Designing Deterministic AI Agents in Enterprise Platforms: A Schema-Guided Reasoning Approach

Ignatii Anokhin
Department of Business Informatics
National Research University Higher School of Economics
Moscow, Russia
igananokhin@edu.hse.ru

https://doi.org/10.63962/MABC9179

Abstract—The adoption of AI agents in enterprise systems is constrained by limited controllability, reproducibility, and transparency of their reasoning processes, which restricts their use in business-critical scenarios. This study aims to explore how reasoning in enterprise AI agents can be structured to support reliable multi-stage task execution. To address this, the paper investigates the application of Schema-Guided Reasoning (SGR), a structured approach that decomposes reasoning into stages with predefined schemas, validation checkpoints, and iterative control mechanisms. The contribution of the study is the validation of SGR applicability in an enterprise support scenario and the formulation of a reasoning workflow suitable for corporate environments.

Keywords—AI agents, deterministic reasoning, enterprise platforms, schema-guided reasoning, LLM orchestration, enterprise automation

I. Introduction

A. Context

Recent advances in large language models (LLMs) have enabled the development of autonomous AI agents capable of planning, contextual interpretation, and interaction with external systems. These agents are increasingly applied across domains, including enterprise environments, where the focus is shifting toward goal-oriented and proactive systems [1].

In enterprise platforms such as CRM, ERP, and xRM systems, AI agents offer significant potential for automating routine operations, supporting decision-making, analyzing data, and managing organizational resources [2]. However, enterprise deployment introduces stricter requirements compared to consumer use cases. Key expectations include predictable behavior, reproducibility, controllability of reasoning, compliance with internal policies, auditability, and secure handling of sensitive data [2], [4].

Additionally, enterprise agents must operate within complex IT ecosystems, integrating with internal systems, tools, and data sources while maintaining governance and reliability [6], [12]. As a result, the challenge extends beyond output quality to the organization and control of the reasoning process itself.

B. Problem Statement

Despite growing adoption, enterprise implementation of AI agents faces several persistent challenges.

First, reasoning processes are often non-deterministic: similar inputs may produce different outputs, limiting their applicability in business-critical workflows [2], [17].

Second, multi-step scenarios require handling large and evolving contexts, while LLMs are constrained by context window limitations and may fail to retain relevant information across reasoning steps [1], [13].

Third, many agent systems are tightly coupled with specific models, prompting strategies, and tool-calling mechanisms, reducing architectural portability and complicating long-term evolution [1], [12], [13].

Fourth, enterprise constraints frequently necessitate the use of local or smaller models due to security, cost, and infrastructure requirements. However, such models typically demonstrate weaker reasoning capabilities, especially in complex multi-step tasks [3], [9], [15].

Finally, existing approaches often lack sufficient traceability of intermediate reasoning steps, limiting testing, debugging, verification, and audit capabilities [2], [5], [17].

These limitations are also reflected in empirical benchmarks. For example, CRMArena demonstrates that even advanced LLM agents struggle with realistic CRM tasks in professional environments [8].

C. Research Gap

Prior research proposes various techniques to improve agent reliability, including structured planning, context management, intermediate validation, governed execution, and separation of architectural logic from the underlying model [5], [7], [13], [17]. However, these approaches remain fragmented and do not form a unified, widely adopted framework for structuring reasoning in enterprise agents.

In practice, organizations must combine multiple techniques and validate them independently within specific business scenarios [1], [2]. This creates a gap between available architectural concepts and the need for a coherent mechanism that enables reasoning to function as a controlled, reproducible, and verifiable process in complex enterprise workflows.

D. Research Question

This study addresses the following research question: How can reasoning processes in enterprise AI agents be structured to enable reliable multi-stage task execution?

A related sub-question examines how such structuring can support determinism, reproducibility, model-agnostic operation, and traceability of reasoning.

