Frontiers in Computer Science and Artificial Intelligence
ISSN: 2978-8048
DOI: 10.32996/fcsai
Journal Homepage: www.al-kindipublisher.com/index.php/fcsai

FCSAI
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Cost-Aware Prompt Engineering: A Governance Architecture for Optimizing Enterprise AI Agents

Prasad Maderamitla
Independent Researcher, California, USA
Corresponding Author: Prasad Maderamitla**, E-mail:** prasad.madera@gmail.com

| ABSTRACT

As enterprises deploy large language model (LLM)-based and agentic AI systems across critical business functions, prompt engineering has emerged as a significant operational bottleneck. Prompts that are poorly structured, excessively verbose, or manually tuned without systematic comparison introduce hidden costs: elevated token consumption, increased inference latency, inconsistent output quality, and deployment risk. Despite its importance, prompt engineering in most enterprise environments remains an informal, trial-and-error discipline without governance, reproducibility, or measurable quality criteria. This paper proposes a Cost-Aware Prompt Engineering Governance Architecture (CAPEGA) designed for enterprise production environments. The architecture integrates four core components: iterative prompt optimization using SELF-REFINE techniques, structured prompt compression with instruction-preservation controls, metric-driven prompt comparison with cost and latency artifact generation, and CI/CD-ready governance workflows for prompt promotion and rollback. CAPEGA's contribution is architectural integration rather than novel algorithmic techniques: it combines established optimization methods into a unified, production-ready governance pipeline. Together, these components transform prompt engineering from an ad hoc craft into a reproducible, auditable engineering discipline. The framework is model-agnostic in design, model-specific in execution, and intended for integration into existing enterprise AI release pipelines. By establishing measurable quality gates for prompts, CAPEGA enables organizations to reduce token overhead, improve output consistency, control inference cost, and generate audit-ready evidence for production deployment decisions. An implementation architecture with component-level design is presented alongside the conceptual framework.

| KEYWORDS

Prompt Engineering, Enterprise AI, Cost-Aware AI, SELF-REFINE, LLM Optimization, AI Governance, CI/CD, Agentic AI, Token Efficiency, Production Readiness

| ARTICLE INFORMATION

ACCEPTED: 01 April 2026 **PUBLISHED:** 28 May 2026 **DOI:** 10.32996/jcsts.2026.5.8.2

1. Introduction

The deployment of large language model (LLM)-based systems and AI agents in enterprise environments has grown substantially in recent years. Organizations across financial services, healthcare, technology, and operations now use LLM-based systems to power customer-facing assistants, internal knowledge tools, automated decision support, and multi-step agentic workflows. As these systems move from experimental pilots into production, the engineering challenges associated with managing, evaluating, and optimizing prompts have become increasingly visible and costly.

Prompt engineering—the practice of designing, testing, and iterating on the natural language instructions that govern LLM behavior—is central to how these systems perform. A well-designed prompt improves output quality, reduces hallucination, enforces instruction-following behavior, and controls the scope of agent actions. A poorly designed prompt introduces

ambiguity, increases token overhead, produces inconsistent outputs, and in multi-step agentic systems, can cause cascading failures across tool calls and decision chains.

Despite its centrality to AI system quality, prompt engineering in most enterprise environments is treated as informal iteration. Engineers write prompts based on intuition, test them manually, and promote changes based on subjective judgment. There is rarely a systematic method for comparing prompt versions, measuring the cost or latency impact of a change, preserving critical instruction components during optimization, or creating audit-ready documentation of why a particular prompt was selected for production.

This paper addresses this gap by proposing a Cost-Aware Prompt Engineering Governance Architecture (CAPEGA). CAPEGA is an integration architecture for making prompt engineering a structured, measurable, and governable engineering discipline within enterprise AI development pipelines. It is designed to integrate with existing release workflows, is model-agnostic in design, and does not require new infrastructure beyond what most enterprise AI teams already operate.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 defines the problem. Section 4 presents the CAPEGA framework and its four core components. Section 5 presents the implementation architecture. Section 6 discusses implementation considerations. Section 7 discusses implications and limitations. Section 8 concludes.

2. Related Work

Research on prompt engineering has grown rapidly alongside the adoption of large language models. Early foundational work established the importance of prompt structure, few-shot examples, and chain-of-thought reasoning on model output quality (Brown et al., 2020; Wei et al., 2022). These contributions demonstrated that prompt design has a measurable and significant effect on LLM behavior, establishing prompt engineering as a legitimate technical discipline.

