GateLens: A Reasoning-Enhanced LLM Agent for Automotive Software Release Analytics

Arsham Gholamzadeh Khoee
Chalmers University of Technology
Gothenburg, Sweden
khoee@chalmers.se

Shuai Wang
Chalmers University of Technology
Gothenburg, Sweden
shuaiwa@chalmers.se

Yinan Yu
Chalmers University of Technology
Gothenburg, Sweden
yinan@chalmers.se

Robert Feldt
Chalmers University of Technology
Gothenburg, Sweden
robert.feldt@chalmers.se

Dhasarathy Parthasarathy
Volvo Group
Gothenburg, Sweden
dhasarathy.parthasarathy@volvo.com

Abstract

Ensuring the reliability and effectiveness of software release decisions is critical, particularly in safety-critical domains like automotive systems. Precise analysis of release validation data, often presented in tabular form, plays a pivotal role in this process. However, traditional methods that rely on manual analysis of extensive test datasets and validation metrics are prone to delays and high costs. Large Language Models (LLMs) offer a promising alternative but face challenges in analytical reasoning, contextual understanding, handling out-of-scope queries, and processing structured test data consistently; limitations that hinder their direct application in safety-critical scenarios.

This paper introduces GateLens, an LLM-based tool for analyzing tabular data in the automotive domain. GateLens translates natural language queries into Relational Algebra (RA) expressions and then generates optimized Python code. It outperforms the baseline system on benchmarking datasets, achieving higher F1 scores and handling complex and ambiguous queries with greater robustness. Ablation studies confirm the critical role of the RA module, with performance dropping sharply when omitted.

Industrial evaluations reveal that GateLens reduces analysis time by over 80% while maintaining high accuracy and reliability. As demonstrated by presented results, GateLens achieved high performance without relying on few-shot examples, showcasing strong generalization across various query types from diverse company roles. Insights from deploying GateLens with a partner automotive company offer practical guidance for integrating AI into critical workflows such as release validation. Results show that by automating test result analysis, GateLens enables faster, more informed, and dependable release decisions, and can thus advance software scalability and reliability in automotive systems.

Keywords

Large Language Models, Tabular Question Answering, Software Release Analytics, Automotive Software Testing, Test Result Interpretation, Interpretable Reasoning

1 Introduction

Validating software releases in the automotive industry is a multifaceted challenge, particularly for embedded software in safety-critical systems. Modern vehicles integrate numerous subsystems, making this process complex and resource-intensive. Each integration phase involves *gating* steps—critical checkpoints where tests verify compliance with predefined quality standards. Failures at these gates can ripple through the system, delaying dependent subsystems regardless of their individual quality. Release managers tasked with safeguarding quality must analyze vast quantities of test results and validation data. While essential for ensuring safety and reliability, this process is time-consuming and prone to human error in data interpretation.

The software industry's transition from manual to automated processes has entered a new era with the emergence of Large Language Models (LLMs) [4, 18]. Companies are increasingly integrating these AI agents into their workflows, seeking more cost-effective and optimized solutions for complex software engineering tasks [14]. However, the direct application of LLMs to the software release validation process is hindered by limitations in interpretable reasoning and understanding of technical specifications [1, 20].

To address these challenges, we introduce *GateLens*, a reasoning-enhanced LLM agent to support release validation in the automotive domain. GateLens integrates structured relational analysis with domain-specific expertise, leveraging a reasoning layer built on Relational Algebra (RA) to break down complex validation tasks into systematic analytical steps. A domain-specific knowledge base—including automotive software specifications and data schemas—improves accuracy. GateLens simplifies three critical aspects of release validation:

1. Test Result Analysis: Analyzing test execution outcomes is foundational to release validation. This involves analyzing pass/fail patterns across comprehensive test suites, identifying recurring failures, and validating test coverage metrics. In automotive software, where a single release might involve a large number of test cases across multiple vehicle functions, this task becomes particularly demanding. Release engineers must not only identify failed tests but also understand their patterns, assess coverage adequacy, and evaluate test execution stability.

2. Impact Assessment: Impact Assessment is a systematic process for evaluating how software issues affect vehicle functionality and safety during release validation. It involves three phases: first, a critical failure analysis identifies the root cause and immediate effects of an issue, such as an ABS module causing a 200ms brake signal delay that exceeds the 100ms threshold. Second, a component-level

impact evaluation traces how the issue propagates through interconnected systems, assessing both direct effects, like problematic emergency braking, and indirect effects, such as reduced stability control performance. Finally, an integration risk assessment quantifies the severity of these impacts against safety thresholds and functional requirements, categorizing issues like the ABS delay as system-wide risks with critical severity. This structured process enables engineers to understand system-wide effects, ensuring all safety and functionality requirements are met before release.

3. Release Candidate Analysis: The final quality gate involves evaluating Release Candidates (RCs) against predefined quality gates and criteria. This encompasses analyzing whether a particular RC meets all quality thresholds, identifying potential release blockers, and validating compliance with release requirements. In automotive software, where releases must meet stringent safety and quality standards, this analysis requires careful validation of each RC against established criteria, ensuring all prerequisites for a safe and reliable release are satisfied.

The traditional release validation process demands extensive manual effort. Release engineers meticulously analyze test results, assess impacts, verify RCs against quality gates, and report findings to stakeholders, such as release managers. As automotive software systems grow increasingly complex, these manual workflows become more challenging, time-consuming, and error-prone.

This work aims to streamline release validation by automating key analysis workflows, enabling engineers to focus on high-value analysis and discussion. By providing deeper analytical insights, the proposed approach reduces the time needed to deliver accurate validation results, empowering release managers to make informed decisions more efficiently.

Our contributions can be summarized as follows:

- Integrating a *novel approach that combines domain-specific relational modeling with an analytical reasoning mechanism based on RA*, significantly enhancing the reasoning capabilities of LLMs when working with tabular data. By leveraging RA, the approach introduces analytical steps as RA expressions, enabling precise handling of domain-specific queries without any fine-tuning.
- A *systematic framework for automotive software release validation* that builds upon our existing deployed LLM-based multi-agent system in production. While our current system serves core release validation needs, increasing user diversity and query complexity revealed opportunities for improvement. GateLens addresses these limitations through enhanced query processing capabilities, as demonstrated in our comparative experiments, ensuring more robust and scalable test result analysis while maintaining the reliability of the release process.
- An *optimized system architecture for time-critical operations*, minimizing LLM invocations while maintaining high accuracy. By using only a single reasoning layer or one additional invocation of the LLM, the system is lighter than approaches requiring multiple invocations for planning and execution, reducing analysis time and enhancing decision support for release managers.
- *Empirical evaluation and industrial deployment* demonstrate GateLens's effectiveness in real-world automotive software

release scenarios, and that it can support diverse groups of stakeholders. Performance comparisons with GPT-4o and Llama 3.1 70B models and ablation of key system modules further highlight design trade-offs and key performance contributors.

