Gatelens: A reasoning-enhanced llm agent for automotive software release analytics

The Journal of Systems and Software 240 (2026) 112961

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

GateLens: A reasoning-enhanced LLM agent for automotive software release analytics☆

Arsham Gholamzadeh Khoee [a,b,*], Shuai Wang [a,b], Robert Feldt [a], Dhasarathy Parthasarathy [b], Yinan Yu [a,b]

[a] Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden
[b] Volvo Group, Gothenburg, Sweden

ARTICLE INFO

Keywords:
Large language models
Tabular question answering
Software release analytics
Automotive software testing
Test result interpretation
Interpretable reasoning

ABSTRACT

Ensuring reliable data-driven decisions is crucial in domains where analytical accuracy directly impacts safety, compliance, or operational outcomes. Decision support in such domains relies on large tabular datasets, where manual analysis is slow, costly, and error-prone. While Large Language Models (LLMs) offer promising automation potential, they face challenges in analytical reasoning, structured data handling, and ambiguity resolution. This paper introduces GateLens, an LLM-based architecture for reliable analysis of complex tabular data. Its key innovation is the use of Relational Algebra (RA) as a formal intermediate representation between natural-language reasoning and executable code, addressing the reasoning-to-code gap that can arise in direct generation approaches. In our automotive instantiation, GateLens translates natural language queries into RA expressions and generates optimized Python code. Unlike traditional multi-agent or planning-based systems that can be slow, opaque, and costly to maintain, GateLens emphasizes speed, transparency, and reliability. We validate the architecture in automotive software release analytics, where experimental results show that GateLens outperforms the existing Chain-of-Thought (CoT) + Self-Consistency (SC) based system on real-world datasets, particularly in handling complex and ambiguous queries. Ablation studies confirm the essential role of the RA layer. Industrial deployment demonstrates over 80% reduction in analysis time while maintaining high accuracy across domain-specific tasks. GateLens operates effectively in zero-shot settings without requiring few-shot examples or agent orchestration. This work advances deployable LLM system design by identifying key architectural features—intermediate formal representations, execution efficiency, and low configuration overhead—crucial for domain-specific analytical applications where accuracy, traceability, and stakeholder trust are paramount.

1. Introduction

Reliable decision-making in data-intensive domains depends on the ability to accurately analyze large volumes of structured data. In sectors such as automotive manufacturing, healthcare, finance, and regulatory compliance, critical decisions hinge on interpreting tabular datasets that capture test results, operational metrics, or validation records. These datasets often pass through gating steps—critical checkpoints where predefined quality or compliance standards must be met. Failures at these gates can cascade through interconnected processes, delaying dependent workflows regardless of their individual quality. Analysts tasked with safeguarding decision quality must process vast quantities of data. While essential for ensuring accuracy 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) (Liu et al., 2024; Chang et al., 2024). Companies are increasingly integrating these AI agents into their workflows, seeking more cost-effective and optimized solutions for complex software engineering tasks (Leung and Murphy, 2023). However, direct application of LLMs to structured data analysis for decision support is hindered by limitations in interpretable reasoning and understanding of technical specifications (Marques, 2024; Austin et al., 2021).

To address these challenges, we introduce *GateLens*, a reasoning-enhanced LLM agent (Miehling et al., 2025) for reliable tabular data

☆ Editor: Dr. Dario Di Nucci.

* Corresponding author at: Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden.
E-mail addresses: arsham.khoee@chalmers.se (A.G. Khoee), shuaiwa@chalmers.se (S. Wang), robert.feldt@chalmers.se (R. Feldt), dhasarathy.parthasarathy@volvo.com (D. Parthasarathy), yinan@chalmers.se (Y. Yu).

https://doi.org/10.1016/j.jss.2026.112961
Received 30 November 2025; Received in revised form 1 March 2026; Accepted 18 May 2026
Available online 23 May 2026

