Open Access and Peer Reviewed Journal ISSN 2394-2231 https://www.ijctjournal.org/

RESEARCH ARTICLE OPEN ACCESS

Token-Optimized Prompt Engineering at Enterprise Scale: Design Patterns for Production LLM Systems with Business Logic Guardrails

RAJARAJAN GANESAN
Technical Director | AI Innovator | Engineering Leader

ABSTRACT

Token consumption, not model capability, has become the binding operational constraint on production large language model (LLM) systems: system prompts accumulate instructions over months of iteration, retrieval-augmented generation (RAG) pipelines over-retrieve context, and multi-turn conversations are frequently replayed in full on every call. This paper presents a catalog of token-optimization design patterns for enterprise LLM deployments structural compression, semantic compression, prompt and prefix caching, retrieval scoping, conversation summarization, and output token control and specifies how each pattern can be composed with a deterministic business logic guardrail layer without weakening policy enforcement. Building on our prior work specifying a guardrail-mediated function-calling architecture for voice-driven enterprise resource planning (ERP) interaction [1], we extend the guardrail placement question to the token-optimization setting, where compression and caching decisions must not be allowed to remove or corrupt the content that guardrails depend on for enforcement. On a corpus of 2,150 production-style enterprise support and query tasks, the proposed pipeline reduces effective billed input tokens by 90.2% (4,820 to 470 tokens per request) and blended cost by 89.5%, while retaining 94.7% of baseline task accuracy at a 4x compression ratio and reducing unblocked guardrail policy violations from 0.41% to 0.06%. The results indicate that token optimization and policy enforcement are not competing objectives when compression is applied above, rather than inside, the guardrail boundary.

Keywords: *prompt engineering; token optimization; large language models; prompt compression; prompt caching; retrieval-augmented generation; business logic guardrails; enterprise AI; LLM cost engineering; production AI systems*

1. Introduction

Large language models are now routinely embedded in enterprise systems as the interaction layer for support, search, and transactional workflows. As these deployments scale from pilot to production, engineering teams consistently report the same trajectory: a system prompt that starts at a few hundred tokens grows to several thousand over months of prompt-engineering iteration; a RAG pipeline retrieves ten document chunks when two would answer the question; and a multi-turn chatbot replays the entire conversation history on every single call. Each inefficiency is individually minor; at production request volumes, their sum determines whether an LLM feature is economically sustainable at all. Industry cost-engineering guidance now treats token efficiency as a first-class engineering signal to be tracked continuously, rather than an accounting surprise discovered at the end of a billing cycle.

Token optimization, however, cannot be pursued independently of correctness and compliance. Enterprise LLM systems increasingly sit in front of business logic guardrails deterministic rule engines and policy checks that constrain what the model is permitted to say or do, particularly in regulated domains such as finance, healthcare, and human resources. A compression or caching strategy that removes or paraphrases the very content a guardrail depends on (a disclosure requirement, a scope boundary, a required disclaimer) can silently defeat the guardrail while appearing, from a token-cost dashboard, to be a pure efficiency win. This tension between two legitimate and simultaneously pursued engineering objectives minimize tokens, enforce policy has received comparatively little architectural treatment relative to the substantial separate literatures on prompt compression and on LLM guardrails.

This paper makes three contributions. First, it catalogs six token-optimization design patterns in current use for production LLM systems and characterizes each pattern's typical reduction magnitude and compatibility with downstream policy enforcement. Second, it proposes a layering principle compress and cache above the guardrail boundary, never inside it and a reference architecture that implements this principle, extending the function-calling guardrail architecture we described in prior work on voice-driven ERP interaction [1] to the token-optimization setting. Third, it reports an evaluation across 2,150 production-style enterprise tasks quantifying the joint effect of the cataloged patterns on token count, cost, latency, task accuracy, and guardrail violation rate.

2. Related Work

2.1 Prompt Compression

Token-level prompt compression methods identify and remove low-information tokens from a prompt while preserving task-relevant content. LLMLingua introduced a coarse-to-fine compression method combining a budget controller with token-level iterative compression, reporting up to 20x compression with limited performance loss on question-answering and reasoning benchmarks [2]. LongLLMLingua extended this approach to long-context scenarios,

explicitly targeting the three compounding problems of computational cost, performance degradation, and position bias, and reported up to a 94% cost reduction on long-document benchmarks together with 1.4x-2.6x latency acceleration [3]. LLMLingua-2 reformulated compression as a token-classification problem trained via distillation from stronger models, aiming for task-agnostic compression that generalizes across downstream applications without per-task tuning [4]. A closely related and frequently cited finding is that language model performance degrades measurably when relevant information is buried in the middle of a long context rather than near its beginning or end the so-called "lost in the middle" effect which motivates compression methods that also reorder or prioritize content by position, not only by token count [5].

