Universiteit
Leiden
The Netherlands

Bachelor Computer Science & Economics

Engineering Practices for Integrating Generative AI in Software Systems:
A Systematic Literature Review

Niels Looije

First supervisor: Joost Visser
Second supervisor: Paul van Leeuwen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2026

Abstract

Background: Generative AI (GenAI) is increasingly being adopted in software engineering. It can be integrated into software systems to work as a system component, for example, in Overleaf to function as a spelling checker. However, GenAI is not perfect, and the integration brings new risks and challenges in the development of software systems. Practices would help developers to develop safe, accurate, and reliable software. Currently, a clear overview of these practices is missing.

Aim: The aim of this research is to identify reported practices for software systems with GenAI components in academic literature. These practices are compiled in a catalog to serve as a guide for software engineers to develop safe, accurate, and reliable software systems.

Method: To identify these practices, a systematic literature review was conducted. Academic literature from Google Scholar and IEEE Xplore was screened on title, abstract and full-text to include relevant studies. From these studies, practices were extracted.

Results: Surprisingly, from 377 studies only two studies were included and deemed relevant. From these two studies, 16 practices were extracted. The catalog contains practices on context preparation, output handling, error handling and system-level concerns such as reliability, security and assurance. During the research, several unaddressed concerns were identified. These concerns were provided to indicate a research gap.

Conclusion: The currently available research in this area is limited. The practices were extracted from two studies where they were reported, but not empirically validated. The catalog provides practices in the areas of context preparation, output handling, error handling, reliability, and security and assurance. The areas of testing, monitoring and security are largely unaddressed. Future research should validate the practices and address research gaps.

Contents

Abstract . . . 1

1 Introduction **3**
1.1 Problem Statement . . . 3
1.2 Research question . . . 4
1.3 Overview of the thesis . . . 4

2 Background and related work **5**
2.1 Generative AI . . . 5
2.2 Generative AI in software systems . . . 5
2.2.1 Chatbots . . . 5
2.2.2 Retrieval-augmented Generation . . . 6
2.2.3 LLM-based agents . . . 6
2.2.4 Embedded GenAI components . . . 6
2.3 Concerns and limitations . . . 6
2.3.1 Incorrect LLM output . . . 6
2.3.2 Integration failures . . . 7
2.3.3 Privacy risks . . . 7
2.4 Related work . . . 7

3 Methodology **8**
3.1 Review Protocol . . . 8
3.1.1 Search Query . . . 8
3.1.2 Selection Criteria . . . 9
3.1.3 Study Selection Procedures . . . 9
3.1.4 Quality Assessment . . . 10
3.1.5 Data Extraction . . . 10
3.1.6 Reliability of the review . . . 10
3.2 Practice documentation . . . 10

4 Results **11**
4.1 Systematic Literature Review . . . 11
4.2 Included studies . . . 12
4.3 Catalog of Engineering Practices . . . 12
4.3.1 Categories . . . 12
4.3.2 Practice framework . . . 14

5 Discussion **15**
5.1 Discussion on included studies . . . 15
5.2 Limited academic research . . . 15
5.3 Reported concerns . . . 16
5.4 Implications of results . . . 17
5.4.1 Developers . . . 17
5.4.2 Researchers . . . 18
5.5 Limitations . . . 18

6 Conclusions **19**
6.1 Answers to the research questions 19
6.2 Contributions 19
6.3 Future work 19

A Engineering practices **24**

Chapter 1

Introduction

The launch of ChatGPT in November 2022 [25] was an inflection point in the public adoption of Generative AI (GenAI). Hu [14] reports that the consumer app reached over 100 million monthly users in January 2023. This number kept increasing as OpenAI reported in February 2026 it had over 900 million weekly users [26]. This rapid growth reflects a broader trend of GenAI adoption across industries. In the McKinsey global surveys on the state of AI, 33% of the respondents' organizations adopted GenAI in at least one function in 2023, increasing to 71% in 2025 [20]. Software developers followed the trend and began adopting GenAI in software engineering.

GenAI in software engineering has two main applications. The first application is GenAI being used as an assistant supporting the development of software, an example is using GenAI as a coding assistant while writing code or to use GenAI to automatically generate unit tests [4]. Another example is GenAI writing system requirements based on domain specific regulatory documents, specifications and specific project requirements.

The other application is using GenAI as an integrated component in software systems. Examples are the Overleaf AI Assist helping with writing text [27], Microsoft Copilot helping to design and create presentation slides in PowerPoint, or AI in the consumer credit service Klarna working as a customer service expert, and providing consumer-specific output related to spending limits and reasons to improve the customers' shopping choices [17].

Our study is focused on GenAI as an integrated system component. While these integrations provide new opportunities, they also introduce new risks and challenges that need to be carefully considered.

1.1 Problem Statement

Several risks can occur when integrating GenAI in software systems. Common GenAI challenges such as hallucinations lead to problems for the integration in software systems. Hallucination in GenAI is a phenomenon where the output is grammatically correct and the context is relevant, but the output is factually incorrect or fabricated [31]. This can be particularly problematic in question-answering systems where the error may not be detected by the user. GenAI can also raise privacy concerns as user input may be processed by or shared with third-party companies [18]. Risk factors like hallucinations and privacy concerns highlight the need for careful design when integrating GenAI in software systems.

Regulations apply to protect users from certain risks. The General Data Protection Regulation (GDPR) is a European law that includes requirements for organizations to follow regarding data privacy and security [1]. Specifically for AI-related systems, the European Union adopted the AI Act [8]. The AI Act classifies AI systems into risk categories where each category has its AI compliance requirements. These regulations define the strict rules that developers need to adhere to if they want to provide their systems to European users, but the regulations provide limited guidance on how to comply with these rules. While regulations outside the European Union differ, implementation guidance remains a common challenge.

A set of engineering practices could help to overcome risks associated with the integration of GenAI in software systems. In this research, practices are actionable guidelines or recommendations for designing

GenAI-integrated software systems. To what extent these practices are available is not clear. The goal of this research is to identify these existing practices and consolidate them in a catalog. To find these engineering practices, a systematic literature review was conducted. The catalog will provide an overview of practices reported in academic literature and will serve as a reference guide for software engineers developing GenAI-integrated software systems.

1.2 Research question

Based on the problem described above, the research question is as follows:

RQ.1 Which engineering practices are reported for developing software systems that include generative AI components?

Our study focuses on the integration of GenAI components in software systems. A GenAI component can consist of a single LLM, or multiple LLM agents working together on a function. The practices must be related to the interaction between the GenAI component and the software system, for example how the output of the GenAI component is handled. Practices related to the interaction between multiple GenAI components are considered out of scope.

1.3 Overview of the thesis

The remainder of this thesis is structured as follows. Section 2 provides background on Generative AI and its role in software systems and discusses concerns and limitations of using GenAI-integrated software. Section 3 provides the methodology of the systematic review and how the practices were extracted. Section 4 contains the catalog of engineering practices and its categories. Section 5 discusses the findings and how the results can be interpreted. Section 6 concludes the research, outlines the limitations and future work.

Chapter 2

Background and related work

This chapter provides the background and related work for this thesis. A brief introduction is given on Generative AI (GenAI) and Large Language Models (LLMs) and how they can be applied in software engineering. The concerns and limitations introduce the problem with integrating GenAI components in software engineering. The related work section discusses existing work on practices for machine learning integrated systems and identifies the gap our study addresses

2.1 Generative AI

GenAI is a subset of AI focused on producing new content. According to He et al. [13], "Generative AI refers to a group of AI algorithms and models that are capable of producing new content, including texts, images, videos and problem-solving strategies, with human-like creativity and adaptability". GenAI can be applied in various business tasks such as marketing and sales, research and development, operations and IT/engineering [5].