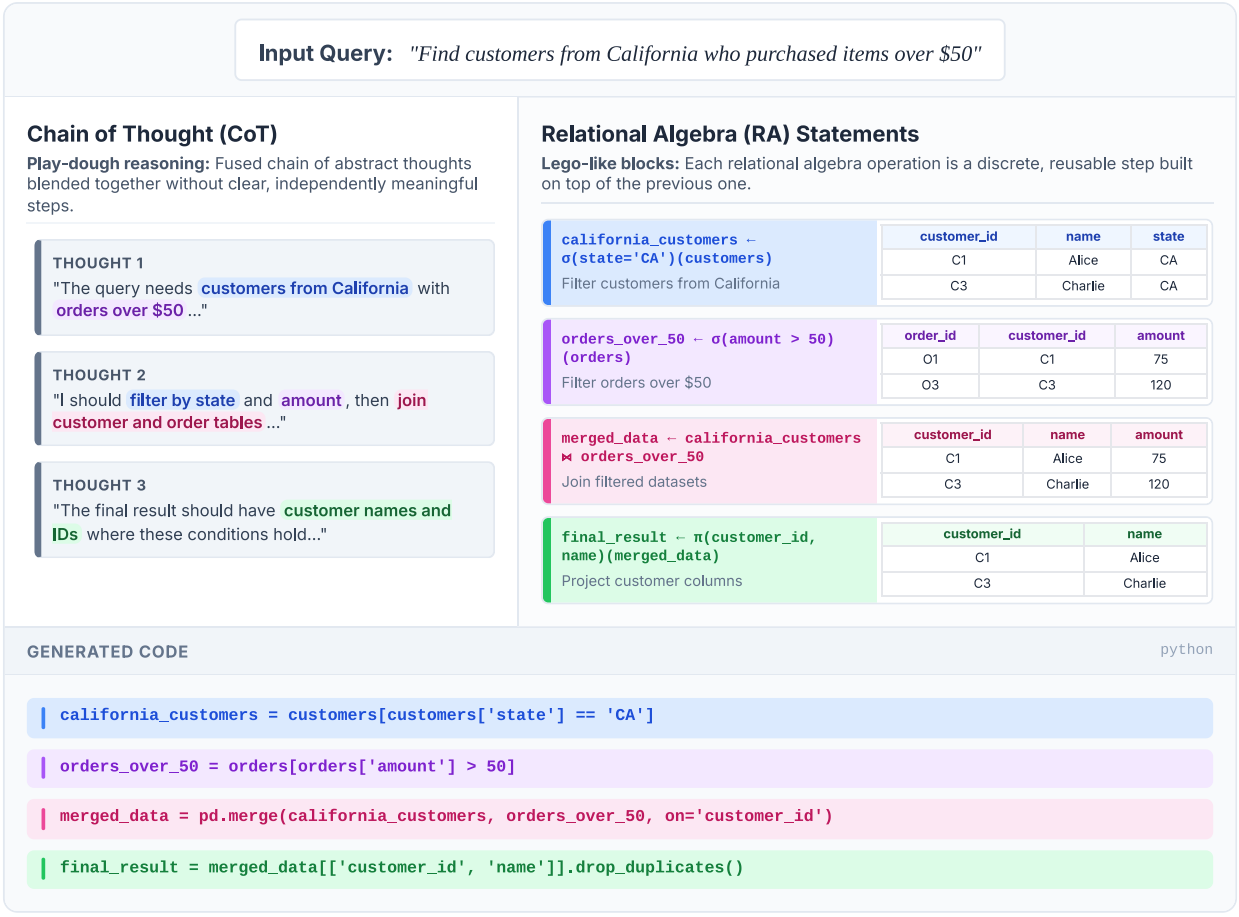


Fig. 1. Comparison of conventional Chain-of-Thought (CoT) reasoning and its extension using Relational Algebra (RA) statements. (Left) CoT employs unstructured, fused reasoning where multiple analytical concepts are blended together in informal thoughts without clear separation. Operations cannot be independently mapped to executable code snippets, making reasoning steps opaque and difficult to debug in isolation. (Right) RA-based reasoning adopts a structured, compositional approach where each relational algebra operation (filtering, joining, projecting) is a discrete, formally grounded step. Each operation is independently interpretable, reusable, and can be directly mapped to executable code. The color-coding illustrates how RA maintains clear boundaries between distinct analytical steps, whereas CoT conflates multiple operations within single reasoning thoughts. This structural distinction enables transparent, formally grounded reasoning that can be systematically verified and debugged at each stage.

analysis to support decision-making in domain-specific contexts. We validate the architecture in the automotive software release domain, where the need for precise, transparent, and efficient analysis is particularly acute.

A key challenge in LLM-based analytical agents is the potential mismatch between natural-language reasoning and the actual computation implemented in generated code—a reasoning-to-code gap that becomes more pronounced as analytical complexity increases. GateLens addresses this challenge by integrating structured relational analysis with domain-specific expertise through a reasoning layer built on Relational Algebra (RA) as an extension of Chain-of-Thought (CoT). This reasoning layer systematically breaks down complex validation tasks into discrete, formally grounded analytical steps. We adopt this approach to address essential limitations of vanilla CoT for our work: its reasoning steps are opaque and often cannot be mapped directly to executable code, its operations are not compositional and cannot be independently reused or debugged, and its reasoning is unstructured with operations blended together without clear intent or formal grounding. In contrast, our RA-based reasoning layer aims to ensure that each step is independently interpretable and reusable, reasoning is grounded in formal relational algebra, and intent is made explicit through well-defined operations. Fig. 1 illustrates this distinction between our structured RA-based approach and conventional CoT reasoning.

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 200 ms brake signal delay that exceeds the 100 ms 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:

- We design an *architecture optimized for time- and safety-critical environments*, minimizing LLM invocations while preserving reasoning depth for reliable tabular analysis. GateLens operates in a zero-shot setting, avoiding the need for few-shot examples or multi-agent coordination, which improves generalizability, execution speed, reduces maintenance overhead, and enhances transparency.
- We introduce a *scalable and maintainable framework for automotive software release validation*, developed in response to observed limitations in traditional planning-based multi-agent system. GateLens handles diverse user queries with higher robustness and clarity, supporting effective decision-making across a wider range of release engineering tasks.
- We conduct a *comprehensive empirical evaluation*, including comparisons with a CoT+SC system, ablation of the RA module, and performance across multiple LLMs (GPT-4o and Llama 3.1 70B). These experiments demonstrate GateLens's performance advantages in complex and ambiguous queries.
- We report on *real-world industrial deployment* in a partner automotive company, where GateLens reduces analysis time by over 80% and demonstrates strong generalization across user roles, highlighting its practical value and deployment-readiness in safety-critical release validation workflows.

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.

This diversity is reflected not only in perspective but also in the level of granularity required from the underlying data. For example, senior management may ask high-level questions such as: "List all vehicles that have not yet received global approval", seeking a consolidated overview without reference to schema details or subsystem-specific attributes. In contrast, release managers or verification engineers may pose highly specific queries such as: "What are the baseline, phase, RC, and EUF values for RM-320 in the latest test suite?" These questions rely on detailed knowledge of domain terminology and schema structure. Although both types of queries operate on the same data foundation, they demand substantially different views and levels of abstraction.

Release managers function as gatekeepers in the software deployment pipeline. They handle 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.

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 (L. Wang et al., 2024). 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.

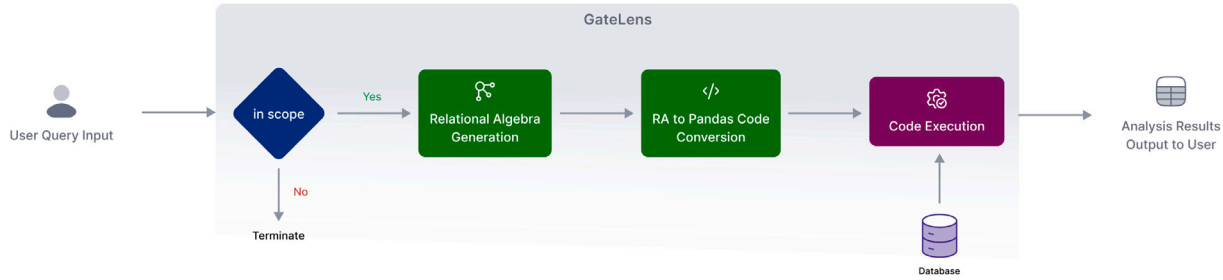


Fig. 2. GateLens top-level architecture: The system processes high-level queries from the end user, generates the necessary data manipulation code using the enhanced reasoning layer with the help of RA, executes it, and outputs the result table as a decision-support resource.

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 Fig. 2, 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 Fig. 3.

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, field descriptions, data types, and enriched metadata capturing domain-specific acronyms and terminology mappings. This glossary-enhanced schema is injected into the prompt context, enabling accurate resolution of domain-specific terms into formal schema attributes during RA construction.

To handle imprecise queries without compromising data privacy or exceeding LLM context windows, GateLens employs a selective strategy for exposing categorical information. For low-cardinality attributes (e.g., fields with only a few distinct categories such as test status), all valid options are explicitly enumerated within the schema metadata. In contrast, for high-cardinality attributes, actual database values are never exposed; instead, the schema specifies structural patterns or expected formats (e.g., standard prefixes or identifier conventions). This hybrid design keeps the prompt context compact and privacy-preserving while providing the LLM with sufficient context to generate accurate filtering conditions.

