	3	7 - 20	A (1), B (2)
2	Software architect	2	11 - 23	A (1), B (1)
3	Quality assurance engineer	3	8 - 19	A (1), B (2)
4	Legal engineer	3	9 - 23	A (1), B (2)
5	Security engineer	3	6 - 17	A (2), B (1)
6	Operations engineer	3	7 - 15	A (1), B (2)
7	Product owner	2	13 - 22	A (1), B (1)

6.3.5 Archival data

In addition to interviews, we collected archival data [32] (e.g., internal documents) to triangulate our findings and gain deeper insights into GenAI adoption and quality evaluation practices.

We examined a total of 184 internal wiki/web pages, which are counted as unique URLs, not A4 pages. Pages ranged from short information (1–2 paragraphs) to multi-section guides, and the count indicates breadth rather than physical length. These pages span a wide range of topics related to GenAI adoption, including GenAI development portals, legal compliance and risk assessment pages, GenAI experimentation guidelines, system development processes, and tooling documentation (e.g., API consoles and quality dashboards). Key categories included "GenAI risk and com-

pliance", "AI model profiles", "development and assessment procedures", "GenAI initiative board" overviews, and "handbooks for GenAI practices". These resources provided important context on how quality evaluation is performed within organizations.

Furthermore, we analyzed 15 internal documents (e.g., PDF, Docx, Excel, slides), focused on how GenAI is adopted and evaluated in practice, such as "GenAI risk guardrails" (legal and technical risks), "AI adoption assessment" forms used at checkpoints, "Responsible GenAI" checklists, model registration and vendor assessment records, security and privacy reviews, metric definitions and dashboard specifications, and training slide decks.

These materials formed the primary evidence base, with concrete procedures, criteria, and role ownership. Firstly, they provided a high-level view of the GenAI adoption process, including handoffs and ownership that individual practitioners did not always see. Secondly, they enabled a cross-check through comparison with documented procedures and templates. Thirdly, they revealed organizational expectations for GenAI quality that were not yet consistently enacted in teams' daily work.

Although the archival data strengthened credibility, they also have limits. Access was restricted to internal platforms, so coverage reflects what was available during November 2024 to June 2025. As documents varied in detail and maintenance, we mitigated these risks by cross-checking with interviews.

6.3.6 Interviews

We conducted semi-structured interviews [32] to investigate how industrial practitioners adopt GenAI in software development and how they evaluate quality during this process. The interviews allowed for in-depth exploration of practices, decision rationales, and cross-role responsibilities not captured in internal pages, complementing the archival analysis.

A total of 19 interviews were held between November 2024 and June 2025. Twelve were in person and seven online, due to the geographical distribution of participants. The interviews were conducted by the first author and were held in English using an interview guide of 21 questions. The interview guide is available on Figshare. Each interview lasted between 46 and 60 minutes.

We took several actions to enhance interview validity. Firstly, we ensured role diversity in participant selection. Secondly, we refined the interview guide after pilot interviews based on emerging themes, then used the finalized guide consistently across all remaining interviews. Thirdly, we conducted member checking by sharing our analyzed data and proposed framework with participants for review through email and follow-up conversations to confirm our interpretations.

However, certain limitations exist. As participants came from two companies in a large corporate group with ongoing AI initiatives, the sample may not reflect

settings with minimal tooling, restricted data access, or without in-house legal and security support. Moreover, self-reporting can introduce recall bias. We used internal documents and web pages mentioned or shared by participants to confirm statements and elaborate on details from interview transcripts. However, we acknowledge that these sources may reflect similar organizational perspectives rather than providing fully independent triangulation.

6.3.7 Data synthesis and analysis

We employed thematic analysis [48] to identify patterns and themes in the collected data. This analysis followed a four-step process, supported by iterative collaboration and joint interpretation sessions.

1. *Initial coding*: A starting set of codes was generated from interview transcripts and archival documents. Codes were generated based on relevance to themes connecting to our research questions. For example, codes reflect GenAI adoptions (e.g., "prompting"), evaluation (e.g., "risk checks", "metrics"), and stakeholder roles (e.g., "product owner", "security").

2. *Refinement and disambiguation*: The initial codes were iteratively reviewed to improve granularity and avoid semantic overlap. For example, "correctness" was split into "output accuracy" and "code validity" to capture domain-specific distinctions.

3. *Theme construction*: We organized codes along two categories: process area (adoption phase and step) and quality dimension (ISO/IEC 25059 characteristics). We formed themes when repeated evidence appeared in both categories. Edge cases were resolved through team discussion to avoid overlap.

4. *Mapping to research questions*: Themes were organized according to their relevance to each research question. For example, codes concerning risk evaluations were connected to RQ2, while those related to developer responsibilities were connected to RQ3.

To support internal validity, coding was reviewed by the first three authors and reconciled through multiple discussion rounds. A shared coding protocol was used to align definitions, coding rationale, and theme development. Preliminary interpretations were verified through feedback sessions with selected participants, ensuring alignment with their experiences.

6.3.7.1 Deriving GenAI adoption and use processes

We present a four-step method to derive GenAI adoption processes based on the collected data. Firstly, *prompt piloting* in Ideation was derived from repeated interview

statements about "quick prompt trials before building" and the Ideation step in Figure 6.2, together with internal guidance on experimentation. Secondly, *legal and security risks* in Ideation were grounded in the risk checklist in Table 6.3 and interview accounts that approvals were required "before PoC," making it a checkpoint rather than an ad hoc review. Thirdly, *model selection* followed piloting and bridged into Development, as teams chose a provider on output suitability and integration ease, and then proceeded to *system integration* (Development step in Figure 6.2). Fourthly, *runtime monitoring* in Operation was supported by the verification case evidence of dashboards and metrics (e.g., validity and latency) shown in Figure 6.7, which then fed *continuous improvements*. We grouped labels such as "prompt tries," and "sandbox runs" into *prompt piloting*, and merged "legal review," "supplier terms," and "risk guardrails" into the *legal and security* category. For traceability, each step in the combined process was linked to its supporting interviews and documents, with an overview available in Figshare – a third-party research data repository (access here).

6.4 Results

We present study findings to answer our research questions in this section.

6.4.1 Results of RQ1 -- What processes do industrial engineers follow when adopting GenAI in software development?

Figure 6.2 shows the identified processes in phases for GenAI adoption and use in software development. The phases and subsequent steps were derived from our coding procedure in Section 6.3.7.1. The process is organized into three main phases: *Ideation*, *Development*, and *Operation*. These phases reflect a combination of exploratory use, iterative design, and deployment-oriented practices.

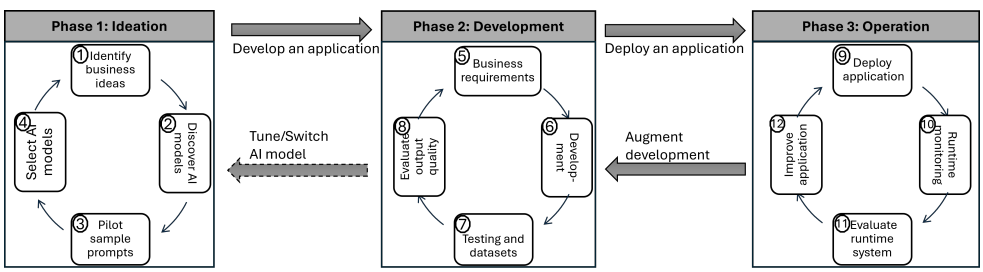

Figure 6.2: Phases and steps followed to adopt GenAI.

6.4.1.1 *Phase 1: Ideation*

This phase is initiated by identifying high-level business opportunities (Step 1). The ideas are usually sourced by product teams, often without strong technical feasibil-

ity input at this point. This reflects a trend where business-driven interest precedes technical alignment.

Once candidate ideas are selected, engineers explore potential GenAI solutions by surveying available models (Step 2). At this stage, engineers screen for early fit and risk rather than formal accuracy. They check task fit (outputs match the intended task and domain), data and policy fit (inputs allowed under privacy and security rules), legal acceptability of license, basic output quality on a few representative prompts, and integration feasibility (API access, hosting, deployment path).

Piloting sample prompts (Step 3) serves as a lightweight feasibility check. It allows teams to assess whether GenAI output aligns with expectations before further investment. In the studied companies, this step was often used to discard ideas quickly, reflecting a low-cost experimentation practice.

If the output appears promising, teams proceed to select a specific AI model (Step 4). Selection is commonly based on empirical prompt testing, vendor trust, and ease of integration, rather than formal quality guarantees. This also implies that model selection is frequently revisited as the project progresses.

6.4.1.2 Phase 2: Development

In the development phase, the focus shifts from ideation to system integration. Teams refine business requirements (Step 5) to accommodate GenAI capabilities and constraints, often reducing traditional specification granularity. In contrast to conventional software, requirements are shaped around acceptable variation in model output, making this step inherently iterative.

The development step (Step 6) involves incorporating the selected GenAI model into an application or service. This may include prompt engineering, workflow orchestration, or API-level integration. Participants noted that development often advances without complete clarity on evaluation criteria, reinforcing the need for later adjustments.

Testing and dataset preparation (Step 7) varies considerably in formality. Some teams curate specific test cases to validate expected behaviors, while others rely on production-like data to simulate real-world input. The level of rigor here depends on the perceived risk and target user.

Output evaluation (Step 8) is where the prototype is reviewed for its usefulness, correctness, and consistency. Unlike deterministic software, evaluation here involves human-in-the-loop judgment and often lacks predefined acceptance thresholds. This introduces challenges in comparing iterations or tracking progress over time.

6.4.1.3 Phase 3: Operation

After deployment (Step 9), systems enter the operation phase, where GenAI becomes part of production environments. This transition often marks a shift in responsibility from development teams to maintenance and operations teams.

Runtime monitoring (Step 10) is implemented to track anomalies, drift, or failure cases. However, monitoring practices remain immature; few teams employ systematic mechanisms for runtime evaluation beyond basic usage statistics.

Evaluation of the runtime system (Step 11) focuses on stability, user satisfaction, and reliability in production contexts. This step is typically reactive—issues are addressed after surfacing, rather than through predictive assessments.

Improvement cycles (Step 12) are driven by operational feedback and usage data. These iterations may involve switching models, re-engineering prompts, or redesigning workflows. However, the improvement process is rarely governed by formal change management, reflecting the still-exploratory nature of GenAI operations.

6.4.1.4 Observations

We observed that GenAI adoption in industry follows a typical pattern of iterative experimentation, model probing, and incremental discovery, similar to how organizations adopt other new technologies. This reflects a shift from deterministic system design toward empirical convergence, where systems evolve through usage feedback and feasibility trials.

Moreover, we learned that quality evaluation occurs in most phases, but practices are often informal and subjective (e.g., accepting outputs without predefined thresholds for accuracy, latency, or safety). Practices such as prompt piloting, runtime observation, and prototype judgment serve as implicit gates for decision-making. This raises concerns about traceability and accountability in GenAI-based systems.

For practitioners, the identified three phases can serve as a reference to position their own efforts. It also offers researchers an example of how GenAI adoption unfolded in two industrial settings.

6.4.2 Results of RQ2 -- What quality aspects are evaluated across GenAI adoption and use in software development?

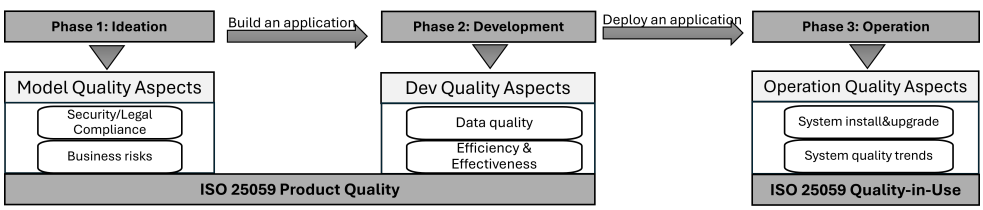

Figure 6.3: Primary quality aspects considered throughout GenAI adoption and use, mapped to ISO/IEC 25059.

Figure 6.3 summarizes the quality aspects identified by participants across the three phases of GenAI adoption and use. Quality aspects in Phases 1 and 2 connect to *Product Quality*, and Phase 3 is about operation, which belongs to *Quality-in-Use* characteristics according to ISO/IEC 25059. These aspects reflect where practitioners

currently focus their attention, shaped by their roles and organizational constraints.

6.4.2.1 Model Quality Aspects (Phase 1) --- ISO 25059 Product Quality

Security and legal compliance Participants highlighted the need to understand legal implications early, especially concerning intellectual property, data privacy, and third-party model licensing. This concern is not only about selecting 'safe' models but also about whether the use case is legally viable. The motivation stems from the uncertainty around AI model behavior and ownership of training data, which introduces legal risks. Practitioners noted that without early legal validation, entire business ideas may later be blocked or delayed.

Business risks Feasibility was often framed in business rather than technical terms, for example, unclear value propositions, unpredictable behaviour in user-facing contexts, or prohibitive costs. Participants noted that even technically functional GenAI models might conflict with user trust or supportability goals. As a result, early-phase quality evaluation often prioritizes business viability alongside compliance.

Actions At this phase, quality evaluation focuses on compliance and purpose. Organizations are advised to involve legal, compliance, and product strategy teams early to assess feasibility before prototyping.

6.4.2.2 Development Quality Aspects (Phase 2) --- ISO 25059 Product Quality

Data quality Data quality emerged as a central concern. Participants were uncertain whether test datasets and prompts accurately reflected real-world adoption, especially when outputs were highly sensitive to subtle input changes. In some teams, overfitting prompts to hand-crafted cases reduced generalization after deployment.

Efficiency and Effectiveness Effectiveness was interpreted contextually: some defined it as "acceptable output for task completion", others included time-to-answer or resource efficiency (e.g., latency, token cost). Formal benchmarks were rarely used, according to participants. Instead, teams used quick iterations with subjective judgment. While agile, this informality makes scaling and repeatability difficult.

Actions Informal assessments should be complemented with measurable criteria, for example, output consistency, processing time, and task success rates. Data quality should be revisited during development, not assumed from earlier checks.

6.4.2.3 Operation quality aspects (Phase 3) --- ISO 25059 Quality-in-Use

System installability and upgradeability Participants reported deployment friction for GenAI components due to version mismatches, hosting constraints, and unclear dependencies. Such issues are more disruptive for GenAI than traditional systems, given their dynamic dependencies and licensing restrictions.

System quality trends Monitoring quality trends in production was valued but underdeveloped. Most participants lacked formal systems in place to track model drift, unexpected failures, or changing behavior across environments. Instead, most relied on user complaints or ticket reports from support teams. This limits feedback loops and hinders root-cause analysis.

Actions Practitioners should treat GenAI-based components as evolving artifacts. Continuous monitoring, logging, and sampling should be used to detect unexpected changes, supported by feedback channels from users and operations teams.

6.4.2.4 Observations

While the identified aspects align with ISO/IEC 25059, several characteristics remain under-addressed:

- **Functional adaptability** — system responsiveness to dynamic environments or evolving data.
- **Robustness** — stable performance under bias, adversarial inputs, or novel conditions.
- **Transparency** — clear documentation of model logic, data provenance, and decision rationale.
- **Societal and ethical risk mitigation** — adherence to fairness, privacy, and accountability principles.

During the interviews, these quality characteristics were rarely mentioned unless directly prompted. For instance, *functional adaptability* – an important reason for choosing GenAI in the first place—was not systematically evaluated. *Robustness* and *transparency* were typically implicit or informal (e.g., "seems to work well") rather than captured through structured criteria or logging. Without explicit evaluation, systems may degrade in unobserved ways, particularly after deployment.

Engineering teams should integrate selected ISO/IEC 25059 attributes into lightweight quality evaluation checklists or design review templates. The following aspects are actionable:

- **Transparency:** Document prompt structures, data origins, and model versioning. Use structured logging for prompt-output behavior over time.
- **Robustness:** Evaluate and guard against structured-output failures and unresponsive model calls (e.g., JSON, XML, CSV). Use schema validators (e.g., JSON Schema, XSD), constrained decoding or typed function calling, and strict parsers. Add automatic repair, enforce timeouts, and use retries with backoff and circuit breakers. Define fallbacks for known failure cases (e.g., cached answers, rule-based generation, or human-in-the-loop).
- **Societal and ethical risk mitigation:** Align GenAI adoption with Responsible AI frameworks [144]. Include non-engineering stakeholders early, such as legal, security, compliance officers, domain experts, and representative users.