Subsequent work introduced systematic approaches to prompt optimization. Yao et al. (2023) proposed ReAct, a framework combining reasoning and action in prompts for agent tasks. Madaan et al. (2023) introduced SELF-REFINE, an iterative technique in which the model critiques and revises its own outputs over multiple passes, improving quality without requiring additional training data or human labeling. Prompt compression has been explored through approaches such as LLMLingua (Jiang et al., 2023), which reduces prompt token count while preserving semantic content, achieving 2-5x compression ratios without degrading output quality on standard benchmarks.

Programmatic prompt optimization has been advanced by DSPy (Khattab et al., 2023), which treats prompts as modular programs and optimizes them through compilation rather than manual iteration. Yang et al. (2023) proposed OPRO, using LLMs themselves as optimizers to search over prompt space. These approaches automate prompt improvement but focus on optimization algorithms rather than enterprise governance, cost tracking, or deployment lifecycle management.

Parallel work in the MLOps and AI engineering communities has examined how software engineering practices can be applied to AI systems. Sculley et al. (2015) identified the operational complexity of machine learning systems and the hidden costs of technical debt in ML pipelines. More recent work has extended these concerns to LLM-based systems, highlighting the need for systematic evaluation, version control, and governance (Liang et al., 2022; Bommasani et al., 2022).

Despite this body of work, a gap remains in enterprise-grade governance of the prompt engineering lifecycle. Existing optimization techniques (SELF-REFINE, DSPy, OPRO, LLMLingua) address individual dimensions of prompt quality but are not designed for integration into enterprise release governance, cost tracking, or audit workflows. CAPEGA addresses this gap by providing an integrated architecture that combines optimization, compression, evaluation, and governance into a production-ready engineering framework, complementing rather than replacing these individual techniques.

3. Problem Statement

In enterprise AI deployments, prompt engineering is a high-stakes activity currently governed by informal, non-reproducible practices that do not produce the audit-ready evidence required for responsible production deployment. Organizations face the following interconnected challenges:

Token overhead and cost growth. As LLM systems scale across business functions, inference cost grows proportionally with prompt length and call volume. Prompts with redundant instructions, excessive context, or poorly compressed background information consume more tokens per call than necessary. At enterprise scale, this represents a significant and often untracked operational cost.

Absence of prompt version governance. Unlike application code managed through version control with review processes and rollback capabilities, prompt changes are made informally. There is no systematic record of what changed between versions, why a change was made, or what its measured impact was.

No standardized quality criteria. Organizations promoting a prompt to production rely on qualitative assessment: manual tests and subjective judgment. There are no standardized criteria for production-ready prompts analogous to test coverage or performance benchmarks in software engineering.

Latency and reliability impact. Long prompts increase time-to-first-token and overall response latency. In real-time applications such as customer-facing assistants, latency is a user-facing quality measure. Unoptimized prompts directly degrade user experience.

Instruction preservation risk. Naive compression or rewriting can inadvertently remove critical instructions related to safety constraints, access boundaries, output format requirements, or compliance-relevant behaviors. Without explicit preservation controls, optimization introduces regression risk.

Non-deterministic model behavior. AI agents produce probabilistic responses that can fail through hallucination, poor grounding, incomplete reasoning, policy violation, or inconsistent instruction following. The platform must evaluate reliability across repeated runs.

These challenges mean most organizations cannot answer basic questions about their prompts: Is this prompt as efficient as it could be? What did it cost last month? What changed between production and the tested version? Why was this version promoted? CAPEGA is designed to make these questions answerable.

4. The CAPEGA Framework

The Cost-Aware Prompt Engineering Governance Architecture consists of four core components that operate in sequence, forming a governed engineering pipeline from prompt development through production deployment.

4.1 SELF-REFINE Iterative Optimization

The first component applies SELF-REFINE techniques (Madaan et al., 2023) to improve candidate prompts before they enter compression and comparison stages. In the CAPEGA implementation, SELF-REFINE is applied at the prompt level: a candidate prompt is submitted to a refinement loop in which the model evaluates the prompt against criteria including instruction clarity, specificity, potential for ambiguity, redundancy, and alignment with task scope. The model generates a revised prompt and a critique explaining changes.

This process serves two purposes. First, it improves prompt quality before optimization, reducing the likelihood that compression will need to compensate for structural problems. Second, it generates a documented revision history showing the prompt at each iteration and what improvements were made, becoming part of the governance artifact.

A critical design decision: the system tracks the global best prompt across all iterations, not just the final iteration. Prompt optimization is non-monotonic—the last refinement may not be the best. Selection is based on measured performance, not iteration order.

Critique criteria should be defined at the organizational level and reflect deployment-specific quality requirements: system type, audience, compliance requirements, and costliest failure modes.