2 Background and Motivation

Software release decisions in the automotive industry involve multiple stakeholders and extensive data analysis. Modern vehicles integrate hundreds of software components, each requiring rigorous testing and validation. The process advances through distinct phases: component-level testing, integration testing, system-level validation, and vehicle validation testing. Component-level testing verifies individual software modules, integration testing ensures proper interaction between components, system-level validation examines the complete system behavior, and vehicle validation testing evaluates software performance under real vehicle conditions on closed tracks.

The development cycle grows in complexity with each integration phase. This increasing complexity presents challenges in managing large-scale test results, tracking interdependencies between components, correlating test failures across different subsystems, and maintaining historical context for recurring issues. The iterative nature of software testing and validation further expands this data ecosystem.

The wide range of stakeholders in the release process creates additional challenges in data interpretation and presentation. Project managers need high-level progress indicators, verification engineers require detailed technical insights, quality engineers focus on trend analysis and improvement metrics, and release engineers need specific release-readiness indicators. This variety of perspectives necessitates different views of the same underlying data, making the analysis process more complex.

Release managers function as gatekeepers in the software deployment pipeline. Their responsibilities encompass test result analysis, cross-system impact assessment, decision-making, stakeholder coordination, and safety compliance verification. The manual workflow introduces vulnerabilities: time-intensive processing, potential errors in interpretation, decision delays, and communication barriers between technical and business teams.

Within this process, statisticians provide an overall view of the data to project managers and quality engineers for future business decisions. The existing manual approach faces several limitations, particularly regarding time and resource constraints. These include labor-intensive data analysis, delayed response to critical issues, limited capacity for comprehensive analysis, and bottlenecks in the release pipeline. Communication challenges further complicate the process, with misalignment between technical analysts and statisticians, varying interpretations of project requirements, inconsistent reporting formats, and knowledge transfer gaps.

Internal Testing on Closed Track represents a crucial validation phase. Release managers must analyze extensive datasets to evaluate progression readiness. The manual query process for report generation can impact release timelines, business objectives, and subsystem integration schedules.
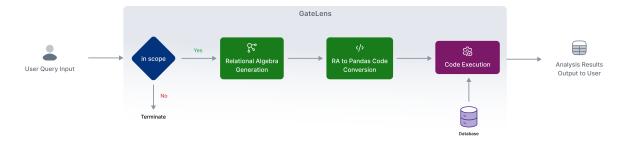
Figure 1: GateLens top-level architecture: The system processes high-level queries from the end user, generates the necessary data manipulation code using the enhanced reasoning layer with the help of RA, executes it, and outputs the result table as a decision-support resource.

The deployment of intelligent assistants presents opportunities to address these challenges through automated data processing and analysis, standardized reporting frameworks, real-time insights generation, and stakeholder-specific view generation. However, current automation solutions, including basic LLM implementations, face limitations in understanding complex technical specifications, maintaining structured analysis steps, handling domain-specific requirements, and processing automotive validation data systematically.

These limitations highlight the need for enhanced solutions combining domain expertise with advanced analytical reasoning capabilities. Such solutions must facilitate efficient decision-making while maintaining high safety and quality standards in automotive software development [30]. Effective intelligent assistants can transform the release decision process, enabling release managers to prioritize result interpretation and strategic decision-making over routine data analysis.

3 Approach and Methodology

The complexity of automotive software release validation demands a system that can bridge the gap between human-centric inquiries and precise technical analysis. GateLens addresses this challenge by utilizing LLM agents to transform natural language queries into actionable insights through systematic analysis.

At its core, GateLens must fulfill three fundamental requirements:

1. Query Understanding: The system must accurately interpret diverse user queries, ranging from high-level management questions to detailed technical inquiries about specific test cases.

2. Query Transformation: The system needs to transform these interpretations into structured formal expressions, ensuring consistent and verifiable reasoning structures.

3. Analysis Execution: The system must generate and execute precise analysis code that processes validation data according to these formal expressions.

The architecture of GateLens is driven by these fundamental requirements, establishing a systematic pipeline for transforming user queries into analytical results. The system leverages RA to enhance LLMs' reasoning capabilities and transform user queries into formal relational operations. To support this transformation, GateLens employs domain-specific data schemas that guide its relational modeling and enhance the generation of RA expressions. This

approach ensures both accessibility for non-technical stakeholders and the precision required for automotive software validation.

3.1 System Overview

The primary objective of GateLens is to generate executable code that performs precise test data analysis based on user queries. The system's workflow consists of two main phases: query interpretation and code generation. As illustrated in Figure 1, GateLens first processes user queries through an LLM agent that translates natural language inputs into formal RA expressions. This translation incorporates a detailed relational model of the test data, ensuring precise specification of automotive domain concepts. The resulting RA expressions serve as a pivotal intermediate representation that is more transparent to both LLM agents and humans. In the second phase, these formal expressions are passed to the coder agent, which generates executable desirable code, such as SQL or Python code, to perform the required analysis on the test data and produce results.

3.2 Core Components

The system architecture consists of two primary components that work in tandem to transform natural language queries into executable code: the query interpreter agent and the coder agent. The query interpreter agent first translates user queries into RA expressions, providing a structured framework for analytical reasoning. The coder agent then converts these formal expressions into executable code, completing the transformation pipeline. This two-stage approach ensures both analytical precision and efficient implementation, where the prompt engineering flow and prompt structure are presented in Figure 2.

3.2.1 Query Interpreter. The query interpreter agent is responsible for converting user queries into formal RA expressions, providing a precise framework for analytical reasoning. Before initiating this translation, the agent consults the knowledge base, comprising the data schema and domain-specific context. The data schema provides a detailed understanding of the dataset, including its relational modeling, detailed field descriptions, and data types. Using this information, the agent verifies whether the query is relevant and within the scope of the dataset [19]. This validation step ensures that only supported and meaningful queries are processed, improving both accuracy and efficiency. Once the query is confirmed to be

in scope, the agent leverages the knowledge base to interpret and decompose the query into formal RA expressions.