These steps do not require full ISO compliance but can raise quality awareness and connect daily GenAI practice with standardized guidance.

6.4.3 Results of RQ3 -- Who is responsible for evaluating quality throughout GenAI adoption?

GenAI integration into software development creates new quality challenges, with responsibility for addressing these challenges remaining fragmented across roles. Figure 6.4 summarizes the roles identified in each phase of GenAI adoption—ranging from business ideation to runtime operations—and the specific quality concerns associated with them.

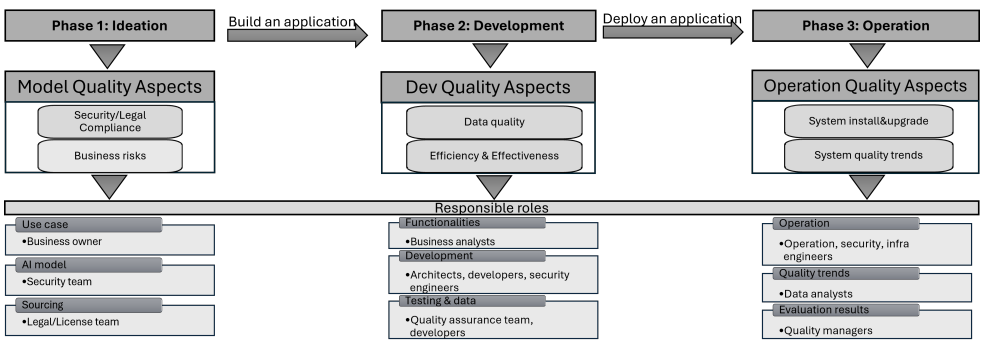

Figure 6.4: Responsible roles for quality evaluation throughout GenAI adoption.

6.4.3.1 Responsible roles

In the **Ideation** phase, responsibility is distributed among business owners, legal and licensing teams, and security units. Business owners assess use case viability and business risks, while security and legal experts validate compliance with regulations

and intellectual property policies when selecting or procuring models. As a participant stated: "We had to get the legal team involved early because we weren't sure if the use case was even allowed with the data we had."

In the **Development** phase, developers and architects lead integration, while quality engineers and analysts support testing and validation. Yet evaluation practices often remain informal or rely on domain intuition. Several participants reported: "It is mostly trial and error—engineers test prompts, tweak outputs, and we go with what feels right."

Some teams assign evaluation duties to developers by default, and in the absence of structured criteria, they rely on domain expertise and human judgement. Several participants noted: "There is no fixed checklist. We just look at whether the response makes sense or is dangerous."

In the **Operation** phase, roles such as infrastructure engineers and support analysts become involved. However, runtime feedback mechanisms are often weakly connected to earlier evaluation decisions. Participants stated that "We rely on support tickets to learn when something's off. There's no automated alerting tied to model output yet."

6.4.3.2 *The need for a dedicated role in ownership*

Across all phases, we found limited unified ownership for GenAI quality evaluation. Although responsibilities are scattered among domain experts, the lack of coordination and traceability reduces the accountability and repeatability of evaluation activities. For example, a participant stated: "Everyone is CERTAIN that SOMEONE is checking the quality, but no one knows who actually does it." This ambiguity is not unique to GenAI. Conventional projects may also suffer from diffuse quality ownership in siloed organizations. GenAI magnifies the problem because quality decisions cut across legal, security, compliance, and software development.

A recurring theme in interviews was the absence of a designated role that integrates technical, legal, and ethical concerns across the GenAI adoption and use. Therefore, we propose the role of **GenAI quality lead**. This role is not merely a test engineer or auditor but a cross-functional *driver*. The driver runs quality checks, records decisions, and can block or escalate unresolved risks. The aim is to ensure the evaluability, explainability, and appropriateness of GenAI features across the identified phases. As participants repeated that "It would help to have someone who knows what to check, not just technically, but in terms of risk or impact too."

Responsibilities of the quality lead include:

- Facilitate early-phase risk identification (e.g., legal, security, bias).
- Establish and adapt quality criteria with legal, security, and domain stakeholders.
- Defining evaluation criteria with product and engineering teams.

- Drive evaluation practices across development and operations (e.g., output testing, runtime monitoring).
- Translating quality standards (e.g., ISO/IEC 25059) into project-specific practices.
- Serve as a contact point for review boards, audits, and compliance alignment.

This role builds upon principles of SE4AI, which emphasizes quality assurance throughout the AI engineering lifecycle. By formalizing such a role, companies can better integrate emerging standards (e.g., ISO/IEC 25059) and internal Responsible AI guidelines into everyday practice. Importantly, the role is not a replacement for domain-specific experts but a coordinating role that improves quality accountability and traceability. It helps move beyond ad hoc evaluations by embedding structured thinking into the day-to-day engineering of GenAI-based solutions.

6.4.4 Results of RQ4 -- How is quality evaluation conducted throughout GenAI adoption?

Based on our findings from RQ1 to RQ3, we synthesized a process-oriented framework that integrates GenAI adoption phases with quality evaluation practices and role responsibilities. This framework shows HOW quality is evaluated during GenAI adoption in phases and indicates WHO is accountable.

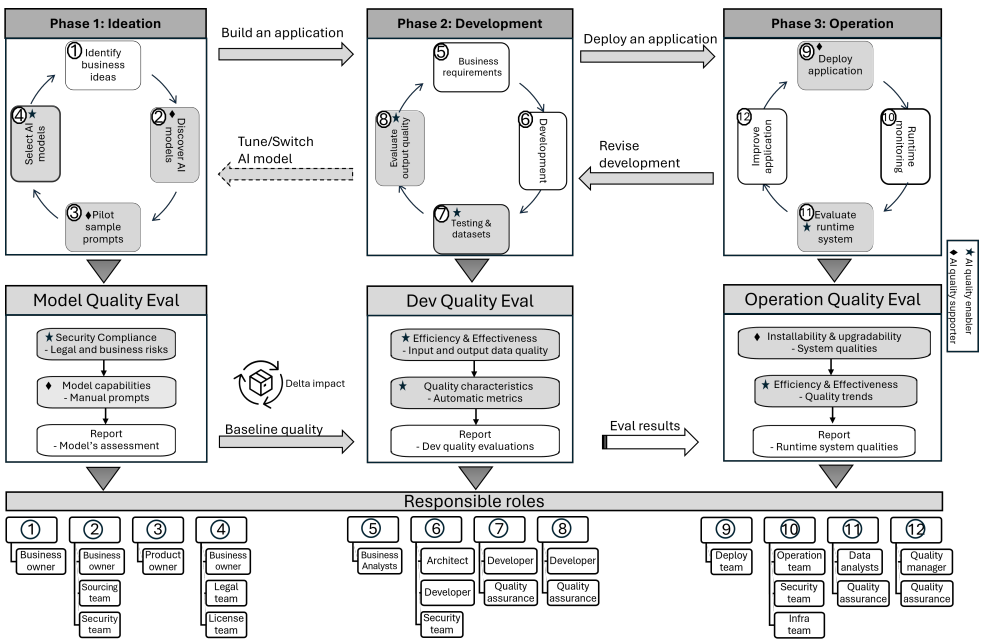

Figure 6.5: Overview of quality evaluation framework during GenAI adoption in software development.

Figure 6.5 consolidates our findings into a process-oriented view of GenAI adoption, with quality evaluations integrated into each phase and corresponding roles.

Each phase in the figure contains steps (white boxes), among which specific actions related to quality evaluation are highlighted. The ⋆ marks indicate points where evaluation is **directly enabled**, for example, assessing legal risks, verifying outputs, or analyzing system qualities. The ♦ marks identify **supporting checks**, for example, model prompting or installability that shape or constrain quality assurance decisions. Together, they form a network of quality signals – some proactive, others reactive.

Below each phase, the gray panels describe evaluation practices, which range from compliance vetting to metric-based evaluations. At the bottom, the roles responsible are connected, which reveals a distributed accountability structure, with legal, engineering, and quality teams all involved at different stages.

6.4.4.1 *Quality evaluation practices in GenAI adoption*

Based on collected data, three interconnected evaluations emerge: Gatekeeping – evaluation before development, Validation – testing during development, and Monitoring – evaluation during operation.

1. Gatekeeping (before development) Evaluations in the Ideation phase are often conducted to determine whether a GenAI model or idea can be pursued. These gatekeeping checks are typically compliance-driven, involving legal and security teams.

Table 6.3: Legal risk assessment checklist and owners

Legal risk type	Evaluation(s)	Responsible
Intellectual property infringement	Who is liable for infringement of copyrights in training data, e.g., the supplier or user?	Legal and Security teams
Training data	What data can be used to train/tune AI models?	Legal team
License	What Open Source Software has been used to develop the model(s), and which license applies?	Licensing team
Inputs of AI models	How do the AI suppliers treat users' confidential information, e.g., publicize user inputs or user inputs used to train AI models?	Legal and Security teams.
Outputs of AI models	Who is legally liable for the output of AI models? The supplier may disclaim all liabilities for use of their models	Legal team
Data privacy	Who is responsible for any illegal processing of personal data during the adoption of AI?	Legal and Security teams
	Can the AI model provider guarantee that the user data is not used to train the model, or in any other way processed by the provider of the system, or other third party for other purposes?	
	For any use cases of AI leading to recommendations, conclusions, or predictions related to individuals, can data protection for individuals be mitigated from the legal aspect?	

The checklist in Table 6.3 was referenced by multiple participants as a tool to assess risk exposure and vendor accountability before prototyping. Participants from legal teams mentioned: *"We block use cases if the supplier can't commit to data privacy or copyright terms."* These evaluations serve as safeguards, especially in large organizations where regulatory pressure is high.

2. Validation (during development) In the development phase, evaluation activities shift from permission to execution.

Here, teams examine if the model behaves appropriately in the target setting. This includes assessing data quality, response consistency, efficiency, and effective-

ness with internal policies. We found that evaluation practices often lack mature tooling and rely on custom metrics. For example, a senior developer shared that *"We often create our specific key performance indicators in our implementation, aiming to give a signal on hallucinations."*

Table 6.4 outlines a cross-functional practice that was used in multiple settings to structure quality assessments during implementation. These assessments are rarely isolated. They are tightly coupled to development iterations.

Table 6.4: Quality evaluations for GenAI application implementation

Development tasks	Evaluation(s)	Responsible
Information Security	How are objectives on confidentiality, integrity, and availability of information managed?	Security team
Data governance	How to handle data quality issues, unauthorized access, and improper use of data?	Security team
Contractual Frameworks	What are the terms of use for both internal AI adoption and for delivery towards the customer?	Legal team
Intellectual Property Rights	Avoid leakage of confidential information.	Licensing team
	No licensing out of patents.	
	Strict adherence to open source policies.	
Product Security	No use if no full control over AI tools/services.	Architect team
	A small and simple proof-of-concept is required.	
	AI is a support tool, not a decision maker.	
	Not be solely used for sensitive functionalities where incorrect outcomes cannot be tolerated.	
Data privacy	Complying with data protection principles/laws.	Legal team
	Processing user data only according to the customer contracts.	
	Maintaining records of processing use cases is a regulatory requirement.	
	No violations of individuals' rights and freedoms.	
Ethical and responsible outcomes	For in-house built models, training data should be explained and evaluated for bias.	Product owner Development team Quality team Management team
	For sourced models or tools, vendors must provide information about their bias analysis.	
	The ethical and responsible consequences of model error should be evaluated for the use case	
	Human-in-the-loop controls should be implemented.	
	Analyze all AI use cases as to their status against regulations. Ban those prohibited by law.	
	Classify AI use cases as limited- or high-risk under the AI Act.	
Quality of outcomes	PoC use cases to determine the seriousness of erroneous responses.	Product owner Development team Quality assurance team Management team
	Avoid use cases where the consequences are serious or no opportunity to check the response before action is taken.	
	Implement controls on the model training (data quality).	
	Inform the user about the possibility of error and the appropriate caution that should be taken.	
	Use application designs that allow source checking with source references.	
	Provide explainability tooling to help users understand how answers were determined.	
	Implement monitoring, response and support mechanisms for dealing with errors detected during production use.	

To better understand the relationship between the observed practices and software quality, we mapped the evaluation activities to ISO/IEC 25059 characteristics in Figure 6.6.

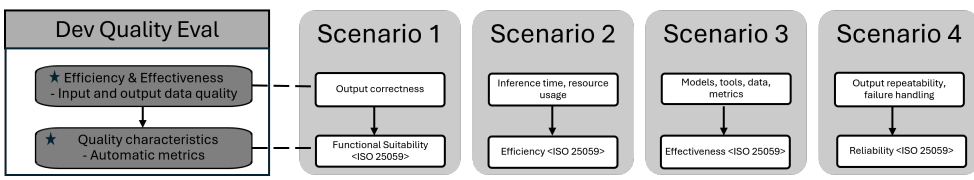

Figure 6.6: Relation between observed Dev Quality Evaluation practices and ISO quality characteristics.

As shown in Figure 6.6, observed practices (e.g., prompt testing, the use of tools, and metrics) map implicitly to ISO quality characteristics. Specifically, they reflect:

- **Functional Suitability**. It includes the correctness and appropriateness of output.
- **Efficiency**. It covers measures for inference time duration and resource usage.
- **Effectiveness**. It covers task success and utility in the target context, assessed through targeted tests and usage evaluation.

- **Reliability**. It refers to output repeatability and failure handling.

These mappings indicate *what* to evaluate; they do not prescribe *how*. In our cases, metrics were selected per context, and some remained unsettled. Future tooling and policy definitions may benefit from translating these ISO sub-characteristics into practical checkpoints and observable metrics.

3. Monitoring (during operation) Evaluation does not end at deployment. Once a system is deployed successfully, teams perform runtime assessments based on logs, live quality trends, and user feedback. This mode focuses on identifying performance degradation, systemic errors, or violations of usage policies. As shared by an operations engineer, *"It is required that we detect issues not when they happen, but when someone flags them downstream."* These practices are increasingly important in GenAI, where behavior can drift over time. Importantly, monitoring efforts feed back into development and legal reassessments, completing the lifecycle.

The integrated view in Figure 6.5 reveals a decentralized architecture of quality assurance in GenAI adoption. Evaluation is not owned by a single role or function. It is distributed, contextual, and embedded in day-to-day decisions. Tables 6.3 and 6.4 reflect how companies attempt to formalize what would otherwise remain tacit.

However, several challenges remain: metrics are inconsistent, practices vary across teams, and accountability is shared but unclear. Some participants described this as a lack of a coherent QA strategy tailored to GenAI, *"We have a security master in each dev team. But who is the quality champion for GenAI?"*

This observation motivates the discussion of a new role in Section 6.7.2, and underscores the practical value of treating quality evaluation not as an afterthought, but as an ongoing design activity.

6.4.4.2 Framework reuse

The quality evaluation framework presented in Figure 6.5 offers a reusable structure for organizations seeking to evaluate GenAI-enabled software.

By clarifying when and how evaluations are conducted, and by whom, the overview can serve as a reference for organizations lacking established quality practices for GenAI. It also helps bridge informal routines and ISO-inspired evaluation dimensions, as illustrated in Figure 6.6. Therefore, the framework is a synthesized process reflecting actual engineering practice.

To further examine its practical utility, we conducted a case implementation in one of the studied companies. The aim was to apply the evaluation structure to a real GenAI adoption and assess its feasibility. The next section presents this framework verification case.

6.5 Verification of the Quality Evaluation Framework

To verify the applicability and practicality of our quality evaluation framework, we implemented it in an internal software development project at one of the selected companies.

6.5.1 Case description

The case was a GenAI-assisted application for generating XML code used in Unstructured Supplementary Service Data (USSD) menus. USSD is a widely used communication protocol in mobile services that allows interaction through short codes without requiring internet access. The target users of this tool were merchants who needed to create dynamic menus for product browsing and purchasing without writing XML code manually. The tool enabled these users to provide natural language descriptions of their menu logic, which the system converted into valid, executable XML files for integration into mobile financial platforms.

This project was selected as a verification case for two reasons. First, it represents a realistic and commercially relevant GenAI adoption, involving code generation in a critical domain. Second, it required cross-role collaboration between developers, quality assurance engineers, legal reviewers, and product owners, reflecting the multi-stakeholder structure of GenAI adoption.

6.5.2 Application of the quality evaluation framework

The development process was structured around the phases and evaluation steps outlined in the quality evaluation framework.