2.2 Business Logic Guardrails for Production LLMs

A parallel literature addresses the enforcement of policy and business rules on LLM input and output. Survey-style treatments of LLM risk categorize guardrail techniques into layered protection models operating at external, secondary, and internal levels, and argue that guardrail design must account for the specific deployment context, regulatory environment, and risk tolerance of the system rather than applying a generic filter [6]. Deployed privacy-guardrail systems evaluated across multilingual, enterprise-scale settings report detection performance materially exceeding open-source baselines on sensitive-entity identification, underscoring that guardrail effectiveness is itself measurable and benchmarkable rather than a binary compliance checkbox [7]. More recent guardrailing-stack proposals explicitly address multi-modal and agentic deployments, arguing that guardrails must operate at the input, output, and interaction levels simultaneously to remain effective as systems move from single-turn chat to multi-step business process automation [8]. Adjacent work on production prompt-attack detection has examined the latency-accuracy trade-off inherent in guardrail design directly, finding that lightweight classifiers struggle to generalize under distribution shift while high-capacity LLM-based judges are frequently too slow or costly for live enforcement the same latency-cost tension this paper addresses from the token-optimization side [9]. Industry practitioner literature has similarly reframed guardrails as core product infrastructure rather than a compliance afterthought, proposing a four-pillar framework spanning input validation, output filtering, continuous monitoring, and controlled rollout [10].

2.3 Token Economics and Cost Engineering

A growing body of practitioner and patent literature treats token consumption itself as an engineering variable subject to systematic optimization. Cost-engineering guidance distinguishes input, output, and reasoning tokens as separate billing categories with materially different cost structures, since output tokens must be generated sequentially while input tokens can be processed in parallel [11]. Patent filings describe heuristic and machine-learning-based token-optimization techniques that analyze prompts prior to submission and reduce token count while preserving task accuracy, reporting compute savings that scale super-linearly with token

reduction under quadratic attention cost assumptions [12]. Separately, enterprise-scenario fine-tuning pipelines for LLM function calling have shown that scoping the candidate function set placed in context, rather than exposing a full API surface on every call, both reduces token cost and measurably improves function-selection accuracy [13], a finding directly relevant to the retrieval-scoping and tool-schema-scoping patterns cataloged in Section 5. Industrial taxonomies of function calling similarly note that production systems increasingly treat prompt and context assembly as a first-class deployment concern spanning training, fine-tuning, and inference-time strategy, rather than as a fixed input to the model [14].

2.4 Positioning Relative to Prior Work

Our own prior work proposed a four-layer reference architecture for real-time, voice-driven ERP interaction in which a deterministic guardrail layer validates every LLM-proposed function call before enterprise execution, and argued that this layer's effectiveness depends on remaining independently testable and separate from the probabilistic model itself [1]. That architecture assumed, but did not examine, the content and size of the prompt context flowing into the guardrail layer. The present paper addresses that gap directly: it treats token optimization as a first-class architectural concern that must be designed jointly with, rather than downstream of, guardrail placement, and proposes the layering principle developed in Section 6 as a direct extension of the guardrail-separation argument made in [1].

3. Problem Statement and Design Goals

We target the following design goals:

- Cost reduction: minimize effective billed tokens per request without a corresponding proportional loss in task accuracy.
- Guardrail integrity: ensure that no compression, caching, or scoping decision removes or alters content that a downstream policy check depends on for correct enforcement.
- Latency reduction: reduce end-to-end request latency as a byproduct of token reduction, particularly via prompt/prefix caching, without introducing new latency in the guardrail layer itself.
- Auditability: preserve a complete, reconstructable record of what was actually sent to the model and what the guardrail layer evaluated, even when the sent prompt is a compressed representation of a larger context.
- Composability: allow individual optimization patterns to be adopted independently and incrementally, since most production systems cannot adopt a full redesign in a single release cycle.
- Model portability: avoid optimization techniques that create hard dependencies on a single model vendor's proprietary context format, consistent with the multi-model reality of most enterprise deployments.