E. Contributions

This paper investigates the application of a structured reasoning control mechanism in an enterprise scenario. The approach aims to make multi-step reasoning explicit, decomposed, and verifiable.

The study contributes by:

- demonstrating how structured reasoning can be applied in a real enterprise use case,
- defining a reasoning workflow aligned with enterprise requirements,
- and evaluating its applicability and potential effectiveness based on initial industrial validation.

Importantly, the contribution lies in the application and adaptation of existing but not widely accepted principles of structured reasoning, rather than proposing a new framework.

II. Methodology

A. Research Design

This study follows the Design Science Research (DSR) methodology, focusing on the development and practical validation of an architectural artifact addressing a real-world problem. The research is conducted within a Russian enterprise vendor of an xRM low-code platform used for ITSM solutions. This setting enables the investigation of AI agents in realistic operational contexts, including incident handling, user request processing, knowledge base utilization, etc. Consequently, the proposed artifact is designed with direct applicability to production environments.

The research process consists of several stages. First, key limitations of existing agent solutions are identified and formalized. Second, relevant architectural approaches are analyzed. Third, a target reasoning design is developed and applied to a selected enterprise scenario. Finally, the proposed approach is evaluated through a controlled experiment.

B. Problem Identification

Problem identification was based on analyzing existing AI agents deployed within the company's products and collecting feedback from key stakeholders. Three groups participated: product managers, engineers responsible for configuring agent scenarios, and end users.

The study included 10 interviews (4 internal teams and 6 customer teams) and analysis of acceptance testing results, including expert evaluations of scenario performance. This enabled the construction of a representative set of enterprise automation scenarios and identification of recurring limitations.

The most common scenarios were related to user support automation, including ticket classification, operator copilots for solution search, report generation, incident pattern detection, and monitoring data processing. Detailed feedback was collected for each scenario, including specific failure cases and instability patterns.

The analysis revealed that the identified issues are systemic and consistent across scenarios. These findings align with the limitations described in Section 3, including non-deterministic reasoning, model dependency, and limited traceability.

A support copilot scenario for solution retrieval was selected as the primary evaluation case. This choice is justified by its high business impact, availability of quantitative feedback (current quality score: 6.8/10), and its representation of a typical multi-step reasoning process involving retrieval, analysis, and synthesis.

C. Knowledge Exploration

To determine an appropriate architectural approach, existing research on AI agents and structured reasoning was reviewed.

LLM agent architectures. Prior work shows that LLM agents typically consist of planning, memory, and execution components and are applied across diverse domains, including enterprise systems [1]. However, enterprise adoption highlights challenges related to reliability and reproducibility [2], while many architectures emphasize communication and tool integration [6].

Structured reasoning and controlled generation. Schema-aware generation and grammar-constrained decoding improve output consistency and reduce errors by enforcing predefined structures [17]. Frameworks such as Routine introduce step-wise planning and validation mechanisms to enhance robustness [9].

Schema-Guided Reasoning (SGR). SGR organizes reasoning through predefined schemas, intermediate validation, and structured outputs [19]. Prior studies indicate improvements in reproducibility, debuggability, and testing, although the approach requires careful schema design [18], [19].

Tool orchestration and coordination. Enterprise-oriented architectures propose blueprint-based orchestration and separation of execution logic from language models [12], [17]. The POLARIS framework highlights the role of typed planning and controlled execution in automation scenarios [7].

Context management. Approaches such as SagaLLM and DataLab emphasize structured context handling, validation, and state management across reasoning stages [13], [14], underscoring the importance of managing intermediate reasoning states.

D. Solution Design

The solution design followed a structured process.

First, the existing assistant workflow (AS IS) was analyzed in detail, including processing steps, system components, and decision points. This allowed formalizing the implicit reasoning process.

Second, failure points identified during problem analysis were examined, with a focus on context loss, uncontrolled generation, and absence of intermediate validation.