Using this information, the agent verifies whether the query is relevant and within the scope of the dataset (Manik et al., 2021). 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 data schemas 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 (Khot et al., 2022). This approach extends traditional Chain-of-Thoughts (CoT) (Wei et al., 2022) reasoning by constraining the model to think within a formal system framework (Zhang et al., 2023). Instead of generating free-form solutions, the agent must express analytics using standard relational algebra notations 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

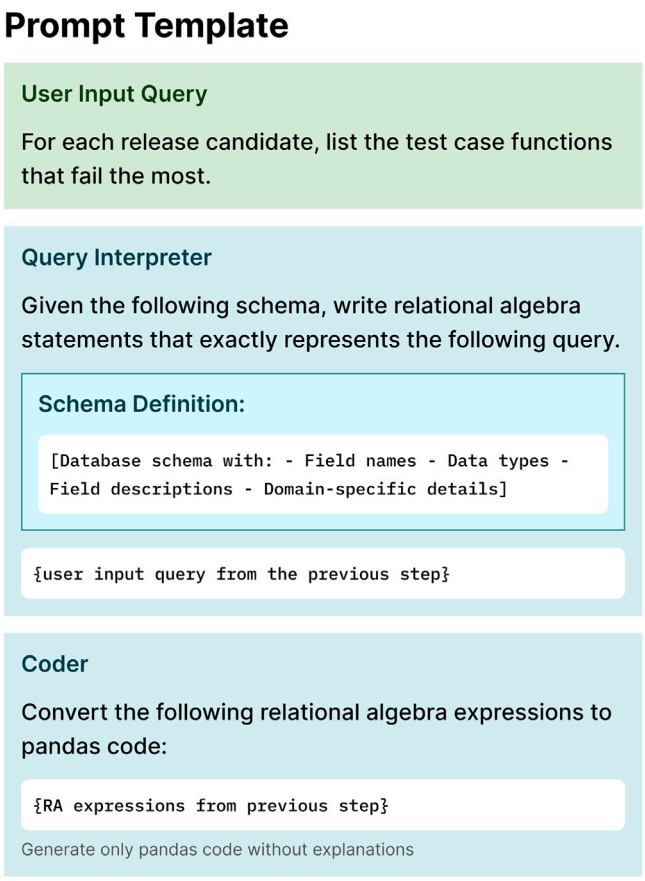


Fig. 3. Overview of the prompt engineering flow and prompt structure within the GateLens system architecture.

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.

To ensure the generated code meets quality standards, we designed prompts that specifically instruct the LLM to include essential validation mechanisms. The prompts explicitly require data type validation instructions for numerical and categorical field handling, null value checking procedures to maintain data integrity, validation of numerical operations, and validation of join conditions to ensure proper matching of key columns between tables.

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 yet precise approach ensures both efficiency and reliability in the analysis pipeline, fully aligned with the formal rigor of RA expressions and improving overall 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 LLM agents, which could raise privacy concerns (Boudewijn et al., 2023) 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
- **Scalability:** The system can handle large-scale test datasets that would exceed LLM context windows
- **Knowledge Integration:** Data schemas and relational models serve as an essential 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: To what extent does RA-based reasoning eliminate the need for in-context learning?

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. To assess GateLens's performance across a spectrum of query complexities, these queries are categorized into four difficulty levels. 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 at our partner company. These queries were sourced from the historical logs of an agentic system that employed a well-established tabular data reasoning approach combining CoT (Wei et al., 2022) prompting with Self-Consistency (SC) (Wang et al., 2022). This system was used in production to support software release analytics, and the collected queries reflect a wide range of user roles, query types, and domain-specific requirements. While this system performs effectively in many scenarios, its limitations become apparent as the range of roles and users expands, leading to a significant diversification of queries. This broader query diversity exposes the system'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. Nevertheless, this preliminary system played a critical role in data collection for GateLens by providing query logs used to develop and validate our approach. From these logs, we filtered out near-duplicates and selected 244 frequently repeated unique queries, which we then organized into eight functional categories based on their purposes. The ground truth for the 244 real-world queries was established through a two-stage manual annotation process: two domain experts independently solved an initial subset of 20 queries to align on interpretation and expected outputs, after which the remaining queries were divided and annotated following the agreed-upon criteria, with periodic cross-checks to ensure consistency.

Table 1
Comparison of average token consumption and reduction between CoT+SC and GateLens across four difficulty levels. The evaluation was conducted on the first benchmark, which consists of 50 designed queries with annotated difficulty levels, with both agents utilizing GPT-4o.

Level	# Queries	GateLens with GPT-4o			GateLens with Llama 3.1 70B			CoT+SC with GPT-4o			CoT+SC 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 the CoT+SC system 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			CoT+SC 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%**

In order to assess GateLens's performance, we run experiments with two large language models (LLMs): GPT-4o, a leading commercial model, and Llama 3.1 70b, a recently released open-source model. We also benchmark GateLens against the CoT+SC (Khoee et al., 2024) agentic system currently used to support the company's release decisions. Our comparative analysis is designed to quantify the improvements introduced by GateLens's novel architecture.

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, specifically when the final output of the executed code matches the expected ground-truth output. A **False Positive (FP)** occurs when the system provides an incorrect result, meaning the executed code produces output that differs from the ground-truth. A **False Negative (FN)** occurs when the system fails to provide any result for a query. **True Negatives (TNs)** are not applicable since we focus on valid queries producing 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. By relying on a closed-set benchmark with established ground truths, these metrics enable us to rigorously isolate and measure the impact of our architectural choices during lab validation, setting the stage for the real-world industrial evaluation detailed in Section 6.

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 CoT+SC. 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 for interpreting and generating RA. Similarly, CoT+SC with GPT-4o outperforms its Llama 3.1 70B variant, with the performance gap growing as query complexity increases. CoT+SC performance declines with query complexity. This underscores the importance of the RA reasoning mechanism in GateLens, which enables effective handling of complex, unstructured queries by decomposing them into logical, structured expressions. Most notably, GateLens with GPT-4o achieved optimal performance on this benchmark, maintaining 100% accuracy across all difficulty levels. This stems from integrating RA reasoning into our framework. By translating queries into RA expressions, GateLens explicitly captures the logical structure of operations, enhancing both the clarity and precision of the generated code. The intermediate RA conversion allows the system to focus on the relevant table operations while filtering out irrelevant elements in the query, greatly enhancing the problem-solving capabilities of the LLM agent.

For the second benchmark, results in Table 2 show that GateLens (GPT-4o) and CoT+SC (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.