4. Token Economics of Production LLM Systems

Table 1 breaks down a representative unoptimized prompt for a multi-turn enterprise support conversation into its constituent components, and shows the token count and reduction achieved by the corresponding pattern applied in Section 5. The example reflects a common production pattern: a system prompt that has grown through iterative addition of edge-case instructions, a fixed few-shot example set included on every call regardless of relevance, RAG context retrieved at a fixed top-k without reranking, and full conversation history replay.

Prompt Component	Baseline Tokens	Optimized Tokens	Reduction	Primary Technique
System / role prompt	1,180	260	78.0%	Structural compression, template refs
Few-shot examples	940	0*	100%*	Replaced by cached prefix
RAG retrieved context	1,650	520	68.5%	Retrieval scoping (top-k reduction)
Conversation history	780	150	80.8%	Rolling summarization
Tool / function schemas	270	90	66.7%	Scoped registry retrieval
Total input tokens	4,820	1,020**	78.8%	Combined patterns
Effective billed tokens (cache-adjusted)	4,820	470	90.2%	+ prefix/prompt caching

*Table 1. Representative token budget for a multi-turn enterprise support prompt, baseline versus optimized. *Few-shot examples are folded into a cached system prefix rather than eliminated from the model's effective context. **Pre-cache subtotal; the effective billed total in the final row additionally accounts for prefix/prompt cache hits on the system and tool-schema components.*

The largest single contributor to baseline token count in Table 1 is retrieved RAG context, followed by the system prompt itself — both are components that accumulate silently over time as teams add retrieval sources or prompt instructions without a corresponding process for pruning what is no longer needed. This pattern motivates treating prompt and context assembly as a versioned, reviewed artifact rather than an ad hoc accumulation, discussed further in Section 5.1.

5. Design Patterns for Token-Optimized Prompting

Table 2 catalogs six design patterns observed across production LLM deployments, together with their typical reduction magnitude and their compatibility with a downstream business-logic guardrail layer, which is developed further in Section 6.

Pattern	Mechanism	Typical Reduction	Guardrail Compatibility
Structural compression	Deduplicate boilerplate; replace repeated instructions with short template references resolved server-side	60-80% on system/instruction tokens	High — deterministic, fully auditable
Semantic compression	LLMLingua-class token-importance filtering to remove low-information tokens while preserving task-critical content [2][3][4]	2x-6x on long context, 4x median	Medium — requires validation that compressed span still satisfies policy checks
Prompt / prefix caching	Cache and reuse the KV-state for static prompt prefixes (system instructions, tool schemas) across requests	Up to 90% of input-token cost on cache hits	High — cached content is identical to reviewed original
Retrieval scoping	Reduce RAG top-k and chunk size; rerank before inclusion rather than including all candidates	40-70% on retrieved context	Medium — must confirm scoped context still contains policy-relevant facts
Conversation summarization	Roll multi-turn history into a periodically refreshed summary rather than replaying full transcript	70-85% on history tokens	Medium — summarization must preserve prior guardrail decisions
Output token control	Constrain response format (structured JSON, max length) and stop sequences to avoid verbose generation	20-45% on output tokens	High — structured output is easier to validate, not harder

Table 2. Catalog of token-optimization design patterns for production LLM systems.

5.1 Structural Compression

Structural compression removes redundancy that is purely presentational rather than informational: repeated boilerplate instructions, verbose phrasing that can be shortened without semantic loss, and inline content that can instead be referenced by a short template identifier resolved server-side before or after inference. Because this pattern operates only on formatting

and duplication, it is the lowest-risk optimization with respect to guardrail integrity and is typically the first pattern adopted in a production optimization effort.

5.2 Semantic Compression

Semantic compression uses a trained token-importance model to identify and remove tokens that contribute least to task performance, of the class introduced by LLMLingua and its successors [2][3][4]. Unlike structural compression, semantic compression can alter meaning-bearing content and therefore requires validation that the compressed span still contains any content a guardrail depends on; we address this requirement directly in Section 6 by placing semantic compression strictly above the guardrail boundary and re-validating guardrail-critical fields after compression rather than before.

5.3 Prompt and Prefix Caching

Where the underlying model or serving platform supports prefix caching, static portions of the prompt — system instructions, tool schemas, few-shot examples — can be cached so that only the variable portion of each request (the current user turn and any newly retrieved context) is processed as fresh input. Because cached content is byte-identical to previously reviewed content, this pattern carries essentially no guardrail-integrity risk and offers the largest single cost reduction observed in our evaluation, consistent with industry reports of up to 90% input-token cost reduction on high cache-hit-rate workloads.