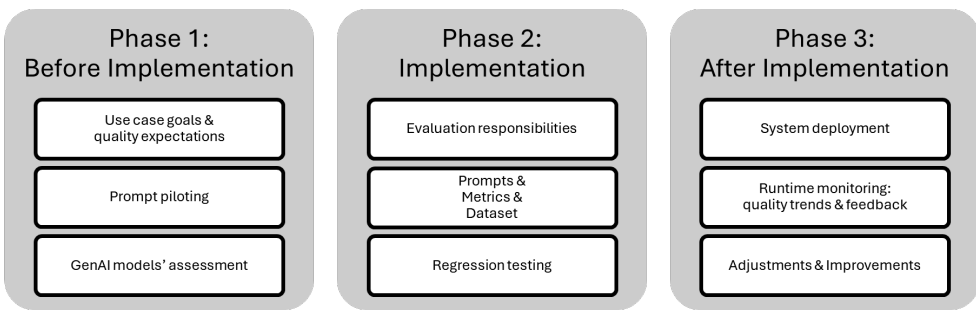

Figure 6.7: The process of applying the quality evaluation framework

As shown in Figure 6.7, in the **Before implementation** phase, the team specified use-case goals and quality expectations for GenAI-generated outputs, focusing on XML validity and semantic alignment. Prompt design and baseline testing were conducted with a suite of curated merchant inputs, and multiple GenAI models were assessed, including GPT-4o, Llama 3, and Gemini. Quality characteristics such as

correctness, adaptability, and reliability were defined in relation to ISO/IEC 25059 and ISO/IEC 25023 guidance. The team selected GPT-4o based on output consistency and schema adherence in prompt testing. Legal and security experts reviewed the use case and verified that no sensitive or personal data would be handled, minimizing legal risk exposure.

In the **During implementation** phase, development work was integrated into quality evaluation steps. Specific responsibilities were assigned to developers, quality assurance (QA) engineers, and architects. Developers implemented prompt engineering improvements based on failure cases observed in a validation dataset. QA teams conducted regression testing and introduced a customized quality checklist inspired by the tables proposed in Section 6.4 and 6.3.

Evaluation covered functional suitability, efficiency, and correctness, using metrics such as XML validity rate, response time, and coverage of logic branches. This aligns with the middle phase of the framework, where evaluation activities are embedded in GenAI application tasks.

In the **After implementation** phase, the system was deployed to a live production environment with monitored usage. Grafana dashboards [145] were used to visualize runtime quality indicators. Metric scores, such as output validity and latency, were collected using Prometheus Pushgateway and logged over time. Feedback from merchant users was also monitored to detect misunderstood or misrendered menu structures. Operational adjustments were made based on findings, including prompt template refinement and rule-based XML post-validation. This phase corresponded to the framework's final stage and enabled long-term quality tracking and post-deployment tuning.

6.5.3 Verification outcomes

The proposed framework was applied to all three phases of the USSD XML Generator case. It enabled the team to address quality concerns with defined responsibilities, metrics, and checkpoints.

In the early phase, the model assessment helped filter out unsuitable options and anticipate risks. Legal and security engineers used the checklist to validate compliance concerns before deployment. One engineer noted, "The checklist turned vague worries into actionable points, such as licensing, datasets, inputs, and outputs." During development, quality metrics were embedded into the implementation process. Accuracy and latency were measured continuously using in-house scripts and Prometheus monitoring [146]. A quality assurance engineer remarked, "The quality characteristics gave us criteria early, so we knew what to test and why." Operational monitoring added a feedback loop post-deployment. Monitoring dashboards showed runtime trends that directly informed model prompt adjustments. "Seeing the correctness rate of generated outputs improve week by week gave us real confidence," said

an operations engineer.

Despite its practical value, the framework had limitations during the case implementation. First, the success of its application relied on the willingness and capacity of team members to take ownership of additional evaluation tasks. Second, although the framework included evaluation practices with ISO alignment, the evaluation results still depended on the quality of test data and prompt engineering, where guidance was limited. Third, the monitoring approach, while useful, required specific tooling and metrics setup that may not generalize to other organizations.

We position the framework as a meta-level structure with a stable core (checkpoints, role duties, and a minimal metric set) and configurable parts (attributes, datasets, and safeguards). Teams can instantiate it per feature or domain using shared templates and steps, rather than building a new method or bringing new tools each time. In the USSD case, the core metrics were reused; prompts and test data were tailored.

6.6 Validity Threats

We follow Runeson et al.'s guideline to address validity threats: construct, internal, external, and conclusion validity [147].

Construct validity To identify quality evaluation practices in GenAI adoption, we conducted 19 interviews with practitioners who had hands-on experience with GenAI-supported software development. Participants were selected from multiple roles, including development, testing, architecture, legal, and management, to ensure diverse perspectives. Although each participant had experience with parts of the entire process, their combined accounts provided a holistic view across its constituent steps. However, participants may have used inconsistent terminology or lacked shared definitions of "quality," "evaluation," or "GenAI adoption." To mitigate this, we clarified questions during interviews (e.g., "What did your team do to ensure the model was suitable?"), followed by clarifying questions and validation of summaries during transcription checks.

Internal validity Our thematic analysis followed a staged coding process, including initial open coding, coding for pattern identification, and iterative refinement. Multiple researchers reviewed the coding structure and figure derivation to minimize personal bias. However, participant statements may still include misinterpretations. We mitigated this through three group sessions to review collected artifacts (e.g., internal documents and metrics). For the verification case, the same research team collaborated with practitioners during implementation, which may introduce bias toward confirming our proposed framework. To reduce this, evaluation metrics and improvement steps were interpreted by study participants.

External validity In this study, we selected two software development companies from the same corporate group, which may introduce shared culture and governance effects. The framework was derived in a setting with development teams, in-house legal and security support, and cross-site collaboration. Organizations that lack these prerequisites, such as smaller teams without dedicated risk or operations functions, may need to adapt roles, checkpoints, and metrics before adoption. Therefore, we present the framework as process-oriented guidance rather than a universal prescription, and we distinguish elements from configurable ones to aid tailoring. The verification case demonstrates feasibility in the studied context, but applying the framework elsewhere will require adaptations to local constraints.

Conclusion validity We documented research procedures, coding steps, and analytical decisions to support transparency. The first three researchers analyzed the data and resolved differences to ensure reliability. Participants' feedback was used to validate our interpretations of key themes.

We acknowledge potential threats to conclusion validity. For example, if the data concerning legal evaluation practices were incomplete or misinterpreted, an alternative conclusion might be that legal responsibilities are decentralized rather than coordinated through a central unit. However, our collected data showed that legal evaluation was explicitly addressed by multiple participants. Furthermore, our multistep verification, such as peer checking, participant feedback, and triangulation, reduces the risk of such misinterpretations.

Additionally, we reported both positive outcomes and challenges during the verification of the proposed framework to mitigate confirmation bias. Future studies are needed to understand the framework's long-term effects in other industrial settings.

6.7 Discussions

6.7.1 Legal risk and security compliance come first?

In our cases, preliminary legal and security checkpoints blocked or delayed pilots. The reasons included supplier terms that allowed training on user inputs, unclear ownership of generated code, and data-residency constraints that conflicted with company policy (Table 6.3). While many engineering activities are iterative and exploratory, legal and compliance violations can trigger irreversible consequences, such as breach of contract, regulatory sanctions, or reputational damage. This risk profile justifies the placement of legal and security evaluation as the first quality gates in our framework.

The necessity of prioritizing these concerns became evident in our interviews. As one participant noted: "We did not even get to try the model, legal shut it down because of the terms of service." Unlike performance issues or usability flaws, le-

gal violations typically cannot be patched silently once a product reaches customers. This means the cost of deferred evaluation is not technical debt but organizational risk. Treating these issues as upstream concerns helps ensure that software built on GenAI is legally viable and security-resilient before other quality aspects are even meaningful.

Additionally, the placement of these assessments early in the workflow promotes shared accountability across teams. Rather than delegating compliance checks to external gatekeepers, integrating them into quality evaluation encourages engineers to consider these aspects as part of their technical responsibility. This shift from reactive compliance to proactive governance requires extra resources to deal with the risk and compliance evaluation.

6.7.2 Introducing a GenAI quality lead role

Our findings show that quality evaluation responsibilities in GenAI-supported software development are distributed across several functional roles, including legal teams, developers, security engineers, analysts, and quality managers. While each role addresses a specific concern—such as compliance, technical performance, or operational monitoring—no designated function oversees quality holistically across the entire lifecycle.

This absence of centralized responsibility leads to fragmentation in evaluation practices and limited traceability of decisions. Practices such as legal risk assessment or runtime evaluation are conducted in isolation, without a unifying perspective that ensures GenAI-specific quality concerns are systematically addressed.

To mitigate this gap, we propose the establishment of a dedicated role: the **GenAI Quality Lead (GQL)**. The GQL is envisioned as a cross-functional role that ensures quality considerations are proactively embedded throughout the GenAI adoption lifecycle—from model selection and data preparation to deployment and continuous operation.

Specifically, the GenAI Quality Lead would coordinate the following responsibilities:

- Collaborate with legal and security teams during model sourcing to address compliance and licensing risks.
- Work with developers and analysts to define evaluation criteria aligned with business and technical requirements.
- Ensure structured assessment of output quality and runtime behavior, drawing on relevant metrics and feedback loops.
- Facilitate alignment with external standards, such as ISO/IEC 25059, by translating quality characteristics into actionable practices.

This role draws conceptual support from the SE4AI (Software Engineering for AI) perspective, which emphasizes the need for dedicated quality assurance practices tailored to AI-based systems. The GenAI Quality Lead does not replace existing domain experts, but rather functions as a coordinating actor that integrates diverse quality concerns into a consistent and traceable process.

Establishing such a role may improve organizational capability to systematically evaluate GenAI applications and adapt to emerging regulatory and standardization landscapes.

6.7.3 Why ISO quality characteristics matter

Across the organizations we studied, quality was interpreted through different aspects: efficiency by developers, explainability by managers, and compliance by legal teams. This fragmentation created challenges in aligning expectations, coordinating evaluations, and making trade-offs explicit. What was missing was a common foundation to reason about quality without flattening its complexity.

ISO/IEC 25059 fills this gap by offering a standardized structure of quality characteristics tailored to AI-enabled systems. It may be more recognizable to software engineering professionals compared to custom system qualities. In our implementation, ISO characteristics helped participants translate vague concerns into actionable targets. For instance, instead of "the model should be fast and safe," teams articulated expectations as "inference time $<$ 45 seconds (efficiency)" and "no confidential leakage from enterprise information (security)." This shift enabled traceable decisions and role-specific accountability throughout the lifecycle.

Moreover, ISO's separation of characteristics and sub-characteristics (e.g., from reliability to robustness) allowed practitioners to tailor depth without discarding structure. This flexibility appears important in real-world development, where not all quality attributes are equally relevant. The standard helps teams organize what they have already done, while exposing what might be missed in terms of software quality.

We acknowledge that ISO 25059 is still under refinement and does not address all GenAI-specific characteristics. However, our evidence suggests that it provides a shared model grounded in both academic rigor and industrial recognition.

Relation to the EU AI Act While ISO/IEC quality standards help structure technical quality, they do not by themselves address regulatory duties under the EU AI Act (e.g., risk classification, documentation, conformity assessment, post-market monitoring). When this study started, the operational guidance for the Act was still evolving in the studied companies, and the usage of software development was considered unclear. Therefore, teams mapped existing ISO-based quality characteristics to internal policies for GenAI adoption and usage, conducted legal and privacy reviews, and prepared for later alignment with evolving standards and regulatory requirements,

not limited to ISO/IEC 25059.

6.7.4 Future work

Future studies should investigate how GenAI quality evolves over time and how teams update evaluation criteria accordingly. Moreover, the GenAI Quality Lead role needs deeper exploration, including required competencies, training models, and its interaction with existing quality assurance structures.

6.8 Conclusions

The use of GenAI technologies in industrial software development introduces new challenges in software quality assurance. Through a case study in two industrial organizations, we investigated the process of GenAI adoption in practice, what quality aspects matter, who is responsible for evaluation, and how these activities are carried out across the software development.

In the study, we contribute a framework: a process-oriented and role-aware view of GenAI quality evaluation. The framework is built based on observed practices and practitioners' feedback. This framework clarifies when and how quality should be assessed between legal, technical, and operational responsibilities.

By visualizing where evaluation takes place and what characteristics are considered, we help practitioners gain a deeper understanding of GenAI-related quality. The introduction of the GenAI Quality Lead role addresses a missing coordination function, supporting traceability, consistency, and compliance in multi-role environments. The real-world implementation further confirms the framework's applicability and exposes opportunities for refinement.

Looking forward, GenAI adoption can accelerate across sectors and development contexts. It raises the necessity for quality evaluation frameworks that are not only theoretically grounded but also implementable in practice. Our study offers such a framework for practitioners to follow, and also provides a lens to rethink software quality in the era of GenAI.

Study F

Evaluating the Sufficiency of Single-Agent LLM Systems for Algorithmic Problem Solving in Support and Operations

Abstract

Support and operations engineers frequently develop lightweight scripts to parse logs or reconcile data — tasks that rely on algorithmic code patterns. While Multi-Agent systems are often advocated for their reliability, it remains unclear if their coordination overhead is justified for these self-contained tasks. We analyzed 27000+ industrial source files to validate the prevalence of small, common, but non-trivial algorithmic patterns used in practice (hash maps, string operations, sorting, and simple tree or graph traversals). We then conducted a controlled experiment on 150 LeetCode algorithmic problems that are representative proxies for these patterns, using four state-of-the-art LLMs (GPT-5.1, Claude-4.5, Deepseek-chat-v3.1, Gemini-2.5-pro). Contrary to the 'more is better' assumption in AI agents, we observed a capability saturation effect: the evaluated models already solved most tasks with a single-turn prompt, leaving limited room for Multi-Agent orchestration to improve acceptance. Single-Agent baselines achieved high acceptance rates (95–99%), statistically indistinguishable from Multi-Agent systems. However, the Multi-Agent approach introduced an increase in latency and token cost without yielding meaningful quality gains in code quality, as measured by cyclomatic complexity and lines of code. Our study results indicate that for well-defined and self-contained algorithmic scripting tasks, the four selected LLMs have crossed a sufficiency threshold: they already provide suitable single-turn solutions, so extra agent orchestration is not required for these tasks. Although the single-agent solution performed equally to the multi-agent solution in the context of this study, the results provide no evidence to suggest that similar behaviors should be observed for other, or more general, software engineering tasks.

7.1 Introduction

The advent of Large Language Models (LLMs) has fundamentally transformed software engineering, enabling automated code generation at an unprecedented scale [29, 130]. While early tools like GitHub Copilot [148] focused on completing single lines or functions, the field is rapidly shifting towards AI agents/tools — autonomous systems capable of planning, executing, and verifying complex tasks [149]. Frameworks such as AutoGen [150], MetaGPT [151], and ChatDev [152] advocate for multi-agent architectures, where specialized agents (e.g., Architect, Engineer, Tester) collaborate to solve problems. The prevailing assumption in recent literature is that multi-agent orchestration inherently outperforms single-agent generation by mimicking human team dynamics [149, 153].

However, this "more is better" hypothesis remains untested mainly for fundamental, self-contained algorithmic tasks. Benchmarks like SWE-bench [154] assess multi-file feature work, but do not speak directly to the short scripts that support/-operations engineers produce under time pressure. For these engineers, the primary need is for correct, efficient, and self-contained solutions [155]; extra orchestration is only helpful if it measurably improves reliability or quality.

We combine an observational scan with a controlled experiment. First, we analyze ten actively maintained industrial repositories that underpin core services in a large enterprise, characterizing the prevalence of small, common, but non-trivial algorithmic patterns that we observe in practice, such as hash maps, string operations, sorting, and basic tree or graph traversals, to ground the study in industrial reality. Second, we evaluate four state-of-the-art LLMs (GPT-5.1, Claude-4.5, Deepseek-chat-v3.1, and Gemini-2.5-pro) on a dataset of 150 LeetCode problems (50 Easy, 50 Medium, 50 Hard) that are representative proxies of these patterns. We evaluate two orchestration strategies: a Single-Agent prompt (baseline) and a sequential Multi-Agent pipeline (Analyzer $\rightarrow$ Designer $\rightarrow$ Executor $\rightarrow$ Verifier).