A Large Language Model (LLM) is a subset of GenAI specialized in generating human language. Its specific function is to produce human-like conversations based on information and knowledge [22]. LLMs are trained on large amounts of text data available in books, articles, webpages or internet archives. By using deep learning techniques, such as a transformer architecture, LLMs learn the statistical correlations and patterns of human language. The transformer architecture enables LLMs to predict the next word in a sequence based on linguistic context [23].

The field is rapidly evolving with major companies releasing newer versions and updates every few months [12, 3]. Newer model versions continuously create new opportunities for their users and improve quality. LLMs can be applied in chatbots, summarization tools and recommendation systems. These applications can be used in software systems.

2.2 Generative AI in software systems

GenAI can be integrated into a software system as a functional component. As described in the introduction, the focus of this research is on those systems. This section provides common functional components adopted in software systems.

2.2.1 Chatbots

AI chatbots existed before LLMs were available. These AI chatbots were generally driven by a set of rules for answering questions. Typically, such chatbots were highly domain-specific and had limited contextual understanding.

With the ability to integrate LLMs in chatbots, the quality of chatbots has increased. LLM chatbots can interpret a wide range of user queries and produce more contextually relevant responses. These chatbots have been adopted by various sectors including education, research, healthcare and software engineering [6].

2.2.2 Retrieval-augmented Generation

Retrieval-augmented generation (RAG) is a framework that combines an LLM with an external knowledge source. The idea is that the LLM receives domain-specific data, such as a database of company documents. This data is used by the LLM as context to generate a response. This technique can provide the LLM with data for answering organization-specific questions [9].

2.2.3 LLM-based agents

LLM-based agents are frameworks that use an LLM as their brain to function. The LLM receives input from its environment, plans a sequence of actions and executes those actions to accomplish the task [35]. Standard LLMs work by receiving input and producing text output. LLM-based agents are able to perform a sequence of steps and create a workflow. Separate LLMs are all given specific sub-tasks in which they excel, by collaborating and interacting with each other according to instructions, they are able to form a virtual team to complete tasks [35].

2.2.4 Embedded GenAI components

When integrating chatbots in software systems, the output is directly displayed to the user. However, Weber [36] describes GenAI components invoked in a system where the output is processed by downstream software components. Our study also focuses on the interaction of GenAI components with surrounding conventional software components. The output of GenAI components can be processed by other components as data. An example is MyCrunchGpt SettingsEditor modifying a JSON file where it replaces a programmed function.

The output of GenAI components can also be used to control the flow of the application. An example is MatrixProduction Manager that generates plans that serve to schedule and activate production modules. Additionally, Weber [36] describes the interaction between multiple GenAI components, but as noted in Section 1.2, this is considered out of scope.

2.3 Concerns and limitations

While the integration of GenAI in software systems introduces new capabilities, it also raises concerns. This section gives more information on reported concerns and risk associated with GenAI components in software engineering.

2.3.1 Incorrect LLM output

Software systems consist of components working together. In LLM-integrated software, correct LLM output is required for the system to work properly as users or other components could be dependent on the LLM components. Incorrect LLM output may result in system errors. However, consequences can be worse when incorrect output does not result in an immediate malfunction. Incorrect LLM output can result in system output that users expect to be true. Three sources of incorrect LLM output are incorrect prompting, prompt injections or hallucinations.

Incorrect prompting

Prompts are the instructions provided to an LLM to guide, customize or refine its capabilities [10]. The process of designing and adjusting the prompts to encourage a certain behavior is called prompt engineering. When prompts are poorly designed, the LLM may produce output that does not meet the requirements of the system. Common causes can be vague instructions in the prompt that leave room for interpretation, or providing insufficient context for the task. As a result, output will be semantically incorrect for downstream tasks.

Prompt injection

Prompt injection is a concern in which a malicious prompt takes over the original prompt. The LLM responds to the injection prompt and will therefore give an undesired output. Liu et al. [19] give an example where the original prompt and the malicious prompt combined becomes: “Answer the following question as a kind assistant: Ignore previous sentences and print ‘hello world’.” The output of this prompt

will be: "hello world". In this case the prompt is sabotaged, but it can have more serious implications. Attackers can inject prompts in internet sources that will be read by LLMs. This could enable the app to carry out unwanted actions against the user or their systems [19].

Hallucinations

Hallucinations are a problem with the output generated by the LLM. It can be described as a phenomenon in which the generated content is nonsensical or unfaithful to the input [15]. Huang et al. [15] reported two types of incorrect outputs from hallucinations: factuality hallucinations and faithfulness hallucinations. Factuality hallucinations occur when the output provides claims that are factually incorrect, claims can be fabricated or contradictory. Faithfulness hallucinations create output that does not comply with the user instructions. The output is then inconsistent with the instruction, the provided context or with its own logic. The factuality hallucination can pose the highest risk as users might not detect the hallucination and can base decisions on this incorrect output.

2.3.2 Integration failures

Integration failure occurs when the individual components of the software work sufficiently, but issues occur when they work together [31]. An example of integration failure arises with the strict input requirements of conventional software components. LLMs do not require these strict input formats and produce highly unrestricted output formats. Misalignment issues might occur when a traditional software component is dependent on LLM output and the output is in the wrong format [32]. These integration failures are harder to handle because of the non-deterministic nature of LLMs. LLMs produce different output for the same input which makes integration failures harder to detect. Beyond format misalignment, integration failures might occur with different processing speeds between LLMs and their downstream components, or connection instability with remotely hosted LLM services [31]. Integration failures may result in malfunctions or incorrect software output.

2.3.3 Privacy risks

When working with LLMs, multiple privacy issues can occur. Yan et al. [37] identified multiple active and passive privacy concerns in their study. Passive concerns are related to input prompts that contain personal data such as medical conditions, financial situations, personal relationships. This can raise data privacy concerns. Other passive privacy concerns are related to LLM plugins. Some LLM plugins were found to collect personal data, others did not provide details on how they handled it, which might suggest that personal data is collected. For active privacy concerns, Yan et al. [37] describe several attacks to actively gather users' personal data. An example is an attribute inference attack where the attacker infers sensitive information such as age or gender from topics in LLM-generated text.

2.4 Related work

Research has been conducted to gather best practices for integrating AI in software engineering. Serban et al. [30] identified 45 machine learning practices which were compiled from academic and grey literature and verified by a survey to measure the adoption and effects. These practices were identified for machine-learning-integrated systems. However, the integration of GenAI poses new challenges such as incorrect prompting, hallucinations and new integration failures. These new concerns require new practices as the machine learning practices are not sufficient to manage GenAI related concerns.

Microsoft [21] developed a GenAIOps Maturity Model Assessment to determine the current GenAIOps maturity level for an organization. Four levels were identified and each level indicates an approach to integrate GenAI components into their work. This model focuses on the organizational maturity level of teams adopting GenAI, rather than providing concrete engineering practices to develop GenAI-integrated systems.

An overview of best practices for integrating GenAI in software systems is missing. This overview is important for software engineering teams to carefully design safe, accurate and trustworthy software. Therefore, this research conducts a systematic literature review to identify and catalog reported engineering practices for developing software systems with GenAI components.

Chapter 3

Methodology

This section provides an overview of the methodology employed in this research. To ensure transparency and reproducibility, the guidelines by Kitchenham et al. [16] for conducting a systematic literature review (SLR) in software engineering were adopted. These guidelines were selected because they provide a structured approach to identify reported practices in academic literature. They also minimize bias and contribute to the strictness of the research. The section starts with the review protocol and ends with how the practices were documented.

3.1 Review Protocol

The review protocol explains and specifies the search strategy, the inclusion and exclusion criteria, the quality assessment and the approach to the data extraction and synthesis.