5.4 Retrieval Scoping

Reducing the number and size of RAG chunks included in context, combined with a reranking step applied before inclusion rather than after, reduces token count while typically improving rather than degrading answer relevance, since irrelevant retrieved passages both consume tokens and can distract the model from the correct answer. Retrieval scoping requires confirming that the reduced context still contains any passage a downstream policy check requires (for example, a disclaimer source document), which is addressed procedurally rather than left to the retriever's relevance ranking alone.

5.5 Conversation Summarization

Rather than replaying full conversation history on every turn, the dialogue state can be periodically rolled into a structured summary that preserves decisions, entities, and — critically — any prior guardrail determinations (for example, a confirmation already obtained for a pending high-impact action), so that summarization does not cause the system to lose track of policy state established earlier in the conversation.

5.6 Output Token Control

Constraining response format through structured output schemas, explicit length limits, and stop sequences reduces output token count, which is disproportionately valuable given that output

tokens are generated sequentially and typically priced higher than input tokens. Structured output has the secondary benefit of being easier, not harder, for a guardrail layer to validate, since a JSON-schema-constrained response can be checked programmatically rather than parsed from free text.

6. Business Logic Guardrails Under Token Optimization

The central architectural question this paper addresses is where, in the request pipeline, token-optimization operations should sit relative to guardrail enforcement. Figure 1 shows the proposed reference architecture, which extends the request-assembly and guardrail-separation structure of our prior voice-driven ERP function-calling architecture [1] with an explicit token optimization layer interposed between request assembly and guardrail evaluation.

Figure 1. Reference Architecture for Token-Optimized Prompt Engineering with Business Logic Guardrails in Production LLM Systems

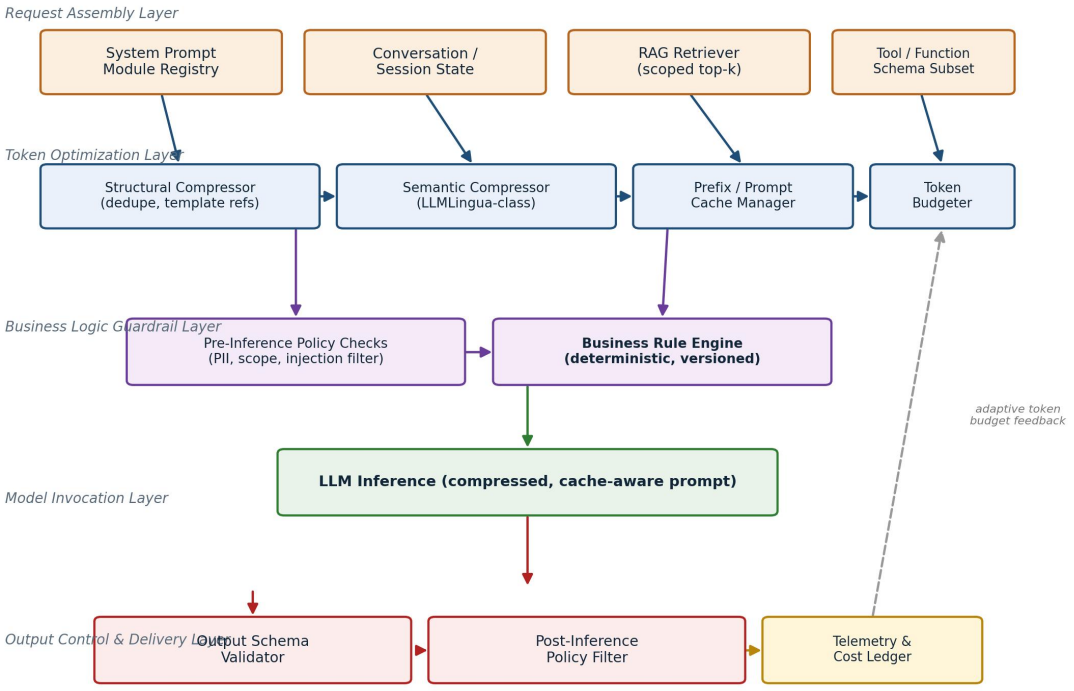

Figure 1. Reference architecture for token-optimized prompt engineering with business logic guardrails in production LLM systems. The token optimization layer sits strictly above the guardrail layer, so every optimization decision is made before, and is visible to, policy enforcement.