The agent's primary function is to map natural language queries into formal RA expressions, enhancing LLM reasoning through structured decomposition [13]. This approach extends traditional chain-of-thoughts (CoT) [33] reasoning by constraining the model to think within a formal system framework [36]. Instead of generating free-form solutions, the agent must express analytics through a limited set of standard operations: selection, projection, union, set difference, cartesian product, and rename as basic operations, as well as derived operations such as join, intersection, and division and complemented by aggregation functions like average, minimum, maximum, sum, and count.

By limiting operations to this standard set, the agent effectively handles ambiguous queries through formal translation, ensures technical precision, and prevents deviation from analytical requirements. The formal nature of RA enables query optimization, which the agent incorporates by prioritizing data reduction operations early in the expression chain. This optimization strategy involves applying filters first, then performing expensive operations on the reduced dataset, thereby minimizing processing time and resource utilization.

The translation to RA offers two significant advantages. First, it makes the analytics more transparent in technical terms, allowing for clear interpretation and validation of the reasoning process. Second, it ensures that every solution generated is precisely defined and feasible for implementation, preventing the agent from proposing impractical or undefined analytical approaches.

3.2.2 Coder. The coder agent is responsible for generating executable code from given RA expressions. Upon receiving an RA expression, the agent follows precise instructions to produce code that delivers the final analytical results. This capability allows the agent to generate complete, self-contained code at once, eliminating the need for step-by-step generation and execution phases.

The generated code is immediately ready for execution and incorporates robust error handling and validation mechanisms. These include data type validation to ensure correct handling of numerical and categorical fields, null value handling to maintain data integrity, validation of numerical operations, and validation of join conditions to ensure proper matching of key columns between tables. Exception handling is also included to address potential runtime errors, ensuring reliable execution.

By generating the entire code in a single pass rather than through iterative refinement, the agent significantly reduces processing overhead, system response time, and resource consumption while minimizing potential errors that could arise from multiple execution steps. This streamlined approach to code generation and execution ensures both efficiency and reliability in the analysis pipeline while maintaining the precision established by the formal RA expressions. Producing the executable code in a single step also significantly accelerates the overall analysis process, improving system responsiveness to user queries.

3.3 Data Handling

A key architectural decision in GateLens is its indirect interaction with test data. Rather than exposing sensitive test data directly to
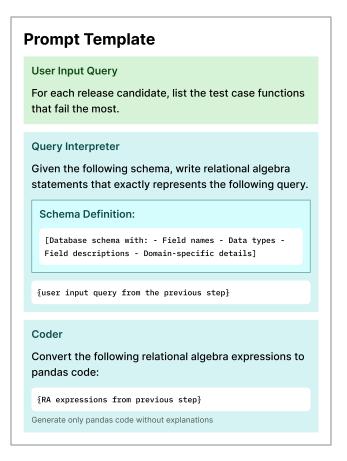

> **Prompt Template**
>
> **User Input Query**
>
> For each release candidate, list the test case functions that fail the most.
>
> **Query Interpreter**
>
> Given the following schema, write relational algebra statements that exactly represents the following query.
>
> > **Schema Definition:**
> >
> > `[Database schema with: - Field names - Data types - Field descriptions - Domain-specific details]`
>
> `{user input query from the previous step}`
>
> **Coder**
>
> Convert the following relational algebra expressions to pandas code:
>
> `{RA expressions from previous step}`
>
> Generate only pandas code without explanations

Figure 2: Overview of the prompt engineering flow and prompt structure within the GateLens system architecture.

LLM agents, which could raise privacy concerns [2] and exceed input context limitations, the system operates on data schemas and relational models. This approach serves multiple critical purposes:

- **Privacy Protection:** Sensitive automotive test data remains secure within the organization's infrastructure
- **Hallucination Prevention:** By limiting LLMs' access to actual data, the system prevents potential hallucinations or incorrect inferences
- **Scalability:** The system can handle large-scale test datasets that would exceed LLM context windows
- **Knowledge Integration:** Data schemas and relational models serve as a comprehensive knowledge base, providing necessary structural understanding without raw data exposure

The final execution of the generated code runs on the test data in the target environment, maintaining data privacy while delivering precise analytical results.

4 Experimental Evaluation

In this section, we aim to answer the following research questions:

RQ1: How effectively does GateLens address user queries and deliver accurate results across various query categories?

RQ2: How robust is GateLens in handling out of scope queries and imprecise queries?

RQ3: How does the RA reasoning procedure contribute to the overall performance of GateLens?

RQ4: How does the number of few-shot examples impact GateLens's performance?

4.1 Experimental Setup

To address the research questions introduced in Section 4, we designed and conducted extensive experiments to evaluate the performance of GateLens.

The experimental data comprises two distinct benchmarks. The first benchmark consists of 50 queries designed with the assistance of release engineers, quality engineers, and verification engineers. These queries are categorized into four levels of difficulty, following the same approach introduced in GoNoGo [12], to assess GateLens's performance across varying levels of difficulty. The four levels of query difficulty are defined as follows:

Level 1 Simple queries involving a single operation such as filtering or sorting.

Level 2 Queries combining two or three basic operations, such as multiple filtering followed by sorting.

Level 3 Queries involving more than three operations, potentially including grouping and aggregating.

Level 4 Complex queries requiring multiple advanced operations beyond basic filtering and sorting such as grouping and aggregating for statistical calculations.

The second benchmark is derived from real-world user queries collected from production logs. These queries were curated from the historical logs of the first-generation system, GoNoGo, a multi-agent system designed to support software release decision-making using planning and few-shot learning, which is currently deployed in production at our partnering industrial company [12]. While GoNoGo performs effectively in many scenarios, its limitations become evident as the range of roles and users expands, leading to a significant diversification of queries. This increased diversity exposes GoNoGo's reliance on few-shot examples, making it less capable of handling highly complex, ambiguous, or ill-defined queries that require greater flexibility and adaptability. Despite its limitations, the GoNoGo system played a critical role in data collection for GateLens by providing query logs that aided in developing and validating future iterations. Leveraging these historical logs, we filtered out very similar queries and selected 244 of the most frequently repeated unique queries. These queries were then categorized into eight functional categories based on their purposes.