GateLens with GPT-4o outperformed CoT+SC (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", the CoT+SC system often fails to identify the correct field. A common failure mode in CoT+SC 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

Table 3
Comparison of average token consumption and reduction between CoT+SC and GateLens across four difficulty levels. The evaluation was conducted on the first benchmark, which consists of 50 designed queries with annotated difficulty levels, with both agents utilizing GPT-4o.

Level	Agent	Avg input tokens	Avg output tokens	Avg total tokens	Token reduction
1	CoT+SC	11,905	186	12,091	–
	GateLens	2239	420	2658	↓ 78%
2	CoT+SC	13,747	368	14,116	–
	GateLens	2428	701	3129	↓ 78%
3	CoT+SC	14,726	441	15,168	–
	GateLens	2432	698	3130	↓ 79%
4	CoT+SC	17,103	452	17,555	–
	GateLens	2505	847	3352	↓ 81%

fuzzy matching to detect and correct field names, enabling the system to resolve typographical errors and ambiguous queries effectively. This approach improves accuracy and resilience, particularly in real-world scenarios where user queries may not always adhere to strict formatting standards.

In addition, compared to the CoT+SC solution, GateLens uses significantly fewer tokens due to its effective use of RA as the intermediate representation and its zero-shot architecture. In Table 3, we compare the token usage of both systems across different difficulty levels. Notably, while the CoT+SC approach requires increasingly massive input contexts, primarily to accommodate few-shot examples, GateLens maintains a highly compact input footprint. As the query difficulty increases, the token reduction becomes even more significant, demonstrating the efficiency and scalability of the RA-based approach in handling complex queries.

RQ1 findings: GateLens with GPT-4o achieved 100% F1 score on the first benchmark across all difficulty levels and 83.51% F1 score on the second benchmark with 244 real-world queries. It outperformed CoT+SC (GPT-4o) by approximately 13 percentage points on real-world queries, indicating that the RA reasoning mechanism effectively addresses both complex logical operations and real-world noise, such as typos and ambiguous field names.

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 4. 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 4
Model comparison 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%
CoT+SC with GPT-4o	51.10%	89.19%	64.97%

Table 5
Model comparison 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%
CoT+SC with GPT-4o	90%	36%	51.43%

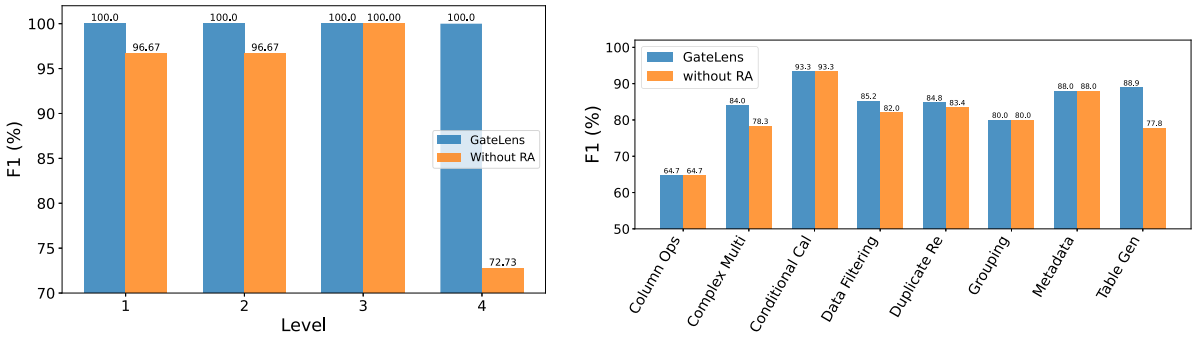
The superior precision of GateLens with GPT-4o can be attributed to two key aspects of its design. First, its robust filtering mechanism ensures that out-of-scope queries are identified and excluded early in the processing pipeline, preventing irrelevant results. Second, the conversion of raw natural 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.

CoT+SC showed significantly lower precision due to the variability of real-world queries and the inconsistency of user narratives, which often contain a mix of relevant and irrelevant content. This variability increases uncertainty and poses challenges for models that struggle to identify task-relevant information. Although all models demonstrated high recall, this did not translate into accurate processing.

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 5. As shown, GateLens with GPT-4o 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 substantially higher F1 score compared to other methods, demonstrating that it not only processes most queries but also produces accurate results for them.



(a) The first benchmark with annotated difficulty levels.

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

Fig. 4. Comparison of the original method and the method without the RA module across different datasets.

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 execution and reduced accuracy. On the other hand, CoT+SC exhibits low recall, as it is highly susceptible to confusion by ambiguous query elements. This causes CoT+SC to frequently generate incorrect code that fails execution, significantly lowering its recall rate.

RQ2 findings: GateLens demonstrates high robustness in real-world conditions. It effectively filters out-of-scope inputs, achieving approximately 40% higher precision than the baseline system. Furthermore, it handles imprecise or informal queries with superior performance, more than doubling the recall (78% vs. 36%). This confirms that the system can manage ambiguous user inputs without sacrificing the accuracy of the generated code.

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 Fig. 4, 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 key bridge between user intent and code execution. This translation process enables the code generator to produce accurate, efficient executable code for data analysis tasks.

RQ3 findings: The RA module is a critical component for handling query complexity. Removing it results in substantial performance degradation (over 27% for complex queries), confirming that translating natural language to RA provides necessary structural guidance for accurate code generation.

4.5. Role 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 CoT+SC. This experiment is performed on the first benchmark containing 50 designed queries, with results illustrated in Fig. 5.

The results demonstrate that GateLens relies heavily on its RA translation process, achieving 100% F1 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 (dropping to 95.92% at 2 examples). However, it regains optimal performance at 3 examples and remains at 100% thereafter. In contrast, CoT+SC's performance heavily depends on in-context examples, achieving only 42% F1 with 2 examples and showing suboptimal results without sufficient few-shot examples. Performance improved steadily with more examples, reaching 84.57% at 50 examples.

RQ4 findings: GateLens does not require many-shot examples to achieve high performance, delivering optimal results in a 0-shot setting. This contrasts with CoT+SC, which depends on carefully curated examples. This independence from few-shot examples makes GateLens more efficient for real-time applications by reducing context size and computational overhead.

5. Qualitative assessment examples

To provide a closer view into the inner workings of the GateLens framework, we analyze two representative examples. Each example presents the relational algebra solution generated from the input query using a domain-specific relational model, the corresponding executable code created from the relational algebra statements, and the resulting output after execution.

Table 6 represents an example that produces the desirable result. The agent correctly decomposes the input query into a sequence of relational algebra statements that include filtering, grouping by release candidate and function, counting entries with the status "NotRun", joining the intermediate results, and finally computing the relative frequency. This process shows how the system is capable of understanding the semantics of the query, applying the required relational transformations, and leveraging the domain-specific schema to produce the correct result. This stepwise reasoning lays the foundation for semantic decomposition and precise query interpretation, which then naturally translates into valid executable code.