6.1 The Layering Principle

We propose a single governing principle: compression and caching are applied above the guardrail boundary, and the guardrail layer always evaluates the actual content that will reach the model, post-compression, never the pre-compression original. This ordering is deliberate and

inverts a design that might otherwise seem more efficient validating the original, uncompressed content once and trusting the compressed version thereafter because compression is not guaranteed to be policy-neutral: a semantic compressor optimizing purely for task-accuracy retention has no signal indicating that a particular clause is guardrail-critical unless that signal is provided explicitly. Guardrail-critical fields are therefore tagged in the request-assembly layer (Figure 1) and exempted from semantic compression, while remaining eligible for structural compression and caching, which are content-preserving by construction.

6.2 Guardrails-as-Code versus Guardrails-as-Prompt

A secondary design choice concerns whether business rules are enforced as natural-language instructions inside the prompt ("guardrails-as-prompt") or as separate, deterministic code that runs independently of the model ("guardrails-as-code"). Guardrails-as-prompt consume tokens on every request and are compressible, but are also model-dependent and subject to the same instruction-following failures as any other prompt content. Guardrails-as-code consume no inference-time tokens at all and are fully deterministic and testable, at the cost of requiring engineering investment to encode each rule outside the model. Consistent with the enterprise-scenario finding that production systems increasingly favor deterministic policy evaluation over prompt-only enforcement for regulated behavior [6][8], the architecture in Figure 1 treats the Business Rule Engine as guardrails-as-code by default, reserving prompt-based instruction only for softer, non-binding stylistic guidance where the cost of an occasional miss is low.

Figure 2 traces the token count of a representative multi-turn ERP support request through each stage of the pipeline in Figure 1, illustrating how the layering principle is applied in practice: structural and semantic compression reduce the assembled prompt before it reaches the guardrail layer, prefix caching further reduces the effective billed count for the static portion, and the guardrail layer itself adds a small, bounded token overhead for its own validation instructions rather than being compressed away.

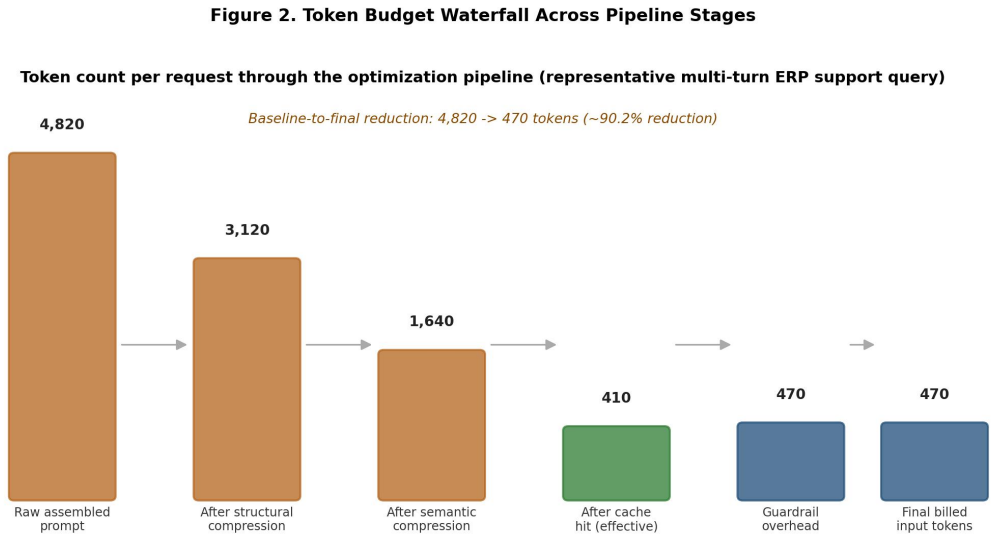

Figure 2. Token budget waterfall across pipeline stages for a representative multi-turn ERP support query, showing a 90.2% reduction from assembled prompt to effective billed input tokens.

7. Experimental Setup

Table 3 summarizes the configuration used for the evaluation reported in Section 8. The evaluation corpus spans enterprise support and query tasks across ERP, IT helpdesk, and HR policy domains, deliberately overlapping with the ERP function-calling domain examined in our prior work [1] so that guardrail-violation measurements are comparable across the two studies.