In order to assess GateLens's performance, a series of experiments are conducted using two large language models (LLMs): GPT-4o, a leading commercial model in its class, and Llama 3.1 70b, a recently introduced open-source model. GateLens is also compared with the latest version of the first-generation system, GoNoGo [12], our recently deployed production system that has demonstrated significant value in industrial settings. While GoNoGo effectively serves the company's release engineering needs, this comparative analysis aims to evaluate potential improvements in our new system design. It is important to note that this version of GoNoGo differs slightly from the one introduced in the original paper, as it incorporates minor updates in prompt engineering and replaces GPT-3.5 with the aforementioned recent advanced LLMs, while still maintaining its core approach based on Chain-of-Thought (CoT) [33] reasoning and Self-Consistency [31] techniques.

To address the challenge of handling out of scope user queries during real-time interactions, GateLens incorporates an in-scope filtering mechanism as explained in Section 3.2. This mechanism ensures that the system only attempts to process queries that fall within its scope, thereby improving reliability and reducing errors. Performance evaluation focused on two key aspects:

(1) **Quality of responses**: Measured using precision, recall, and F1 Score, which reflect the system's ability to address relevant queries correctly.

(2) **Coverage of relevant queries**: Ensuring the system does not reject a significant proportion of valid queries, thus maintaining broad applicability.

Additionally, an ablation study is conducted to examine the contribution of the RA reasoning mechanism of GateLens.

In our experiments, the evaluation of system performance is based on the following definitions: A **True Positive (TP)** occurs when the system produces a result that matches the manually generated ground-truth result. A **False Positive (FP)** occurs when the system provides an incorrect result. A **False Negative (FN)** occurs when the system fails to provide any result for a query. **True Negatives (TNs)** are not applicable in this evaluation, as we focus on the system's ability to handle valid queries that result in meaningful output.

Based on these definitions, we calculate Precision, Recall, and F1 scores to assess the performance of the system. Precision ensures that incorrect results are minimized, recall ensures relevant queries are addressed, and the F1 score balances the two to provide an overall assessment of system performance.

4.2 Performance in Addressing User Queries (RQ1)

We conducted experiments to compare the performance of GateLens across the two introduced benchmarks. The first benchmark, consisting of 50 queries categorized by difficulty levels, was used to evaluate and compare the performance of GateLens and GoNoGo. Both systems were tested using GPT-4o and Llama 3.1 70B as their underlying LLMs. The results are summarized in Table 1.

The results demonstrate that GateLens with GPT-4o significantly outperforms GateLens with Llama 3.1 70B, indicating GPT-4o's superior capability in interpreting and generating RA. Similarly, GoNoGo with GPT-4o outperforms its Llama 3.1 70B variant, with the performance gap growing as query complexity increases. Most notably, GateLens with GPT-4o achieved optimal performance on this benchmark, maintaining 100% accuracy across all difficulty levels. This superior performance can be attributed to the RA reasoning mechanism integrated into our framework. By translating queries into RA expressions, GateLens explicitly captures the logical structure of operations, enhancing the clarity and precision of the generated code. This intermediate RA conversion allows the system to focus on the relevant table operations while filtering

Table 1: Performance comparison across different models and difficulty levels on the first benchmark, which consists of 50 designed queries with annotated difficulty levels.

Level	# Queries	GateLens with GPT-4o			GateLens with Llama 3.1 70B			GoNoGo with GPT-4o			GoNoGo with Llama 3.1 70B		
		Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
1	16	100%	100%	**100%**	100%	43.75%	60.87%	93.33%	87.5%	90.32%	100%	93.75%	96.77%
2	16	100%	100%	**100%**	100%	62.5%	76.92%	100%	81.25%	89.66%	92.31%	75%	82.76%
3	12	100%	100%	**100%**	100%	50%	66.67%	91.67%	91.67%	91.67%	90.91%	83.33%	86.96%
4	6	100%	100%	**100%**	100%	33%	49.62%	66.67%	66.67%	66.67%	60%	50%	54.55%
Total	**50**	**100%**	**100%**	**100%**	**100%**	**47.31%**	**63.52%**	**87.91%**	**81.77%**	**84.57%**	**85.81%**	**75.52%**	**80.26%**

Table 2: Performance comparison of GateLens and GoNoGo across different categories on the second benchmark, which consists of 244 real-world queries.

Category	# Queries	GateLens with GPT-4o			GateLens with Llama 3.1 70B			GoNoGo with GPT-4o		
		Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Column Operations	17	64.7%	64.7%	64.7%	50%	11.76%	19.04%	76.47%	76.47%	**76.47%**
Complex Multi-Condition Queries	77	86.3%	81.82%	**84%**	100%	24.68%	39.59%	84.75%	64.94%	73.53%
Conditional Calculations	8	100%	87.5%	**93.3%**	100%	37.5%	54.55%	87.5%	87.5%	87.5%
Data Filtering	32	89.66%	81.25%	**85.25%**	90.91%	31.25%	46.51%	86.21%	78.13%	81.97%
Duplicate Removal	78	87.67%	82.05%	**84.77%**	100%	23.08%	37.5%	75.93%	52.56%	62.12%
Grouping and Aggregation	10	80.0%	80.0%	**80.0%**	100%	30%	46.15%	83.33%	50%	62.5%
Metadata Queries	13	91.67%	84.61%	**88.0%**	100%	15.38%	26.66%	46.15%	46.15%	46.15%
Table Generation	9	88.89%	88.89%	88.89%	100%	44.44%	61.53%	88.89%	88.89%	**88.89%**
Total	**244**	**86.02%**	**81.14%**	**83.51%**	**92.61%**	**27.26%**	**41.44%**	**83.15%**	**63.52%**	**70.61%**

out irrelevant elements in the query, which greatly enhances the problem-solving capabilities of the LLM agent.

For the second benchmark, results in Table 2 show that GateLens with GPT-4o and GoNoGo with GPT-4o significantly outperformed GateLens with Llama 3.1 70B. This performance disparity is primarily due to the strict code generation requirements of the task, including table filtering, merging strategies, and key-value mapping operations, where GPT-4o demonstrated markedly superior capabilities compared to Llama 3.1 70B for this purpose.

GateLens with GPT-4o outperformed GoNoGo with GPT-4o across most categories, particularly evident in Metadata Queries—those seeking basic table information. For example, when processing the query "Give me the list of release candidates," GoNoGo often struggles with proper field identification. A common failure mode in GoNoGo occurred when user queries included typographical errors or incorrect casing in field names, with the system directly using the erroneous fields without correction. GateLens addresses this limitation through its query-to-RA transformation process, which incorporates the database's relational model, adjusts query fields to match table formats, and can handle fuzzy matching to detect and correct field names, enabling the system to resolve typographical errors and ambiguous queries effectively. This robust approach significantly improves the system's accuracy and resilience, particularly in real-world scenarios where user queries may not always adhere to strict formatting standards.