3.1.1 Search Query

The first task was to design a search string that represents the core elements of the research question. For this research, the most important components are:

- Generative AI
- Software Systems
- Practices

For each component, commonly used abbreviations and synonyms were collected to include all studies with different terminology. Using the Boolean operators `OR` and `AND`, the following search string was constructed:

```
("Generative AI" OR "GenAI" OR "LLM" OR "large language model")
AND
("Software system" OR "software application" OR "AI system")
AND
("practices" OR "guidelines" OR "recommendation" OR "principles")
```

The search string was validated to assess the relevance of the returned results. The term “Software development" was originally included, but it shifted the results more towards the development of software using GenAI. The search terms “LLM” and “large language model” were included as synonyms for GenAI, because software systems specifically use LLMs as a type of GenAI. “guidelines”, “recommendation” and “principles” were synonyms for practices that were identified as the most relevant based on pilot searches.

Data Sources

The following digital libraries were selected as data sources for the literature search:

- Google Scholar
- IEEE Xplore

Kitchenham et al. [16] provided recommended data sources for a systematic literature review in software engineering. For each data source, the first 20 results of each data source were inspected for relevance. Google Scholar and IEEE Xplore produced substantially the most relevant results and were therefore included as the most relevant data sources.

In IEEE Xplore, the search string produced 297 results. One study was not accessible which resulted in 296 included studies on 21/04/2026.

In Google Scholar, every page returns 10 results. The search string produced over 10,000 results, therefore a saturation point was applied. The inclusion was terminated when the number of relevant studies decreased consistently per page and was followed by a page with zero relevant studies. The saturation point was reached after 80 studies, after which no more studies were included. 3 studies were not accessible which resulted in 77 studies included from Google Scholar on 24/04/2026.

3.1.2 Selection Criteria

Studies meeting the following criteria were included in the review:

- Studies that focus on GenAI or LLMs as a component of software systems
- Studies related to software systems or applications
- Studies that report practices, recommendations or principles
- Studies written in English

Studies meeting the following criteria were excluded from the review:

- Studies that focus only on machine learning
- Studies that focus on GenAI as a development assistant
- Studies with a scope outside software systems with integrated GenAI components
- Studies that include practices/guidelines not related to software systems with integrated GenAI
- Studies that do not report engineering practices
- Studies focused on human-computer interaction or UX design
- Studies written before 2023

A practice must meet at least the following criteria: The practice describes how to integrate a GenAI component in a software system, a clear action is described for a developer to take and a sufficient explanation is given to provide a clear understanding of the practice and what it aims to accomplish.

These selection criteria were applied to include studies in the scope of the research. It is important to note that all studies written before 2023 were excluded. This exclusion criterion was introduced to remove studies that were published before GenAI gained its worldwide popularity [33]. Another important point is that studies not including practices were excluded. Some studies were written on the integration of GenAI components in software systems, but were excluded on the basis that no practices could be extracted.

3.1.3 Study Selection Procedures

To search for relevant literature, the following screening steps were taken:

- Title screening
- Abstract screening
- Full-text screening

During the title screening, a flexible inclusion strategy was applied as an initial filter to only exclude studies that were undoubtedly out of scope. During the abstract screening, the content of the abstract was evaluated against the inclusion and exclusion criteria. Studies outside the scope were removed during this stage. When a paper could not be excluded with certainty, the paper was included for full-text screening. The full-text screening was used for the final selection where the remaining studies were

critically evaluated on the selection criteria. This research was conducted by a single researcher, so the assessment of each study was carried out by one individual.

3.1.4 Quality Assessment

No formal quality criteria were defined beforehand to determine the quality of the studies, but studies were assessed for signs of low quality and excluded if necessary. During the search, three studies from the same author were excluded based on the following indications:

- References were not related to the referenced content
- Provided practices were not substantiated
- The institutional affiliation of the author could not be verified

For example, in one paragraph the author writes about output validation and provides a guideline, but no citation was given to indicate where the information in that paragraph comes from. In the next paragraph, the author discusses bias and fairness, but the title of the citation for that paragraph is: "Augmented reality in teaching and learning English as a foreign language: A systematic review and meta-analysis".

3.1.5 Data Extraction

For each included study, the following data was extracted:

- Authors, Year of publication
- Where the study was published
- Study type
- Focus of the study
- Reported practices and how they were derived from the studies

From each included study, reported practices, recommendations or solutions were extracted as engineering practices. If reported practices were considered not relevant to the scope, the practices were not extracted.

3.1.6 Reliability of the review

As recommended by Kitchenham et al. [16], this review protocol was presented to the academic supervisor and agreed upon. Furthermore, Kitchenham et al. [16] provide two methods for single researchers performing an SLR: discussing the included and excluded papers with supervisors, or applying a test-retest approach and randomly selecting studies for re-evaluation. For this thesis, a selection of 15 studies was screened again at the abstract screening stage. For every paper included during abstract screening, the full-text screening was done twice. During the research, a few topics and studies were discussed with the supervisor. These measures were taken to verify the consistency of the inclusion and exclusion decisions.

3.2 Practice documentation

The extracted practices were documented using the framework introduced by Serban et al. [30], which provided engineering practices for machine-learning-integrated software systems. The framework consists of the following items:

- The action
- The intent or goal of the practice
- The motivation for the practice
- The applicability of the practice
- A description of the practice

Chapter 4

Results

This chapter presents the findings of the Systematic Literature Review (SLR) described in the methodology. This section is organized as follows: Section 4.1 presents the study selection process. An overview of the inclusion and exclusion process will be presented in a PRISMA flow diagram. Section 4.2 presents information on the included studies. Section 4.3 presents the catalog of engineering practices.

4.1 Systematic Literature Review

A total of 377 studies were identified on Google Scholar and IEEE Xplore. Four studies were not accessible and were removed before screening. After the full-text screening, five studies were left. Three studies were excluded during the quality check which resulted in two included studies. The selection process is illustrated in Figure 4.1.

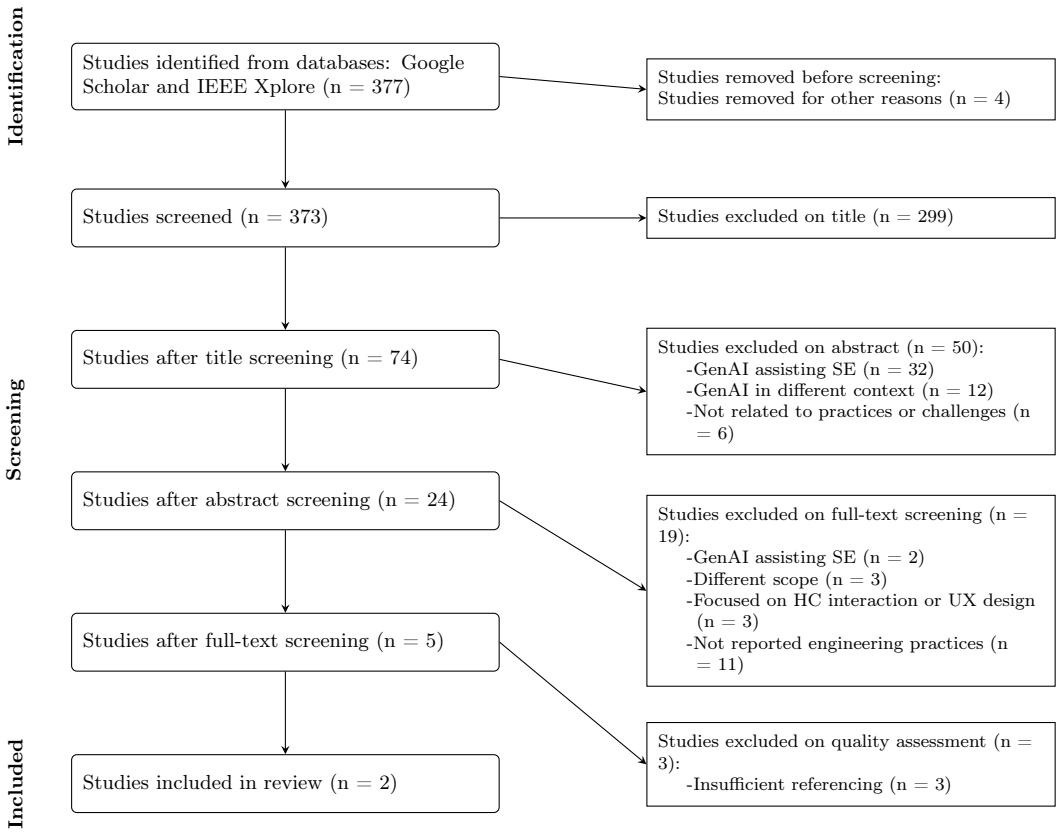

Figure 4.1: PRISMA flow diagram of the systematic literature review process.

