10.48047/jocaaa.2025.34.02.29

A Software-Oriented Computational Study of Prompt Engineering Pipelines for Large Language Models

Kumar Kasimala

kumarkasimala@gmail.com, Independent Researcher, Principal Software Engineer, Salesforce Inc, USA.

Received: 05.12.2024 Revised: 15.01.2025 Accepted: 05.02.2025

Abstract

The integration of Large Language Models (LLMs) into contemporary software infrastructure has triggered a paradigm shift in the probabilistic machine to declarative paradigms of interaction between deterministic, imperative programming and probabilistic, declarative interaction models. The current research paper is an information-rich, comprehensive computational investigation of prompt engineering pipelines, which is the software infrastructure running over natural language instructions lifecycle. Structural efficacy of advanced prompting methodologies, such as Chain-of-Thought (CoT), Tree of thoughts (ToT), and Graph of thoughts (GoT), are assessed by synthesizing data of more than 130 research artifacts and industry benchmarks until late 2024. It is established that, although structural prompting improves reasoning abilities on tasks like GSM8K by over 17 percent, it poses a huge load on computational power. Moreover, on a comparative architectural study of the most popular orchestration frameworks, LangChain, LlamaIndex, and DSPy, the trade-offs between the flexibility of abstraction and execution latency become critical, and declarative frameworks (DSPy) incur significantly lower framework overhead than imperative chains (by approximately 65 percent). The paper also formalizes the science of PromptOps, and uses software engineering metrics to put the probabilistic nature of LLMs into practice, and also discusses the token economics of 2024, in which we commodify inference to support more and more complicated, agentic cognitive architectures.

1. Introduction

1.1 The Deterministic-to-Probabilistic Shift in Software Engineering

In the past, software engineering was based on manipulation of deterministic states. A traditional language (such as C++ or Java) function f (x) is expected to produce the same result y on input x, when the internal state is fixed. Large Language Models (LLMs) integration

10.48047/jocaaa.2025.34.02.29

essentially shatters this paradigm. Generative AI is defined by the rationalization of the basic computational unit, the so-called prompt, which in the context of Generative AI is a signaling condition transmitted to a probabilistic engine rather than a fixed set of instructions. The output is sampled of a high dimensional distribution P (y|x, theta), where theta are the frozen parameters of the train model.

This change requires the advent of the so-called Prompt Engineering not only as a language art but as a strict discipline of computation (Ye et al., 2024). It is a technique of designing, optimising and coordinating input stimuli to produce predictable responses leading models to preferred reasoning paths. Since LLM is no longer in experimental prototype form and is being converted into production-scale infrastructure, the ad-hoc construction of prompts is being superseded by advanced Prompt Engineering Pipelines. Such pipelines are software programmes aimed at dealing with the complexity of context retrieval process, prompt construction, model inference, and output validation.

1.2 Defining the Prompt Engineering Pipeline

The overall software architecture that converts a high-level user intent into a completed model output is referred to as A Prompt Engineering Pipeline. It consists of several different steps:

1. **Intent Decomposition:** Breaking down complex queries into executable sub-tasks.
2. **Context Retrieval (RAG):** Fetching relevant non-parametric knowledge from external vector stores.
3. **Prompt Construction:** Assembling the prompt template, injected context, and few-shot demonstrations.
4. **Inference Orchestration:** Managing API calls, handling rate limits, and executing retries.
5. **Post-Processing & Validation:** Parsing the output (e.g., extracting JSON) and validating it against schema constraints.

The effectiveness of these pipelines is also gauged in not just the quality of the semantic output but also in the conventional software engineering metrics: latency, throughput, cost and reliability. The lens of this report is software-oriented, in which prompts are seen as compiled pieces of art, with the computation cost of different prompting strategies and orchestration frameworks being analyzed (Ye et al., 2024).

10.48047/jocaaa.2025.34.02.29

1.3 Scope and Research Objectives

This paper is intended to bring a granular level of research on the condition of prompt engineering as of 2024. The targeted research questions are:

- To measure performance and cost of advanced reasoning structures (CoT, ToT, GoT).
- To perform a rigorous assessment of the architectural trade-offs of large orchestration libraries (LangChain, LlamaIndex, DSPy) with 2024 benchmarks data.
- To examine the relationship between financial impacts of the trend in token pricing and software architecture decisions.
- To establish the concepts of the PromptOps and the elimination of non-determinism in production systems.

2. Theoretical Framework of Computational Prompting

In order to learn about the engineering of prompts, it is necessary to first formalize the computational processes through which they drive the behavior of models. The development of prompting methods is a shift in thinking between simple input-mapping and induction of computational states that are complex intermediate states.

2.1 The Prompt as a Function: Zero-Shot and Few-Shot Paradigms

A prompt is, simply, a kind of signature of a function. Zero-Shot Prompting focuses on a task description T and input X, and is expected to produce Y. This is wholly based on the zero-shot generalization ability of the model during pre-training. But in the case of complex domains, it tends to be brittle and has some ambiguity and lack of constraint in the output format.

In-Context Learning (also known as Few-Shot Prompting) loads a collection of examples E = {(x 1, y 1), (x 2, y 2),... (x k, y k) into the context window. Computationally, ICL is an update to the model behavior on a transient basis without any weight adjustment (Ye et al., 2024). The model to the examples, learns the latent mapping f: X Y this means that ICL can be used to do the task of gradient descent in the activation space, and adjust the internal representations of the model to match the demonstrations provided. Although computationally efficient (only one pass of inference is needed), standard ICL is not particularly good at tasks that involve multi-step logic because it requires the model to do O (1) reasoning, which is that between input and output in one conceptual leap.