Based on the literature review, a target approach was selected that decomposes reasoning into structured stages with explicit control over inputs and outputs. The design emphasizes the ability to introduce this mechanism without modifying the core platform, using an external orchestration layer.

As a result, a target reasoning workflow (TO BE) was defined, separating reasoning, retrieval, and response generation into distinct, controlled stages.

E. Evaluation

The proposed approach is evaluated using an A/B testing methodology. Two assistant implementations are compared: the baseline (AS IS) and the structured approach (TO BE). Both operate on the same dataset and use the same language model, isolating the impact of architectural changes.

The dataset consists of 100 test cases representing real user requests for solution retrieval from knowledge bases and ticket history. Outputs are evaluated by ITSM product experts using a 10-point scale.

Evaluation metrics include average quality score, reproducibility, performance on small language models (SLM), stability across different models, and token consumption. This set of metrics enables assessment not only of answer quality but also of robustness, portability, and cost efficiency.

III. Preliminary Results

A. Existing Workflow

The current assistant workflow (AS IS) follows a standard retrieval-augmented generation (RAG) architecture widely used in enterprise applications. A user query is first encoded into a vector representation, which is then used to retrieve relevant documents from a vector database. Retrieved results are re-ranked and combined with the original query during prompt construction, after which the language model generates the final response.

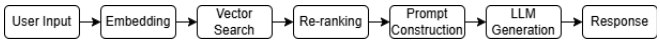


Fig. 1. AS IS Workflow

While this approach enables basic retrieval and answer generation, it does not provide explicit control over the reasoning process. Decision-making logic remains encapsulated within the language model, limiting transparency, controllability, and adaptability.

The component structure includes an embedding model (USER-bge-m3), a vector search engine (pgvector), a reranker (colbert-xm), and a generation model (saiga_llama3_8b). The main limitations of this pipeline are the lack of task structuring, sensitivity to query formulation, and non-deterministic outputs.

TABLE I. AS IS PIPELINE COMPONENTS

№	Component	Function	Tool / Model	Limitations
1	Embedder	Encode user query into vector	USER-bge-m3	No task structure awareness
2	Retriever	Retrieve relevant documents	pgvector	Sensitive to query wording
3	Reranker	Improve document relevance	colbert-xm	Partial noise filtering
4	Prompt Constructor	Combine query and context	Prompt template	
5	Answer Generator	Generate final response	saiga_llama3_8b	Non-deterministic output

B. Analysis of Limitations

The analysis of the existing workflow highlights several critical issues related to the absence of explicit reasoning control.

First, the input task is processed in an unstructured form, leading to ambiguous interpretation of user requests. This is particularly problematic in support scenarios, where a single case may involve multiple interrelated issues.

Second, retrieval relies on one or a limited number of queries derived directly from the input text. As a result, retrieval quality depends heavily on wording rather than the underlying problem structure.

Third, there is no mechanism for validating intermediate results. The system proceeds directly to response generation without verifying the completeness or correctness of retrieved information, increasing the likelihood of hallucinations.

Fourth, the reasoning process is not traceable. It is not possible to reconstruct which data sources were used or how the final response was derived, which complicates debugging, testing, and auditing.

Finally, the approach demonstrates limited robustness when applied to smaller models. As model quality decreases, both answer accuracy and reasoning capability degrade, since all logic is embedded within a single generation step.

C. Selection of Target Approach

To address these limitations, Schema-Guided Reasoning (SGR) is selected as the target approach for structuring reasoning [19].

The core principle of SGR is the use of structured outputs, where both intermediate and final results are represented through predefined schemas (e.g., JSON). This reduces variability, enables validation, and enforces consistency [19], [20].

At the process level, SGR relies on three key patterns:

- **Cascade**, which decomposes tasks into sequential subtasks with structured outputs;
- **Routing**, which dynamically selects execution paths based on intermediate results;
- **Cycle**, which enables iterative refinement (e.g., repeated retrieval or validation).

Their combination forms a controlled reasoning pipeline with explicit transitions between stages.