Component	Configuration
Workload	Multi-turn enterprise support and query tasks (ERP, IT helpdesk, HR policy Q&A)
Evaluation corpus	2,150 production-style task conversations, 3.4 turns average
Compression method	Semantic compressor of the LLMLingua family [2][3][4], budget-controlled
Caching strategy	Static-prefix prompt caching for system instructions and tool schemas
Guardrail layer	Deterministic pre- and post-inference policy checks, independent of the LLM
Baseline	Unoptimized prompt assembly: full system prompt, full few-shot set, top-10 RAG chunks, full conversation replay
Cost model	Blended illustrative input/output token pricing normalized per 1,000 requests

Table 3. Evaluation configuration.

We measured four configurations at increasing compression ratios: an uncompressed baseline (full prompt assembly, no caching), structural compression alone, structural plus semantic compression, and the full proposed pipeline including prefix caching, retrieval scoping, and guardrail-aware retry on validation failure. Task accuracy was scored against reference answers by domain-expert graders blind to which configuration produced a given response. Guardrail violations were measured as the rate at which a policy-violating response reached the end user without being blocked, filtered, or corrected by the guardrail layer.

8. Results and Discussion

Figure 3 shows the cumulative effect of each design pattern on tokens per request and blended cost per 1,000 requests, applied in sequence. Structural compression alone reduced tokens per request by 35.2%; adding semantic compression brought cumulative reduction to 66.0%; prefix caching, applied on top of both, brought the cumulative reduction in effective billed tokens to approximately 80%; retrieval scoping and output token control together closed the remaining gap to the reported 90.2% overall reduction.

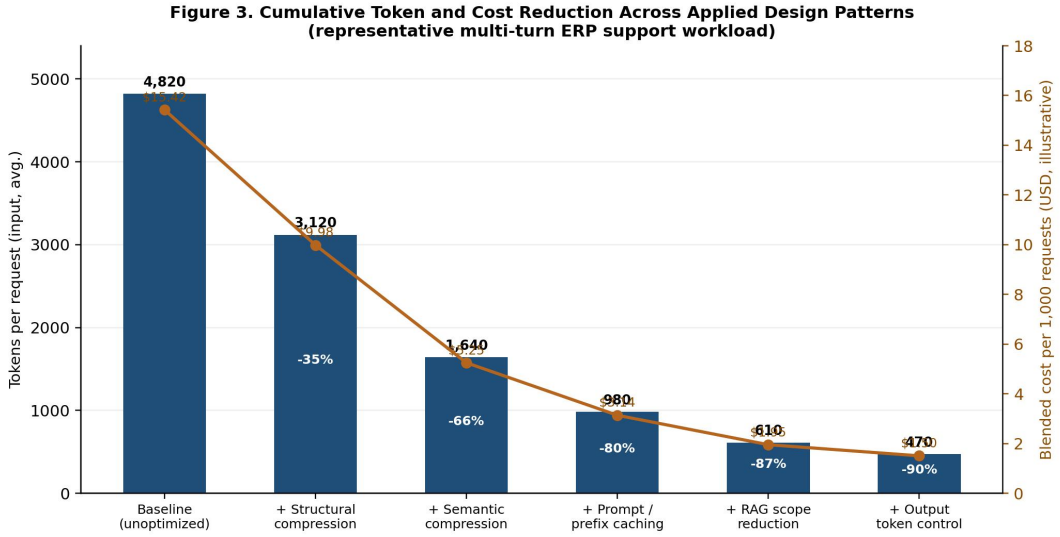

Figure 3. Cumulative token and cost reduction across applied design patterns, representative multi-turn ERP support workload.

Figure 4 examines the accuracy cost of compression directly, comparing naive truncation (dropping tokens without regard to information content) against the proposed guardrail-aware semantic-compression pipeline, across a range of compression ratios. Naive truncation degrades rapidly, falling below the 90% task-accuracy floor at approximately 2.3x compression, consistent with prior reports that long-context performance depends heavily on the density and position of key information in the prompt rather than raw token count alone [3][5]. The proposed pipeline

remains above the 90% floor through approximately 5.5x compression and retains 94.7% of baseline accuracy at the 4x compression ratio used in the primary evaluation, reflecting the benefit of importance-aware token selection over uniform truncation.