Table 7 is an example that illustrates a failed case. Although the relational algebra translation is semantically sound, the resulting code does not correctly apply projection—it returns all occurrences of the

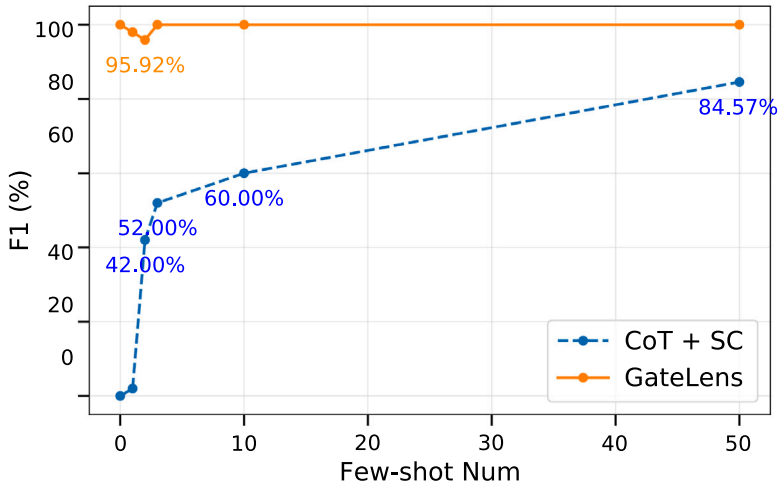


Fig. 5. Comparison of GateLens against CoT+SC across different numbers of few-shot examples.

Table 6
An example of an input query and its corresponding relational algebra expressions translated using a domain-specific database schema, along with the executable code generated from the relational algebra and the resulting output produced when executed over the database.

Stage	Content
Input Query	*Compute the relative frequency of not run tests for each RC and function*
Relational Algebra	**Step 1:** $R0 \leftarrow \sigma_{\text{result\_status="NotRun"}}(\text{df})$ **Step 2:** $R1 \leftarrow \gamma_{\text{releasecandidate,function,countNotRun}\rightarrow\text{count}}(R0)$ **Step 3:** $R2 \leftarrow \gamma_{\text{releasecandidate,function,totalcount}\rightarrow\text{count}}(\text{df})$ **Step 4:** $R3 \leftarrow R1 \bowtie_{\text{R1.releasecandidate=R2.releasecandidate} \wedge \text{R1.function=R2.function}} R2$ **Step 5:** Result $\leftarrow \gamma_{\text{R3.releasecandidate,R3.function,relativeFrequency}\rightarrow \frac{\text{R3.count}}{\text{R3.totalcount}}}(R3)$

Python Code	`# Step 1: Select rows where result_status is 'NotRun'`
	`R0 = df[df['result_status'] == 'NotRun']`
	
	`# Step 2: Group by releasecandidate and function, count 'NotRun' entries`
	`R1 = R0.groupby(['releasecandidate', 'function']).size().reset_index(name='count')`
	
	`# Step 3: Group all rows by releasecandidate and function, count total entries`
	`R2 = df.groupby(['releasecandidate', 'function']).size().reset_index(name='totalcount')`
	
	`# Step 4: Join R1 and R2 on releasecandidate and function`
	`R3 = pd.merge(R1, R2, how='inner', left_on=['releasecandidate', 'function'], right_on=['releasecandidate', 'function'])`
	
	`# Step 5: Compute relative frequency of 'NotRun' tests`
	`R3['relativeFrequency'] = R3['count'] / R3['totalcount']`
	`result = R3[['releasecandidate', 'function', 'relativeFrequency']]`

Output	**RC** **Function** **Freq** ALIEN func_001 0.833333 ALIEN func_002 0.142857 ALIEN func_003 0.125000 ALIEN func_004 0.142857 PACMAN func_001 0.325000 ⋮ ⋮ ⋮ Total rows: 3662

same baseline value rather than the distinct value expected by the relational algebra. This issue arises because the plan omits the duplicate-elimination operator (δ), which should be used to ensure the final result contains unique entries. As a result, while the output does not match the intended result, it is still potentially useful to the end user, as it contains relevant baseline information.

Overall, these examples highlight how the reasoning-enhanced LLM agent supports accurate and explainable code generation by decomposing user queries into relational algebra operations that align closely with domain-specific requirements. The relational modeling serves as a crucial guide for transforming natural language queries into semantically coherent, executable plans.

6. Industrial deployment: Lessons learned

The deployment of GateLens at a partner automotive company has provided valuable insights into integrating AI-assisted analytics into

Table 7
An example of an incorrect implementation where the relational algebra translation is semantically correct but the generated code fails to properly implement the projection operation, returning all occurrences of the baseline value rather than the distinct value as specified by the relational algebra.

Stage	Content
Input Query	*What is the latest baseline on RM-320?*
Relational Algebra	**Step 1:** $R0 \leftarrow \sigma_{\text{name="RM-320"}}(\text{df})$ **Step 2:** $\text{max\_date} \leftarrow \text{MAX}(\pi_{\text{date}}(R0))$ **Step 3:** $R1 \leftarrow \sigma_{\text{date=max\_date}}(R0)$ **Step 4:** $\text{Result} \leftarrow \pi_{\text{baseline}}(R1)$

Python Code	`# Step 1: Select rows where the truck name is RM-320`
	`R0 = df[df['name'] == 'RM-320']`
	
	`# Step 2: Identify the latest date from the selected rows`
	`max_date = R0['date'].max()`
	
	`# Step 3: Select rows that have this latest date`
	`R1 = R0[R0['date'] == max_date]`
	
	`# Step 4: Project the baseline field`
	`result = R1['baseline'].reset_index(drop=True)`

Output	**Baseline** 23_T2_w2509_ABC_12 23_T2_w2509_ABC_12 23_T2_w2509_ABC_12 23_T2_w2509_ABC_12 23_T2_w2509_ABC_12 ⋮ Total rows: 132 (all duplicates)

Table 8
GateLens (zero-shot) vs CoT+SC (few-shot) performance across different roles on the second benchmark. For each role tested, CoT+SC was trained using examples from the other two roles only (leave-one-role-out approach).

Roles	# Queries	GateLens	CoT+SC (Few-shot with All Roles)	CoT+SC (Few-shot with Leave-One-Role-Out)		
	(244 in total)	F1 Score	F1 Score	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%)

complex industrial workflows, specifically for streamlining decision-making in automotive software release validation.

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 bottlenecks and delays in the decision-making process.

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 more efficient decision-making. 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 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.

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.