SGR aligns with enterprise requirements by supporting determinism, reproducibility, traceability, and model-agnostic design. It can be implemented as an external orchestration layer without modifying the underlying platform.

Compared to alternative approaches, SGR provides stronger formalization. Methods such as Chain-of-Thought and Tree-of-Thought rely on implicit textual reasoning without structured validation. Graph-of-Thought extends these ideas but introduces computational overhead without ensuring control. ReAct and

Plan-and-Execute incorporate tool usage and planning but lack strict schema enforcement. Program-of-Thought improves precision through code-like reasoning but is limited to formalizable tasks. Multi-agent and hierarchical planning approaches increase architectural complexity without guaranteeing reproducibility.

In contrast, SGR enforces structured reasoning through schemas, typed data, and validation checkpoints, making it more suitable for enterprise scenarios requiring stability and auditability.

Existing studies indicate that schema-constrained generation reduces errors and improves consistency [19], while SGR-based implementations enhance reproducibility and testing capabilities [20], [21]. The contribution of this work lies in applying and adapting these principles to enterprise ITSM scenarios rather than proposing a new framework.

D. Proposed Workflow

The proposed reasoning workflow (TO BE) implements SGR principles as a multi-stage controlled pipeline with explicit decomposition and validation. Each stage operates on typed inputs and produces validated outputs, enabling deterministic transitions between stages.

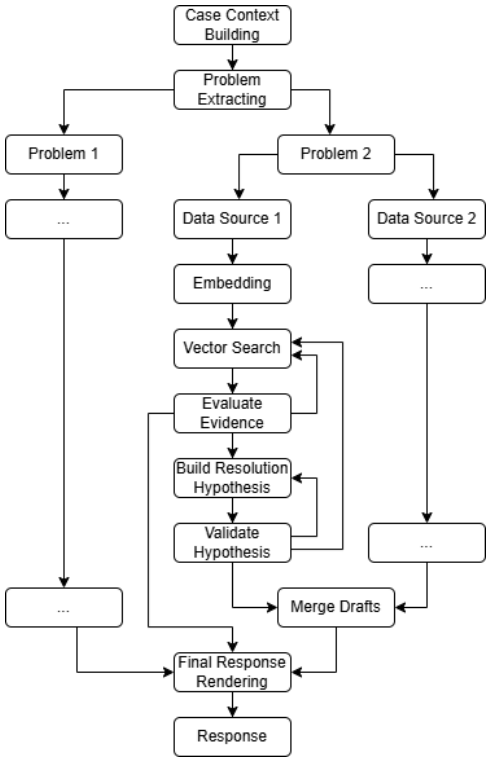


Fig. 2. TO BE Workflow

The process begins with Case Context Building, where the input is normalized into a structured representation. The Problem Extractor then applies cascade decomposition, splitting the case into independent structured problems (including symptoms, entities, and retrieval queries), reducing ambiguity.

Each problem is processed through a retrieval cycle. The Embedder converts structured queries into vector form, and the Retriever retrieves candidate evidence. The Source Evaluator applies routing and cycle mechanisms by assessing relevance and completeness, triggering additional retrieval if needed.

Based on validated evidence, the Resolution Hypothesis Builder generates structured solution hypotheses. The Resolution Hypothesis Evaluator verifies these hypotheses against the problem definition and schema, enabling feedback loops to earlier stages when inconsistencies are detected.

Validated hypotheses are combined in the Draft Builder, preserving the relationship between problems and solutions. Finally, the Final Response Renderer converts the structured draft into a user-facing response. This separation of reasoning and generation improves controllability and reproducibility.

Each component performs a distinct function within the reasoning process, eliminating the “black-box” effect and enabling traceability.