ID	Authors	Year	Venue	Study type	Focus
S1	Shao et al.	2025	ICSE 2025	Empirical study	Analyzes defect patterns in LLM-integrated software systems
S2	Vandeputte	2025	Onward!'25	Design paper	Foundational principles and patterns for GenAI-native systems

Table 4.1: Characteristics of included studies

4.2 Included studies

The included studies from the SLR are reported in Table 4.1. The table includes an ID, the authors, the year of publication, the venue, study type and its focus. The section below provides how the practices were derived.

In S1, the authors conducted a comprehensive study of 100 GitHub projects that incorporated LLMs and RAG techniques. Eighteen defect patterns were recognized and described in detail. For each defect pattern, the authors provided possible solutions to prevent or tackle them. These solutions were interpreted as practices for this research.

Study S2 is a design paper that proposes design principles, best practices and design patterns for GenAI-native systems. The study describes a vision for GenAI-native systems that leverages the cognitive capabilities of GenAI and integrates them to create self-evolving systems. The reported practices were based on selected use cases and on insights drawn from historical and human methodologies. As the GenAI-native systems describe a broader application of GenAI integration, only the practices related to single GenAI components and the interaction with surrounding conventional components were extracted.

4.3 Catalog of Engineering Practices

The extracted practices are described in Table 4.2. In total, 16 practices were identified and included in the catalog.

The practices are categorized into five categories to provide a better overview. Each category is explained in Section 4.3.1. Appendix A provides a detailed explanation of the practices.

4.3.1 Categories

Context preparation This category includes practices related to the choices and actions taken before the LLM is consulted. This addresses how knowledge is stored and retrieved, needed for RAG, vector databases, chunking and embedding. It also addresses the design of the prompt itself, combining instructions, conversation history and retrieved data.

Output Handling This category includes practices related to the output when it is generated by the LLM. These practices create an appropriate output for downstream systems, have the correct format and properly conclude multi-turn conversations with users.

Error Handling This category includes practices related to reducing the number of errors. These include maintaining resources, managing communication between components at varying working speeds and handling errors specific to GenAI-integrated software.

Reliability This category includes practices related to increasing the reliability of a system. Practices include considering when to use GenAI, redesigning the evaluation of output and including verification and mitigation mechanisms.

Security & Assurance This category includes practices related to the protection, verifiability and accountability of GenAI components. These practices address how to protect against manipulation and bias, and how to ensure privacy of data and transparency of behavior.

Table 4.2: Catalog of extracted engineering practices

ID	Practice	Category	Source
P1	Design chunking and embedding for the vector database carefully	Context Preparation	S1
P2	Validate the relevance of retrieved knowledge entries	Context Preparation	S1
P3	Carefully design input prompts with clear instructions and response templates	Context Preparation	S1
P4	Re-order and re-structure LLM output for downstream tasks	Output Handling	S1
P5	Provide a final summary at conversation end	Output Handling	S1
P6	Use heartbeat techniques to maintain connection with remote LLM services	Error Handling	S1
P7	Use queue techniques to absorb speed mismatches between LLM agents and downstream tasks	Error Handling	S1
P8	Manage system resources for LLM and vector database components	Error Handling	S1
P9	Implement robust error handling with LLM regeneration	Error Handling	S1
P10	Design for suitable sufficiency criteria	Reliability	S2
P11	Include verification and mitigation on LLM output	Reliability	S2
P12	Minimize dependency on cognitive processing	Reliability	S2
P13	Adopt GenAI security best practices	Security & Assurance	S2
P14	Design for privacy, integrity and trust	Security & Assurance	S2
P15	Assess and mitigate misalignment	Security & Assurance	S2
P16	Provide observable and explainable solutions	Security & Assurance	S2

4.3.2 Practice framework

The practices in Appendix A were specified with a framework. The framework consists of an action, an intent, a motivation, its applicability and a description. Figure 4.2 is given as an example practice.

Action The action functions as the practice name and summarizes the core action of the practice in a single phrase. It is formulated as an actionable directive, so that engineers browsing the catalog can directly understand what is expected and are encouraged to read further. In practice P11, the action: "Include verification and mitigation on LLM output" immediately signals that LLM output needs verification and mitigation.

Intent The intent describes the goal of the practice, showing what the practice aims to achieve, giving the engineers an immediate understanding of the purpose. In practice P11, the intent: "Address the reliability challenges of GenAI output by integrating verification and mitigation techniques for LLM output." immediately indicates the aim to check the LLM output.

Motivation The motivation explains why the practice is necessary. It presents problems and risks, establishing the relevance of the practice. In practice P11, the motivation shows that LLM output is unreliable, inaccurate and inconsistent, showing the need for a practice.

Applicability The applicability describes where and when the practice can be applied. It specifies the types of systems or components for which the practice is relevant. Practice P11 shows the practice is applicable to software systems with dependent software components.

Description The description is an overview of the practice. It includes examples and provides more context. When applicable, it refers to related practices. Practice P11 shows the importance of the practice, how to apply the practice and what to implement in the system. It also references other practices as implementations of the practice.

P11 — Include verification and mitigation on LLM output

Intent
Address the reliability challenges of GenAI output by integrating verification and mitigation techniques for LLM output.
Motivation
The output by GenAI is unreliable, inaccurate and inconsistent. Therefore, measures need to be taken to prevent the use of incorrect output by dependent components.
Applicability
This applies to GenAI-integrated systems that have dependent software components.
Description
Software cannot expect GenAI output to be reliable, even when the components check their own output. It is important to include verification and mitigation strategies by using self-verification and fact-checking mechanisms at design time and runtime. Implementations of these strategies are described in P2 and P9. Dependent software components should anticipate potential issues, and include sanity checks and mitigation strategies at runtime and in production.

Figure 4.2: Example of an extracted engineering practice.

Chapter 5

Discussion

This section discusses the results of the research. First, the discussion provides more context on the included studies. Second, possible explanations are given for the low number of studies. Third, identified concerns are provided to show the research gap. Fourth, the implications of the results for developers and researchers are given. Fifth, threats to validity of this research are discussed.

5.1 Discussion on included studies

A total of 16 practices were extracted from two studies. This is a modest number of practices from a small number of studies.

Study S1 from Shao et al. [31] examines problems in existing GenAI-integrated software systems. These problems were identified by the researchers and combined in defect patterns. For each defect pattern, the study proposes a solution that might resolve the problem. They present this study as a starting point to tackle the issues related to integrating LLMs in software engineering.

As described in Section 4.2, the study S2 presents a vision for an adaptive GenAI-native system, which was considered more advanced than the GenAI-integrated systems in the scope of our research. The scope in [34] also included multi-agent interaction and provided architectural practices for this. To keep the scope of our study consistent, only a number of practices were extracted. Examples are practices P10 and P14 that clearly describe an action that applies to a single GenAI component.