We explicitly define our Single-Agent baseline as a direct, single-turn interaction to mimic the time-constrained workflow of support and operations engineers who work with CI/CD pipelines and production systems. When a site or CI environment is down, engineers prioritize immediate script generation over complex, autonomous iterative loops. By contrasting this baseline against a structured Multi-Agent pipeline, we aim to isolate the coordination cost — the additional time consumed by decomposing a task that might already fall within the model's default single-agent capabilities, without extra prompt augmentation or multi-agent reasoning.

The main contributions of this study are:

1. **Industrial Relevance Verification**: A scan of 27000+ source code files from ten industrial projects, establishing that abstract algorithmic patterns (e.g., Hash Maps) serve as the critical 'glue code' for data reconciliation and log parsing in enterprise systems.

2. **Evidence of Capability Saturation**: Empirical results from 150 selected algorithmic problems showing that, for this domain, the evaluated single-agent and four-agent solutions both achieve acceptance rates above 95%.

3. **Evidence of Cost**: An analysis of resource overhead reveals that Multi-Agent orchestration increases latency and token usage by approximately 4x to 6x, while yielding only small, statistically non-significant changes in code quality metrics (cyclomatic complexity and lines of code).

Regarding the study scope and limits, our findings apply to small, self-contained algorithmic problems with deterministic oracles, to a sample of 150 LeetCode problems (50 Easy, 50 Medium, 50 Hard) randomly selected from three study plans among the top 25% most-liked problems and filtered to exclude tasks requiring images, interaction, or external services, and to the four models tested during the study period. We chose LeetCode because it offers a varied set of widely used pure code-generation tasks while maintaining a comparable problem structure across difficulty levels. We do not claim that these results generalize to repository-level feature work (e.g., SWE-bench [154] or FEA-bench [156]) or to stronger single-agent baselines that include self-reflection or tool use.

The remainder of this paper is organized as follows: Section 7.2 provides relevant research on LLM-based code generation and multi-agent systems. Section 7.3 describes our experimental design and implementation following Wohlin's guidelines. Section 7.4 presents our findings, followed by discussion in Section 7.5. Section 7.6 addresses threats to validity, and Section 7.7 concludes this study.

7.2 Related work

Large Language Models have transformed automated programming. Existing studies established the capabilities of models like Codex [29] and AlphaCode [157] on competitive programming tasks. More recently, the focus has shifted to real-world software engineering tasks. Benchmarks such as SWE-bench [154] and FEA-bench [156] evaluate agents on their ability to resolve GitHub issues or implement new features in existing repositories. While these benchmarks provide high ecological validity, they report very low success rates (<20% for unassisted agents) [156]. For a support or operations engineer, a tool that succeeds 20% of the time is often less helpful than one that reliably solves smaller, well-defined sub-problems. Our work complements these studies by focusing on the atomic units of scripting — algorithmic patterns — where high reliability is achievable and expected.

7.2.1 Agentic code generation

Recent advancements have moved beyond simple prompting to agentic loops that employ tool use and self-correction. Systems like SWE-agent [158] and AutoCodeRover [159] demonstrate that agents can resolve complex GitHub issues by navigating repositories, editing files, and running tests. AgentCoder [160] and Reflexion [161] show that iterative self-refinement and multi-agent collaboration can significantly improve performance on algorithmic benchmarks. However, these systems often incur high latency and cost. Our work complements this literature by isolating the orchestration factor on atomic algorithmic tasks, determining whether the overhead of multi-agent coordination is justified for small, self-contained problems where a single prompt might suffice.

7.2.2 Multi-Agent systems in software engineering

Multi-agent frameworks, such as AutoGen [150], MetaGPT [151], and ChatDev [152], employ role-based collaboration (e.g., Product Manager, Architect, Engineer) to handle complex tasks. These systems excel at decomposition and planning for large-scale software construction. However, it remains an open question whether this heavyweight orchestration is beneficial for the shorter, self-contained tasks typical of infrastructure scripting. Recent work by Hou et al. [153] suggests that the benefits of multi-agent systems can be task-dependent. We contribute to this discourse by isolating the orchestration factor for a specific class of algorithmic problems, providing evidence that "less is more" in this domain.

7.2.3 Algorithmic patterns in industry

While algorithmic problems on LeetCode are often criticized as being detached from reality, algorithmic efficiency is crucial for infrastructure at scale. Studies on code smells and performance bugs [162] often point to inefficient loops or improper data structure usage. Our work provides empirical data from actual industrial repositories to quantify exactly which LeetCode patterns appear in production code, bridging the gap between interview practice and industrial reality.

7.3 Research Methodology

We follow the guidelines for controlled experiments in software engineering by Wohlin et al. [31]. The controlled-experiment process consists of three main phases: Scoping, Planning, Design & Instrumentation, and Operation. Figure 7.1 illustrates our research processes.

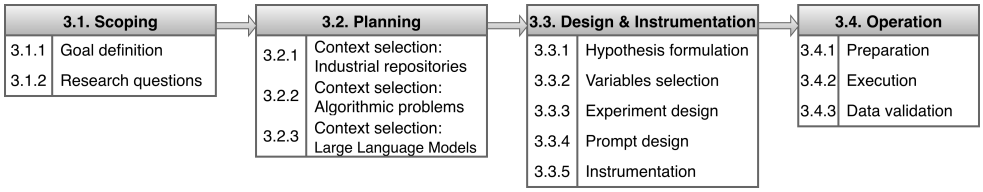

Figure 7.1: Overview of the research process following Wohlin et al. [31].

7.3.1 Scoping

The scoping step includes the goal definition and research questions [31].

7.3.1.1 Goal definition

The primary goal of this study is to evaluate the effectiveness of agent-based LLM systems in supporting engineers in efficiently generating scripts of sufficient quality for small, self-contained algorithmic tasks in industrial contexts. As a prerequisite to this evaluation, we quantify the frequency at which the corresponding algorithmic patterns occur in industrial repositories, ensuring that the studied tasks reflect the realistic needs of practitioners. This goal guided the formulation of the research questions and the selection of metrics.

7.3.1.2 Research questions

Derived from the research goal, we formulate three research questions:

RQ1 (Prevalence): How frequently do small algorithmic patterns that could be supported by LLM-based scripting assistance appear in the selected industrial code repositories?

RQ2 (Effectiveness): How do Single-Agent and Multi-Agent LLM systems perform in solution acceptance rate on the selected small, self-contained algorithmic problems across different LLMs?

RQ3 (Quality): What differences exist in code quality metrics (complexity and lines of code) between code generated by single-agent and multi-agent systems for the selected small, self-contained algorithmic problems?

7.3.2 Planning

The planning primarily involves selecting the context, including industrial code repositories and algorithmic problems.

7.3.2.1 Context selection: Industrial repositories

To answer RQ1, we employed convenience sampling to select ten internal software repositories from a large industrial organization. These repositories (anonymized as Project 1 to Project 10) represent active microservices written in Java/Python for ap-

plications deployed in a FinTech business domain. The purpose of this analysis is to ground the studied algorithmic patterns in actual production code and to show their prevalence and face validity in industrial contexts. Across these projects, we analyze over 27K source files, providing a large code base in which we expect the most common small-scale patterns to appear, later verified through the analysis results.

We used an offline pattern-based scanner to traverse each source file, detect predefined pattern types (hash map operations, sorting calls, string operations, and simple tree or graph constructs), and count matches per project. The scanning scripts and pattern definitions are included in the artifact for transparency and replication. We chose a code-level scan rather than a raw text search to reduce false positives from comments or unrelated tokens, but acknowledge that heuristic pattern matching can still undercount or overcount depending on coding style. Some practitioners view LeetCode as a collection of abstract puzzles [163]. However, our scan of 27K industrial files reveals that these 'puzzles' are the atomic units of 'glue code.'

7.3.2.2 Context selection: Algorithmic problems

While LeetCode problems are closed-ended, they proxy the logic core of many operational scripting tasks carried out by support and operations engineers. By isolating this core, we evaluate the model's ability to handle the specific logical density found in industrial scripting, separate from environmental factors.

To answer RQ2 and RQ3, we selected 150 LeetCode problems to serve as controlled proxies for the identified algorithmic patterns. The selection criteria were:

- *Sampling*: We randomly selected 50 problems from each difficulty category (Easy, Medium, Hard). This difficulty-balanced sampling avoids over representation of any single difficulty level and ensures that we include a substantial number of harder problems where multi-agent decomposition is hypothesized to provide the most benefit.
- *Popularity*: Problems were randomly sampled from the top 25% most-liked problems in each category to ensure quality and community relevance.
- *Exclusion criteria*: We excluded problems requiring visual inputs (images), interactive elements, or external HTTP interfaces, so that all tasks reduce to pure code generation in the target programming language.

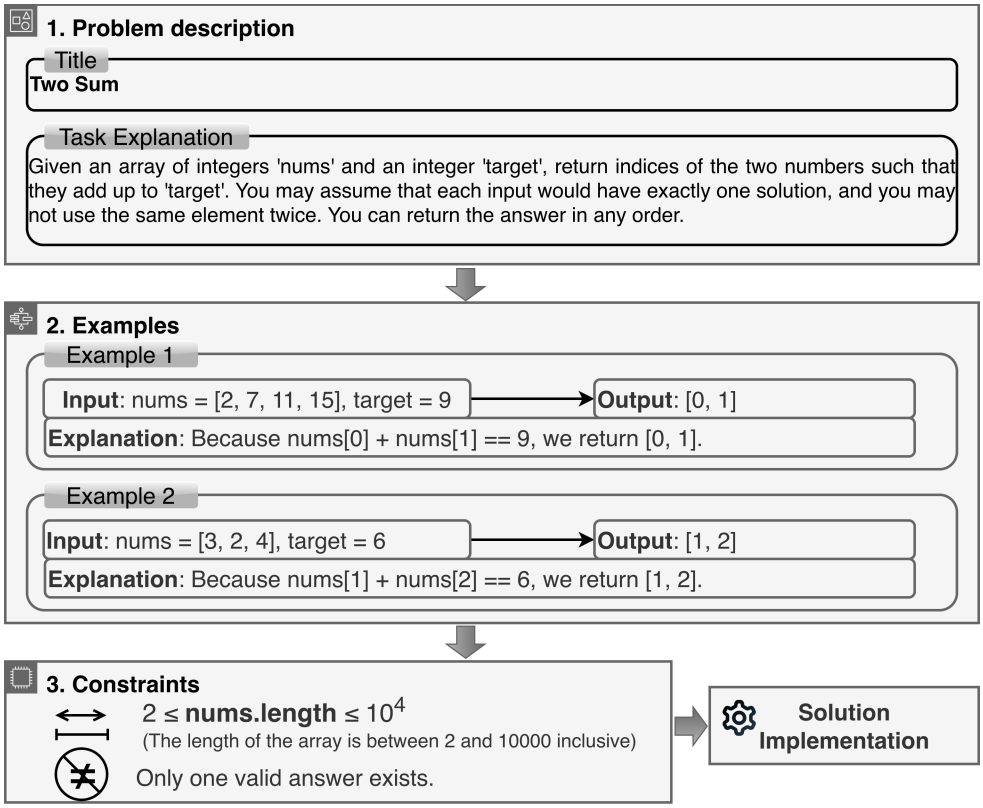

Figure 7.2: Structure of a selected algorithmic problem --- Two Sum. The explicit constraints and examples provide a ground truth for verifying the correctness of implemented solutions.

Figure 7.2 illustrates the structure of the *Two Sum* algorithmic problem. This problem asks for the indices of two numbers in an array whose sum equals a target value. For example, when `nums = [2, 7, 11, 15]` and `target = 9`, the correct output is `[0, 1]` because `2 + 7 = 9`. Constraints restrict the array length to the range $[2, 10^4]$ and guarantee exactly one valid answer. Editorial solutions and community discussions become available after submission, supporting cross-checking of generated code.

We chose these algorithmic tasks because they are self-contained, text-described, and possess a deterministic oracle (pass/fail) [164]. User submissions are evaluated against predefined, automated tests, and the platform reports whether all cases passed. Figure 7.3 illustrates this automation pipeline, where the deterministic oracle enables scalable and objective evaluation of solutions without human intervention.

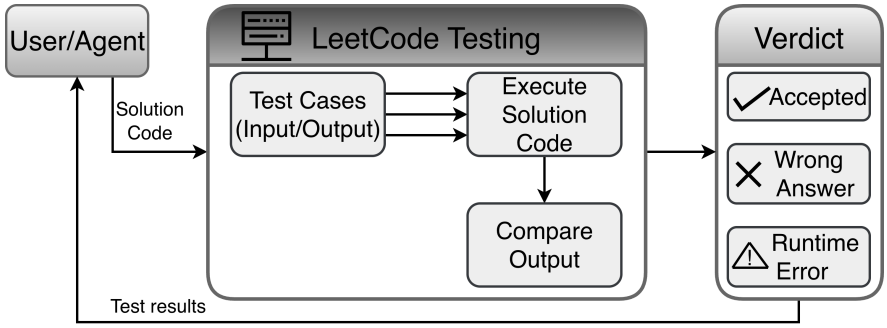

Figure 7.3: The LeetCode automation pipeline. The platform provides a deterministic execution environment and test cases, enabling objective, scalable verification of submitted solutions.

7.3.2.3 Context selection: Large Language Models

To ensure our findings represent the current state-of-the-art, we selected four frontier models representing diverse model families and providers. Our selection criteria prioritized models that ranked among the top entries on coding benchmarks such as SWE-bench [165] as of November 2025. The selected models are:

- **GPT-5.1**: The latest iteration from OpenAI (released Nov 2025), chosen for its reported improvements in complex reasoning.
- **Claude-4.5** (Sonnet): Anthropic's leading model, selected for its strong performance in code generation and instruction following.
- **Deepseek-chat-v3.1**: A specialized coding model that represents an open-weight alternative with high reasoning capabilities.
- **Gemini-2.5-pro**: Google's multimodal model, included to assess performance diversity across different architectures.

All models were accessed via the OpenRouter [1] API during November 2025, using the exact model versions listed above to ensure reproducibility and version consistency.

7.3.3 Design and Instrumentation

This phase covers hypothesis formulation, variable selection, experiment design, instrumentation, and validity evaluation.

7.3.3.1 Hypothesis formulation

We focus on hypotheses in the comparative research questions (RQ2 and RQ3):

For RQ2 (Acceptance):

[1] https://openrouter.ai

- H_{01}: There is no difference in solution acceptance rate between single-agent and multi-agent systems.
- H_{11}: Multi-agent systems achieve a different solution acceptance rate than single-agent systems.

For RQ3 (Code quality):

- H_{02a}: The mean cyclomatic complexity is equal between single-agent and multi-agent systems.
- H_{12a}: The mean cyclomatic complexity differs between single-agent and multi-agent systems.
- H_{02b}: The mean lines of code are equal between single-agent and multi-agent systems.
- H_{12b}: The mean lines of code differ between single-agent and multi-agent systems.

7.3.3.2 Variables selection

Table 7.1 presents the variables used in our experimental design. The first author identified and operationalized these variables based on the research questions and established software engineering metrics.

Table 7.1: Selected variables

Variable Type	Variable name	Description
Independent	System Type	Single-agent vs Multi-agent
Dependent	Acceptance rate	Percentage of solutions marked "Accepted" by LeetCode (ratio)
	Lines of Code	Total executable lines in generated solution (ratio)
	Cyclomatic Complexity	McCabe complexity measure (ratio)
Resource	Generation Time	Time required to generate solution
	Token Usage	Input and output tokens consumed
Controlled	LLM	GPT-5.1, Claude-4.5, Deepseek-chat-v3.1, Gemini-2.5-pro
	Problem Set	Identical 150 algorithmic problems for both systems
	Evaluation Criteria	Same test cases on LeetCode server

Each variable in Table 7.1 was carefully defined to ensure measurability and reproducibility. Acceptance Rate records the proportion of submissions that LeetCode labels “Accepted.” Lines of Code excludes comments and blank lines using a language-aware parser. Cyclomatic Complexity is computed on the generated abstract syntax tree via the standard McCabe algorithm. Generation Time measures the time from issuing the first model call until the final code artifact is produced. Token usage comes directly from the API response metadata. Lines of code and cyclomatic complexity are widely used structural code metrics in empirical software

engineering [166], capturing the size and control-flow complexity of generated programs, which are relevant indicators for maintainability and potential defect proneness [166].