Across both datasets, GateLens with GPT-4o consistently delivered the best performance, achieving perfect accuracy across difficulty levels in the first benchmark and superior handling of diverse query categories in the second benchmark. This superior performance can be attributed to the system's capability to extract logical conditions from natural language queries and transform them into RA expressions. This transformation not only clarifies

the underlying logic of the queries but also ensures that all conditions are accurately identified and correctly applied, minimizing errors and omissions in the generated code. These results highlight how the proposed framework significantly enhances the ability of LLMs to handle complex queries while improving their robustness and reliability in real-world applications.

4.3 Robustness: Handling Out of Scope and Imprecise Queries (RQ2)

To assess the robustness of our approach in handling diverse user queries under real-world conditions, we conducted further experiments focusing on filtering out of scope queries as well as processing imprecise queries. For this purpose, the data analysis team at our industrial partner company manually selected 37 out of scope queries and 50 imprecise queries from the historical logs of the first-generation system, which are used to perform targeted evaluations.

Out of scope queries are those that cannot be meaningfully answered using the available data. For example, a query like "What is the most beautiful truck?" requires subjective judgment and cannot be resolved through database operations; it should be identified and filtered as out of scope. On the other hand, imprecise queries are those that can be answered using the database but contain ambiguous or inexact terms. For instance, a query such as "Find some trucks for cases that are NOK" is considered imprecise because while it seeks truck names where test results are "NOK" (failed), it uses ambiguous terminology - referring to "trucks" instead of the actual database field "name", and mentions "NOK" without specifying the "test_result" field. Such imprecise queries require mapping informal language to precise database fields and conditions for proper execution.

4.3.1 Handling Out of Scope Queries. We compared GateLens with other models; the results can be found in Table 3. The results demonstrate that GateLens with GPT-4o achieved the best performance, particularly in terms of precision, which is approximately 40% higher than other models, indicating GateLens' ability to avoid generating incorrect results.

Table 3: Comparison of different models for out of scope queries.

Model	Precision	Recall	F1 Score
GateLens with GPT-4o	92.5%	100%	**96.10%**
GateLens with Llama 3.1 70B	52.94%	97.30%	68.57%
GoNoGo with GPT-4o	51.10%	89.19%	64.97%

The superior precision of GateLens with GPT-4o can be attributed to two key aspects of its design. First, its robust filtering mechanism ensures that out of scope queries are identified and excluded early in the processing pipeline, preventing irrelevant results. Second, the conversion of raw natural language queries into structured RA expressions enables the model to isolate and capture task-relevant components of a query. This structured approach considerably decreases erroneous outcomes and enhances the model's ability to handle complex and diverse query formulations in real-world scenarios.

In contrast, the precision of GoNoGo was notably weaker. This can be attributed to the diversity of real-world queries and the variability in user narratives, which often contain a mixture of relevant and irrelevant parts. Such queries increase the degree of uncertainty, making it challenging for models that cannot effectively isolate task-relevant parts to process them appropriately. Furthermore, while the recall rates of all models were relatively high, indicating their ability to generate some results, this does not necessarily translate to accurate query processing.

A significant limitation was observed with GoNoGo, which relies on few-shot learning, as it occasionally generated code for out of scope queries. This behavior likely stems from the task-relevant examples in its prompt, which inadvertently bias the model toward attempting code generation even for out of scope queries. In summary, our approach demonstrates a strong ability to filter out of scope queries while maintaining a low error rate when processing in-scope queries.

4.3.2 Handling Imprecise Queries. To further assess the robustness of our approach, we evaluated its performance in handling imprecise queries, which posed two primary challenges. First, some queries are informal and conversational in style, appearing unrelated to data analysis but actually carrying relevant intent. Second, many queries referred to fields using terms differing from the column headers.

The results of these experiments are presented in Table 4. As shown, GateLens with GPT-4 demonstrates the best overall performance. In terms of precision, all methods performed relatively well, indicating that when results are generated, they are likely to be correct. However, our method significantly outperformed the others in recall, highlighting its ability to handle a larger portion of the imprecise queries. As a result, GateLens with GPT-4 achieved a

Table 4: Comparison of different models for imprecise queries.

Model	Precision	Recall	F1 Score
GateLens with GPT-4o	92.86%	78%	**84.78%**
GateLens with Llama 3.1 70B	92.86%	26%	40.63%
GoNoGo with GPT-4o	90%	36%	51.43%

substantially higher F1 score compared to other methods, demonstrating that it not only processes most queries but also produces accurate results for them.

The observed performance gap between GateLens with GPT-4o and the other models can be attributed to their inherent limitations. Specifically, the Llama 3.1 70B model struggled to interpret user queries that deviated from the exact column header descriptions in the database schema. In such cases, Llama 3.1 70B often converted only the clearly defined parts of the query into RA, leading to incomplete query execution and reduced accuracy. On the other hand, GoNoGo exhibits low recall, as it is highly susceptible to confusion by ambiguous query elements. This causes GoNoGo to frequently generate incorrect code, which, when executed, fails to produce valid results, significantly lowering its recall rate.

Overall, our approach demonstrated a strong ability to handle imprecise queries by maintaining high precision and significantly improving recall. This robustness ensures that the system is adaptable to variations and ambiguities in query formulations, allowing users to freely express their queries without being constrained by strict adherence to column header names. Such flexibility enhances the system's practicality and applicability in real-world scenarios, making it a reliable and user-friendly tool for diverse interactions.
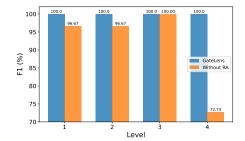
4.4 Effectiveness of the RA module (RQ3)

To evaluate the impact of the RA module that converts user queries into RA expressions, we conducted experiments by removing the RA module from the framework and comparing the results to the original system. The outcomes, shown in Figure 3, demonstrate significant performance degradation across both benchmarks when operating without the RA module.
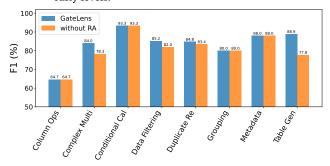
In the first benchmark, performance declined most notably for Level 4 queries, showing a drop exceeding 27%. These queries, which involve advanced operations like grouping, aggregating, and statistical calculations, demonstrated that RA translation is particularly crucial for handling queries with multiple, intricate operations. Similarly, the second benchmark shows decreased performance in complex tasks such as multi-condition filtering, duplicate data removal, and table generation, further emphasizing RA's effectiveness in managing complex database operations.