TABLE II. TO BE PIPELINE COMPONENTS

№	Component	Function	Tool / Model	Reasoning Role
1	Case Context Builder	Normalize input and assemble structured case context	saiga_llama3_8b	Reduces ambiguity before reasoning
2	Problem Extractor	Decompose case into one or more structured problems	saiga_llama3_8b	Makes reasoning explicit and decomposed
3	Embedder	Convert structured problem into vector representations for retrieval	USER-bge-m3	Enables semantic alignment between problem and knowledge
4	Retriever	Retrieve evidence using multiple queries and sources	pgvector	Provides initial evidence set
5	Source Evaluator	Filter and rank evidence, detect insufficiency (loop back to retrieval if needed, trigger response rendering if no relevant evidence found)	saiga_llama3_8b	Ensures evidence quality and relevance
6	Resolution Hypothesis Builder	Generate candidate solution based on structured problem and evidence	saiga_llama3_8b	Separates reasoning from final answer
7	Resolution Hypothesis Evaluator	Validate hypothesis against initial problem, evidence and schema (loop back to retrieval if needed)	saiga_llama3_8b	Enables intermediate validation and control
8	Draft Builder	Merge validated hypotheses into structured operator draft	saiga_llama3_8b	Preserves structure across subproblems
9	Final Response Renderer	Convert structured draft into final response	saiga_llama3_8b	Separates reasoning from final answer

Structured schemas play a central role in this design. Cases are represented as collections of problems with formalized attributes such as symptoms, entities, and retrieval queries.

Listing 1. Structured Problem Representation Output (JSON)

```
{
  "case_id": "INC-10452",
  "problems": [
    {
      "problem_id": "P1",
      "label": "VPN authentication failure",
      "symptoms": [
        "VPN login rejected",
        "issue started after password reset"
      ],
      "retrieval_queries": [
        "VPN login failed after password reset",
        "authentication error after credential change"
      ]
    }
  ]
}
```

Typed data models (e.g., using validation frameworks such as Pydantic) ensure consistency, reduce generation errors, and simplify integration between components [19], [20].

Listing 2. Typed Problem Representation (Python)

```
class Problem:
    problem_id: str
    label: str
    symptoms: list[str]
```

```
    retrieval_queries: list[str]

def extract_problems(case_text: str):
    """
    Extract structured problems
    using schema-guided LLM output
    """
    result = LLM(case_text,
                 schema=Problem)
    return result
```

E. MVP Limitations and Evaluation

An MVP implementation is developed using Python, with LangChain and LangGraph as the orchestration framework. Integration with the xRM platform is achieved via MCP, minimizing changes to the existing architecture.

A mid-sized local model (saiga_llama3_8b) is used as the baseline to evaluate whether structured reasoning can compensate for the limitations of smaller models.

Preliminary qualitative observations on a subset of cases indicate a reduction in irrelevant responses compared to the baseline RAG approach. A full quantitative evaluation is designed as an A/B test on 100 cases with expert scoring.

The current MVP is limited by partial platform integration, a restricted set of scenarios, and the need for manual schema design. Future work includes comprehensive quantitative evaluation using metrics such as answer quality, reproducibility, model robustness, and computational cost.

IV. Conclusion

This study examines the limitations of controllability and reproducibility in reasoning processes of enterprise AI agents, which restrict their use in business-critical contexts.

The paper explores the application of Schema-Guided Reasoning (SGR) as a mechanism for structuring reasoning through predefined stages, schemas, and validation checkpoints. The proposed workflow enables explicit task decomposition, intermediate validation, and separation of reasoning from response generation, supporting enterprise requirements such as traceability, determinism, and robustness.

Future work includes completing the MVP, conducting controlled A/B testing, and performing quantitative evaluation, followed by potential extension to production environments and other enterprise use cases.

References

[1] T. Zhao, C. Ma, X. Feng, Z. Zhang, and H. Yang, “A survey on large language model based autonomous agents,” *Frontiers of Computer Science*, 2024, doi: 10.1007/s11704-024-40231-1.