7.3.3.3 Experiment design

To rigorously evaluate the sufficiency of the Single-Agent, we established a Multi-Agent system for comparison. This allows us to determine whether the increased complexity of the Multi-Agent setup yields measurable performance gains or whether the Single-Agent baseline has already reached a saturation point for solving these algorithmic tasks. We subjected both the Single-Agent and Multi-Agent systems to the same set of 150 selected algorithmic problems. This controlled design maintains constant problem instances across models and isolates system type (Single- or Multi-Agent) as the sole independent variable.

Figure 7.4 illustrates the overall experimental flow, highlighting the direct prompting baseline against the sequential multi-agent processing.

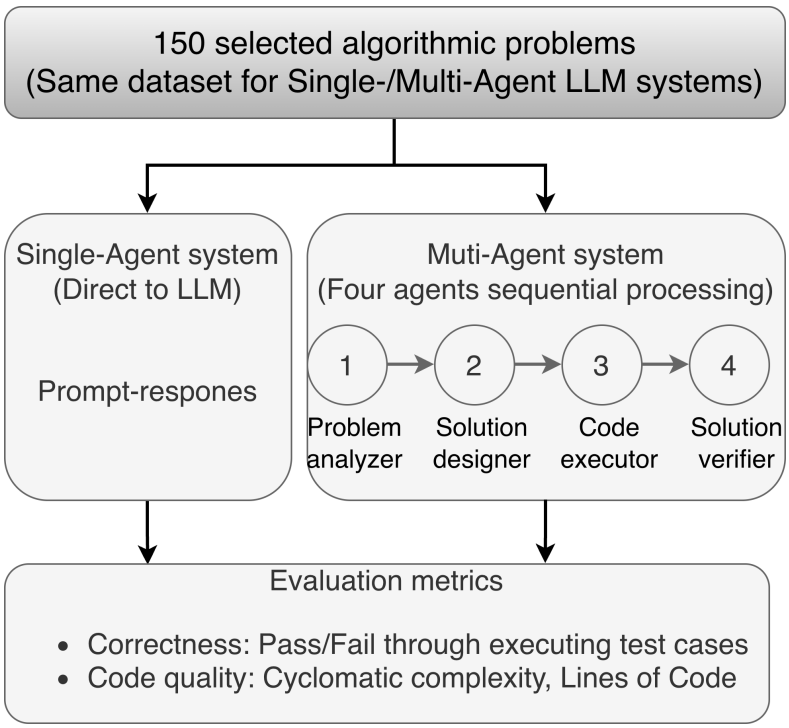

Figure 7.4: Experimental design evaluating Single-Agent direct prompting and Multi-Agent sequential processing on the selected algorithmic problems.

Single-Agent Baseline (Direct prompting): The left path of Figure 7.4 represents the Single-Agent system. The model receives the complete problem statement and a coding instruction prompt, returning a solution without intermediate visible reason-

ing steps.

Multi-Agent System (Sequential decomposition): The right path of Figure 7.4 details the Multi-Agent chain: *Analyzer* → *Designer* → *Executor* → *Verifier*. We selected this sequential architecture (rather than a hierarchical swarm) to enforce a separation of concerns that mirrors the Software Development Life Cycle (SDLC), including requirements, design, implementation, and verification [167, 168]. By decomposing the generation process, we aim to:

1. **Externalize Chain-of-Thought (CoT) [169]:** Transform the latent reasoning of the LLM into discrete architectural components (e.g., separating design from implementation).
2. **Isolate Error Propagation [170]:** Allow for the analysis of specific failure points, such as whether a correct design leads to a failed implementation.
3. **Mimic Industrial Review:** The *Solution Verifier* simulates a peer-review step, replaying examples to flag logic errors before final submission.

The specific roles function as follows:

- **Problem Analyzer:** Extracts requirements and constraints.
- **Solution Designer:** Outlines the algorithm and complexity targets.
- **Code Executor:** Implements the solution strictly following the Designer's blueprint.
- **Solution Verifier:** Replays examples and flags fixes if the code fails the provided tests.

Together, these four roles form a minimal decomposition that covers requirement analysis, algorithm design, implementation, and basic checking for the scripting tasks.

Both Single- and Multi-Agent systems use the same evaluation metrics: pass/-fail verdicts from the LeetCode evaluation, code-quality measures (cyclomatic complexity and lines of code), and efficiency indicators (generation time and token usage).

7.3.3.4 Prompt design

To ensure reproducibility and transparency, we explicitly defined the prompt structures for both systems. The full text of all prompts is available in our replication package on Github.

Single-Agent prompt: The Single-Agent system employs a direct, zero-shot prompting strategy. The prompt is designed for rapid scripting, providing the model with the problem statement, a code snippet with fixed signatures, and a set of constraints. Figure 7.5 illustrates the template used. We instruct the model to output only valid source code, minimizing conversational filler that would require post-processing.

Single-Agent prompt template

Please write your solution in the {language} programming language. Your code must:

* Solve the problem fully and correctly.
* Pass all provided sample test cases.
* Run within acceptable time and memory limits.

Here is the problem statement: {question} Here is the code snippet, which you should expand with your solution: {snippet} Important Requirements:

- Do not change any provided function signatures.
- Output only valid source code.

Figure 7.5: The prompt template used for the Single-Agent baseline. The variables {language}, {question}, and {snippet} are dynamically populated.

Multi-Agent prompts: The Multi-Agent system utilizes a sequential chain of specialized agents. Each agent receives a distinct system prompt defining its role, inputs, and expected outputs. Table 7.2 summarizes the responsibilities and key instructions for each agent in the pipeline. This separation of concerns ensures that the *Code Executor* receives a verified design plan rather than raw requirements, and the *Solution Verifier* acts as an independent quality gate.

Table 7.2: Overview of Multi-Agent roles and prompt strategies

Agent Role	Input	Key Instructions
Problem Analyzer	Raw Problem Statement	Analyze problem type, constraints, edge cases, and difficulty. Output structured analysis.
Solution Designer	Problem + Analysis	Design a high-level algorithm, select data structures, and provide step-by-step workflows.
Code Executor	Problem + Analysis + Design	Implement the solution in the target language following the design exactly. No explanations, code only.
Solution Verifier	Problem + Design + Code	Review the code for correctness, edge case handling, and complexity. Walk through examples.

7.3.3.5 Instrumentation

To ensure reproducibility and controlled evaluation, we implemented scripts and interfaces in GO to automate problem downloading, LLM interactions, submissions

of generated code, and extraction of LeetCode testing verdicts. The complete implementation is included in our replication package, which enables repetition of the experimental process.

Our implementation consists of several integrated components. The problem acquisition (download.go) module automatically downloads problem descriptions, constraints, and test cases from the three selected LeetCode study plans using the LeetCode API. The solution generation module implements both single-agent (prompt.go) and multi-agent (multiagent.go) approaches, managing API requests to the selected model with appropriate prompts and coordination logic for the multi-agent system.

The evaluation component automatically submits generated solutions to LeetCode's testing platform, capturing acceptance rates, runtime performance, and error messages. Our metrics extraction component analyzes the generated code to compute cyclomatic complexity using established algorithms, counts lines of code (excluding comments and blank lines), and measures token usage and generation time.

The data module stores experiment data in structured JSON format, maintains detailed logs of API interactions and system responses, and provides automated scripts for statistical analysis and result compilation. Quality assurance mechanisms include validating problem downloads against expected formats, verifying solution submissions and capturing results, and conducting consistency checks across all generated metrics.

Our goal is repeatability rather than bitwise determinism. We used the same problem set and logged all inputs/outputs for re-execution. Because LLM APIs are not strictly deterministic, we resubmitted when platform errors occurred and report aggregated results for each model. We release code, prompts, logs, and outputs to enable independent re-runs.

7.3.3.6 Statistical analysis

For acceptance rate comparison (RQ2), we used Fisher's exact test to compare the binary outcomes (pass/fail) between systems. Fisher's exact test is preferred over chi-square tests for categorical data with minor expected frequencies, as it provides exact p-values regardless of sample size. The test evaluates the null hypothesis that both systems have identical acceptance rates by calculating the probability of observing the actual contingency table (or a more extreme one) under this assumption.

For code quality metrics (RQ3), we employed paired two-sample t-tests to compare continuous variables (lines of code and cyclomatic complexity) between systems. We verified normality assumptions using Shapiro-Wilk tests and reported both statistical significance (p-values) and practical significance (Cohen's d effect sizes). Latency and token usage are reported descriptively as indicators of resource utilization. All statistical analyses were conducted using established software packages with an $\alpha = 0.05$ as the significance threshold.

7.3.4 Operation

Following Wohlin's guidelines [31], the operation phase executes the plan: prepare the controlled setup, run both systems on the same tasks, and validate outputs.

7.3.4.1 Preparation

We prepared a controlled environment to ensure the reliability and reproducibility of the experiments. This environment ran Ubuntu 20.04 LTS with 64 GB RAM and stable internet connectivity to minimize network-related variability. The preparation phase involved several critical steps, such as configuring API access, accommodating LeetCode submission constraints, and validating the problem dataset.

The first step, API access configuration, required obtaining Azure credentials with sufficient quota to handle the expected token volume (about 597 K across all experiments) and implementing rate limiting to respect API constraints while maintaining consistency. The token estimate was derived from pilot runs on the entire problem set.

LeetCode submissions occasionally failed due to rate limiting and service protections (HTTP 403/500). To remain compliant with the platform, we throttled request rates and scheduled runs during off-peak hours. These operational constraints extended the preparation time but did not change the evaluation protocol. We logged all submission outcomes for auditability.

7.3.4.2 Execution

We executed single-agent and multi-agent runs for each of the four models on the same 150 problems. Runs were scheduled during off-peak hours to reduce submission variability. For each model, we ran two sequential phases: first, the single-agent system, and then the multi-agent system. We repeated this full process three times per model and agent type, resulting in three replications for each configuration. We reset the environment between phases by opening a fresh session, clearing conversation history, and reloading the problem prompt and tests. We applied exponential backoff with capped retries, timeouts for stalled responses, and fallbacks for malformed handoffs in the verification step.

We logged timestamps for each API call and recorded per-problem metrics (accept/fail outcome, cyclomatic complexity, lines of code, generation time, and token counts). Our analysis aggregates these per-problem metrics for each model and orchestration.

7.3.4.3 Data validation

We implemented multiple validation layers covering both systems to ensure data integrity and experimental validity.

Real-time validation comprised automated checks for API response completeness, automated linters that ensured generated code compiled in Python, and confir-

mation that LeetCode acknowledged each submission.

Post-execution validation re-ran integrity checks. We verified that both systems processed all 150 problems, reconciled LeetCode submission verdicts with our local logs, and recomputed code metrics to confirm consistency. When we detected cases where the platform returned inconsistent verdicts for identical code, we immediately resubmitted within the same run to obtain stable results.

We also manually reviewed a uniform random sample of 27 multi-agent conversations (18% of the dataset) to verify consistency between the logs and outputs. The review confirmed that the implementation matched the intended design, and all planned metrics were collected for every problem.

7.4 Results

7.4.1 RQ1: Prevalence of algorithmic patterns

The scan of 10 industrial projects revealed heavy use of hash maps and string operations, with sorting and tree or graph patterns also present but less frequent.

Table 7.3: Algorithmic patterns in 10 industrial projects (27206 files)

Project ID	Files	Hash Maps	String Operations	Sorting	LOC of the patterns	Project LOC
Project 1	1,265	885	27	6	909	~145K
Project 2	329	259	24	4	272	~43K
Project 3	254	1370	8	0	1368	~38K
Project 4	21459	89461	10743	277	99399	~8.76M
Project 5	643	341	29	0	358	~87K
Project 6	458	348	8	7	359	~65K
Project 7	273	209	29	7	225	~36K
Project 8	792	144	2	0	145	~85K
Project 9	856	2252	81	2	2313	~163K
Project 10	877	1259	125	6	1361	~273K
Total	**27206**	**96528**	**11076**	**309**	**106709**	**~9.7M**

Table 7.3 summarizes the findings. Across 27,206 source code files, hash map usage was counted approximately 96K times, and string operations were counted about 11K times. We also measured ‘Pattern LOC’ — the number of lines of code containing at least one of these patterns — finding 106709 such lines. This represents approximately 1.1% of the total 9.7M LOC, which is a large proportion, especially considering these are specific algorithmic constructs rather than generic boilerplate. Sorting (309 instances) and tree or graph traversals (89 instances) were less common but still present, often related to data reconciliation and hierarchical data processing. Hash maps and string operations are prevalent throughout the projects, with Project 4 accounting for the majority of the usage due to its significantly larger size; however, smaller services also exhibit steady usage. Sorting and simple tree or graph constructs are rare, but their presence across projects indicates that these patterns are not confined to a single codebase. These results are observational: the scan is a convenience sample of industrial systems. It relies on heuristic pattern detection without

measured precision/recall, so that counts may under- or over-estimate actual usage.

We observed that the string operations identified in Project 4 were not merely API calls, but rather complex regular expression (regex) logic used for sanitizing inputs in CI/CD pipelines, which directly mirrors the complexity of LeetCode's text processing tasks.

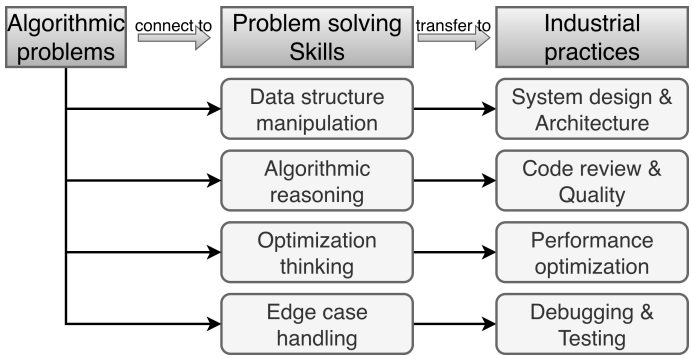

Figure 7.6: Conceptual mapping between algorithmic patterns and real-world tasks. The mapping justifies using LeetCode algorithmic problems as controlled proxies for industrial practices.

Figure 7.6 illustrates a mapping between algorithmic patterns and industrial tasks. The problem-solving skills (e.g., algorithmic reasoning patterns) mirror the logic required for data reconciliation tasks, such as matching records from two different logs or counting error types. Similarly, string manipulation in engineers' daily work is not just a coding puzzle but the core logic for parsing unstructured logs or sanitizing inputs for a CI/CD pipeline. By validating agents on these isolated patterns, we gain insight into their reliability for the corresponding industrial work.

7.4.2 RQ2: Effectiveness (Acceptance Rate)

Table 7.4 presents the acceptance rates on the 150 LeetCode problems across all four models. Single-Agent systems performed well: GPT-5.1 and Gemini-2.5-pro reached 99.33%, Deepseek-chat-v3.1 achieved 97.33%, and Claude-4.5 attained 95.33%. Multi-Agent orchestration provided only minor changes: GPT-5.1 remained at 99.33%, Claude-4.5 and Deepseek-chat-v3.1 both reached 99.33%, and Gemini-2.5-pro reached 97.99% (146/149, one system error). These minor differences came at the cost of increased latency and token usage.

Table 7.4: Average Acceptance Rate (150 Problems, three runs per agent type using the same model during November 2025)

Model	Single-Agent	Multi-Agent
GPT-5.1	99.33%	99.33%
Claude-4.5	95.33%	99.33%
Deepseek-chat-v3.1	97.33%	99.33%
Gemini-2.5-pro	99.33%	97.99%

To test hypothesis H_{01} (no difference in acceptance rate), we performed Fisher's exact test comparing Single-Agent and Multi-Agent outcomes for each model. The statistical analysis yielded non-significant results for all models at the $\alpha = 0.05$ level: GPT-5.1 ($p = 1.0$), Claude-4.5 ($p = 0.068$), Deepseek-chat-v3.1 ($p = 0.37$), and Gemini-2.5-pro ($p = 0.62$). Consequently, we **accept the null hypothesis** H_{01}. The data confirm that introducing a multi-agent setup for these self-contained algorithmic tasks does not produce a statistically significant improvement in solution acceptance rates compared to the single-agent baseline.