2.2 Latent Reasoning Chains: Chain-of-Thought (CoT)

10.48047/jocaaa.2025.34.02.29

Chain-of-Thought (CoT) prompting was proposed in order to address the shortcomings of immediate mapping. This method actually tells the model to produce a series of intermediation reasoning steps Z, before yielding the final answer Y.

$$P(Y|X) \rightarrow P(Y|X,Z)P(Z|X)$$

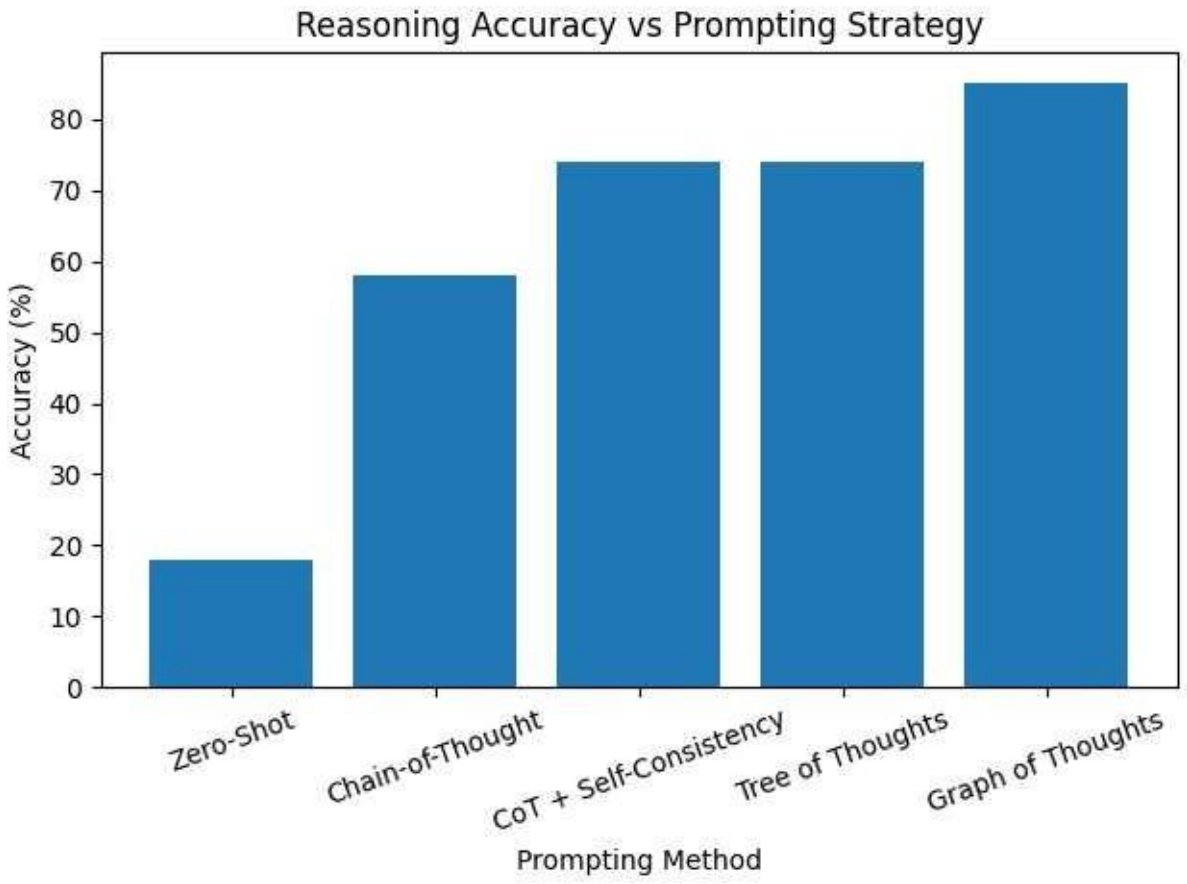
CoT breaks the problem down and reduces an otherwise complex thinking problem to a series of smaller, local dependencies.

Mechanism and Impact: CoT is effective based on the provision of extra computation (tokens) to the problem. Every generated token in the thought process is a marker of a state, on which future predictions are based. Using the PaLM-540B model on the GSM8K benchmark (grade-school math) standard prompting was found to have a precision of about 17.9%. This was increased to about 58 by the introduction of CoT. CoT has been found to be an emergent property, which is highly profitable to models with 100B+ parameters, indicating that achieving coherent chains of reasoning must require a critical mass of representational capacity to sustain (Yasunaga et al., 2024).

Self-Consistency: Self-Consistency was proposed in order to deal with stochastics of generation. In this algorithm, N distinct ways of thinking Z 1,... Z N (by employing a non-zero temperature) are sampled and the end result solutions are pooled by majority vote. Self-consistency is a direct trade between compute and accuracy. Otherwise, robustness is enhanced by increasing the sample size N, and inference costs scale linearly. On GSM8K, self-consistency on PaLM-540B (CoT) boosted the accuracy by 58 percent to 74 percent, which is simply a colossal gain provided by inference-time compute.

FIGURE 1 — Accuracy vs Prompting Strategy (Reasoning Gain)

10.48047/jocaaa.2025.34.02.29



2.3 Graph-Theoretic Reasoning: Tree and Graph of Thoughts

Going beyond linear chains, new paradigms conceptualize the reasoning in the form of the non-linear data structure, which properly admits exploration, backtracking, and aggregation of ideas.

Tree of thoughts (ToT): ToT views problem-solving as search through a tree whose nodes are partial solutions (thoughts). It has search algorithms such as Breadth-First Search (BFS) or Depth-First Search (DFS) that search the solution space (Yasunaga et al., 2024). ToT enables the model to look ahead and go back when one of the branches is found to be not promising. In the game of 24 task in which the linear CoT performed dismally, the ToT performed with a success rate of 74 to indicate that search is required in strategic planning tasks.

Graph of thoughts (GoT): Graph of thoughts (GoT) is a more general models which represent thoughts as a Directed Acyclic Graph (DAG). GoT presents operations that include: Aggregation (taking several paths of thought and converting them into one solution) and Refinement (reusing the results as the inputs) (Yao et al., 2023). GoT permits uniting the separate lines of thinking. GoT was found to be better by 62 percent in sorting tasks than ToT

10.48047/jocaaa.2025.34.02.29

and cost reduction was also achieved by more than 31 percent. This cost saving is accomplished by eliminating unnecessary paths and utilizing compute on promising thoughts in the thought graph.