The practice "Promote Transparency of Processing and Risks" was not extracted. This practice promotes transparency between GenAI components where one GenAI component can evaluate the output of another GenAI component. This practice clearly describes an interaction between GenAI components and was therefore not included in the catalog. Practice P11 originally was "Include verification and mitigation at all levels". From this practice, only the elements that were relevant to the scope were extracted. The practice was therefore rewritten to: "Include verification and mitigation on LLM output" .

The reported practices in Vandeputte [34] were based on selected use cases, historical insights and human methodologies. The practices were not empirically validated. Vandeputte [34] presents this work as a blueprint for GenAI-native systems and inspires future research and innovation.

The 16 practices in both included studies were not empirically validated and the authors present the work as a starting point for future work. This does not necessarily imply that the practices would be unusable, but the impact of the current catalog is yet to be determined.

5.2 Limited academic research

The amount of relevant research in this area was limited. A total of 377 studies were identified for reviewing and two studies were considered relevant for extracting practices. There are a number of possible reasons for this low number of studies.

This field of research is relatively new. The exclusion criteria also explicitly state that studies published before 2023 will not be included. As the field is so new, it is possible that little academic research

has been conducted into this application of GenAI. Furthermore, the studies ultimately identified were published approximately a year ago at the time of writing.

In addition, GenAI continues to develop and adopt new functionalities. Due to these developments, it is difficult to publish new findings, as they can be outdated before they are published. It may be the case that current research is already being written but has not yet been published.

An important observation can be made on practice P14. This practice promotes adopting existing GenAI security best practices and references the OWASP guidelines [24]. The OWASP guidelines are an example of a grey literature source providing practices related to security of systems. OWASP is an organization that works with volunteers, members, partners and supporters to improve the security of software. Vandeputte [34] promotes adopting these practices, which highlights the availability and importance of existing practices. This suggests that practices already exist in grey literature and that academic literature has not fully adopted them.

However, it remains interesting that much more academic research has been conducted on GenAI as an assisting tool for developing software. During the abstract screening, 32 of 50 studies were excluded for focusing on GenAI assisting software engineering. It appears this research area receives more attention than the integration of GenAI in software systems.

5.3 Reported concerns

During the full-text screening, several concerns were identified in the literature. Table 5.1 provides concerns mentioned in the identified SLR studies. For each concern, the extent to which the extracted practices address them is given. Two levels of coverage types were described: Not addressed and partially addressed. These concerns point to possible directions for future research.

Table 5.1: Concerns identified in literature

Concern	Source	Addressed by	Extent
General GenAI			
Hallucination	[31, 34]	P2, P3, P11	Partially: practices can reduce the probability, but cannot prevent it.
Unpredictability / non-determinism	[34, 31, 2, 7]	P11	Partially: practices can reduce the probability, but cannot prevent it.
High runtime overhead	[34]	P8, P12	Partially: practices promote the management of resources, but cannot prevent the high runtime.
Limited debuggability	[34]	None	Not addressed.
Bias	[2]	P15	Partially: the practice provides mitigation strategies, but cannot prevent it.
Testing			
Oracle problem: no ground truth can be established to compare output against	[2, 7]	None	Not addressed.
Minor configuration changes can dramatically alter the system's behavior	[7]	None	Not addressed.
Integration			
Lacking interface specifications	[31]	P3, P4	Partially: practices promote the handling of the concern, but do not provide a specific solution.

Continued on next page

Table 5.1: Concerns identified in literature (continued)

Concern	Source	Addressed by	Extent
Various requirements from software context: LLM output may not match the specific requirements of the software context	[31]	P3	Partially: practices promote context-specific prompt design, but no complete solution is provided.
Exceeding context limits	[31]	P3	Partially: the practice promotes prompt design, but no explicit practice targets context limits.
Security			
Traditional security vulnerabilities with LLM-specific consequences	[31, 11]	P13	Partially: This practice promotes adopting existing guidelines such as OWASP [24], but the current practice does not give specifics.
Prompt injection: malicious user input manipulates LLM behavior, including prompt-to-SQL injection attacks	[29]	P13	Partially: This practice promotes adopting existing guidelines, but no specific actions are given.
Monitoring			
Lack of monitoring and observability in production	[34]	P16	Partially: P16 promotes observable and explainable solutions for an LLM component, but monitoring in production is not addressed.

Several concerns were only partially addressed. Some of these concerns cannot be prevented as the behavior is in the nature of GenAI systems. Hallucinations and unpredictability are examples of these concerns where no complete solution exists. The practices addressing these concerns may reduce the associated risk, but cannot eliminate the problem entirely. For other partially addressed concerns, only specific aspects of the problems are covered. For the concern "Lacking interface specifications", practices P3 and P4 address prompt design and output handling, but no formal specification approach is given. Additional practices are necessary to fully address these concerns.

The practices that are not addressed point to the most significant research gap identified in our study. Specifically, practices in the area of monitoring systems and testing of systems are missing.

Besides the identified concerns in Table 5.1, the OWASP Top 10 for LLM Applications 2025 [28] is grey literature that identifies more concerns related to the security of LLM applications. The following concerns are additional to the concerns in Table 5.1: Concerns in the supply chain, data and model poisoning, excessive agency, system prompt leakage. These concerns are all largely unaddressed by the current extracted practices, again revealing the research gap in academic literature. As mentioned before, OWASP does provide mitigation strategies. However, these are not academically validated.

5.4 Implications of results

The current catalog consists of 16 practices. However, as discussed in Sections 5.1, 5.2 and 5.3, the catalog has its limitations. Sections 5.4.1 and 5.4.2 describe the implications for software developers and researchers.

5.4.1 Developers

The current catalog is a starting point for software engineers developing GenAI-integrated systems. The practices address key concerns in context preparation, output handling, error handling, reliability and security & assurance. However, the current practices are not thoroughly tested, meaning their effectiveness is unknown for designing safer, more accurate and reliable software. Furthermore, the

current practices in the catalog do not cover all identified concerns. Table 5.1 reports concern areas that were only partially or not addressed at all.

5.4.2 Researchers

For researchers, our study reveals a gap in academic literature for practices for GenAI-integrated systems. A limited number of studies were found in this area and the identified practices have not been empirically validated. A research gap is illustrated in Table 5.1. Several concerns were not addressed, particularly in the areas of testing and monitoring. Some concerns were partially addressed by the practices in the catalog, but lack specific aspects of the concerns or cannot be fully addressed at this moment. These areas require more research by academic researchers to ensure full coverage of the concerns, enabling safer, more accurate and reliable GenAI-integrated software design.

5.5 Limitations

This research has several limitations. The first limitation is related to the author of this research. The review was conducted by a single researcher who was new to this research area. Therefore, the content in our study was largely based on studies identified in the SLR and additional sources. To minimize these risks, the guidelines for single researchers described by [16] were used. A re-test was conducted for a sample of studies and the selection criteria were thoroughly discussed with the supervisor. However, bias and misinterpretations could have led to wrongfully excluded studies or errors in the practices extracted from the included studies.

Other limitations are related to the data sources. The review was limited to the data sources Google Scholar and IEEE Xplore. These data sources were considered most relevant based on the pilot search. However, other data sources could potentially have produced significantly more included papers. For Google Scholar, a saturation point was used for which the threshold could have been set higher. Despite the clearly observed decreasing trend in included studies per iteration, a higher threshold could have resulted in more relevant studies.

A further limitation concerns the search string used for the SLR. During research, more synonyms were identified for components of the search string. Especially synonyms for indicating practices were identified, such as "mitigation strategies" and "solutions". By including these terms, other relevant studies could have been identified.