4.2 Structured Prompt Compression with Instruction-Preservation Controls

The second component applies structured compression to the refined candidate prompt with explicit controls to protect instruction elements that must not be removed or altered. Compression reduces token count by removing redundant phrasing, consolidating overlapping instructions, eliminating verbose clauses, and simplifying structures while preserving semantic content.

The instruction-preservation control mechanism is the key distinguishing feature. Before compression begins, a preservation analysis identifies protected elements:

- Safety and content policy constraints restricting model output
- Access boundary instructions limiting data sources, tools, or actions
- Output format specifications that downstream systems depend on for parsing
- Compliance-relevant instructions governing regulated data handling
- Identity and persona instructions defining the model's enterprise role

- Structured data sections (schemas, examples, reference data)

Protected elements are tagged before compression and excluded from modification. After compression, a preservation verification step confirms all tagged elements remain intact. Any compression requiring modification of a protected element is flagged for explicit human review.

The compression pipeline operates in three stages: rule-based compression (removing filler phrases, deduplication), semantic deduplication (identifying semantically equivalent instructions), and LLM-based compression (restructuring for conciseness). Quality validation occurs after each stage, with fallback to the prior prompt if validation fails.

The output is a compressed prompt with a preservation log documenting: original token count, compressed token count, percentage reduction, protected elements identified, and confirmation of preservation integrity.

4.3 Metric-Driven Comparison and Cost/Latency Artifact Generation

The third component establishes a systematic basis for comparing prompt versions and generates a structured artifact documenting results. This artifact serves as primary evidence for governance decisions.

Prompt comparison is structured around five measurement dimensions:

Output quality. Evaluation cases representing realistic enterprise scenarios are run against both the baseline (current production) and candidate prompt. Quality is assessed using both LLM-as-judge scoring and rule-based metrics across categories:

Category	Metrics
Factual quality	Factuality, answer correctness
Structural quality	Coherence, completeness, conciseness
Instruction quality	Instruction following compliance
Context quality	Context recall, context precision, context relevance
Semantic quality	LLM semantic similarity, cosine similarity, answer similarity

Token efficiency. Total token count per version, average tokens per inference call, and percentage change between versions.

Inference latency. Time-to-first-token and total response latency: mean, median, and 95th percentile values under representative load conditions.

Estimated cost impact. Based on token consumption and provider pricing, estimated cost per call and cost at production scale.

Reliability. Pass-rate across repeated runs, capturing stochastic variance and consistency characteristics.

The comparison artifact includes all five dimensions for both prompts (absolute values and percentage differences), the SELF-REFINE revision history, and the preservation log. A key design principle: no prompt version should be promoted based on quality alone. Cost, latency, and reliability data must be present for a promotion decision to proceed.

4.4 CI/CD-Ready Governance Workflow for Prompt Promotion and Rollback

The fourth component provides the governance layer integrating outputs of the first three components into an enterprise release pipeline.

Development. A prompt change is submitted to Stage 1 (optimization) and Stage 2 (compression). Outputs are committed to a version-controlled prompt repository with the original draft and preservation log.

Evaluation. The Stage 3 comparison artifact is generated against baseline and candidate.

Review gate. The promotion request and artifact are reviewed by designated stakeholders (engineering, product, compliance). Reviewers must approve with recorded rationale or reject with documented reasons. Promotion cannot proceed without recorded approval.

Deployment. Approved prompts deploy through the standard CI/CD pipeline. Deployment is logged with timestamp, approving reviewers, and comparison artifact.

Rollback. If regression is detected post-deployment, the prior production version is restored from the repository. The rollback event is logged with timestamp and reason. The failed candidate is flagged for investigation.

Promotion criteria (configurable, recommended baseline):

- All protected elements verified present in compressed candidate
- At least one quality dimension showing improvement or parity
- No latency increase above defined threshold
- Cost impact documented and reviewed
- Pass-rate above minimum reliability threshold

Requests not meeting criteria are blocked from the review gate until satisfied.

5. Implementation Architecture

This section presents the component-level architecture for a CAPEGA implementation, describing the runtime system that executes the framework defined in Section 4.

5.1 Architecture Overview

The implementation is organized as a layered architecture with the Evaluation Framework acting as a central control plane for agent quality. Every prompt, agent behavior, scoring configuration, and production promotion decision flows through an evaluation lifecycle.

The system comprises the following components:

User Interface / Entry Point. A command-line interface accepts execution mode and parameters, triggers evaluation, optimization, compression, or pass-rate workflows, and routes requests to the runner layer. Future interfaces include REST API, web dashboard, CI/CD job runner, and scheduled batch evaluation.