The RA module maintained consistent performance for simpler queries, demonstrating its versatility across varying complexity levels. By transforming natural language into precise, logical representations, the RA module serves as a critical bridge between user intent and code execution. This translation process enables the code generator to produce accurate, efficient executable code for data analysis tasks.

These results establish RA as a fundamental component in our system's architecture, significantly enhancing its reasoning capabilities and ensuring reliable code generation across diverse query

(a) The first benchmark with annotated difficulty levels.

(b) The second benchmark with real-world user queries

Figure 3: Comparison of the original method and the method without the RA module across different datasets.

types and complexity levels. This enables the system to deliver accurate and efficient solutions for tabular data analysis tasks.

4.5 Impact of Few-Shot Examples (RQ4)

To investigate the effect of including few-shot examples in prompts, we conducted experiments by varying the number of examples provided to both GateLens and GoNoGo. This experiment is performed on the first benchmark containing 50 designed queries, with results illustrated in Figure 4.

The results demonstrate that GateLens relies heavily on its RA translation process, achieving optimal performance even in a 0-shot setting without any examples. Interestingly, when a small number of examples are added, GateLens becomes slightly biased toward them, leading to a minor degradation in performance. However, it regains optimal performance with additional examples as the system learns to generalize better. In contrast, GoNoGo's performance heavily depends on in-context examples, showing suboptimal results without sufficient few-shot examples.

The results further highlight the critical role of RA in enabling effective query handling. GateLens, through its RA-based translation process, achieves robust performance without requiring extensive in-context learning. GateLens's independence from examples not only ensures consistent performance across diverse queries but also offers practical benefits by mitigating issues related to context length limitations and accordingly reducing computational and financial resource consumption, enhancing system scalability, and minimal maintenance needs for adaptation to new tasks. These
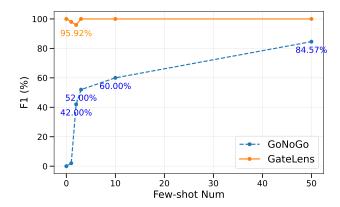

Figure 4: Performance of GateLens and GoNoGo with varying numbers of few-shot examples.

advantages position GateLens as a robust and efficient solution for automating the analysis of tabular data in dynamic environments.

5 Industrial Deployment: Lesson Learned

The deployment we have done of GateLens in a partner automotive company highlights both the challenges and opportunities of integrating AI-assisted analytics into complex industrial workflows. The system was designed to streamline decision-making in automotive software release processes, and its deployment has provided valuable insights into improving decision-making in software release validation for large-scale environments.

Automotive software integration at the company typically occurs across three hierarchical stages: subsystem (control unit), system (multiple control units), and full vehicle levels. Each stage involves extensive testing, with results stored in a central database. Critical Go/No-Go decisions are made at these stages to determine whether a release meets quality thresholds. However, stakeholders from diverse backgrounds—including project managers, mechanical engineers, and software engineers—must query the raw data to evaluate product quality. Many lack expertise in data analytics, creating inefficiencies in decision-making.

Previously, these analytics were managed by a small team of 2–3 full-time analysts, who were often overwhelmed by the volume and diversity of requests. Scaling the team to meet the current demand would have required tripling its size. GateLens addresses this challenge by automating much of the workload, enabling decisions to be made more efficiently and accurately. Currently, GateLens is in an extended pilot phase, supporting a pool of 60-80 users. The analytics team has transitioned to a support role, helping stakeholders articulate their needs into clear, actionable prompts for the system.

User adoption of GateLens has progressed in phases:

- **Small-Scale Pilot**: The initial deployment within the analytics team established baseline benchmarks.
- **Expanded Pilot**: Five additional users from varied backgrounds contributed to refining the benchmarks.
- **Wider Rollout**: The current phase involves a larger group of 60–80 users. Feedback has been highly positive, with stakeholders recognizing GateLens's ability to simplify and accelerate complex analyses.

Table 5: Comparison of GateLens (zero-shot) and GoNoGo (few-shot) performance across different roles on the second benchmark. For each role tested, GoNoGo was trained using examples from the other two roles only (leave-one-role-out approach).

Roles	# Queries (244 in total)	GateLens F1 Score	GoNoGo (Few-shot with All Roles) F1 Score	GoNoGo (Few-shot with Leave-One-Role-Out)		
				Without Mechanic	Without Project	Without Software
Mechanically-oriented	36	94.25%	80.60%	73.85% ↓ (-6.75%)	86.57%	80.60%
Project-oriented	193	80.21%	78.93%	78.93%	76.22% ↓ (-2.71%)	78.40%
Software-oriented	15	100%	100%	100%	100%	82.76% ↓ (-17.24%)

Since the launch, the number of both new and recurring users has grown, encompassing diverse roles and types of queries, thereby demonstrating the tool's increasing utility and trust. GateLens significantly reduces the time and effort required for complex analyses, but the shift towards automation also requires users to take on more responsibility in defining and clarifying their needs. The transition from a primarily supportive tool to a more fully automated system is ongoing, demanding a gradual approach with careful calibration to ensure the tool continues to meet evolving needs.

The diversity in stakeholders' needs and backgrounds is an inevitable factor, leading to a wide range of requirements and queries when interacting with analytics systems. Our previous system, GoNoGo, has performed well when serving a specific group of stakeholders. However, as we opened the system to a broader audience, the variety of queries increased substantially. Systems that rely heavily on few-shot learning face significant challenges in these situations, as they are limited by the relatively small set of few-shot examples, making it difficult to handle a wider range of potentially unexpected inputs. Similarly, fine-tuned models often struggle to adapt to dynamic and diverse environments. Improving generalization is essential to ensure the scalability and reliability of analytics systems in these complex, domain-specific contexts. In our design and development work, we prioritized the system's ability to generalize effectively and meet the needs of a broader and more diverse group of users.