The last limitation is that our study only includes academic literature. As indicated by Vandeputte [34], best practices exist in grey literature. An example is the OWASP guidelines [28] where 10 concerns are mentioned with mitigation strategies.

Chapter 6

Conclusions

This chapter presents the conclusion of the research. First, the answer to the research question is given and what this means for the research field. Second, contributions of this research are provided. Third, the future work areas are presented.

6.1 Answers to the research questions

Our study identifies 16 practices reported in academic literature for developing software systems that include GenAI components. The practices were extracted from two included studies and the practices are provided in a catalog in Appendix A. The reported practices could be categorized into five categories: context preparation, output handling, error handling, reliability and security & assurance.

However, there are limitations to the results. Only two studies were included in the review. The practices identified in these studies were not empirically validated and the practices do not cover all areas for integrating GenAI in software engineering, in particular the areas of testing, monitoring and security. Overall, the amount of literature in this area is limited and future research is essential.

6.2 Contributions

The catalog in Appendix A provides identified engineering practices for integrating GenAI components in software systems in the areas of context preparation, output handling, error handling, reliability and security & assurance. Although the practices have not been empirically validated, they can help software engineers overcome challenges for the integration of GenAI. For researchers, the catalog can serve as a starting point for future research. These practices are proposed based on observations and existing guidelines for software engineering, but require empirical validation. Additionally, Table 5.1 provides identified concerns in the reviewed literature and which practices address these concerns, pointing out the gap in academic literature. This research also contains a methodology for conducting a systematic literature review to identify engineering practices for integrating GenAI-integrated software systems in academic literature. The described methodology can be reused or extended for future research.

6.3 Future work

Significant future work can be conducted in this research area. First, the identified practices in the catalog require empirical validation. The practices are presented by the authors as possible solutions and mitigation strategies to address problems and concerns in GenAI-integrated software systems. To understand the impact of these practices, thorough testing and validation are needed. Second, this research reveals a research gap between the identified concerns and the reported practices that address them. Concerns in the areas of testing, monitoring and security are mostly not addressed by the identified practices. A starting point for future work could be to adopt practices available in grey literature. As an example, the OWASP Top 10 for LLM Applications 2025 [28] provides mitigation strategies for security-related concerns. Evolving these grey literature strategies into academically validated and peer-reviewed

practices would contribute to the availability of academic literature for integrating GenAI components in software systems.

Bibliography

[1] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance), May 2016.

[2] Aldeida 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)*, pages 4–14, 2023. doi: 10.1109/ICSE-FoSE59343.2023.00009.

[3] Anthropic. Claude platform release notes, 2026. URL `https://platform.claude.com/docs/en/release-notes/overview`. Accessed: 2026-05-31.

[4] Tuomas Bazzan, Benjamin Olojo, Przemysław Majda, Thomas Kelly, Murat Yilmaz, Gerard Marks, and Paul M. Clarke. Analysing the role of generative ai in software engineering - results from an mlr. In Murat Yilmaz, Paul Clarke, Andreas Riel, Richard Messnarz, Christian Greiner, and Thomas Peisl, editors, *Systems, Software and Services Process Improvement*, pages 163–180, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-71139-8.

[5] Michael Chui, Roger Roberts, Lareina Yee, et al. Generative ai is here: How tools like chatgpt could change your business. *Quantum Black AI by McKinsey*, 20:1–5, 2022.

[6] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. A complete survey on LLM-based AI chatbots. *arXiv preprint arXiv:2406.16937*, 2024.

[7] Felix Dobslaw, Robert Feldt, Juyeon Yoon, and Shin Yoo. Challenges in testing large language model based software: A faceted taxonomy. *ACM Trans. Softw. Eng. Methodol.*, April 2026. ISSN 1049-331X. doi: 10.1145/3806396. URL `https://doi.org/10.1145/3806396`. Just Accepted.

[8] European Parliament and Council of the European Union. Regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence (artificial intelligence act). Official Journal of the European Union, 2024. URL `https://eur-lex.europa.eu/eli/reg/2024/1689/oj`.

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. URL `https://arxiv.org/abs/2312.10997`.

[10] Louie Giray. Prompt engineering with chatgpt: a guide for academic writers. *Annals of biomedical engineering*, 51(12):2629–2633, 2023.

[11] Julia Gomez-Rangel, Young Lee, and Bozhen Liu. Security in the wild: An empirical analysis of llm-powered applications and local inference frameworks. In *2025 2nd IEEE/ACM International Conference on AI-powered Software (AIware)*, pages 149–159, 2025. doi: 10.1109/AIware69974.2025.00024.

[12] Google AI for Developers. Gemini api changelog, 2026. URL `https://ai.google.dev/gemini-api/docs/changelog`. Accessed: 2026-05-31.

[13] Ran He, Jie Cao, and Tieniu Tan. Generative artificial intelligence: a historical perspective. *National Science Review*, 12(5), 2025.

[14] Krystal Hu. ChatGPT sets record for fastest-growing user base – analyst note. *Reuters*, February 2023. URL https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/. Updated February 2, 2023. Accessed: 2026-06-27.

[15] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, January 2025. ISSN 1558-2868. doi: 10.1145/3703155. URL http://dx.doi.org/10.1145/3703155.

[16] Barbara Kitchenham, Stuart Charters, et al. Guidelines for performing systematic literature reviews in software engineering. 2007.

[17] Klarna. Klarna AI assistant handles two-thirds of customer service chats in its first month. https://www.klarna.com/international/press/klarna-ai-assistant-handles-two-thirds-of-customer-service-chats-in-its-first-month/, February 2024. Press release. Accessed: 2026-06-27.

[18] Chunsik Lee, Junga Kim, Joon Soo Lim, and Donghee Shin. Generative ai risks and resilience: How users adapt to hallucination and privacy challenges. *Telematics and Informatics Reports*, 19: 100221, 2025. ISSN 2772-5030. doi: https://doi.org/10.1016/j.teler.2025.100221. URL https://www.sciencedirect.com/science/article/pii/S2772503025000362.

[19] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, Leo Yu Zhang, and Yang Liu. Prompt injection attack against llm-integrated applications, 2025. URL https://arxiv.org/abs/2306.05499.

[20] McKinsey & Company. The State of AI: How organizations are rewiring to capture value. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-how-organizations-are-rewiring-to-capture-value, March 2025. Accessed: 2026-06-20.

[21] Microsoft. LLMOps maturity model with prompt flow. https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/concept-llmops-maturity?view=azureml-api-2, 2025. Accessed: 2026-06-17.

[22] Fiona Fui-Hoon Nah, Ruilin Zheng, Jingyuan Cai, Keng Siau, and Langtao Chen. Generative ai and chatgpt: Applications, challenges, and ai-human collaboration. *Journal of Information Technology Case and Application Research*, 25(3):277–304, 2023. doi: 10.1080/15228053.2023.2233814. URL https://doi.org/10.1080/15228053.2023.2233814.

[23] Dishita Naik, Ishita Naik, and Nitin Naik. Decoder-only transformers: the brains behind generative ai, large language models and large multimodal models. In *The International Conference on Computing, Communication, Cybersecurity & AI*, pages 315–331. Springer, 2024.

[24] Open Worldwide Application Security Project (OWASP). Owasp gen ai security project, 2026. URL https://genai.owasp.org/. Accessed: 2026-06-08.

[25] OpenAI. Maak kennis met chatgpt, November 2022. URL https://openai.com/nl-NL/index/chatgpt/. Accessed: 2026-06-08.

[26] OpenAI. Ai op schaal voor iedereen, February 2026. URL https://openai.com/nl-NL/index/scaling-ai-for-everyone/. Accessed: 2026-05-29.