Runner Layer. Orchestrates end-to-end workflows: loading prompts and expected responses, invoking LLM generation, evaluating metrics, aggregating and comparing results, invoking optimization and compression, and persisting output artifacts. Remains workflow-oriented without embedding metric logic or LLM communication details.

LLM Client. Provides asynchronous communication with the enterprise LLM gateway. Sends prompts to approved models, supports single and batch generation, manages authentication, and isolates gateway details from evaluation logic.

Metric Evaluation Engine. Scores generated responses using both LLM-judged and rule-based metrics. Supports factual quality, structural quality, instruction quality, context quality, and semantic quality metrics.

SELF-REFINE Optimizer. Improves prompts through iterative feedback and refinement. Generates baseline response, evaluates quality, generates feedback, refines prompt, validates structure, compares performance, and tracks the global best.

Feedback Evaluator. Measures quality of generated feedback across actionability, specificity, completeness, and impact. Prevents the system from blindly trusting generated critiques.

Prompt Compressor. Reduces prompt length in three stages: rule-based compression, semantic deduplication, and LLM-based compression. Produces compressed prompt and cost report.

Output Artifact Store. Contains evaluation results, reports, optimized prompts, compressed prompts, comparison summaries, and pass-rate reports for debugging, dashboards, audit, and automation.

5.2 Key Execution Flows

Multi-Metric Evaluation Flow. Evaluates a prompt across multiple quality dimensions, producing per-run metric scores, aggregate statistics, metric-specific analysis, dynamic criteria used for scoring, and recommended improvements.

SELF-REFINE Optimization Flow. Iteratively generates responses, evaluates them, produces feedback, refines the prompt, and compares performance. The optimizer preserves critical prompt structure, evaluates each iteration before adoption, and selects the best based on measured performance.

Prompt Comparison Flow. Runs both original and optimized prompts through the shared evaluation layer across repeated runs, producing a promotion decision. Identifies metric-specific regressions and prevents subjective promotion without evidence.

Prompt Compression Flow. Takes the optimized prompt through rule-based compression, semantic deduplication, and LLM-based compression with quality validation at each stage. Produces compressed prompt and cost report. Compressed prompts must be re-evaluated before production promotion.

Pass-Rate Evaluation Flow. Lightweight threshold-based validation for smoke testing and CI/CD gates. Generates N responses, scores similarity, evaluates pass/fail, and produces a CI/CD gate decision. Useful for quick regression checks and deployment gates.

5.3 Output Artifacts

Artifact	Purpose
Metric results JSON	Raw per-run evaluation results
Metric analysis JSON	Metric-level explanations and recommendations
Dynamic criteria files	Requirements extracted from the prompt
Optimized prompt	Final prompt selected by the optimizer
Compressed prompt	Token-reduced prompt
Compression report	Cost and length reduction details
Self-refine history	Iteration-level optimization trace
Prompt comparison report	Original vs. optimized performance
CSV summary	Dashboard and graphing support
Pass-rate report	Threshold-based evaluation results

5.4 Core Data Objects

Object	Description
Prompt	Original or optimized instruction sent to the LLM
Expected response	Ground truth or reference answer
Generated response	Model output produced from a prompt
Metric result	Score and optional explanation for one metric
Evaluation run	Collection of generated responses and metric scores
Optimization iteration	One cycle of generate, evaluate, feedback, refine, and re-evaluate
Compression report	Stage-by-stage compression analysis
Comparison report	Original vs. optimized performance summary

5.5 Dependencies

Dependency	Purpose
Enterprise LLM gateway	Controlled model access
LLM provider	Generation and LLM-as-judge evaluation
Embedding model	Semantic similarity and deduplication
NLP libraries	BLEU, ROUGE, tokenization, and text processing
File system or object store	Input and output artifact storage
CI/CD platform	Regression and deployment gates

6. Implementation Considerations

6.1 Model Agnosticism

CAPEGA is model-agnostic in design but model-specific in execution. The framework does not assume a particular LLM provider or model family. However, prompt quality is inherently model-specific: a prompt refined for one model may behave differently on another. Organizations should apply optimization and evaluation using the same model and version targeted for production, and re-run the framework when upgrading model versions.

6.2 Evaluation Case Set Design

The quality of CAPEGA outputs depends significantly on the evaluation case set used in Stage 3. Organizations should invest in case sets that are drawn from real or realistic production scenarios, cover both common and edge cases, and include cases designed to test failure modes most costly in the deployment context.