As the system was opened to a broader audience, the diversity of query types increased substantially. While the initial CoT+SC-based agent performed well for a relatively homogeneous user group, its performance became increasingly sensitive to the coverage of few-shot examples. Approaches that rely heavily on few-shot prompting are inherently constrained by example selection and may struggle with previously unseen query patterns. This sensitivity limits their scalability in dynamic industrial environments where new query types continuously emerge. For this reason, we prioritized architectural choices that improve robustness and generalization across roles and query styles.

To explore the system's generalizability, we categorized the roles within the company into three groups: mechanically-oriented, project-oriented, and software-oriented roles. 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 8, both GateLens (zero-shot) and CoT+SC (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 CoT+SC'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 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. Crucially, the reported 80% reduction reflects an operational end-to-end improvement that includes the time engineers spend verifying system outputs. In safety-critical industrial environments, AI-generated results are never accepted without validation; therefore, laboratory correctness metrics alone are insufficient to assess real-world impact. A traditional LLM pipeline that directly translates a natural-language query into executable code and returns a final result is not practical in this context. When such black-box outputs are cross-checked against existing dashboards—a natural and common validation strategy—any discrepancies become difficult to diagnose. If the reasoning process is opaque, engineers cannot trace the source of the error, which directly undermines trust and limits usability. On the other hand, when the reasoning process is expressed in natural language, it becomes possible to follow where things went wrong, but precise intervention remains difficult because natural language is inherently fuzzy and imprecise.

In contrast, GateLens generates an explicit relational algebra (RA) plan that decomposes the input query into logical, stepwise building blocks before producing executable code. This intermediate representation mirrors how experienced developers would structure complex analyses. As a result, engineers can inspect whether the decomposed plan is logically sound before or alongside reviewing the final output. This makes discrepancies diagnosable rather than opaque; the RA-based intermediate representation decreases the effort required for validation and actively builds user trust over time. Consequently, 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 (Qiao et al., 2023) or AutoGen (Wu et al., 2023), 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.

Post-deployment monitoring revealed practical failure modes that highlight challenges of LLM adoption in industrial settings. Most issues did not stem from complex reasoning errors, but from ambiguities in user queries. The two most common cases were: (i) implicit constraints, where users assumed a specific test environment or time frame without stating it explicitly, and (ii) highly localized team jargon that led to incorrect schema mappings. To address these issues, we systematically documented recurring patterns, refined prompt guidelines, and extended the schema metadata with an explicit jargon glossary, incorporated as domain-specific context to improve term-to-column alignment. This process also repositioned the central analytics team toward supporting clearer, more explicit query formulation.

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.

7. 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 (Fang et al., 2024; Z. Wang et al., 2024; van Breugel and van der Schaar, 2024). First, the structured tabular data is typically transformed into serialized text. The performance of the LLM may depend on this transformation (Min et al., 2024). Subsequently, the serialized text data is used as input to the LLM for various tasks, such as question-answering, summarization, or logical reasoning. Common approaches to improve LLM performance include prompt engineering, pre-training, fine-tuning, and Retrieval-Augmented Generation (RAG).

Pre-training and fine-tuning (Zhang et al., 2024; Parthasarathy et al., 2024; VM et al., 2024; Dong et al., 2022; Hegselmann et al.) often face scalability concerns. Although resource-efficient training techniques have been proposed to mitigate the substantial computational demands of LLMs (Han et al., 2024; Lin et al., 2024), 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 (Zhao et al., 2024; Gao et al., 2023). 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. Costs would also be especially high with fine-tuning since re-tuning would be needed when new and improved base LLMs are released and should be incorporated.

Prompt engineering techniques are among the most resource-efficient methods for improving LLM output (Sahoo et al., 2024; Jin and Lu, 2023). 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 (Li et al., 2023), 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 (Pham et al., 2024). 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) (Ye et al., 2024), query languages (e.g. SQL) (Li et al.; Dong et al., 2023; Mouravieff et al., 2024), configuration formats (e.g. YAML or JSON), or other Domain-Specific Languages (DSLs) (Glenn et al., 2024; Dai et al., 2024). These languages 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 (Pan et al., 2023).

In this paper, we introduce a novel prompt-only (training-free) approach that bridges natural language and executable code through RA, a symbolic formalism designed for relational modeling and ideally suited for analyzing tabular data. Unlike prior work that often relies on complex multi-agent planning, our approach leverages RA as a lightweight intermediate representation to enable precise query normalization, disambiguation of natural language input, and efficient code generation. RA acts as an abstraction layer that can target multiple execution backends (e.g., Python, SQL), providing adaptability across systems. In GateLens, we generate Python code to support practical industrial deployment and high-performance execution. GateLens is *training-free*, *feed-forward* (single-pass, without looping or multi-agent orchestration), and thus easier to verify, trace, maintain, and trust — qualities critical for safety-critical industrial applications.

8. Discussion and conclusions

This study introduced GateLens, a reasoning-enhanced LLM architecture for reliable tabular analysis, applied to domain-specific software release validation in the automotive industry. By introducing RA as an intermediate representation before code generation, GateLens addresses the "Unfaithful Chain-of-Thought (CoT) reasoning" problem in code generation, where reasoning steps in CoT explanations do not accurately reflect the model's actual thought process (Turpin et al., 2023). Specifically, GateLens divides the analysis into two steps: (1) natural language queries are first translated into RA expressions, and (2) these RA expressions are then converted into executable code. We use Python as our target language in step (2) due to its widespread use in our partner company and the model's strong performance in Python, which benefits from more extensive training data. However, this step is also compatible with RA-to-SQL generation, enabling flexibility across backends. The inclusion of the RA reasoning module is a critical factor in improving robustness and scalability, as evidenced by superior F1 scores in both benchmarking and industrial evaluations.

Our findings regarding few-shot learning (RQ4) highlight a notable architectural advantage of GateLens over conventional few-shot learning approaches. While CoT+SC's performance improves with more examples, this approach introduces several practical and technical challenges. Increasing example count expands input context size, which increases inference time and computational cost due to the quadratic complexity of Transformer-based models—particularly problematic for real-time and resource-constrained applications (Cui et al., 2025; Agarwal et al., 2024). This also escalates operational costs through increased token usage (higher cloud API fees) and computational demands (Cui et al., 2025). Additionally, the quality and selection of examples significantly impact performance (Huang et al., 2023); poorly chosen or noisy examples can degrade reasoning and lead to overfitting or failure to generalize on complex tasks (Qin et al., 2024; Zhou et al., 2024). As more examples are added, the risk of including irrelevant or contradictory rationales amplifies, potentially confusing the model and reducing accuracy (Cui et al., 2024; Zhou et al., 2024). Moreover, the larger context risks exceeding the maximum length, which can lead to truncation or lost information (Agarwal et al., 2024), compromising the model's response quality. In long contexts, critical content may be ignored, resulting in a phenomenon known as "lost in the middle", which adversely affects overall performance (Liu et al., 2023). Finally, crafting high-quality, task-specific CoT examples is labor-intensive (Tai et al., 2023; Stechly et al., 2024), and while many-shot in-context learning (ICL) may improve performance on specific tasks, generalization to new tasks remains limited without careful prompt engineering. GateLens circumvents these issues by achieving optimal performance in a zero-shot setting, relying on logical RA translation rather than using manually designed many-shot examples. By maintaining a compact input footprint, GateLens reduces average total token consumption by approximately 79% compared to the CoT+SC baseline in our production logs, leading to proportionally lower inference latency and API usage costs.