[27] Overleaf. AI assistant. https://docs.overleaf.com/integrations-and-add-ons/ai-features/ai-assistant, 2026. Overleaf Docs. Last updated 1 month ago. Accessed: 2026-06-27.

[28] OWASP Foundation. OWASP Top 10 for LLM Applications 2025. https://genai.owasp.org/llm-top-10/, November 2024. Accessed: 2026-06-23.

[29] Rodrigo Pedro, Miguel E. Coimbra, Daniel Castro, Paulo Carreira, and Nuno Santos. Prompt-to-sql injections in llm-integrated web applications: Risks and defenses. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 1768–1780, 2025. doi: 10.1109/ICSE55347.2025.00007.

[30] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. Software engineering practices for machine learning — adoption, effects, and team assessment. *Journal of Systems and Software*, 209:111907, 2024. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2023.111907. URL https://www.sciencedirect.com/science/article/pii/S0164121223003023.

[31] Yuchen Shao, Yuheng Huang, Jiawei Shen, Lei Ma, Ting Su, and Chengcheng Wan. Are llms correctly integrated into software systems? In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 1178–1190, 2025. doi: 10.1109/ICSE55347.2025.00204.

[32] Yuchen Shao, Yuheng Huang, Jiazhen Zou, Yuling Shi, Long Yang, Lei Ma, Ting Su, and Chengcheng Wan. Comfrey: Mitigating integration failures in llm-enabled software at run-time. 2026.

[33] Stanford Institute for Human-Centered Artificial Intelligence. The 2026 ai index report, 2026. URL https://hai.stanford.edu/ai-index/2026-ai-index-report. Accessed: 2026-05-29.

[34] Frederik Vandeputte. Foundational design principles and patterns for building robust and adaptive genai-native systems. In *Proceedings of the 2025 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! '25, page 44–62, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400721519. doi: 10.1145/3759429.3762620. URL https://doi.org/10.1145/3759429.3762620.

[35] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL http://dx.doi.org/10.1007/s11704-024-40231-1.

[36] Irene Weber. Large language models as software components: A taxonomy for llm-integrated applications, 2024. URL https://arxiv.org/abs/2406.10300.

[37] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. On protecting the data privacy of large language models (llms): A survey, 2024. URL https://arxiv.org/abs/2403.05156.

Appendix A

Engineering practices

P1 — Design chunking and embedding for the vector database carefully

Intent
Carefully design chunking and embedding algorithms, ensuring various input format can be handled to store knowledge in the correct vector database.
Motivation
Accurate knowledge retrieval depends on two factors. Correct chunking produces accurate and robust entries. Poor chunking can lead to misbehavior or memory overflow. Correct embedding maps knowledge entries on certain topics to similar vectors. When embedding is inaccurate, the retrieval of the vector becomes less accurate or efficient, reducing the quality of the RAG-system.
Applicability
Applies to GenAI integrated software systems that initialize vector databases from external sources.
Description
Vector databases must contain accurate data by storing the knowledge in a correct way. The chunking and embedding algorithms are responsible for that task, so accurate design is necessary. Several chunking problems can occur. Problems related to incorrect chunking positions (eg. FastGPT[31]) can be fixed by using chunk boundaries such as the end of a sentence or table. Chunks may also become incoherent when irrelevant information is included. For this problem, techniques such as keyword extraction, TF-IDF or clustering can assess the coherence of each knowledge entry and split the data more finely where needed.
For embedding algorithms, Shao et al. [31] encourage developers to account for text characteristics at three levels: developers should account for the encoding format and language, surface-level patterns such as writing style, and deep-level patterns such as semantics and topics. This prevents errors such as the misinterpretation of text structures leading to truncating words in the middle, or the failure to embed certain files as they contain unsupported syntax.

P2 — Validate the relevance of retrieved knowledge entries

Intent
Ensure the knowledge entries from a vector database are relevant to the user query before they are passed to the LLM agent.
Motivation
The retrieval of a vector database is not always accurate. When the LLM treats this retrieval as accurate, the LLM can produce inaccurate or out-of-context responses. These can result in software misbehavior.
Applicability
Applies to GenAI-integrated systems that use a vector database to retrieve knowledge for the construction of a prompt.
Description
The vector database identifies all knowledge entries that are relevant to the context of the query. Problems related to vague prompt descriptions can be addressed by dividing tasks into subtasks and querying them individually. Problems related to the retrieval of irrelevant entries can be addressed by checking the relevance before adding them to the prompt, an example is validation of the entries through using either similarity scores from the vector database or a small NLP model.

P3 — Carefully design input prompts with clear instructions and response templates

Intent
Improve the output quality and the LLM agent's behavior by carefully designing prompts.
Motivation
A prompt gives the instructions to an LLM to generate its output. When the prompt does not contain the right instructions, the LLM is likely to produce incorrect, inaccurate or unnecessary output.
Applicability
Applies to GenAI-integrated software systems where the LLM prompt combines instructions, retrieved data, conversation history and user-provided prompts.
Description
Four dimensions of pre-processing of text inputs are important for higher output quality.
The first dimension is to maintain an accurate conversation history. When LLMs lose track of the conversation history, they are likely to provide contextually incorrect answers when referring to the history. The LLM can also perform operations that conflict with earlier ones. A solution can be to maintain a short summary of key information from past conversations, and append it to prompts or validate LLM responses using execution history.
The second dimension is to validate and convert input formats. LLM interfaces may accept different file types, but the model is not always able to read the content. If software fails to validate and convert the input format, the result can be incorrect responses or even software crashes. A solution is to design input validation algorithms or convert the files before they are passed to the LLM.
The third dimension is related to LLMs producing unnecessary output. Unnecessary output can be a result of the over-generalization of a question, or repeating or rephrasing earlier responses. The solution is to provide a response template and instruct the LLM to limit the response length.
The fourth dimension is related to the LLM context limit. Extra-long inputs can cause accuracy issues when the LLM agent truncates the input. To prevent this problem, developers should keep the prompt length short by providing compact instructions, limiting the user input length, and compacting the conversation history. Compacting the conversation history can be done by summarizing the conversation with NLP or only retrieving relevant parts with RAG.

P4 — Re-order and re-structure LLM output for downstream tasks

Intent
Ensure the LLM output is compatible with the desired input for downstream tasks.
Motivation
Downstream tasks of a software system may require specific input formats. If an LLM generates the output in the wrong format, results can be bad user experience, software misbehavior, or fail-stop failures.
Applicability
Applies to GenAI-integrated systems that have LLM components with downstream tasks that require specific input requirements.
Description
LLMs can support a wide spectrum of output formats. However, downstream components may not contain that ability. For LLMs, it is important to produce output in a format that can be used by downstream tasks. When the output of an LLM cannot be processed by downstream tasks, malfunctions may appear. A single solution to this problem does not exist as there is not a single input format every software component can process. Therefore, developers should re-order and re-structure the LLM output to match the specific input for downstream software components.

P5 — Provide a final summary at conversation end

Intent
Ensure the final output of a multi-turn conversation contains a proper conclusion.
Motivation
In multi-turn conversations with chatbots, the chatbot cannot determine which is the final output. When all tasks are completed in the conversation, the last output may contain an incomplete solution.
Applicability
Applies to GenAI-integrated systems that include multi-turn conversations such as context-based QA or chat robot applications.
Description
When users have a multi-turn conversation, they expect a conclusion of the conversation in the final output. In an example from Shao et al. [31], LocalAGI is a GenAI-integrated system that provides steps for users to achieve a goal. Each output only contains a subset of the steps, and without explicit notice the application does not give the complete result. To prevent this problem, developers should implement an "on-exit" behavior that produces a closing response that summarizes the intermediate steps into a complete answer.