[2] V. Muthusamy, Y. Rizk, K. Kate, P. Venkateswaran, and V. Ishakian, “Towards large language model-based personal agents in the enterprise: Current trends and open problems,” in *Proc. EMNLP Findings*, 2023, doi: 10.18653/v1/2023.findings-emnlp.461.

[3] Z. Wang et al., “A comprehensive survey of small language models in the era of large language models,” arXiv:2411.03350, 2024.

[4] Y. Yang, H. Chai, Y. Song, S. Qi, M. Wen, N. Li, et al., “A survey of AI agent protocols,” arXiv:2504.16736, 2025.

[5] Z. Moslemi, K. Koneru, Y.-T. Lee, S. Kumar, and R. Radhakrishnan, “POLARIS: Typed planning and governed execution for agentic AI in back-office automation,” arXiv, 2026.

[6] R. Shu, N. Das, M. Yuan, M. Sunkara, and Y. Zhang, “Towards effective GenAI multi-agent collaboration: Design and evaluation for enterprise applications,” arXiv, 2024.

[7] Y. Zeng et al., “Routine: A structural planning framework for LLM agent system in enterprise,” arXiv:2507.14447, 2025.

[8] K.-H. Huang, A. Prabhakar, S. Dhawan, Y. Mao, and H. Wang, “CRMArena: Understanding the capacity of LLM agents to perform professional CRM tasks in realistic environments,” arXiv:2411.02305, 2024.

[9] L. Tuggener et al., “So you want your private LLM at home? A survey and benchmark of methods for efficient GPTs,” in *Proc. Swiss Conf. Data Science*, 2024, doi: 10.1109/sds60720.2024.00036.

[10] B. Reitemeyer and H.-G. Fill, “Applying large language models in knowledge graph-based enterprise modeling: Challenges and opportunities,” arXiv:2501.03566, 2025.

[11] M. A. Ferrag, N. Tihanyi, and M. Debbah, “From LLM reasoning to autonomous AI agents: A comprehensive review,” arXiv:2504.19678, 2025.

[12] E. Kandogan, N. Bhutani, D. Zhang, L. Chen, and S. Gurajada, “Orchestrating agents and data for enterprise: A blueprint architecture for compound AI,” arXiv:2504.08148, 2025.

[13] J. Chang et al., “SagaLLM: Context management, validation, and transaction guarantees for multi-agent LLM planning,” *Proc. VLDB Endowment*, vol. 18, no. 5, pp. 1234–1247, 2025, doi: 10.14778/3750601.3750611.

[14] C. Weng et al., “DataLab: A unified platform for LLM-powered business intelligence,” arXiv:2412.02205, 2024.

[15] V. Vemulapalli, “Choosing the right model for enterprise AI applications: A comprehensive analysis of small language models versus large language models in enterprise architecture,” *European Modern Studies Journal*, vol. 9, no. 5, p. 25, 2025, doi: 10.59573/emsj.9(5).2025.25.

[16] Y. Liu et al., “Towards robust multi-modal reasoning via model selection,” arXiv:2310.08446, 2023.

[17] L. Qiu, Y. Ye, Z. Gao, X. Zou, and J. Chen, “Blueprint first, model second: A framework for deterministic LLM workflow,” arXiv:2508.02721, 2025.

[18] Cognia Lab, “Schema-Guided Reasoning: A new method of structuring reasoning in LLMs to reduce errors,” [Online]. Available: https://cognia-lab.com/schema-guidedreasoning

[19] R. Abdullin, “Schema-Guided Reasoning (SGR),” 2025. [Online]. Available: https://abdullin.com/schema-guided-reasoning/

[20] vamplabAI, “sgr-agent-core,” GitHub repository. [Online]. Available: https://github.com/vamplabAI/sgr-agent-core

[21] Redmadrobot, “Schema-Guided Reasoning: как научить языковые модели последовательно рассуждать,” Habr, [Online]. Available: https://habr.com/ru/companies/redmadrobot/articles/962236/