For support roles such as CI pipeline engineers, scripting engineers, testers, developer productivity engineers, operations staff, or troubleshooting engineers, these high acceptance rates suggest that a single capable LLM can reliably handle small, well-specified algorithmic tasks. For example, a troubleshooting engineer narrowing down a self-contained parsing bug can ask a single agent to identify and fix the issue, rather than manually searching through documentation. LLM use in these cases can reduce manual searching or hand-coding, provided the task has a clear oracle and matches the bounds of our problem set. We avoid extending this implication to longer, less-defined tasks or to domains outside the tested models and time frame.

Table 7.5: Failed problems by model and agent type

Model	Agent type	Problem	Status
GPT-5.1	Single	Palindrome Number	Wrong Answer
Claude-4.5	Single	Count Odd Numbers in an Interval Range	Runtime Error
Claude-4.5	Single	Excel Sheet Column Number	Runtime Error
Claude-4.5	Single	N-th Tribonacci Number	Runtime Error
Claude-4.5	Single	Paint House III	Wrong Answer
Claude-4.5	Single	Shortest Common Supersequence	Runtime Error
Claude-4.5	Single	Smallest Sufficient Team	Wrong Answer
Claude-4.5	Single	Tallest Billboard	Wrong Answer
Claude-4.5	Multi	Longest Duplicate Substring	Wrong Answer
Deepseek-chat-v3.1	Single	Binary Trees With Factors	Wrong Answer
Deepseek-chat-v3.1	Single	Permutation Sequence	Runtime Error
Deepseek-chat-v3.1	Single	Regular Expression Matching	Wrong Answer
Deepseek-chat-v3.1	Single	Sliding Window Median	Wrong Answer
Deepseek-chat-v3.1	Multi	Sliding Window Median	Wrong Answer
Gemini-2.5-pro	Multi	Constrained Subsequence Sum	Runtime Error
Gemini-2.5-pro	Multi	Max Dot Product of Two Subsequences	Runtime Error
Gemini-2.5-pro	Multi	Word Ladder II	Time Limit Exceeded

For a better understanding of the differences in acceptance rates, we examined the failed algorithm problems, as shown in Table 7.5. Most failures were runtime errors or timeouts on more challenging cases; a few were incorrect answers on edge cases. LeetCode does not expose partial test counts, so only the final verdicts are reported. These cases show that even on short tasks, a quick edge-case check remains useful. We note that several failures occur on combinatorial or graph-heavy problems (e.g., **Word Ladder II**, **Sliding Window Median**) or on corner-case-sensitive tasks (e.g., **Palindrome Number**, **Regular Expression Matching**). This suggests that performance pitfalls and edge conditions remain common failure modes, and prompts or verifier steps that emphasize edge cases and basic complexity checks could be

beneficial.

7.4.3 RQ3: Code quality (Complexity & LOC)

Table 7.6 shows the code quality metrics. Multi-agent systems did not consistently reduce complexity or the number of lines of code. For Gemini-2.5-pro (single agent), the code was more verbose (29.35 LOC) compared to GPT-5.1 (20.73 LOC), highlighting a trade-off between correctness and conciseness.

Table 7.6: Code Quality Metrics (Average)

Model	Complexity		LOC	
	Single	Multi	Single	Multi
GPT-5.1	4.60	4.77	20.73	19.94
Claude-4.5	4.25	4.29	22.41	22.70
Deepseek-chat-v3.1	4.56	4.73	19.87	19.79
Gemini-2.5-pro	6.32	4.53	29.35	23.33

Differences in cyclomatic complexity and LOC between single-agent and multi-agent systems are minor for the tested models. GPT-5.1 shows a slight LOC reduction in the multi-agent setup with a small increase in complexity, whereas Claude-4.5 and Deepseek-chat-v3.1 exhibit negligible changes. Gemini-2.5-pro produces more verbose single-agent code, and its multi-agent variant lowers LOC and complexity, but at a lower acceptance rate and on 149 completed problems. Overall, orchestration delivers limited structural improvements on these small algorithmic tasks; the main overhead remains latency and resource use.

We conducted paired two-sample t-tests to evaluate the hypotheses regarding code quality. Regarding cyclomatic complexity (H_{02a}), we observed no significant difference between the systems ($p > 0.05$ for all models), with negligible effect sizes (Cohen's $d < 0.2$). Thus, we **accept the null hypothesis** H_{02a}. Regarding Lines of Code (H_{02b}), while Gemini-2.5-pro showed a reduction in verbosity in the multi-agent setup, the aggregate difference across all problems was not statistically significant ($p = 0.14$), and differences for GPT-5.1, Claude-4.5, and Deepseek-chat-v3.1 were statistically negligible ($p > 0.05$). Therefore, we **accept the null hypothesis** H_{02b}. These results indicate that the Multi-Agent orchestration did not fundamentally alter the structural complexity or verbosity of the generated solutions for this specific domain.

Table 7.7: Latency and token usage (Average per Problem)

Model	Latency (s)		Tokens	
	Single	Multi	Single	Multi
GPT-5.1	6.47	38.46	905	5414
Claude-4.5	8.41	18.64	1077	6177
Deepseek-chat-v3.1	6.53	32.79	858	6611
Gemini-2.5-pro	122.08	274.17	15550	38955

In addition to the quality metrics, we also examined resource usage. Table 7.7 details this resource overhead. Multi-agent orchestration consistently increases both latency (End-to-end generation time) and token consumption. For instance, GPT-5.1 experiences a roughly 6× increase in latency (from 6.47s to 38.46s) and token usage (from 905 to 5414) when transitioning to the multi-agent system. Similar trends are observed for other models, with Gemini-2.5-pro showing the highest absolute costs. This reinforces the trade-off: while multi-agent systems can offer marginal stability gains or self-correction capabilities, they come at a cost that may not be justifiable for simple, well-defined tasks.

7.5 Discussion

We discuss study results and their impacts in this section.

7.5.1 Complexity threshold: when do we need Multi-Agent Systems?

To understand the practical limits of our findings, we must distinguish between the *algorithmic* tasks in this study and the *repository-scale* tasks in benchmarks like SWE-bench [154]. Comparing the acceptance rates directly (95% vs. 76%) is misleading because the tasks are different. Instead, we propose a *complexity threshold* — a specific point where the difficulty shifts from "solving logic" to "managing context." Table 7.8 outlines these structural differences.

Table 7.8: Complexity differences between this study and SWE-bench [154]

Feature	Algorithmic Scripting (This Study)	Repository Development (SWE-bench)
The Main Challenge	*Algorithmic Logic*: Solving a hard math or data problem within strict rules.	*Navigation*: Locating relevant code in repositories and understanding how it connects.
Information	*Complete*: The prompt contains full requirements.	*Incomplete*: The agent must search the codebase to discover requirements.
Risk of Error	*Isolated*: If it fails, only this script breaks.	*Systemic*: A mistake here can crash a totally different part of the software.
Conclusion	**Single-Agent is sufficient.**	**Multi-Agent is necessary.**

Our results confirm that modern LLMs have effectively saturated the algorithmic scripting domain. When the context is provided in the prompt, Single-Agent baselines achieve $> 95\%$ acceptance, rendering additional agents redundant. However, Table 7.9 reveals where this sufficiency ends. We present the performance of the same models on *SWE-bench Verified* [154], a benchmark that represents the "Repository Development" side. When the task shifts from self-contained algorithmic logic to multi-file navigation (spanning 12 repositories, such as Django and

scikit-learn), performance drops from our observed $\approx 95\%$ to the 68–76% range.

Table 7.9: Repository-Scale Tasks (SWE-bench Verified [154], leaderboard accessed November 22, 2025). The *SWE-bench Verified* dataset [171] contains **500 human-validated** tasks sampled from **12 popular Python repositories** (e.g., Django, scikit-learn, Flask). Unlike the self-contained algorithmic problems in this study, these tasks require navigating a large codebase, identifying dependencies, and integrating changes across multiple files.

Model	SWE-Agent Acceptance Rate [165]
GPT-5.1	76.00%
Gemini-2.5-pro	75.20%
Claude-4.5	70.60%
Deepseek-chat-v3.1	68.20%

This performance gap quantifies the *complexity threshold*. It suggests that the value of Multi-Agent orchestration is not universal, but conditional. For 'glue code' and self-contained scripts, Single-Agent is sufficient. The overhead of Multi-Agent systems (latency and token) is justified when the task crosses this threshold, specifically, when the difficulty lies in navigating the unknown rather than solving the known.

7.5.2 Error propagation in sequential agents

When agents operate in a chain-of-thought pattern where each agent builds upon the previous agent's output, a single error in the middle of the chain can cascade through all subsequent agents. Potentially, this kind of error may lead to a completely incorrect final solution regardless of how well the later agents perform their individual tasks.

Consider our four-agent architecture: Problem Analyzer $\rightarrow$ Solution Designer $\rightarrow$ Code Executor $\rightarrow$ Solution Verifier. If the Problem Analyzer misunderstands the problem requirements, this misunderstanding propagates to the Solution Designer, who creates a design based on incorrect assumptions. The Code Executor then implements this flawed design, and the Solution Verifier validates against the wrong criteria. In this scenario, the entire system's quality is bounded by the Problem Analyzer's initial error, regardless of how sophisticated the subsequent agents are.

Thus, error propagation could pose a scalability problem for sequential multi-agent systems used for code generation purposes. Mathematically, if each agent has a probability p of correctly processing its input, then a chain of n agents has an expected success probability of p^n. As the chain length approaches infinity, this probability approaches zero: $\lim_{n\to\infty} p^n = 0$ for any $p < 1$. This simple probability reasoning assumes independent pass/fail and binary correctness; real agent interactions can be more nuanced because outputs may be ambiguous, partial, or open to multiple valid interpretations (e.g., different sorting algorithms with similar behavior). Nonetheless, even highly reliable individual agents (e.g., $p = 0.95$) will see overall reliability decline as chains grow longer: a four-agent chain would have approximately 81% success ($0.95^4 \approx 0.81$). In contrast, a ten-agent chain would drop to about 60%.

Consequently, the Multi-Agent system in this study did not act as a quality booster but as a resource consumer. The error propagation theory suggests that as n (agents) increases, the probability of system success (p^n) decreases unless p is significantly higher than the single-agent baseline. Since the Single-Agent baseline (p) was already between 0.95 and 0.99, the Multi-Agent decomposition offered no statistical room for improvement, serving only to introduce latency and potential coordination failures.

7.5.3 Industrial applicability

Our scan of 27206 internal source files revealed over 96K instances of hash map operations and 11K string manipulations, confirming that algorithm-related skills are essential to enterprise implementation logic. These patterns often constitute the "glue code" in a functioning system. In an operational context, a string manipulation task often translates to parsing unstructured logs or sanitizing user inputs for a CI/CD pipeline. Similarly, hash map operations serve as the computational core for correlating alerts, aggregating metrics by region, or caching configuration states.

The value of using LLMs in this domain lies not just in speed but in cognitive offloading. During a high-pressure incident, a site reliability engineer must focus on system architecture and failure modes, rather than the syntax of a regular expression or the boilerplate of a map-reduce script. By reliably delegating these algorithmic tasks to an agent, engineers can reduce their cognitive load and potentially lower the Mean Time To Recovery [172]. However, this delegation requires human-in-the-loop review and confirmation: a "glue" script that fails silently or corrupts data is worse than no script at all.

7.6 Threats to Validity

We followed Wohlin et al.'s guidelines [31] to address four types of validity threats.

7.6.1 Internal Validity

We aimed to attribute differences to orchestration rather than hidden factors. We adjusted the prompts and used the same 150 tasks for all models, running single-agent before multi-agent within each model during off-peak hours to reduce volatility. Run order is a possible concern because the single-agent phase preceded the multi-agent phase in each replication. We opened new sessions, cleared the history, and reloaded prompts to block cross-run leakage; however, timing and background load can still vary and affect latency. Aggregation over three replications mitigates this risk but does not remove it. The task sample can also bias results within our scope. We drew from three LeetCode study plans under the inclusion criteria for self-contained, text-

only problems. Design choices are another source of threat. Our four-role chain is a defensible mapping of parse → plan → code → check; however, other role sets, tighter handoff schemas, or verifier gating could alter token use, generation time, and even quality.

7.6.2 Construct Validity

The acceptance rate is operationalized as LeetCode *accepted* on the server side. This maps to functional correctness on the LeetCode platform but does not cover all industrial test scenarios or non-functional requirements. Code quality is proxied by cyclomatic complexity and lines of code. These proxies capture structural simplicity and size, not readability, naming, architecture, or security. We report both metrics to avoid reliance on a single proxy and to enable replication. Latency is the end-to-end *generation time* from the first request to the final submission for each model. Token counts are reported as resource trace, not as monetary or energy cost. Therefore, our findings speak to orchestration at the capability levels of the four tested models; stronger or weaker models could shift absolute levels while preserving or altering the observed gaps.

7.6.3 External Validity

A threat to our findings lies in the rapid evolution of LLM capabilities. We evaluated four models available during the study period, but the AI landscape changes dramatically every few months. Future models may possess fundamentally different reasoning capabilities, which could shift the balance between single-agent and multi-agent approaches. Another concern is platform drift: LeetCode's test cases and problem statements can change over time. To mitigate this, we recorded problem identifiers and our tooling downloaded the statements. We release the problem list and timestamps so future studies can either reuse the problem set or re-evaluate it. Even with these steps, accepted results should be read as time-bound to the LeetCode state during our runs. Generalizability is also limited by context. Our tasks are self-contained algorithmic problems, and they do not represent other non-coding problem types.

7.6.4 Conclusion Validity

Our industrial scan was limited to source code repositories from one organization. The identified algorithmic patterns might differ in Python or Java codebases. However, the fundamental algorithmic needs (e.g., hashing and sorting) are language-agnostic.

7.7 Conclusion

We examined the prevalence of small algorithmic patterns in a set of 10 industrial repositories and investigated the effectiveness of LLMs in solving these algorithmic problems. We scanned 27k source code files and found that hash maps and string operations appear frequently, with sorting and tree or graph patterns also present but less frequently. This demonstrates that these simple yet non-trivial patterns are relevant in real-world code. We then ran a controlled experiment with four LLMs in solving 150 algorithmic problems, comparing single-agent and multi-agent setups. Both Single-Agent and Multi-Agent setups achieved acceptance rates above 95%, but the Multi-Agent chains introduced higher latency and token cost with only minor quality differences. We therefore conclude that for self-contained algorithmic scripting tasks, a single capable model provides comparable effectiveness but greater efficiency than the evaluated Multi-Agent setup.

An implication for practitioners is that, when a task has a clear oracle and can be addressed in a single prompt, we recommend focusing on a strong Single-Agent LLM solution. Our results indicate that, for self-contained algorithmic problems, Single-Agent setups consume fewer computational resources while providing outcomes comparable to those of the evaluated Multi-Agent setup. As such, these results may generalize to other problems of a similar nature. However, this recommendation should be limited to small, well-defined tasks for the four LLMs evaluated in this study (GPT-5.1, Claude-4.5, Deepseek-chat-v3.1, and Gemini-2.5-pro) as used during our study period.

While our results demonstrate single-agent sufficiency for self-contained algorithms, recent studies on SWE-bench [154] and FEA-bench [156] suggest that, for repository-level tasks that require complex decomposition, external tools, or navigation, multi-agent or tool-augmented solutions can be more effective than Single-Agent baselines. These bodies of work enable us to formulate the hypothesis that single-agent solutions may perform equally effectively as multi-agent solutions for small algorithmic problems. This hypothesis suggests that future research should investigate the attributes, aspects, or characteristics of problems where single-agent solutions are perceived as more suitable than multi-agent solutions.

References

[1] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech. “Generative ai”. In: *Business & Information Systems Engineering* 66.1 (2024), pp. 111–126.

[2] J. Prather, J. Leinonen, N. Kiesler, J. Gorson Benario, S. Lau, S. MacNeil, N. Norouzi, S. Opel, V. Pettit, L. Porter, et al. “Beyond the hype: A comprehensive review of current trends in generative AI research, teaching practices, and tools”. In: *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (2025), pp. 300–338.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[4] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. “Improving language understanding by generative pre-training”. In: *OpenAI* (2018).

[5] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. “Pre-trained models for natural language processing: A survey”. In: *Science China technological sciences* 63.10 (2020), pp. 1872–1897.

[6] T. Bazzan, B. Olojo, P. Majda, T. Kelly, M. Yilmaz, G. Marks, and P. M. Clarke. “Analysing the Role of Generative AI in Software Engineering-Results from an MLR”. In: *European Conference on Software Process Improvement*. Springer. 2024, pp. 163–180.