To explore the system's generalizability, we categorized the roles within the company into three groups: mechanically-oriented, project-oriented, and software-oriented roles. The queries from these roles have distinct focuses. Mechanically-oriented roles typically focus on truck-specific data filtering. Project-oriented roles often combine meta-queries with conditional filters for release management and statistical analysis. Software roles emphasize truck software applications and user functions. We can see from Table 5, both GateLens (zero-shot) and GoNoGo (few-shot) exhibit differences in system performance across these groups, which is likely stemming from the complexity and variety of their typical queries. Nevertheless, the results demonstrate that GateLens is capable of supporting all groups to a high degree. To further evaluate GoNoGo's dependency on few-shot examples, we conducted a **leave-one-role-out** experiment. In this approach, examples from a specific role are excluded in each iteration. For instance, 'without software' indicates that all examples from the software-oriented role have been removed, while the total number of examples is maintained by substituting them with examples from other roles. This highlights the potential challenges with the robustness and generalizability of techniques that rely on few-shot examples. This is a crucial factor to consider in industries where diverse teams collaborate and a wide range of queries may arise.

The impact of automated systems, such as GoNoGo and GateLens, on the release process has been substantial. Compared to the previous manual process, GateLens has reduced the time required for Go/No-Go analytics by more than 80%, significantly improving operational efficiency. Stakeholders can now focus on high-level decision-making, freed from the burden of data preparation and analysis.

A key advantage of GateLens lies in its domain-specific design. Unlike general-purpose tools like TaskWeaver [26], GateLens is tailored to automotive workflows, making it easier to understand, debug, and adapt to automotive common procedures. This focus on domain relevance ensures that the system aligns more closely with stakeholders' needs while providing reliable and nuanced support.

In summary, the deployment of GateLens demonstrates how domain-specific AI solutions can transform critical workflows in the automotive sector. By automating labor-intensive processes and enhancing decision-making, GateLens has delivered measurable improvements in efficiency and user satisfaction. However, its success depends on ongoing refinement and careful management of the transition to full(er) automation. Balancing automation with user empowerment remains crucial, particularly in a complex industry like automotive, where diverse stakeholder needs must be met.

GateLens represents a promising step forward, showcasing the potential of AI-driven systems to improve not only the automotive domain but also other industries requiring robust, scalable solutions for intricate processes.

6 Related Work

General-purpose LLMs are primarily designed for and trained on natural languages. Working with tabular data requires specialized adaptations to effectively handle its structured and heterogeneous nature [7, 28, 32]. First, the structured tablular data is typically transformed into serialized text. The performance of the LLM may depend on this transformation [21]. Subsequently, the serialized text data is used as input to the LLM for various tasks, such as question-answering, summarization, or logical reasoning. Common solutions to improve the LLM performance on various tasks include prompting engineering, pre-training, fine-tuning, and Retrieval-Augmented Generation (RAG).

Pre-training and fine-tuning [5, 10, 24, 29, 35] often face scalability concerns. Athough resource-efficient training techniques have been proposed to mitigate the substantial computational demands of LLMs [9, 17], in safety-critical applications with evolving data and requirements, training LLMs presents significant challenges due to the constant need for rigorous validation and verification. This ongoing necessity substantially increases resource demands for development and maintenance, potentially exceeding the capacities of many companies. Techniques such as RAG have been

employed to dynamically integrate external knowledge bases during inference, reducing the need for frequent model updates [8, 37]. However, such methods can pose challenges in safety-critical industrial settings as well, since both retrieval modules and model components must undergo synchronized updates to maintain relevance, reliability, and compliance with validation and verification requirements.

Prompt engineering techniques are among the most resource-efficient methods for improving LLM output [11, 27]. From a user standpoint, when the input is natural language, prompting techniques can be broadly categorized based on the type of language generated by the LLM. These include outputs in natural language, structured languages [15], or symbolic languages. When the generated language is natural language, LLMs often fail to consistently follow instructions, particularly when the instructions are complex or require precise, step-by-step execution [25]. This inconsistency arises because natural language, while flexible and expressive, can be ambiguous and prone to misinterpretation by LLMs. Structured languages include general-purpose languages (e.g. Python) [34], query languages (e.g. SQL) [6, 16, 22], configuration formats (e.g. YAML or JSON), or other Domain-Specific Languages (DSLs). These lagnguages are subsequently interpreted and/or executed by either external tools, the same LLM, or another LLM agent. This approach offers significant advantages, as it enables precise execution of tasks. Another popular type of output is symbolic languages. Literature shows that symbolic representations provide a more rigorous framework for articulating premises and intent, which can enhance reasoning capabilities [23].

In this paper, we uniquely integrate structured and symbolic languages by converting natural language input into RA expressions, a formalism specifically designed for relational modeling and particularly suited for table analysis, before translating these expressions into Python code for execution. This approach contrasts with prior work that often relies on specialized DSLs or fine-tuning, which can limit flexibility and scalability. By leveraging RA as an intermediate symbolic representation, we achieve precise reasoning and query optimization, while Python's general-purpose capabilities ensure broad applicability and efficient execution across diverse scenarios. Unlike methods that depend heavily on few-shot examples or fine-tuning, GateLens combines interpretability, adaptability, and computational efficiency, making it particularly suitable for dynamic, safety-critical table analysis applications in industrial environments. Notably, RAG systems focus on factuality by retrieving concrete examples without dedicated reasoning modules; nonetheless, they can be integrated with GateLens if needed. It is worth noting that RAG requires constant updates and rigorous validation for safety-critical applications. This ongoing process demands significant effort, specialized expertise, and substantial resources, which can pose a considerable challenge for some organizations.

7   Discussion and Future Work

This study introduced GateLens, a reasoning-enhanced LLM agent specifically designed for automotive software release validation. GateLens demonstrates significant advancements by reducing analysis time by over 80% while maintaining high accuracy in test result interpretation, impact assessment, and release candidate evaluation. The inclusion of a Relational Algebra reasoning module is a critical

factor in improving robustness and scalability, as evidenced by superior F1 scores in both benchmarking and industrial evaluations. Additionally, GateLens successfully handles complex and ambiguous queries, providing practical and reliable support for faster and more informed decision-making in safety-critical software release processes. A phased deployment program is ongoing and shows that multiple stakeholder groups can be supported but also that the roles change and analytical needs evolve which will require continued refinement and discussions.

The validity of our findings is subject to several potential threats. First, the generalizability of GateLens beyond the automotive software release domain is limited, as the system relies on domain-specific relational modeling and datasets, necessitating further validation in other safety-critical industries. Second, the benchmarks and query scenarios used for evaluation, derived from historical data and real-world queries, may not fully capture the diversity and complexity of potential use cases, which could impact the robustness of the system in broader deployments. Finally, the system's performance is influenced by the specific configurations of the LLMs used, such as GPT-4o and Llama 3.1 70B, requiring the capability of interpreting and generating RA expressions. Future work will address these limitations by expanding domain applicability, diversifying evaluation datasets, and testing alternative LLM configurations to enhance reliability and scalability.