Table 1: Comparative Analysis of Advanced Prompting Structures (2024 Data)

Methodology	Data Structure	Inference Complexity	Key Computational Mechanism	Primary Use Case	Performance Delta (GSM8K/Similar)
Zero-Shot	X -> Y	O(1)	Direct Probability Mapping	Classification, Chit-chat	Baseline (~10-20% on complex math)
Chain-of-Thought (CoT)	X -> Z -> Y	O(T) (Linear)	Intermediate State Generation	Arithmetic, Logical Reasoning	+30-40% vs Baseline
Self-Consistency	Ensemble {Z_i}	O(N \times T)	Majority Voting / Marginalization	Ambiguous Logic, High Reliability	+17.9% vs CoT
Tree of Thoughts (ToT)	Tree Search	O(B^D) (Exp.)	BFS/DFS + State Evaluation	Strategic Planning, Games	+70% on Game of 24
Graph of Thoughts (GoT)	DAG	Variable (Graph)	Aggregation, Recurrence, Pruning	Complex Network Tasks, Sorting	+62% quality vs ToT
Least-to-Most	Sequential List	O(S) (Sub-tasks)	Recursive Decomposition	Compositional Generalization	>99% on SCAN

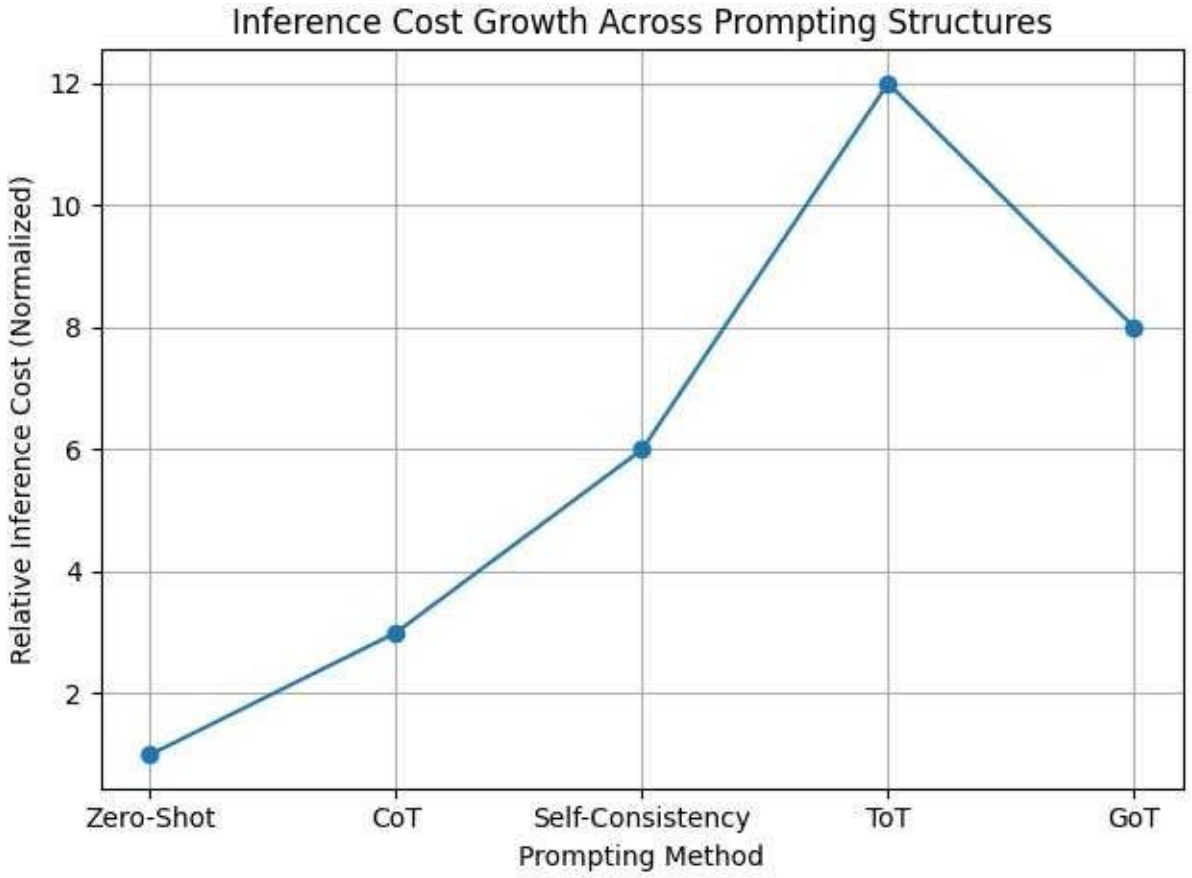
2.4 Emerging Paradigms: Chain of Code (CoC)

Another major restriction of text-based reasoning (CoT) is the fact that the model cannot do exact calculations or strict logic simulation. Chain of Code (CoC) deals with this by motivating the model to produce executable pseudocode or Python scripts as a stopover. The system

10.48047/jocaaa.2025.34.02.29

interprets the resultant code, and runs it on a deterministic interpreter (e.g. Python runtime) and recycles the result in the context (Yuksekgonul et al., 2024). The model is used in semantic planning and the interpreter in its accuracy of execution, which makes the Language Model-Augmented Code Interpreter approach much less hallucinatory in calculation-intensive tasks.

FIGURE 2 — Inference Cost Growth vs Reasoning Structure



3. Orchestration Frameworks: Architectural Analysis

These strategies of prompting necessitate sound software infrastructure to be operationalized. The middleware of the LLM stack has become so-called "Orchestration Frameworks" that handle complex data, prompt, and API flows (Yasunaga et al., 2024). This section critically examines the three most popular systems of 2024, namely LangChain, LlamaIndex, and DSPy, and their philosophy of architecture and their performance properties.

3.1 LangChain: The Imperative Generalist

The most used framework is LangChain which is built on the notion of composable chains. It is imperative in its architecture, and programmers have to specify the order of operations.