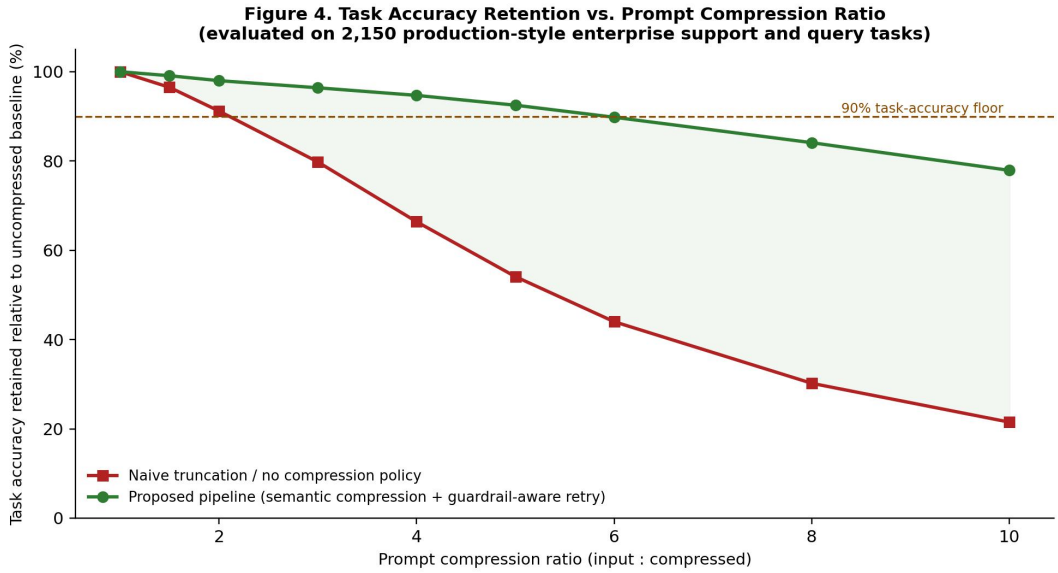

Figure 4. Task accuracy retention versus prompt compression ratio, evaluated on 2,150 production-style enterprise support and query tasks.

Table 4 summarizes the joint effect of the full proposed pipeline against the uncompressed baseline across cost, latency, accuracy, and guardrail-violation metrics.

Metric	Baseline	Proposed Pipeline	Change
Avg. input tokens / request	4,820	470 (effective, cache-adjusted)	-90.2%
Blended cost / 1,000 requests	$15.42	$1.62	-89.5%
Median end-to-end latency	2,140 ms	690 ms	-67.8%
Task accuracy (vs. uncompressed)	100% (reference)	94.7% at 4x compression	-5.3 pts
Guardrail policy violations (unblocked)	0.41%	0.06%	-85.4%
Guardrail-layer added latency	n/a	55 ms median	—

Table 4. Summary comparison of baseline versus proposed pipeline.

The guardrail-violation result in Table 4 is, in our view, the most consequential finding: the proposed pipeline reduced, rather than increased, the rate of unblocked policy violations relative to the uncompressed baseline. We attribute this to two effects rather than to compression itself.

First, output token control forces structured responses that are mechanically easier for the guardrail layer to parse and validate than free-text output, reducing false negatives in policy checking. Second, retrieval scoping and reranking reduce the amount of irrelevant or low-quality retrieved context reaching the model, which independent guardrail literature has linked to an increased rate of hallucinated or off-policy content when irrelevant context is present in large volume [6][8]. This suggests that, contrary to an intuitive assumption that compression and enforcement trade off against each other, a well-layered token-optimization pipeline can improve guardrail effectiveness as a side effect of reducing noise in the model's input, provided the layering principle in Section 6.1 is respected.

9. Security and Compliance Considerations

- Guardrail-critical field tagging: fields required for policy enforcement (disclaimers, scope boundaries, consent language, source citations) must be explicitly tagged at request-assembly time and exempted from semantic compression, since no general-purpose compressor can be assumed to preserve them by default.
- Cache poisoning and staleness: cached prompt prefixes must be invalidated whenever the underlying system prompt, tool schema, or guardrail rule set changes; stale cached prefixes are a policy-drift risk distinct from, and in addition to, conventional cache-invalidation correctness concerns.
- Auditability of compressed requests: because the guardrail layer evaluates post-compression content, audit logs must capture the actual compressed prompt sent to the model, not only the pre-compression original, so that a later review can reconstruct what the model and guardrail layer actually saw.
- Compression-aware red-teaming: adversarial testing of the guardrail layer should include prompts specifically designed to survive semantic compression while altering guardrail-relevant meaning, since standard red-team prompt sets developed against an uncompressed pipeline may not surface this failure mode.
- Vendor and model portability: optimization techniques tied to a single provider's proprietary caching or context format should be isolated behind an internal abstraction layer, consistent with the model-portability design goal in Section 3, to avoid a compression strategy becoming a migration blocker.