8   Conclusions

In automotive manufacturing, efficient software release and status monitoring depend critically on analyzing test results logged in a database. However, the volume and diversity of required queries from verification engineers and stakeholders make manual analysis a significant challenge. GateLens addresses these challenges by leveraging an LLM-based approach that translates natural language queries into structured relational algebra expressions within a defined relational model, in which the agent uses these expressions to produce the desired analytical code. Besides, GateLens has the ability to filter out-of-scope queries based on the database schema, improving the robustness and reliability of the framework.

In real-world deployment serving 60-80 users, GateLens has demonstrated significant practical value through its user-friendly interface and robust query processing capabilities. Users particularly appreciate the flexibility to input, debug, and refine queries easily, marking a substantial advancement in industrial data interaction.

Our implementation insights highlight the advantages of focusing on prompting techniques and in-context learning rather than fine-tuning approaches. This design choice ensures easy integration of newer LLM models without additional training costs while maintaining domain-specific performance [3]. As we move forward, this modular architecture opens opportunities for incorporating emerging LLM capabilities while preserving the system's practical utility in safety-critical industrial applications.

Acknowledgments

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[2] Alexander Theodorus Petrus Boudewijn, Andrea Filippo Ferraris, Daniele Panfilo, Vanessa Cocca, Sabrina Zinutti, Karel De Schepper, and Carlo Rossi Chauvenet. 2023. Privacy measurements in tabular synthetic data: State of the art and future research directions. In *NeurIPS 2023 Workshop on Synthetic Data Generation with Generative AI*.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3 (2024), 1–45.

[5] Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745* (2022).

[6] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306* (2023).

[7] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Jane Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, Christos Faloutsos, et al. 2024. Large language models (LLMs) on tabular data: Prediction, generation, and understanding-a survey. (2024).

[8] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[9] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608* (2024).

[10] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. [n. d.]. TabLLM: Few-shot Classification of Tabular Data with Large Language Models. ([n. d.]). https://doi.org/10.48550/arXiv.2210.10723

[11] Ziqi Jin and Wei Lu. 2023. Tab-cot: Zero-shot tabular chain of thought. *arXiv preprint arXiv:2305.17812* (2023).

[12] Arsham Gholamzadeh Khoee, Yinan Yu, Robert Feldt, Andris Freimanis, Patrick Andersson Rhodin, and Dhasarathy Parthasarathy. 2024. GoNoGo: An Efficient LLM-based Multi-Agent System for Streamlining Automotive Software Release Decision-Making. *arXiv preprint arXiv:2408.09785* (2024).

[13] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406* (2022).

[14] M. Leung and G. Murphy. 2023. On automated assistants for software development: the role of llms. *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2023), 1737–1741. https://doi.org/10.1109/ase56229.2023.00035

[15] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2023. Chain of code: Reasoning with a language model-augmented code emulator. *arXiv preprint arXiv:2312.04474* (2023).

[16] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C C Chang, Fei Huang, Reynold Cheng, and Yongbin Li. [n. d.]. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. ([n. d.]). https://doi.org/10.48550/arXiv.2305.03111

[17] Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. 2024. Data-efficient Fine-tuning for LLM-based Recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 365–374. https://doi.org/10.1145/3626772.3657807

[18] M. Liu, J. Wang, T. Lin, Q. Ma, Z. Fang, and Y. Wu. 2024. An empirical study of the code generation of safety-critical software using llms. *Applied Sciences* 14 (2024), 1046. Issue 3. https://doi.org/10.3390/app14031046

[19] Lindung Parningotan Manik, Zaenal Akbar, Hani Febri Mustika, Ariani Indrawati, Dwi Setyo Rini, Agusdin Dharma Fefirenta, and Tutie Djarwaningsih. 2021. Out-of-Scope Intent Detection on A Knowledge-Based Chatbot. *International Journal of Intelligent Engineering & Systems* 14, 5 (2021).

[20] N. Marques. 2024. Using chatgpt in software requirements engineering: a comprehensive review. *Future Internet* 16 (2024), 180. Issue 6. https://doi.org/10.3390/fi16060180

[21] Dehai Min, Nan Hu, Rihui Jin, Nuo Lin, Jiaoyan Chen, Yongrui Chen, Yu Li, Guilin Qi, Yun Li, Nijun Li, et al. 2024. Exploring the impact of table-to-text methods on augmenting llm-based question answering with domain hybrid data. *arXiv preprint arXiv:2402.12869* (2024).

[22] Raphaël Mouravieff, Benjamin Piwowarski, and Sylvain Lamprier. 2024. Learning Relational Decomposition of Queries for Question Answering from Tables. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10471–10485. https://doi.org/10.18653/v1/2024.acl-long.564

[23] Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295* (2023).

[24] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. 2024. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296* (2024).

[25] Chau Minh Pham, Simeng Sun, and Mohit Iyyer. 2024. Suri: Multi-constraint instruction following for long-form text generation. *arXiv preprint arXiv:2406.19371* (2024).

[26] Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541* (2023).

[27] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).

[28] Boris van Breugel and Mihaela van der Schaar. 2024. Why tabular foundation models should be a research priority. *arXiv preprint arXiv:2405.01147* (2024).

[29] Kushala VM, Harikrishna Warrier, Yogesh Gupta, et al. 2024. Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations. *arXiv preprint arXiv:2404.10779* (2024).

[30] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[31] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[32] Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398* (2024).

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[34] Junyi Ye, Mengnan Du, and Guiling Wang. 2024. DataFrame QA: A Universal LLM Framework on DataFrame Question Answering Without Data Exposure. https://doi.org/10.48550/ARXIV.2401.15463 Version Number: 1.

[35] Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023. Tablellama: Towards open large generalist models for tables. *arXiv preprint arXiv:2311.09206* (2023).

[36] Zhuosheng Zhang, Yao Yao, Aston Zhang, Xiangru Tang, Xinbei Ma, Zhiwei He, Yiming Wang, Mark Gerstein, Rui Wang, Gongshen Liu, et al. 2023. Igniting Language Intelligence: The Hitchhiker's Guide From Chain-of-Thought Reasoning to Language Agents. *arXiv preprint arXiv:2311.11797* (2023).

[37] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473* (2024).