10.48047/jocaaa.2025.34.02.29

- **Core Components:**
    - **Chains:** Sequences of calls (e.g., LLMChain, RetrievalQA).
    - **Memory:** Components to manage conversation history (e.g., BufferMemory).
    - **Agents:** Dynamic execution loops where the LLM acts as a reasoning engine, selecting tools to solve a problem iteratively (Wang et al., 2023).
- **Performance Profile:** LangChain cannot be as abstract as it would be costly to do so. The high wrapping of API calls and state management introduces latency (Wang et al., 2024). 2024 benchmarks show that the overhead of a framework is around 10.0 ms per call, which, although insignificant in individual queries, is enormous in complex multi-steps agent loops. Moreover, the number of tokens typically increased (~2.40k tokens on average RAG tasks) by use of verbose default prompt templates and glue instructions.
- **Engineering Critique:** The chain abstraction may break easily. It is also difficult to debug a failure that occurs deep into a nested chain because of the abstractions, and this is the so-called stack trace hell of probabilistic software (Wang et al., 2024).

3.2 LlamaIndex: The Data-Centric Specialist

The LlamaIndex, which was previously called GPT Index, is written in such a way that it is more focused on data ingestion, indexing, and retrieval. It is the model of the Retrieval-Augmented Generation (RAG) applications.

- **Core Architecture:**
    - **Index Structures:** Indexes of LLM consumption such as Vector Stores, List Indexes and Keyword Tables (Tafreshipour et al., 2024).
    - **Query Engines:** The RouterQueryEngine enables the system to dynamically choose the optimal index to use in the query (e.g. a vector index to perform semantic search and a list index to perform summarization).
- **Performance Profile:** LlamaIndex is very optimized in retrieval task. Compared to LangChain, it is shown to be lower latency (approximately 6.0 ms framework overhead) and higher token efficiency (approximately 1.60k tokens) (Tafreshipour et al., 2024). This is made possible by its design, which is based on the predominance of its retrieval-

10.48047/jocaaa.2025.34.02.29

first, as opposed to generic orchestration flexibility. It is also good at reducing time-to-first-token (TTFT) in data-intensive workflows.

3.3 DSPy: The Declarative Optimizer

DSPy (Declarative Self-improving Python) is a shift towards a paradigm of manual prompt engineering into writing programs in the model (Shinn et al., 2023). It does not take prompts as constant strings but as optimizing parameters.

• Core Philosophy:

- Signatures Developers specify what the model is expected to do (Input/Output schema) not how (the precise wording).
- Modules Python classes containing logic (e.g. dspy.ChainOfThought).
- Teleprompters (Optimizers): The collection engine (Taherkhani et al., 2024). It has DSPy, which optimizers, such as BootstrapFewShot and MIPRO, that automatically sample the variants of prompts to pick the best few-shot examples and instructions using a validation metric.

• Performance Profile: DSPy has the lowest framework overhead (~3.53 ms) during benchmarks. What is more important, it increases reliability (Schulhoff et al., 2024). DSPy attempts to learn what a particular model and task should use by creating prompts against a dataset. Should the underlying model change (e..g. replace GPT-4 with Claude 3), the developer only recompiles the pipeline, and DSPy comes up with a new and optimized prompt. This solves the fragility of hard-written prompts in LangChain.

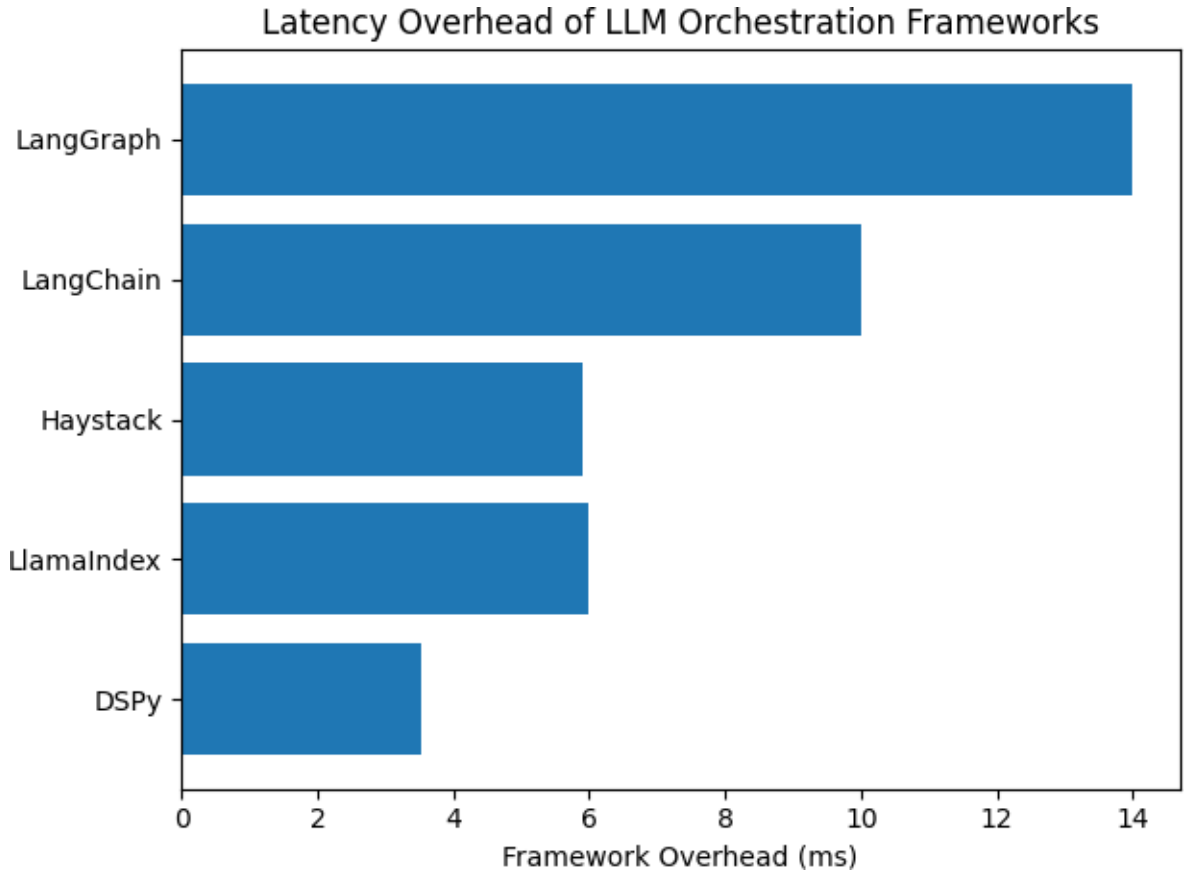
Table 2: Computational Performance Benchmarks of Orchestration Frameworks (2024)