[7] L. Yu. “Paradigm shift on Coding Productivity Using GenAI”. In: *arXiv preprint arXiv:2504.18404* (2025).

[8] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. “Software Engineering for AI-Based Systems: A Survey”. In: *ACM Computing Surveys* 54.4 (2021), pp. 1–38. DOI: 10.1145/3474679.

[9] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. “Software Engineering for Machine Learning: A Case Study”. In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP ’19. IEEE, 2019, pp. 291–300. DOI: 10.1109/ICSE-SEIP.2019.00042.

[10] M. Sinha, S. Menon, and R. Sagar. “Llmops: Definitions, framework and best practices”. In: *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*. IEEE. 2024, pp. 1–6.

[11] H. Luo, Y. Liu, R. Zhang, J. Wang, G. Sun, D. Niyato, H. Yu, Z. Xiong, X. Wang, and X. Shen. “Toward edge general intelligence with multiple-large language model (Multi-LLM): architecture, trust, and orchestration”. In: *IEEE Transactions on Cognitive Communications and Networking* (2025).

[12] A. Aleti. “Software testing of generative ai systems: Challenges and opportunities”. In: *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE. 2023, pp. 4–14.

[13] T. Ramos, A. Dean, and D. McGregor. “AI-Augmented Software Engineering: Revolutionizing or Challenging Software Quality and Testing?” In: *Journal of Software: Evolution and Process* 37.2 (2025), e2741.

[14] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, and M. Du. “Explainability for Large Language Models: A Survey”. en. In: *ACM Transactions on Intelligent Systems and Technology* 15.2 (Apr. 2024), pp. 1–38. ISSN: 2157-6904, 2157-6912. (Visited on 11/22/2024).

[15] K. Kenthapadi, M. Sameki, and A. Taly. “Grounding and evaluation for large language models: Practical challenges and lessons learned (survey)”. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024, pp. 6523–6533.

[16] M. Simaremare and H. Edison. “The state of generative AI adoption from software practitioners’ perspective: An empirical study”. In: *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2024, pp. 106–113.

[17] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang. “Large language models for software engineering: A systematic literature review”. In: *ACM Transactions on Software Engineering and Methodology* (2023).

[18] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. “Hidden Technical Debt in Machine Learning Systems”. In: *Advances in Neural Information Processing Systems* 28 (2015), pp. 2503–2511.

[19] A. Martino, M. Iannelli, and C. Truong. “Knowledge injection to counter large language model (LLM) hallucination”. In: *European Semantic Web Conference*. Springer. 2023, pp. 182–185.

[20] L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek. “Measuring the quality of generative AI systems: Mapping metrics to quality characteristics - Snowballing literature review”. In: *Information and Software Technology* (2025), p. 107802. ISSN: 0950-5849.

[21] International Organization for Standardization. “ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models”. In: *International Organization for Standardization* (2011).

[22] ISO/IEC-25023. *Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): Measurement of System and Software Product Quality*. ISO, 2016.

[23] International Organization for Standardization. “ISO/IEC 25059:2023 Software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model for AI systems”. In: *International Organization for Standardization* (2023).

[24] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. “Bertscore: Evaluating text generation with bert”. In: *arXiv preprint arXiv:1904.09675* (2019).

[25] S. Min, K. Krishna, X. Lyu, M. Lewis, W.-t. Yih, P. W. Koh, M. Iyyer, L. Zettlemoyer, and H. Hajishirzi. “Factscore: Fine-grained atomic evaluation of factual precision in long form text generation”. In: *arXiv preprint arXiv:2305.14251* (2023).

[26] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma. “Codebleu: a method for automatic evaluation of code synthesis”. In: *arXiv preprint arXiv:2009.10297* (2020).

[27] L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek. “Evaluating the Quality of GenAI Applications in Software Engineering: A Multi-case Study”. In: *Empirical software engineering* 14.2 (2025), p. 131.

[28] D. B. Acharya, K. Kuppan, and B. Divya. “Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey”. In: *IEEe Access* (2025).

[29] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. “Evaluating large language models trained on code”. In: *arXiv preprint arXiv:2107.03374* (2021).

[30] C. Wohlin and P. Runeson. “Guiding the selection of research methodology in industry–academia collaboration in software engineering”. In: *Information and software technology* 140 (2021), p. 106678.

[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[32] P. Runeson and M. Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical software engineering* 14.2 (2009), p. 131.

[33] P. Runeson, E. Engström, and M.-A. Storey. “The design science paradigm as a frame for empirical software engineering”. In: *Contemporary empirical methods in software engineering* (2020), pp. 127–147.

[34] U. Sekaran and R. Bougie. *Research methods for business: A skill building approach*. john wiley & sons, 2016.

[35] C. R. Kothari. *Research methodology: Methods and techniques*. New Age International, 2004.

[36] J. S. Molléri, K. Petersen, and E. Mendes. “Survey guidelines in software engineering: An annotated review”. In: *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*. 2016, pp. 1–6.

[37] K. Lewin et al. “Action research and minority problems”. In: *Journal of social issues* 2.4 (1946), pp. 34–46.

[38] C. Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014, pp. 1–10.

[39] I. Khan, X. Zhang, M. Rehman, and R. Ali. “A literature survey and empirical study of meta-learning for classifier selection”. In: *IEEE Access* 8 (2020), pp. 10262–10281.

[40] S. Jalali and C. Wohlin. “Systematic literature studies: database searches vs. backward snowballing”. In: *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM ’12. Lund, Sweden: Association for Computing Machinery, 2012, pp. 29–38. ISBN: 9781450310567. DOI: 10.1145/2372251.2372257. URL: https://doi.org/10.1145/2372251.2372257.

[41] C. Robson. *Real world research*. Vol. 3. Wiley Chichester, 2011.

[42] B. G. Glaser, A. L. Strauss, and E. Strutzel. “The discovery of grounded theory; strategies for qualitative research”. In: *Nursing research* 17.4 (1968), p. 364.

[43] C. Wohlin and A. Aurum. “Towards a decision-making structure for selecting a research design in empirical software engineering”. In: *Empirical Software Engineering* 20 (2015), pp. 1427–1455.

[44] R. Berntsson Svensson, T. Gorschek, and B. Regnell. “Quality requirements in practice: An interview study in requirements engineering for embedded systems”. In: *Requirements Engineering: Foundation for Software Quality: 15th International Working Conference, REFSQ 2009 Amsterdam, The Netherlands, June 8-9, 2009 Proceedings 15*. Springer. 2009, pp. 218–232.

[45] R. K. Yin. *Case study research: Design and methods*. Vol. 5. sage, 2009.

[46] K. L. Gwet. “Computing inter-rater reliability and its variance in the presence of high agreement”. In: *British Journal of Mathematical and Statistical Psychology* 61.1 (2008), pp. 29–48.

[47] A. Agresti. *Categorical Data Analysis. 2013, Hoboken*.

[48] D. S. Cruzes and T. Dyba. “Recommended steps for thematic synthesis in software engineering”. In: *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2011, pp. 275–284.

[49] J. L. Fleiss. “Measuring nominal scale agreement among many raters.” In: *Psychological bulletin* 76.5 (1971), p. 378.

[50] N. Lavesson, V. Boeva, E. Tsiporkova, and P. Davidsson. “A method for evaluation of learning components”. In: *Automated software engineering* 21 (2014), pp. 41–63.

[51] Y. Li, S. Wang, H. Ding, and H. Chen. “Large language models in finance: A survey”. In: *Proceedings of the Fourth ACM International Conference on AI in Finance*. 2023, pp. 374–382.

[52] C. Wang, X. Liu, Y. Yue, X. Tang, T. Zhang, C. Jiayang, Y. Yao, W. Gao, X. Hu, Z. Qi, et al. “Survey on factuality in large language models: Knowledge, retrieval and domain-specificity”. In: *arXiv preprint arXiv:2310.07521* (2023).

[53] L. Chen, Y. Deng, Y. Bian, Z. Qin, B. Wu, T.-S. Chua, and K.-F. Wong. “Beyond Factuality: A Comprehensive Evaluation of Large Language Models as Knowledge Generators”. In: *arXiv preprint arXiv:2310.07289* (2023).

[54] M. F. A. Khan, M. Ramsdell, E. Falor, and H. Karimi. “Assessing the Promise and Pitfalls of ChatGPT for Automated Code Generation”. In: *arXiv preprint arXiv:2311.02640* (2023).

[55] X. Wu, R. Duan, and J. Ni. “Unveiling security, privacy, and ethical concerns of ChatGPT”. en. In: *Journal of Information and Intelligence* 2.2 (Mar. 2024), pp. 102–115. ISSN: 29497159. (Visited on 11/22/2024).

[56] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[57] J. W. Creswell and J. D. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[58] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert. “Ragas: Automated evaluation of retrieval augmented generation”. In: *arXiv preprint arXiv:2309.15217* (2023).

[59] R. Gupta, K. Nair, M. Mishra, B. Ibrahim, and S. Bhardwaj. “Adoption and impacts of generative artificial intelligence: Theoretical underpinnings and research agenda”. In: *International Journal of Information Management Data Insights* 4.1 (2024), p. 100232.

[60] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al. “A survey on evaluation of large language models”. In: *ACM Transactions on Intelligent Systems and Technology* 15.3 (2024), pp. 1–45.

[61] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. “Large language models: A survey”. In: *arXiv preprint arXiv:2402.06196* (2024).

[62] W. Kryściński, N. S. Keskar, B. McCann, C. Xiong, and R. Socher. “Neural text summarization: A critical evaluation”. In: *arXiv preprint arXiv:1908.08960* (2019).

[63] T. He, J. Chen, L. Ma, Z. Gui, F. Li, W. Shao, and Q. Wang. “ROUGE-C: A fully automated evaluation method for multi-document summarization”. In: *2008 IEEE International Conference on Granular Computing*. IEEE. 2008, pp. 269–274.

[64] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, S. Mirjalili, et al. “A survey on large language models: Applications, challenges, limitations, and practical usage”. In: *Authorea Preprints* (2023).

[65] D. Johnson, R. Goodman, J. Patrinely, C. Stone, E. Zimmerman, R. Donald, S. Chang, S. Berkowitz, A. Finn, E. Jahangir, et al. “Assessing the accuracy and reliability of AI-generated medical responses: an evaluation of the Chat-GPT model”. In: *Research square* (2023).

[66] P. Vaithilingam, T. Zhang, and E. L. Glassman. “Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models”. In: *Chi conference on human factors in computing systems extended abstracts*. 2022, pp. 1–7.

[67] Z. Li, W. Qiu, P. Ma, Y. Li, Y. Li, S. He, B. Jiang, S. Wang, and W. Gu. “An empirical study on large language models in accuracy and robustness under chinese industrial scenarios”. In: *arXiv preprint arXiv:2402.01723* (2024).

[68] L. Manduchi, K. Pandey, R. Bamler, R. Cotterell, S. Däubener, S. Fellenz, A. Fischer, T. Gärtner, M. Kirchler, M. Kloft, et al. “On the challenges and opportunities in generative ai”. In: *arXiv preprint arXiv:2403.00025* (2024).

[69] P. Thomas, S. Spielman, N. Craswell, and B. Mitra. “Large language models can accurately predict searcher preferences”. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2024, pp. 1930–1940.

[70] Y. Liu, J. Cao, C. Liu, K. Ding, and L. Jin. “Datasets for large language models: A comprehensive survey”. In: *arXiv preprint arXiv:2402.18041* (2024).

[71] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[72] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. “On the opportunities and risks of foundation models”. In: *arXiv preprint arXiv:2108.07258* (2021).

[73] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al. “A survey on evaluation of large language models”. In: *ACM Transactions on Intelligent Systems and Technology* (2023).

[74] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. “Measuring massive multitask language understanding”. In: *arXiv preprint arXiv:2009.03300* (2020).

[75] R. Rei, C. Stewart, A. C. Farinha, and A. Lavie. “COMET: A neural framework for MT evaluation”. In: *arXiv preprint arXiv:2009.09025* (2020).

[76] W. Zhao, M. Peyrard, F. Liu, Y. Gao, C. M. Meyer, and S. Eger. “MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance”. In: *arXiv preprint arXiv:1909.02622* (2019).

[77] L. Weidinger, J. Uesato, M. Rauh, C. Griffin, P.-S. Huang, J. Mellor, A. Glaese, M. Cheng, B. Balle, A. Kasirzadeh, et al. “Taxonomy of risks posed by language models”. In: *Proceedings of the 2022 ACM conference on fairness, accountability, and transparency*. 2022, pp. 214–229.

[78] B. Wang, Q. Xie, J. Pei, Z. Chen, P. Tiwari, Z. Li, and J. Fu. “Pre-trained language models in biomedical domain: A systematic survey”. In: *ACM Computing Surveys* 56.3 (2023), pp. 1–52.

[79] B. Dhingra, M. Faruqui, A. Parikh, M.-W. Chang, D. Das, and W. W. Cohen. “Handling divergent reference texts when evaluating table-to-text generation”. In: *arXiv preprint arXiv:1906.01081* (2019).

[80] M. K. Eddine, G. Shang, A. Tixier, and M. Vazirgiannis. “FrugalScore: Learning cheaper, lighter and faster evaluation metrics for automatic text generation”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 1305–1318.

[81] H. Le, Y. Wang, A. D. Gotmare, S. Savarese, and S. C. H. Hoi. “Coderl: Mastering code generation through pretrained models and deep reinforcement learning”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 21314–21328.

[82] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui. “AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation”. In: *arXiv preprint arXiv:2312.13010* (2023).

[83] A. F. Akyürek, E. Akyürek, L. Choshen, D. Wijaya, and J. Andreas. “Deductive closure training of language models for coherence, accuracy, and updatability”. In: *arXiv preprint arXiv:2401.08574* (2024).

[84] O. Honovich, R. Aharoni, J. Herzig, H. Taitelbaum, D. Kukliansy, V. Cohen, T. Scialom, I. Szpektor, A. Hassidim, and Y. Matias. “TRUE: Re-evaluating factual consistency evaluation”. In: *arXiv preprint arXiv:2204.04991* (2022).

[85] Y. Liu, H. Zhou, Z. Guo, E. Shareghi, I. Vulic, A. Korhonen, and N. Collier. “Aligning with human judgement: The role of pairwise preference in large language model evaluators”. In: *arXiv preprint arXiv:2403.16950* (2024).

[86] S. Shankar, J. Zamfirescu-Pereira, B. Hartmann, A. Parameswaran, and I. Arawjo. “Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences”. In: *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 2024, pp. 1–14.

[87] Q. Wang and J. M. Gayed. “Effectiveness of large language models in automated evaluation of argumentative essays: finetuning vs. zero-shot prompting”. In: *Computer Assisted Language Learning* (2024), pp. 1–29.

[88] M. Grandini, E. Bagli, and G. Visani. “Metrics for multi-class classification: an overview”. In: *arXiv preprint arXiv:2008.05756* (2020).

[89] Z. Rasheed, M. Waseem, K. Systä, and P. Abrahamsson. “Large language model evaluation via multi ai agents: Preliminary results”. In: *arXiv preprint arXiv:2404.01023* (2024).

[90] S. Banerjee and A. Lavie. “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments”. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[91] K. T. Htar, Y. Wang, J. Wu, G. Hattori, and A. Thida. “BERT-Based Dialogue Evaluation Methods with RUBER Framework”. In: *Advances in Artificial Intelligence: Selected Papers from the Annual Conference of Japanese Society of Artificial Intelligence (JSAI 2020) 34*. 2021, pp. 133–144.

[92] M. Ivarsson and T. Gorschek. “A method for evaluating rigor and industrial relevance of technology evaluations”. In: *Empirical Software Engineering* 16 (2011), pp. 365–395.

[93] C. Wohlin. “Second-generation systematic literature studies using snowballing”. In: *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. 2016, pp. 1–6.

[94] J. R. Landis and G. G. Koch. “The measurement of observer agreement for categorical data”. In: *biometrics* (1977), pp. 159–174.

[95] H. Li, Y. Hao, Y. Zhai, and Z. Qian. “Enhancing static analysis for practical bug detection: An llm-integrated approach”. In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1 (2024), pp. 474–499.

[96] R. Feldt, F. G. de Oliveira Neto, and R. Torkar. “Ways of applying artificial intelligence in software engineering”. In: *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. 2018, pp. 35–41.