While our lab-based quantitative metrics provide a critical, controlled baseline for comparing architectural choices and measuring robustness against ground truths, we acknowledge that precision and recall on curated queries cannot guarantee absolute correctness in open-ended deployment. Therefore, these lab validations serve primarily to complement—rather than replace—our industrial evaluation. In real-world deployment serving 60–80 users, GateLens demonstrates significant practical value through its user-friendly interface and robust query processing capabilities, with users particularly appreciating the flexibility to input, debug, and refine queries easily. This marks a substantial advancement in industrial data interaction, successfully handling complex and ambiguous queries while providing practical support for faster decision-making in safety-critical software release processes. GateLens demonstrates significant practical advancements by reducing analysis time by over 80% while maintaining high accuracy in test result interpretation, impact assessment, and release candidate evaluation.

Our implementation insights highlight the advantages of focusing on training-free and single-pass agent systems by foregrounding the perception phase in the code generation pipeline. This modular architecture opens opportunities for incorporating emerging LLM capabilities while preserving the system's practical utility in safety-critical industrial applications. A phased deployment program is ongoing and shows that multiple stakeholder groups can be supported, though the evolving roles and analytical needs require continued refinement.

Although instantiated in the automotive release domain, the GateLens architecture can be applied to other domains with comparable analytical requirements. Its core components—the RA-based intermediate reasoning layer, the separation between query interpretation and code generation, in-scope validation, and zero-shot operation without reliance on few-shot examples—do not depend on automotive-specific properties. Adapting the system to a new domain primarily requires replacing the schema and domain knowledge base, while preserving the architectural reasoning pipeline. Domains characterized by structured tabular data, complex stakeholder queries, and safety- or compliance-critical decision-making (e.g., healthcare analytics, financial auditing, or certification workflows) share these properties, suggesting broader applicability.

Future work will focus on validating this architectural transferability and testing alternative LLM configurations to enhance reliability across other safety-critical industries. By bridging the gap between flexible natural language interaction and rigorous analytical standards, this approach demonstrates the potential for reasoning-enhanced LLMs to transform industrial workflows across a broad spectrum of critical applications.

9. Threats to validity

The validity of our findings is subject to several potential threats. First, this framework depends on well-defined, static schemas for accurate query decomposition, which fundamentally limits its effectiveness in environments with incomplete or frequently changing schemas. 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 depends on the specific LLM configurations used, such as GPT-4o and Llama 3.1 70B, and their ability to interpret and generate RA expressions. Future work will address these limitations through broader domain testing, expanded evaluation scenarios, and alternative LLM configurations.

CRediT authorship contribution statement

Arsham Gholamzadeh Khoee: Writing – original draft. **Shuai Wang:** Writing – original draft. **Robert Feldt:** Validation, Supervision. **Dhasarathy Parthasarathy:** Data curation, Conceptualization. **Yinan Yu:** Writing – review & editing, Visualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, Sweden, and by the Gender Initiative for Excellence (Genie) at Chalmers University of Technology, funded by the Chalmers University Foundation.

Data availability

The data that has been used is confidential.

References

Agarwal, R., Singh, A., Zhang, L., Bohnet, B., Rosias, L., Chan, S., Zhang, B., Anand, A., Abbas, Z., Nova, A., et al., 2024. Many-shot in-context learning. Adv. Neural Inf. Process. Syst. 37, 76930–76966.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al., 2021. Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

Boudewijn, A.T.P., Ferraris, A.F., Panfilo, D., Cocca, V., Zinutti, S., De Schepper, K., Chauvenet, C.R., 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.

van Breugel, B., van der Schaar, M., 2024. Why tabular foundation models should be a research priority. arXiv preprint arXiv:2405.01147.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al., 2024. A survey on evaluation of large language models. ACM Trans. Intell. Syst. Technol. 15 (3), 1–45.

Cui, Y., He, P., Tang, X., He, Q., Luo, C., Tang, J., Xing, Y., 2024. A theoretical understanding of chain-of-thought: Coherent reasoning and error-aware demonstration. arXiv preprint arXiv:2410.16540.

Cui, Y., He, P., Zeng, J., Liu, H., Tang, X., Dai, Z., Han, Y., Luo, C., Huang, J., Li, Z., et al., 2025. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. arXiv preprint arXiv:2502.13260.

Dai, H., Wang, B., Wan, X., Dai, B., Yang, S., Nova, A., Yin, P., Phothilimthana, M., Sutton, C., Schuurmans, D., 2024. UQE: A query engine for unstructured databases. Adv. Neural Inf. Process. Syst. 37, 29807–29838.

Dong, H., Cheng, Z., He, X., Zhou, M., Zhou, A., Zhou, F., Liu, A., Han, S., Zhang, D., 2022. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. arXiv preprint arXiv:2201.09745.

Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J., Lou, D., et al., 2023. C3: Zero-shot text-to-sql with chatgpt. arXiv preprint arXiv:2307.07306.

Fang, X., Xu, W., Tan, F.A., Zhang, J., Hu, Z., Qi, Y.J., Nickleach, S., Socolinsky, D., Sengamedu, S., Faloutsos, C., et al., 2024. Large language models (LLMs) on tabular data: Prediction, generation, and understanding-a survey.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H., 2023. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997.

Glenn, P., Dakle, P.P., Wang, L., Raghavan, P., 2024. Blendsql: A scalable dialect for unifying hybrid question answering in relational algebra. arXiv preprint arXiv:2402.17882.

Han, Z., Gao, C., Liu, J., Zhang, J., Zhang, S.Q., 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. arXiv preprint arXiv:2403.14608.

Hegselmann, S., Buendia, A., Lang, H., Agrawal, M., Jiang, X., Sontag, D., TabLLM: Few-shot Classification of Tabular Data with Large Language Models. http://dx.doi.org/10.48550/arXiv.2210.10723.