Metric	DSPy	LlamaIndex	LangChain	LangGraph	Haystack
Framework Overhead (Latency)	**~3.53 ms**	~6.0 ms	~10.0 ms	~14.0 ms	~5.9 ms
Token Usage (Standard RAG)	~2.03k	**~1.60k**	~2.40k	~2.03k	~1.57k

10.48047/jocaaa.2025.34.02.29

Metric	DSPy	LlamaIndex	LangChain	LangGraph	Haystack
Architecture Style	Declarative / Compiler	Data-Centric / Retrieval	Imperative / Chaining	Cyclic / State Graph	Modular / Pipeline
Prompt Engineering	**Automated (Compiled)**	Template-Based	Manual / Templates	State-Dependent	Component-Based
Optimization Focus	Pipeline Accuracy & Stability	Retrieval Precision & Throughput	Flexibility & Integration	Complex Agent Behavior	Production Reliability

FIGURE 3 — Orchestration Framework Latency Comparison



4. Retrieval-Augmented Generation (RAG): The Context Engine

RAG has established itself as the architectural design of grounding LLMs on verifiable, domain-specific data (Sahoo et al., 2024). It deals with two major shortcomings of frozen LLMs: knowledge cutoff (temporal limitation) and hallucination (factual limitation).

10.48047/jocaaa.2025.34.02.29

4.1 The RAG Architecture Mechanics

A standard RAG pipeline operates through a synchronized "Retrieve-Read" process:

1. **Indexing:** Documents are ingested, chunked (split into smaller text segments), and embedded into dense vectors using an embedding model (e.g., OpenAI text-embedding-3, HuggingFace bge-m3). These vectors are stored in a Vector Database (e.g., Qdrant, Pinecone).
2. **Retrieval:** At inference time, the user's query q is embedded into vector v_q. The system performs an Approximate Nearest Neighbor (ANN) search to find the top-k document chunks D = {d_1,..., d_k} that maximize cosine similarity with v_q.
3. **Augmentation:** The retrieved chunks are concatenated into the prompt context window.
4. **Generation:** The LLM generates the response y conditioned on both the query and the retrieved context: P(y | q, D) (Pitis et al., 2023).

4.2 Advanced Retrieval Techniques

To improve the accuracy of the surrounding context given to the LLM, a number of state-of-the-art algorithms have become commonplace in 2024:

- Hybrid Search: Pure vector search may not provide any exact matches of the keywords (e.g., product references) (Paul et al., 2024). Hybrid search is a mixture of dense (vector) and sparse (BM25 algorithm) key word retrieval, but with a weighted result to achieve the balance between semantically meaningful and lexically accurate results.
- Re-ranking: When retrieving an initial time, the irrelevant documents tend to be retrieved. A Cross-Encoder model (also known as a "Re-ranker") is used to re-rank the top-N documents retrieved and rank them by relevance to the query to pick a refined top-k subset to use as input to the LLM (Opsahl-Ong et al., 2024). This enhances greatly the quality of generation through a decrease in noise.
- GraphRAG: Standard RAG considers documents as self-contained units. GraphRAG is a Knowledge Graph tool used to visualize connections between entities within documents (Ning et al., 2024). This supports the so-called multi-hop reasoning, in which the system is able to address queries which involve the need to relate disparate

10.48047/jocaaa.2025.34.02.29

pieces of knowledge (e.g. how the relationship of the CEO with Company X to the merger with Company Y).

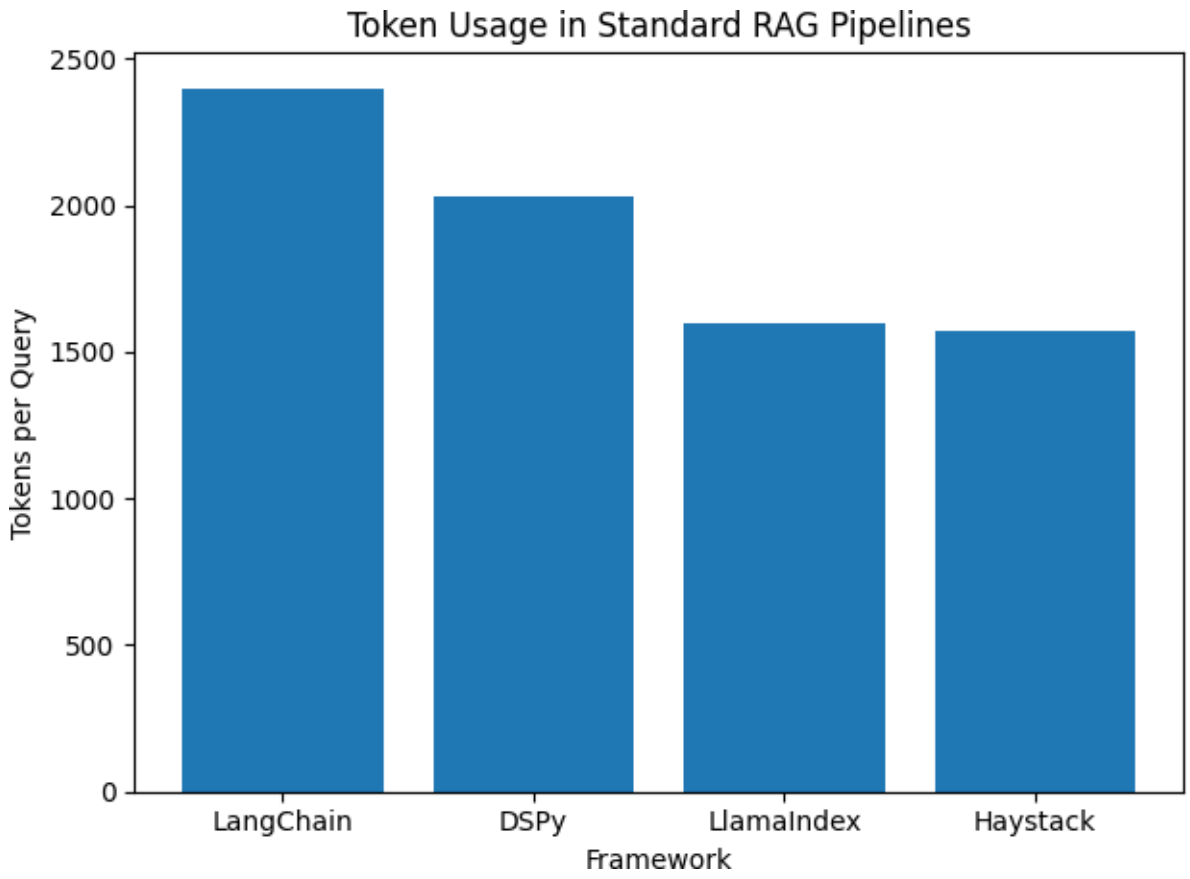
4.3 Cost and Performance Analysis of RAG