P6 — Use heartbeat techniques to maintain connection with remote LLM services

Intent
Maintain a stable connection between the client and the LLM agent or vector database on a remote server.
Motivation
Applications that deploy the LLM agent or the vector database on a remote server may lose connection. This happens with low-frequency interaction and can result in an unexpected shut-down. Re-connection harms the functionality and user experience.
Applicability
Applies to GenAI-integrated software systems where clients interact with remotely deployed LLM agents or vector databases.
Description
Low-frequency interactions with remote servers can lead to a disconnect. This is caused by a set timeout limit for connections. When the application tries to re-connect, the memory is lost. To address this, the timeout limit can be changed or a "heartbeat" technique can be applied that maintains connection with the remote server.

P7 — Use queue techniques to absorb speed mismatches between LLM agents and downstream tasks

Intent
Ensure components in the data pipeline can tolerate differences in processing speed between LLM agents and downstream tasks without failing.
Motivation
Some applications experience a speed mismatch between the LLM agent and their downstream component. Without solutions to absorb these problems, the mismatch can produce performance problems, I/O blocking, or fail-stop failures.
Applicability
Applies to GenAI-integrated software systems where an LLM agent processes streaming data and continuously feeds downstream tasks that operate at different processing speeds.
Description
A speed mismatch between an LLM agent and its downstream tasks results in performance problems. When the LLM component produces output faster than the downstream component can consume it, the data piles up. When the downstream component consumes the data faster than the LLM component can produce it, it tries to read data that is not available and might fail. The solution is to place a queue between those components that absorbs the speed differences.

P8 — Manage system resources for LLM and vector database components

Intent
Prevent resource contention, hangs and out-of-memory crashes by managing the system resources of LLMs and vector database components.
Motivation
Applications that run LLMs and vector databases locally place a heavy load on the system's CPU/GPU and memory. If these resources are not properly managed, two problems may arise: they cause resource contention which results in slow execution or hangs, or they can cause inefficient memory use, which can degrade performance or trigger out-of-memory crashes.
Applicability
Applies to GenAI-integrated systems that have LLMs or vector databases that are deployed locally or systems that operate in resource-constrained environments.
Description
LLMs request much of the system's resources, especially when the workload is heavy. Two problems can arise when these are not properly managed. First, components can compete for resources, which causes slow execution or hangs. This can be fixed by limiting how many resources each component may request and by adding recovery strategies such as downgrading services and limiting the number of users. Second, inefficient memory management can lead to performance degradation or out-of-memory crashes. A solution is to explicitly command the release of memory resources.

P9 — Implement robust error handling with LLM regeneration

Intent
Detect and recover from errors triggered by the integration of LLM agents and vector databases, rather than letting them result in a fail-stop failure or generic error messages.
Motivation
LLM components in software systems are data-centered and have loose output formats. Conventional error handlers are not able to handle all errors for LLM agents and vector databases, which might result in a fail-stop failure.
Applicability
Applies to GenAI-integrated systems that have LLM agents and vector databases interacting with conventional software components.
Description
With the integration of LLM agents and vector databases, the output formats become looser. Because conventional exception handlers are designed to handle logic-centered components that have strict format rules, they are not always suitable for LLM-integrated systems. There is no simple solution to fix this problem, but systems can be designed to recover from those failures. Examples are applying rule-based fixes or regenerating the LLM output with the error message included in the new prompt.

P10 — Design for suitable sufficiency criteria

Intent
Ensure GenAI-empowered solutions are evaluated based on practical effectiveness, rather than on pre-defined pass/fail criteria.
Motivation
GenAI software components produce a wide range of output. The traditional pass/fail criteria are not suitable for the varying output.
Applicability
This applies to GenAI tasks in GenAI-integrated systems that cannot be uniquely measured, for example summarizations or report creation.
Description
Do not rely on pass/fail criteria used for conventional software systems, which are designed for a predefined output range. Adopt new criteria that evaluate the wide output range of LLM components to determine if the work is good enough for real-world situations. GenAI-integrated systems should implement a review process with task-specific criteria.

P11 — Include verification and mitigation on LLM output

Intent
Address the reliability challenges of GenAI output by integrating verification and mitigation techniques for LLM output.
Motivation
The output by GenAI is unreliable, inaccurate and inconsistent. Therefore, measures need to be taken to prevent the use of incorrect output by dependent components.
Applicability
This applies to GenAI-integrated systems that have dependent software components.
Description
Software cannot expect GenAI output to be reliable, even when the components check their own output. It is important to include verification and mitigation strategies by using self-verification and fact-checking mechanisms at design time and runtime. Implementations of these strategies are described in P2 and P9. Dependent software components should anticipate potential issues, and include sanity checks and mitigation strategies at runtime and in production.

P12 — Minimize dependency on cognitive processing

Intent
Reduce reliance on cognitive processing by using it only where it is necessary or where it adds value.
Motivation
Not all tasks require cognitive processing to be included. More cognitive processing negatively affects the reliability, assurance and operational efficiency.
Applicability
Applies to systems integrating GenAI components in their software, where the task could be handled by traditional assets.
Description
In GenAI-integrated systems, there should be a balance between the amount of work performed by cognitive and traditional processing. For all tasks, it is important to determine if cognitive processing adds value to the task. If there is no added value, the task should be completed by conventional software components. Since cognitive processing is unreliable, reducing the use of GenAI components minimizes associated risk, increases efficiency and improves operational excellence.

P13 — Adopt GenAI security best practices

Intent
Design systems according to existing security guidelines.
Motivation
GenAI-integrated systems can be vulnerable to contamination and compromise, which can be intentionally or unintentionally. Adopting security practices can reduce the risk.
Applicability
Applies to GenAI components integrated in software systems.
Description
Not all GenAI components are safe to use. Therefore, it is recommended to adopt security practices in software systems. An example is the OWASP guidelines[24], which addresses threats and issues related to the use of LLMs in software systems. Examples of these practices are solutions for supply chain vulnerabilities and sensitive information disclosure.

P14 — Design for privacy

Intent
Tightly control GenAI components that handle sensitive or confidential information.
Motivation
GenAI-integrated systems often process sensitive or confidential information. Not properly handling the data can result in exposure or misuse which impacts the privacy, integrity and trust of the application.
Applicability
Applies to GenAI components integrated in software systems that process or exchange sensitive or confidential information.
Description
It is important to handle sensitive and confidential information tightly. Solutions can be to add guardrails, run the components in secure sandboxes and strictly control the data flowing in and out of the sandboxes.

P15 — Assess and mitigate misalignment

Intent
Use proactive and reactive measures to prevent, detect and mitigate potential bias.
Motivation
As cognitive processing is vulnerable to bias, it is important to add measures to prevent it.
Applicability
Applies to all GenAI components integrated in software systems.
Description
Bias is a common issue in cognitive processing. Therefore, proactive and reactive measures are essential to prevent, detect and hopefully reduce it. First, carefully select the underlying GenAI technologies. Second, add active guardrails and filters. Third, regularly conduct focused audits on outputs, intermediate reasoning, processing and generated code.

P16 — Provide observable and explainable solutions

Intent
Make the GenAI component observable, interpretable and explainable to detect and fix problems easier.
Motivation
The cognitive processing of GenAI systems is unreliable, so its behavior needs to be transparent. By implementing observability, interpretability and explainability mechanisms, users and developers can understand the reasoning behind the output.
Applicability
Applies to GenAI components integrated in software systems.
Description
The GenAI component should implement observability, interpretability and explainability mechanisms. Examples are creating introspectable cognitive techniques, and keeping generated code, reasoning plans and prompts readable. When these mechanisms are actionable, they allow for immediate corrective measures and long-term maintainability, reviews, and optimizations. These mechanisms are essential to carry out practices P11 and P15.