Huang, X., Zhang, L.L., Cheng, K.-T., Yang, F., Yang, M., 2023. Fewer is more: Boosting llm reasoning with reinforced context pruning. arXiv preprint arXiv:2312.08901.

Jin, Z., Lu, W., 2023. Tab-cot: Zero-shot tabular chain of thought. arXiv preprint arXiv:2305.17812.

Khoee, A.G., Yu, Y., Feldt, R., Freimanis, A., Rhodin, P.A., Parthasarathy, D., 2024. Gonogo: An efficient LLM-based multi-agent system for streamlining automotive software release decision-making. In: IFIP International Conference on Testing Software and Systems. Springer, pp. 30–45.

Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., Sabharwal, A., 2022. Decomposed prompting: A modular approach for solving complex tasks. arXiv preprint arXiv:2210.02406.

Leung, M., Murphy, G., 2023. On automated assistants for software development: the role of llms. In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 1737–1741. http://dx.doi.org/10.1109/ase56229.2023.00035.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K.C.C., Huang, F., Cheng, R., Li, Y., Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. http://dx.doi.org/10.48550/arXiv.2305.03111.

Li, C., Liang, J., Zeng, A., Chen, X., Hausman, K., Sadigh, D., Levine, S., Fei-Fei, L., Xia, F., Ichter, B., 2023. Chain of code: Reasoning with a language model-augmented code emulator. arXiv preprint arXiv:2312.04474.

Lin, X., Wang, W., Li, Y., Yang, S., Feng, F., Wei, Y., Chua, T.-S., 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. SIGIR '24, Association for Computing Machinery, New York, NY, USA, pp. 365–374. http://dx.doi.org/10.1145/3626772.3657807, URL https://doi.org/10.1145/3626772.3657807.

Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P., 2023. Lost in the middle: How language models use long contexts. arXiv preprint arXiv:2307.03172.

Liu, M., Wang, J., Lin, T., Ma, Q., Fang, Z., Wu, Y., 2024. An empirical study of the code generation of safety-critical software using llms. Appl. Sci. 14, 1046. http://dx.doi.org/10.3390/app14031046.

Manik, L.P., Akbar, Z., Mustika, H.F., Indrawati, A., Rini, D.S., Fefirenta, A.D., Djarwaningsih, T., 2021. Out-of-scope intent detection on a knowledge-based chatbot.. Int. J. Intell. Eng. Syst. 14 (5).

Marques, N., 2024. Using chatgpt in software requirements engineering: a comprehensive review. Futur. Internet 16, 180. http://dx.doi.org/10.3390/fi16060180.

Miehling, E., Ramamurthy, K.N., Varshney, K.R., Riemer, M., Bouneffouf, D., Richards, J.T., Dhurandhar, A., Daly, E.M., Hind, M., Sattigeri, P., et al., 2025. Agentic ai needs a systems theory. arXiv preprint arXiv:2503.00237.

Min, D., Hu, N., Jin, R., Lin, N., Chen, J., Chen, Y., Li, Y., Qi, G., Li, Y., Li, N., 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.

Mouravieff, R., Piwowarski, B., Lamprier, S., 2024. Learning relational decomposition of queries for question answering from tables. In: Ku, L.-W., Martins, A., Srikumar, V. (Eds.), Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Bangkok, Thailand, pp. 10471–10485. http://dx.doi.org/10.18653/v1/2024.acl-long.564, URL https://aclanthology.org/2024.acl-long.564/.

Pan, L., Albalak, A., Wang, X., Wang, W.Y., 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. arXiv preprint arXiv:2305.12295.

Parthasarathy, V.B., Zafar, A., Khan, A., Shahid, A., 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.

Pham, C.M., Sun, S., Iyyer, M., 2024. Suri: Multi-constraint instruction following for long-form text generation. arXiv preprint arXiv:2406.19371.

Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., Yang, F., Dong, H., Zhang, J., Wang, L., et al., 2023. Taskweaver: A code-first agent framework. arXiv preprint arXiv:2311.17541.

Qin, C., Zhang, A., Chen, C., Dagar, A., Ye, W., 2024. In-context learning with iterative demonstration selection. In: Findings of the Association for Computational Linguistics: EMNLP 2024. pp. 7441–7455.

Sahoo, P., Singh, A.K., Saha, S., Jain, V., Mondal, S., Chadha, A., 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927.

Stechly, K., Valmeekam, K., Kambhampati, S., 2024. Chain of thoughtlessness? an analysis of cot in planning. Adv. Neural Inf. Process. Syst. 37, 29106–29141.

Tai, C.-Y., Chen, Z., Zhang, T., Deng, X., Sun, H., 2023. Exploring chain-of-thought style prompting for text-to-sql. arXiv preprint arXiv:2305.14215.

Turpin, M., Michael, J., Perez, E., Bowman, S., 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. Adv. Neural Inf. Process. Syst. 36, 74952–74965.

VM, K., Warrier, H., Gupta, Y., et al., 2024. Fine tuning LLM for enterprise: Practical guidelines and recommendations. arXiv preprint arXiv:2404.10779.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al., 2024. A survey on large language model based autonomous agents. Front. Comput. Sci. 18 (6), 186345.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D., 2022. Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171.

Wang, Z., Zhang, H., Li, C.-L., Eisenschlos, J.M., Perot, V., Wang, Z., Miculicich, L., Fujii, Y., Shang, J., Lee, C.-Y., et al., 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. arXiv preprint arXiv:2401.04398.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al., 2022. Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural Inf. Process. Syst. 35, 24824–24837.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., Wang, C., 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. arXiv preprint arXiv:2308.08155.

Ye, J., Du, M., Wang, G., 2024. DataFrame QA: A Universal LLM Framework on DataFrame Question Answering Without Data Exposure. http://dx.doi.org/10.48550/ARXIV.2401.15463, Version Number: 1. URL https://arxiv.org/abs/2401.15463.

Zhang, Z., Yao, Y., Zhang, A., Tang, X., Ma, X., He, Z., Wang, Y., Gerstein, M., Wang, R., Liu, G., et al., 2023. Igniting language intelligence: The hitchhiker's guide from chain-of-thought reasoning to language agents. arXiv preprint arXiv:2311.11797.

Zhang, T., Yue, X., Li, Y., Sun, H., 2024. Tablellama: Towards open large generalist models for tables. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). pp. 6024–6044.

Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., Yang, L., Zhang, W., Cui, B., 2024. Retrieval-augmented generation for ai-generated content: A survey. arXiv preprint arXiv:2402.19473.

Zhou, Z., Tao, R., Zhu, J., Luo, Y., Wang, Z., Han, B., 2024. Can language models perform robust reasoning in chain-of-thought prompting with noisy rationales? Adv. Neural Inf. Process. Syst. 37, 123846–123910.