6.3 Integration with Existing Tooling

CAPEGA complements existing LLM evaluation and observability tooling. Organizations using LangSmith, RAGAS, or custom frameworks can integrate these as quality evaluation mechanisms within Stage 3, with CAPEGA providing the governance layer connecting evaluation outputs to promotion decisions. The framework's contribution is integration of cost measurement, instruction preservation, and governance workflow into a coherent process.

6.4 Cost of the Framework Itself

Running SELF-REFINE iterations, compression, and evaluation consumes tokens. Organizations should establish a break-even threshold: prompts called below a minimum frequency (e.g., fewer than 1,000 calls/month) may not recoup optimization cost. The framework is most valuable for high-frequency prompts where marginal per-call savings compound significantly.

6.5 Compliance and Auditability

The audit trail generated by CAPEGA—version-controlled prompt history, revision logs, preservation records, comparison artifacts, and promotion decision logs—provides documentation that may be required by internal compliance functions or external regulators. Organizations should define retention policies consistent with their compliance obligations.

7. Discussion

CAPEGA addresses a gap that has become increasingly visible as enterprise AI deployments mature. The informal practices that characterize prompt engineering today are sufficient for experimentation but inadequate for production systems that must be reliable, auditable, cost-controlled, and continuously improved.

The integration architecture draws on established engineering principles—iterative refinement, structured testing, artifact-driven review, and CI/CD governance—and applies them to prompt management in enterprise LLM systems. The most significant design decisions include: using both LLM-as-judge and rule-based metrics for evaluation breadth, tracking the global best across optimization iterations rather than assuming monotonic improvement, preserving prompt structure during refinement to protect enterprise-critical instructions, and generating structured output artifacts for auditability.

Limitations

Empirical validation. CAPEGA is presented as an architecture-level framework grounded in published techniques that have individually demonstrated effectiveness. The integrated framework has not yet been evaluated in a controlled study measuring token cost reduction, latency improvement, and governance compliance across multiple deployments. Future work should focus on empirical evaluation in production settings.

Evaluation case set construction. Constructing high-quality evaluation cases requires organizational investment. Future work should explore semi-automated case generation from production logs and user feedback.

Multi-agent scope. The current framework addresses individual prompt optimization. In multi-agent architectures, a change to one prompt may affect downstream agents in ways not captured by single-prompt evaluation. Extending the framework to multi-agent prompt interaction analysis is an important future direction.

Human review bottleneck. High-risk use cases require human approval before deployment. As prompt change velocity increases, the review gate may become a bottleneck requiring risk-based tiering of review requirements.

8. Conclusion

This paper has presented CAPEGA, a Cost-Aware Prompt Engineering Governance Architecture for enterprise LLM and agentic AI systems. The framework integrates four components—SELF-REFINE iterative optimization, prompt compression with instruction-preservation controls, metric-driven comparison with cost and latency artifact generation, and CI/CD-ready governance workflows—into a unified engineering pipeline. An implementation architecture with component-level design demonstrates how the framework maps to a deployable system.

CAPEGA treats prompt engineering as a governed engineering discipline rather than an informal craft. By establishing measurable criteria, generating reproducible comparison artifacts, and embedding prompt changes in formal review workflows, the framework enables organizations to make evidence-based prompt decisions, track cost and performance impact over time, and maintain audit-ready records.

As LLM-based and agentic AI systems become central to enterprise operations, the quality of prompt engineering will have growing impact on system reliability, cost, and compliance posture. Frameworks such as CAPEGA represent a necessary step toward the engineering rigor that enterprise AI systems require.

References

[1]. Bommasani, R., Hudson, D. A., Aditi, E., et al. (2022). On the Opportunities and Risks of Foundation Models. Stanford Center for Research on Foundation Models. arXiv:2108.07258.

[2]. Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 1877-1901.

[3]. Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C. Y., Yang, Y., and Qiu, X. (2023). LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 13358-13376.

[4]. Khattab, O., Singhvi, A., Maheshwari, P., et al. (2023). DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv:2310.03714*.

[5]. Liang, P., Bommasani, R., Lee, T., et al. (2022). Holistic Evaluation of Language Models. *Transactions on Machine Learning Research*.

[6]. Madaan, A., Tandon, N., Gupta, P., et al. (2023). SELF-REFINE: Iterative Refinement with Self-Feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 36.

[7]. Sculley, D., Holt, G., Golovin, D., et al. (2015). Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems (NIPS)*, 28, 2503-2511.

[8]. Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35, 24824-24837.

[9]. Yang, C., Wang, X., Lu, Y., et al. (2023). Large Language Models as Optimizers. *arXiv:2309.03409*.

[10]. Yao, S., Zhao, J., Yu, D., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations (ICLR)*.