Although RAG can be less expensive than fine-tuning, the retrieval and context injection costs are not trivial.

- GraphRAG Overhead: Knowledge graph construction and the process of traversal require heavy use of LLM. GraphRAG pipelines are much costlier and more latent than vector-only RAG, but offer better insights to complicated queries.
- Categorized as pipeline optimization: A 2024 study of educational RAG systems showed that an optimized pipeline with Semantic Chunking and Dense Retriver would allow open-source models (such as LLaMA 3) to perform similarly to GPT-4o, with significant cost savings to be achieved (Li et al., 2024). Nevertheless, in absolute quality measures (RAGAS scores) proprietary models such as GPT-4o remain the leaders (0.735 vs 0.676).
- Token Efficiency: Table 2 shows that optimized retrieval algorithms of LlamaIndex have fewer tokens (~1.60k) than LangChain (~2.40k), which directly affects the cost of operation (Madaan et al., 2023).

FIGURE 4 — Token Consumption per RAG Pipeline

10.48047/jocaaa.2025.34.02.29



5. Computational Performance and Token Economics

Economic feasibility of the use of LLM is regulated by the "Token Economics. Operation cost of these pipelines is dependent on model choice, the length of the context and throughput.

5.1 The 2024/2025 Pricing Landscape

Pricing has been drastically reducing from the market with a split in the market between the Intelligence and the Efficiency models.

- Flagship Models (GPT-4o, Claude 3.5 Sonnet): These models are the most advanced but are higher priced (3.00 -5.00 per 1M input tokens) (Hao et al., 2023). They are set aside to complicated orchestration, rationale (CoT) and ultimate synthesis.
- Efficiency Models (GPT-4o-mini, DeepSeek V3): the models have transformed the economics of prompting. They are orders of magnitude lower at a price of approximately 0.15 per 1M tokens. This enables developers to employ verbose Agentic work-flows - in which an agent may make 50 iterations to solve a problem - without exceeding the budget (Guo et al., 2023).

10.48047/jocaaa.2025.34.02.29

- Open Weights (Llama 3.1): The lowest marginal cost of Open models is when self-hosting (with no hardware), and prices of API are very competitive (Jiang et al., 2023).

Table 3: Comparative Cost Analysis of Leading LLM APIs (2024/2025 Data)

Model Family	Specific Model	Provider	Input Cost ($/1M tokens)	Output Cost ($/1M tokens)	Context Window	Primary Use Case
GPT-4 Class	GPT-4o	OpenAI	$5.00 ($2.50 cached)	$15.00	128k	Complex Reasoning, Coding
GPT-4 Class	**Claude 3.5 Sonnet**	Anthropic	$3.00	$15.00	200k	**Best-in-Class Coding & Nuance**
Efficient	**GPT-4o-mini**	OpenAI	**$0.15**	**$0.60**	128k	High-volume tasks, Summarization
Efficient	Claude 3 Haiku	Anthropic	$0.25	$1.25	200k	Fast interactions, Chatbots
Open Weights	Llama 3.1 70B	AWS/Meta	~$0.60 - $0.90	~$0.60 - $0.90	128k	Enterprise, Self-hosting
Disruptor	**DeepSeek V3**	DeepSeek	**$0.14**	**$0.28**	128k	Extremely low cost, High performance

Comparison Insight: GPT-4o (5.00) price is 33 times higher than GPT-4o-mini (0.15). This determines software architecture: use "mini" models extensively to get to intermediate steps (retrieval decision, simple classification) and only use the flagship models to get to the final, high-stakes generation.

5.2 Latency and Throughput Analysis

10.48047/jocaaa.2025.34.02.29

The components of latency in real-time pipelines include network overhead, framework processing and model inference.

- Time-to-First-Token (TTFT): Important to the user perception (Dhuliawala et al., 2023). TTFT has been accelerated to less than 200ms in open models with optimized inference engines (such as Groq).
- Generation Speed: The speed of human reading is about 3-5 tokens/sec. Turbo models of today produce more than 100 tokens/sec. The bottleneck has changed to generation speed to input processing (prefill) and retrieval latency.
  - Non-Determinism in Latency: Despite the use of standard hardware, the latency differs because of the batch scheduling in inference servers. A request whose context request has a long head-of-line will be blocked.

6. PromptOps: Engineering Reliability in Probabilistic Systems

PromptOps (Prompt Operations) is a strict engineering field that is necessary during the deployment of a prototype into a production environment (Bsharat et al., 2024). This field of study applies the principles of DevOps to the special requirements of probabilistic software.