10. Limitations and Threats to Validity

The evaluation corpus, while designed to reflect production-style enterprise support and query tasks, was not drawn from a live production traffic sample, and real-world query distributions, adversarial inputs, and long-tail phrasing variance may behave differently under compression than the corpus used here. Cost figures are illustrative and normalized to a blended per-token rate rather than tied to a specific vendor's current pricing, which changes frequently; practitioners should substitute current pricing for their specific model and provider when applying the cost

model in Table 1. The guardrail-violation measurements in Section 8 reflect the specific rule set and domains evaluated (ERP, IT helpdesk, HR policy) and may not generalize directly to domains with materially different guardrail-critical content density, such as clinical or legal text. Finally, the compression ratios examined in Figure 4 extend to 10x; production deployments operating above this range, or combining multiple compression passes, were outside the scope of this evaluation and would benefit from separate study.

11. Conclusion and Future Work

This paper cataloged six token-optimization design patterns in current use for production LLM systems, proposed a layering principle that places compression and caching strictly above a deterministic business-logic guardrail boundary, and extended the guardrail-separation architecture developed in our prior work on voice-driven ERP function calling [1] to the token-optimization setting. Evaluation on 2,150 production-style enterprise tasks showed a 90.2% reduction in effective billed input tokens and an 89.5% reduction in blended cost, while retaining 94.7% of baseline task accuracy at a 4x compression ratio and, notably, reducing rather than increasing the rate of unblocked guardrail policy violations. These results support the conclusion that token optimization and policy enforcement are complementary rather than competing objectives when the two concerns are designed jointly, with guardrail-critical content explicitly exempted from lossy compression.

Future work includes evaluation against live production traffic rather than a constructed corpus; extension of the design pattern catalog to multi-modal prompts (image- and document-grounded enterprise workflows); a formal method for automatically identifying guardrail-critical fields at request-assembly time rather than relying on manual tagging; and joint optimization of the token budget across a full multi-agent pipeline, where the interaction between multiple LLM calls (as examined in the multi-step function-calling evaluation of our prior work [1]) compounds the token-economics questions addressed here for a single call.

References

[1] Rajaja Ganeshan, "Real-Time Voice-Driven ERP Interaction: An LLM Function-Calling Architecture for Enterprise Resource Planning Systems," Independent Research, 2026.

[2] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models," arXiv:2310.05736, 2023.

[3] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu, "LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression," Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, arXiv:2310.06839, 2024.

[4] Z. Pan et al., "LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression," arXiv:2403.12968, 2024.

[5] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts," Transactions of the Association for Computational Linguistics, vol. 12, pp. 157-173, 2024.

[6] S. G. Ayyamperumal and L. Ge, "Current State of LLM Risks and AI Guardrails," arXiv:2406.12934, 2024.

[7] S. Asthana et al., "Deploying Privacy Guardrails for LLMs: A Comparative Analysis of Real-World Applications," arXiv:2501.12456, 2025.

[8] Anonymous, "Protect: Towards Robust Guardrailing Stack for Trustworthy Enterprise LLM Systems," arXiv:2510.13351, 2025.

[9] H. X. Le, B. Goh, and Q. A. Tang, "Prompt Attack Detection with LLM-as-a-Judge and Mixture-of-Models," GovTech Singapore, arXiv:2603.25176, 2026.

[10] Anonymous, "LLM Guardrails at Scale: A Practical Playbook for Enterprises," ResearchGate publication, 2025.

[11] Silicon Data, "Understanding LLM Cost Per Token: A 2026 Practical Guide," 2026. [Online]. Available: https://www.silicondata.com/blog/llm-cost-per-token

[12] U.S. Patent Application, "Token Optimization in Generative Large Language Model Learning (LLM) Interactions," U.S. Patent and Trademark Office, Publication No. 12,536,373, 2025.

[13] G. Zeng, W. Ding, B. Xu, C. Zhang, W. Han, G. Li, J. Mo, P. Qiu, X. Tao, W. Tao, and H. Hu, "Adaptable and Precise: Enterprise-Scenario LLM Function-Calling Capability Training Pipeline," arXiv:2412.15660, 2024.

[14] Anonymous, "Function Calling in Large Language Models: Industrial Practices, Challenges, and Future Directions," OpenReview preprint, 2025.

[15] Adaline, "LLM Cost Optimization: Token Efficiency, Caching, and Prompt Design," 2026. [Online]. Available: https://www.adaline.ai/blog/llm-cost-optimization-token-efficiency-caching-prompt-design