[97] R. Ali, S. Lee, and T. C. Chung. “Accurate multi-criteria decision making methodology for recommending machine learning algorithm”. In: *Expert Systems with Applications* 71 (2017), pp. 257–278.

[98] R. Barbudo, S. Ventura, and J. R. Romero. “Eight years of AutoML: categorisation, review and trends”. In: *Knowledge and Information Systems* 65.12 (2023), pp. 5097–5149.

[99] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou. “Identifying, categorizing and mitigating threats to validity in software engineering secondary studies”. In: *Information and Software Technology* 106 (2019), pp. 201–230.

[100] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. “Palm: Scaling language modeling with pathways”. In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[101] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. “Training language models to follow instructions with human feedback”. In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[102] Z. Zhang, C. Chen, B. Liu, C. Liao, Z. Gong, H. Yu, J. Li, and R. Wang. “A survey on language models for code”. In: *arXiv preprint arXiv:2311.07989* (2023).

[103] K. Petersen and C. Wohlin. “The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study”. In: *Empirical Software Engineering* 15 (2010), pp. 654–693.

[104] I. D. Raji, A. Smart, R. N. White, M. Mitchell, T. Gebru, B. Hutchinson, J. Smith-Loud, D. Theron, and P. Barnes. “Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing”. In: *Proceedings of the 2020 conference on fairness, accountability, and transparency*. 2020, pp. 33–44.

[105] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. “Model cards for model reporting”. In: *Proceedings of the conference on fairness, accountability, and transparency*. 2019, pp. 220–229.

[106] A. Celikyilmaz, E. Clark, and J. Gao. “Evaluation of text generation: A survey”. In: *arXiv preprint arXiv:2006.14799* (2020).

[107] M. Esposito, X. Li, S. Moreschini, N. Ahmad, T. Cerny, K. Vaidhyanathan, V. Lenarduzzi, and D. Taibi. “Generative AI for Software Architecture. Applications, Trends, Challenges, and Future Directions”. In: *arXiv preprint arXiv:2503.13310* (2025).

[108] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. “Bleu: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[109] C.-Y. Lin. “Rouge: A package for automatic evaluation of summaries”. In: *Text summarization branches out*. 2004, pp. 74–81.

[110] T. Kynkäänniemi, T. Karras, S. Laine, J. Lehtinen, and T. Aila. “Improved precision and recall metric for assessing generative models”. In: *Advances in neural information processing systems* 32 (2019).

[111] M. Patton Quinn. *Qualitative research & evaluation methods*. 2002.

[112] K. Petersen and C. Wohlin. “Context in industrial software engineering research”. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2009, pp. 401–404.

[113] P. Lago, P. Runeson, Q. Song, and R. Verdecchia. “Threats to Validity in Software Engineering–hypocritical paper section or essential analysis?” In: *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2024, pp. 314–324.

[114] J. Allow and I. Neustadt. “UML 2 and the Unified Process”. In: *Reading: Addison Wesley* (1999).

[115] W. Xu, C. Napoles, E. Pavlick, Q. Chen, and C. Callison-Burch. “Optimizing statistical machine translation for text simplification”. In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 401–415.

[116] C.-k. Lo. “YiSi-a unified semantic MT quality evaluation and estimation metric for languages with different levels of available resources”. In: *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*. 2019, pp. 507–513.

[117] A. R. Fabbri, W. Kryściński, B. McCann, C. Xiong, R. Socher, and D. Radev. "Summeval: Re-evaluating summarization evaluation". In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 391–409.

[118] A. Wang, K. Cho, and M. Lewis. "Asking and answering questions to evaluate the factual consistency of summaries". In: *arXiv preprint arXiv:2004.04228* (2020).

[119] T. Sellam, D. Das, and A. P. Parikh. "BLEURT: Learning robust metrics for text generation". In: *arXiv preprint arXiv:2004.04696* (2020).

[120] L. Regenwetter, A. Srivastava, D. Gutfreund, and F. Ahmed. "Beyond statistical similarity: Rethinking metrics for deep generative models in engineering design". In: *Computer-Aided Design* 165 (2023), p. 103609.

[121] D. Jiang, M. Ku, T. Li, Y. Ni, S. Sun, R. Fan, and W. Chen. "GenAI Arena: An Open Evaluation Platform for Generative Models". In: *arXiv preprint arXiv:2406.04485* (2024).

[122] M. Coutinho, L. Marques, A. Santos, M. Dahia, C. França, and R. de Souza Santos. "The role of generative ai in software development productivity: A pilot case study". In: *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 2024, pp. 131–138.

[123] E. Klotins, T. Gorschek, and M. Wilson. "Continuous software engineering: Introducing an industry readiness model". In: *IEEE Software* 40.4 (2023), pp. 77–87.

[124] E. D. Canedo and G. A. Santos. "Factors affecting software development productivity: An empirical study". In: *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. 2019, pp. 307–316.

[125] B. Omidvar Tehrani, I. M, and A. Anubhai. "Evaluating Human-AI Partnership for LLM-based Code Migration". en. In: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. Honolulu HI USA: ACM, May 2024, pp. 1–8. ISBN: 9798400703317. DOI: 10.1145/3613905.3650896. URL: https://dl.acm.org/doi/10.1145/3613905.3650896 (visited on 02/15/2025).

[126] P. Heng, K. Yongsiriwit, and P. Chaisiriprasert. "Comparing the Effectiveness of Generative AI for Learning and Developing Flutter Application". In: *2024 8th International Conference on Information Technology (InCIT)*. Chonburi, Thailand: IEEE, Nov. 2024, pp. 746–751. ISBN: 9798350366303. DOI: 10.1109/InCIT63192.2024.10810490. URL: https://ieeexplore.ieee.org/document/10810490/ (visited on 02/15/2025).

[127] C. A. Gonçalves and C. T. Gonçalves. “Assessment on the Effectiveness of GitHub Copilot as a Code Assistance Tool: An Empirical Study”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 14969 LNAI (2025). Cited by: 0, pp. 27–38. DOI: 10.1007/978-3-031-73503-5_3. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85210453151&doi=10.1007%2f978-3-031-73503-5_3&partnerID=40&md5=5995203c16d900179f692d2da2a7b458.

[128] V. Corso, L. Mariani, D. Micucci, and O. Riganelli. “Assessing AI-Based Code Assistants in Method Generation Tasks”. en. In: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. Lisbon Portugal: ACM, Apr. 2024, pp. 380–381. ISBN: 9798400705021. DOI: 10.1145/3639478.3643122. URL: https://dl.acm.org/doi/10.1145/3639478.3643122 (visited on 02/15/2025).

[129] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers. “Using an llm to help with code understanding”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–13.

[130] Z. Zheng, K. Ning, Q. Zhong, J. Chen, W. Chen, L. Guo, W. Wang, and Y. Wang. “Towards an understanding of large language models in software engineering tasks”. In: *Empirical Software Engineering* 30.2 (2025), p. 50.

[131] J. Zhang, M. Harman, L. Ma, and Y. Liu. “Machine Learning Testing: Survey, Landscapes and Horizons”. In: *IEEE Transactions on Software Engineering* 46.10 (2020), pp. 1070–1093. DOI: 10.1109/TSE.2019.2962027.

[132] P. d. O. Santos, A. C. Figueiredo, P. Nuno Moura, B. Diirr, A. C. Alvim, and R. P. D. Santos. “Impacts of the usage of generative artificial intelligence on software development process”. In: *Proceedings of the 20th Brazilian Symposium on Information Systems*. 2024, pp. 1–9.

[133] C. T. Ungureanu and A. E. Amironesei. “Legal issues concerning Generative AI technologies”. In: *Eastern Journal of European Studies* 45 (2023).

[134] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella. “Testing Machine Learning Based Systems: A Systematic Mapping”. In: *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM ’20. ACM, 2020, pp. 1–12. DOI: 10.1145/3382494.3410465.

[135] J. Park and S. Choo. “Generative AI prompt engineering for educators: Practical strategies”. In: *Journal of Special Education Technology* (2024), p. 01626434241298954.

[136] A. Donvir, S. Panyam, G. Paliwal, and P. Gujar. “The role of generative AI tools in application development: A comprehensive review of current technologies and practices”. In: *2024 International Conference on Engineering Management of Communication and Technology (EMCTECH)*. IEEE. 2024, pp. 1–9.

[137] J. Wang and Y. Chen. “A review on code generation with llms: Application and evaluation”. In: *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*. IEEE. 2023, pp. 284–289.

[138] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk. “Using Deep Learning to Generate Complete Log Statements”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. ACM, 2022, pp. 883–895. DOI: 10.1145/3510003.3510079.

[139] J. Li, T. Tang, W. X. Zhao, J.-Y. Nie, and J.-R. Wen. “Pre-trained language models for text generation: A survey”. In: *ACM Computing Surveys* 56.9 (2024), pp. 1–39.

[140] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. “The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction”. In: *IEEE Big Data* (2017), pp. 1123–1132. DOI: 10.1109/BigData.2017.8258038.

[141] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. “Beyond Accuracy: Behavioral Testing of NLP Models with CheckList”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 4902–4912.

[142] S. Barke, M. B. James, and N. Polikarpova. “Grounded Copilot: How Programmers Interact with Code-Generating Models”. In: *Proceedings of the 44th ACM/IEEE International Conference on Software Engineering*. ICSE ’22. IEEE, 2022, pp. 319–331. DOI: 10.1145/3510003.3510141.

[143] U.-e. Habiba, M. Haug, J. Bogner, and S. Wagner. “How mature is requirements engineering for AI-based systems? A systematic mapping study on practices, challenges, and future research directions”. In: *Requirements Engineering* 29.4 (2024), pp. 567–600.

[144] K. Kenthapadi, H. Lakkaraju, and N. Rajani. “Generative ai meets responsible ai: Practical challenges and opportunities”. In: *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*. 2023, pp. 5805–5806.

[145] Grafana Labs. *Grafana: Open-Source Visualization and Dashboards*. https://grafana.com/grafana. Accessed 2025-08-01. 2025.

[146] Prometheus. *Prometheus: Monitoring for your systems and services*. https://prometheus.io. Accessed 2025-08-01. 2025.

[147] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Hoboken, NJ: Wiley, 2012. ISBN: 978-1-118-10435-4.

[148] GitHub. *GitHub Copilot*. Accessed: 2025-11-23. 2021. URL: https://github.com/features/copilot.

[149] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, et al. "A survey on large language model based autonomous agents". In: *Frontiers of Computer Science* 18.6 (2024), p. 186345.

[150] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, et al. "AutoGen: Enabling next-gen LLM applications via multi-agent conversation". In: *arXiv preprint arXiv:2308.08155* (2023).

[151] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, et al. "MetaGPT: Meta programming for a multi-agent collaborative framework". In: *The Twelfth International Conference on Learning Representations*. 2023, pp. 1–15.

[152] C. Qian, X. Cong, C. Yang, W. Chen, Y. Su, J. Xu, Z. Liu, and M. Sun. "Communicative agents for software development". In: *arXiv preprint arXiv:2307.07924* (2023).

[153] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang. "Large language models for software engineering: A systematic literature review". In: *ACM Transactions on Software Engineering and Methodology* 33.8 (2024), pp. 1–79.

[154] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" In: *arXiv preprint arXiv:2310.06770* (2023).

[155] S. Joel, J. Wu, and F. Fard. "A survey on llm-based code generation for low-resource and domain-specific programming languages". In: *ACM Transactions on Software Engineering and Methodology* (2024).

[156] G. Li et al. "FEA-Bench: A Benchmark for Feature Engineering Agents". In: *arXiv preprint arXiv:2401.00000* (2024).

[157] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al. "Competition-level code generation with alphacode". In: *Science* 378.6624 (2022), pp. 1092–1097.

[158] J. Yang, C. E. Jimenez, A. Wettig, K. Lunt, S. Yao, and K. Narasimhan. "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering". In: *arXiv preprint arXiv:2405.15793* (2024).

[159] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury. “Autocoderover: Autonomous program improvement”. In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2024, pp. 1592–1604.

[160] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui. “AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation”. In: *arXiv preprint arXiv:2312.13010* (2024).

[161] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. “Reflexion: Language agents with verbal reinforcement learning”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 8634–8652.

[162] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu. “Understanding and detecting real-world performance bugs”. In: *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*. 2012, pp. 77–88.

[163] J. Cui, R. Zhang, F. Zhou, R. Li, Y. Song, and E. Gehringer. “How much effort do you need to expend on a technical interview? a study of leetcode problem solving statistics”. In: *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE. 2024, pp. 1–10.

[164] F. Mehralian, R. Shar, J. R. Rae, and A. Hashemi. “CodeAlignBench: Assessing Code Generation Models on Developer-Preferred Code Adjustments”. In: *arXiv preprint arXiv:2510.27565* (2025).

[165] SWE-bench Team. *SWE-bench Leaderboard*. `https://www.swebench.com/index.html`. Accessed: 2025-11-23. 2025.

[166] M. A. A. Mamun, C. Berger, and J. Hansson. “Correlations of software code metrics: an empirical study”. In: *Proceedings of the 27th international workshop on software measurement and 12th international conference on software process and product measurement*. 2017, pp. 255–266.

[167] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner. “Software engineering for AI-based systems: a survey”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.2 (2022), pp. 1–59.

[168] H.-M. Heyn, E. Knauss, A. P. Muhammad, O. Eriksson, J. Linder, P. Subbiah, S. K. Pradhan, and S. Tungal. “Requirement engineering challenges for ai-intense systems development”. In: *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE. 2021, pp. 89–96.

[169] X. Gu, M. Chen, Y. Lin, Y. Hu, H. Zhang, C. Wan, Z. Wei, Y. Xu, and J. Wang. “On the effectiveness of large language models in domain-specific code generation”. In: *ACM Transactions on Software Engineering and Methodology* 34.3 (2025), pp. 1–22.

[170] L. Myllyaho, M. Raatikainen, T. Männistö, J. K. Nurminen, and T. Mikkonen. “On misbehaviour and fault tolerance in machine learning systems”. In: *Journal of Systems and Software* 183 (2022), p. 111096.

[171] Princeton NLP. *SWE-bench Verified*. `https://huggingface.co/datasets/princeton-nlp/SWE-bench_Verified`. Accessed: 2025-11-23. 2024.

[172] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy. *Site reliability engineering: how Google runs production systems*. O’Reilly Media, Inc., 2016.

Generative Artificial Intelligence (GenAI) is being rapidly adopted in software engineering, introducing a paradigm shift toward human-AI co-creation. However, the non-deterministic, probabilistic, and often black-box nature of GenAI models presents challenges for traditional software quality assurance. Conventional verification and validation techniques are insufficient to handle outputs that are neither predictably correct nor incorrect, but rather stochastically plausible. This discrepancy creates an urgent need for practical processes, metrics, and new governance frameworks to evaluate and manage the quality of GenAI systems in industrial environments.

This thesis examines how industrial organizations adopt GenAI, identify metrics, and evaluate system qualities in alignment with ISO quality standards. Case studies were employed to explore real-world adoption processes, identify contextspecific industrial metrics, and uncover practical insights within organizations. A snowballing literature review was conducted to systematically identify, categorize, and synthesize academic metrics for evaluating the output of GenAI systems. Finally, a controlled experiment was designed to quantitatively test the efficiency (e.g., E2E generation time) and effectiveness (e.g., accuracy) of GenAI agent choices.

The main contributions of this thesis are a synthesized actionable model and framework grounded in both industrial practice and quality standards. The first contribution is a four-stage adoption model, denoted as the IMRM model (Innovate → considerations, Measure → metrics, Realize → values, Manage → improvements) that integrates early-stage risk assessment (e.g., legal, security, and licensing) and quality evaluation throughout the GenAI adoption and usage.

The second contribution presents a detailed framework that connects risks and metrics to concrete decision support, justifying the business value (e.g., quality gates) and technical trade-offs of GenAI solutions. The third contribution provides a structured mapping of GenAI quality to ISO/IEC 25010, 25023, and 25059 characteristics, attempting to ground practical evaluation needs within a standardized vocabulary.

This thesis concludes that a structured quality evaluation process, which prioritizes risks and context, is a valuable approach intended to support building the business confidence required to leverage GenAI for efficient and effective software engineering in industry.

Printed by Media-Tryck, Lund 2026 NORDIC SWAN ECOLABEL 3041 0903

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2026:01
ISSN: 1653-2090
ISBN: 978-91-7295-518-9