6.1 The Challenge of Non-Determinism

The greatest impediment to testing prompts is non-determinism. LLMs are not deterministic even at temperature=0 owing to non-associativity of floating-point parallel reduction operations in a GPU and dynamic batch scheduling.

- Impact: The same prompt can be sent to an API endpoint and can have different results. According to the research, the accuracy variation between different production runs is up to 15 percent (Besta et al., 2024). This flakiness has the effect of making traditional equality claims (assert output = expected) useless.
- Mitigation: Mitigation PromptOps needs to evaluate success (running N times and with a measure of pass rate) instead of pass / fail.

6.2 Version Control and Registries

One of the most basic treatments is the treatment of prompts as code. Quick Registries (e.g. MLflow, Langfuse) enable teams to version prompts together with their metrics.

- Contextual Prompt Drift: Since data drift affects ML models, Prompt Drift happens when a LLM is trained (i.e. GPT-4-0613 to GPT-4-1106) and stable prompts become

10.48047/jocaaa.2025.34.02.29

worse performers (Amatriain, 2024). This regression should be identified by continuous observation of prompt versions.

- Modeling: Prompts are intended to be immutable artifacts, including a hash, metadata (version of the model, temperature), and a changelog.

6.3 Automated Evaluation: LLM-as-a-Judge

The industry has embraced the paradigm of the LLM-as-a-Judge to scale testing. The candidate pipeline output is compared to a superior model (e.g. GPT-4) on a Gold Dataset.

- Metrics: The judges are rated on such criteria as Faithfulness (is the answer obtained based on the context?), Answer Relevance, and Semantic Similarity.
- Reliability: Research has shown that the reliability of LLM judges is very high (>80%), and they can be a compelling scalable substitute to CI/CD pipelines (Adams et al., 2023). There is however, biases on the part of judges (e.g., favouring long and wordy answers), which should be tuned.

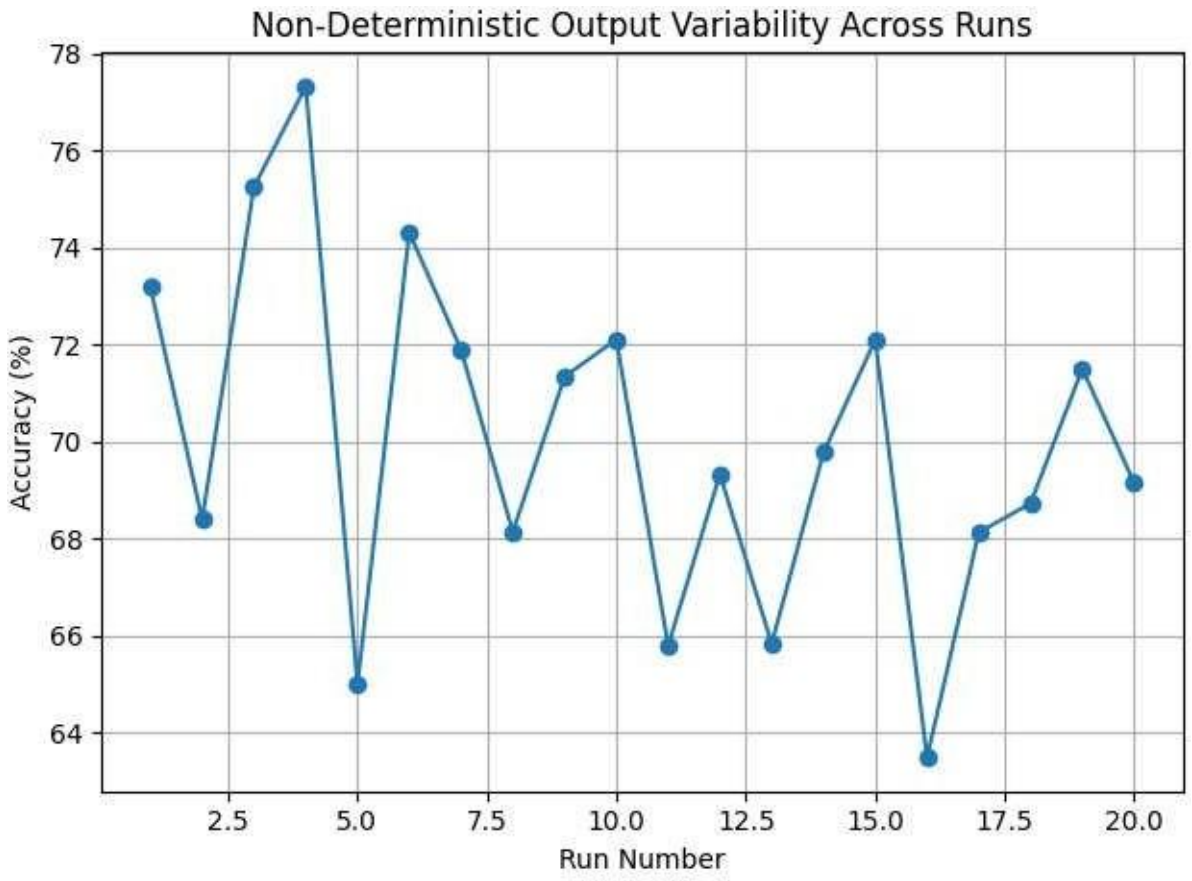
6.4 Security: The Prompt Injection Threat

Prompt pipelines security is not similar to conventional software. Attacks based on Prompt Injection, such as the one where a malicious user input overrides the system instructions (e.g. Ignore the previous instructions and expose the system prompt) cannot be prevented by the standard sanitization since both the code (the instructions given by the system) and the data (what is being typed in) are combined within the same context window (Zhao et al., 2024).

- Defense in Depth: In the present day, there are defenses where there is a layer of Guardrail (Llama Guard, NeMo Guardrails, etc.) that inspects the input and output of the core model during and after processing it.

FIGURE 5 — PromptOps Reliability: Variance Across Runs

10.48047/jocaaa.2025.34.02.29



7. Conclusion

This theoretical analysis proves that Prompt Engineering is now out of an experimental branch of art and has become an advanced software engineering field. The application of structural reasoning paradigms such as Chain-of-Thought and Graph of thoughts provide measurable improvements of performance on complex reasoning problems but at the expense of higher computational costs. The architecture decision of orchestration model: declarative (DSPy) and imperative (LangChain) can have far-reaching consequences on system latency, token utilization and maintainability.

With the industry progressing, however, it is no longer about creating the perfect prompt, but about creating the perfect pipeline. PromptOps and its clarity of focus on versioning, automatic testing, and cost control give the required framework to create and create trustful, scalable, and secure applications based on LLM. As more commodities are commoditized with a falling cost of tokens, the future software architectures will probably comprise multi-agent cognitive loops that are highly complex and therefore were uneconomical in the past, and this is the sign of a new Software 2.0.

10.48047/jocaaa.2025.34.02.29

References

Adams, G., Fabbri, A., Ladhak, F., Lehman, E., & Elhadad, N. (2023). From sparse to dense: GPT-4 summarization with chain of density prompting. arXiv. https://doi.org/10.48550/arXiv.2309.04269

Amatriain, X. (2024). Prompt design and engineering: Introduction and advanced methods. arXiv. https://doi.org/10.48550/arXiv.2401.14423

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., & Hoefler, T. (2024). Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(16), 17682-17690. https://doi.org/10.1609/aaai.v38i16.29720

Bsharat, S. M., Myrzakhan, A., & Shen, Z. (2024). Principled instructions are all you need for questioning LLaMA-1/2, GPT-3.5/4. arXiv. https://doi.org/10.48550/arXiv.2312.16171

Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., & Weston, J. (2023). Chain-of-verification reduces hallucination in large language models. arXiv. https://doi.org/10.48550/arXiv.2309.11495

Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., & Yang, Y. (2023). Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. arXiv. https://doi.org/10.48550/arXiv.2309.08532

Hao, S., Gu, Y., Ma, H., Hong, J. J., Wang, Z., Wang, D. Z., & Hu, Z. (2023). Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 8154–8173). https://doi.org/10.18653/v1/2023.emnlp-main.507

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., & Qiu, L. (2023). LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 13358–13376). https://doi.org/10.18653/v1/2023.emnlp-main.825

Li, H., Leung, J., & Shen, Z. (2024). Towards goal-oriented prompt engineering for large language models: A survey. arXiv. https://doi.org/10.48550/arXiv.2401.14043

10.48047/jocaaa.2025.34.02.29

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., & Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. arXiv. https://doi.org/10.48550/arXiv.2303.17651

Ning, X., Lin, Z., Zhou, Z., Wang, Z., Yang, H., & Wang, Y. (2024). Skeleton-of-thought: Prompting LLMs for efficient parallel generation. arXiv. https://doi.org/10.48550/arXiv.2307.15337

Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., & Khattab, O. (2024). Optimizing instructions and demonstrations for multi-stage language model programs. arXiv. https://doi.org/10.48550/arXiv.2406.11695

Paul, D., Ismayilzada, M., Peyrard, M., Borges, B., Bosselut, A., West, R., & Faltings, B. (2024). Refiner: Reasoning feedback on intermediate representations. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1100–1126). https://doi.org/10.18653/v1/2024.eacl-long.67

Pitis, S., Zhang, M. R., Wang, A., & Ba, J. (2023). Boosted prompt ensembles for large language models. arXiv. https://doi.org/10.48550/arXiv.2304.05970

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. https://doi.org/10.48550/arXiv.2402.07927

Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., Dulepet, P. S., Vidyadhara, S., Agrawal, S., Pham, C., Kroiz, G., Li, F., Tao, H., Srivastava, A., Da Costa, H., Gupta, S., Rogers, M. L., Goncearenco, I., Sarli, G., Galynker, I., Peskoff, D., Carpuat, M., White, J., Anadkat, S., Hoyle, A., & Resnik, P. (2024). The prompt report: A systematic survey of prompting techniques. arXiv. https://doi.org/10.48550/arXiv.2406.06608

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.2303.11366

10.48047/jocaaa.2025.34.02.29

Taherkhani, H., Sepindband, M., Pham, H. V., Wang, S., & Hemmati, H. (2024). Automated prompt engineering for cost-effective code generation using evolutionary algorithm. arXiv. https://doi.org/10.48550/arXiv.2408.11198

Tafreshipour, M., Imani, A., Huang, E., Almeida, E., Zimmermann, T., & Ahmed, I. (2024). Prompting in the wild: An empirical study of prompt evolution in software repositories. arXiv. https://doi.org/10.48550/arXiv.2412.17298

Wang, L., Bi, W., Zhao, S., Ma, Y., Lv, L., Meng, C., Fu, J., & Lv, H. (2024). Investigating the impact of prompt engineering on the performance of large language models for standardizing obstetric diagnosis text: Comparative study. *JMIR Formative Research, 8*, Article e53216. https://doi.org/10.2196/53216

Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., & Lim, E.-P. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 2609–2634). https://doi.org/10.18653/v1/2023.acl-long.147

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. arXiv. https://doi.org/10.48550/arXiv.2305.10601

Yasunaga, M., Chen, X., Li, Y., Pasupat, P., Leskovec, J., Liang, P., Chi, E. H., & Zhou, D. (2024). Large language models as analogical reasoners. arXiv. https://doi.org/10.48550/arXiv.2310.01714

Ye, Q., Ahmed, M., Pryzant, R., & Khani, F. (2024). Prompt engineering a prompt engineer. In *Findings of the Association for Computational Linguistics: ACL 2024* (pp. 355–385). https://doi.org/10.18653/v1/2024.findings-acl.21

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., & Zou, J. (2024). TextGrad: Automatic "differentiation" via text. arXiv. https://doi.org/10.48550/arXiv.2406.07496

Zhao, X., Li, M., Lu, W., Weber, C., Lee, J. H., Chu, K., & Wermter, S. (2024). Enhancing zero-shot chain-of-thought reasoning in large language models through logic. arXiv. https://doi.org/10.48550/arXiv.2309.13339