Marginal Value of Tokens in Business LLM Applications: Measuring Cost, Latency, and Utility of Prompt Components

Md Shafiul Alam

Sentix Labs Inc

md.shafiul.alam@sentixlabs.com

April 2026

Abstract

Large language model (LLM) applications used in business settings are often optimized by informal prompt editing: shortening instructions, adding examples, increasing retrieved context, or imposing output constraints. Such edits are usually evaluated anecdotally, even though each prompt component affects quality, cost, latency, and operational risk. This paper introduces *Marginal Value of Tokens* (MVT), a component-level framework for measuring the incremental business utility of prompt segments relative to their token and cost footprint. The framework treats a prompt as a structured composition of functional components, including system instructions, task rules, business policy, retrieved context, chat history, few-shot examples, tool definitions, and output schemas. We define cost- and latency-adjusted utility, propose paired ablation and coalition-based estimators for component attribution, and give operational rules for classifying prompt components as high-value, low-value, negative-value, reusable, model-dependent, or workflow-dependent. The methodology is designed for common business workloads, including customer support and policy question answering, document summarization, and structured information extraction. The central argument is that business LLM systems should not minimize tokens blindly. They should maximize useful business output per token by preserving necessary context, pruning harmful context, compressing redundant history, caching reusable prefixes, and measuring prompt changes under non-inferiority constraints. The contribution is a practical measurement framework and experimental protocol for cost-efficient LLM adoption in business environments.

Keywords: large language models; prompt optimization; inference cost; prompt ablation; retrieval-augmented generation; prompt caching; business AI systems; evaluation.

1 Introduction

Large language models are increasingly embedded in business workflows such as customer support, document review, invoice processing, policy search, contract analysis, sales operations, and internal knowledge management. In these applications, the prompt is not merely a natural-language query. It is a structured control surface that combines instructions, policy constraints, retrieved evidence, examples, conversation state, tool definitions, and response-format requirements. Although model architecture and serving infrastructure determine the ultimate computational behavior of an LLM, most application teams exercise control at the prompt, retrieval, routing, and evaluation layers. This is especially true for organizations that consume LLM capability through managed services rather than operating their own inference stack.

Prompt length has a direct economic consequence: longer inputs increase prefill work, raise token-based charges, enlarge the surface for irrelevant information, and may increase latency. Yet a short prompt is not necessarily a good prompt. Removing a policy clause may reduce cost while

increasing business risk. Removing stale chat history may reduce cost without changing quality. Removing noisy retrieval may improve both cost and accuracy. Reordering a stable system prompt or tool schema may allow reuse through caching rather than deletion. These examples illustrate the main thesis of this paper: tokens are economically heterogeneous. Some tokens are necessary, some are redundant, some are harmful, and some are valuable primarily because they are reusable.

Current business practice often treats prompt optimization as craft. Teams manually inspect a few examples, remove wording that appears verbose, add examples when outputs look inconsistent, and increase retrieval size when the model hallucinates. Such iteration can improve a prototype, but it is not sufficient for production systems where cost, latency, quality, and compliance must be managed jointly. The absence of measurement is particularly problematic because prompt components interact. Few-shot examples may matter only when an output schema is absent. Retrieved context may help only when it is relevant and positioned effectively. Chat history may be useful in multi-turn workflows but harmful when it contains resolved or contradictory state. A component that is low-value for one model may be high-value for another.

This paper proposes *Marginal Value of Tokens* (MVT) as a framework for measuring prompt-component value. Instead of asking whether a prompt is long or short, MVT asks whether each component contributes enough utility to justify its token, cost, latency, and risk footprint. We define prompt components as functional prompt segments and measure their effect through paired ablation, randomized component coalitions, cost accounting, latency logging, quality scoring, and non-inferiority analysis. The framework is intended for applied AI teams that need to reduce LLM operating costs without degrading task outcomes.

The contributions are as follows:

1. We formalize the concept of Marginal Value of Tokens at the prompt-component level.
2. We define a business utility function combining quality, cost, latency, task failure, and risk penalties.
3. We provide estimators for component-level value using leave-one-component-out ablation and randomized coalition attribution.
4. We propose an experimental methodology for 60 business prompts across customer-support question answering, document summarization, and structured extraction.
5. We translate measured component classes into deployment actions: preserve, remove, compress, cache, retrieve selectively, or route conditionally.

2 Background and Related Work

2.1 Cost-Aware LLM Systems

FrugalGPT showed that LLM usage can be optimized by adapting prompts, approximating expensive models, and cascading requests across models with different cost and performance characteristics [1]. The key insight is that cost reduction need not require training a new model. Application-level decisions can substantially affect economic efficiency. MVT extends this line of work by focusing on the prompt as a decomposable economic object. Rather than selecting only which model to call, the framework asks which prompt components should be retained, removed, compressed, cached, or conditionally included.

2.2 Prompt Compression and Context Pruning

Prompt compression methods reduce inference cost by removing or compressing input text while attempting to preserve task-relevant information. LLMLingua proposes coarse-to-fine prompt compression for accelerated inference [2]. LongLLMLingua extends prompt compression to long-context settings and emphasizes the role of key-information density and position bias [3]. Selective Context prunes redundant context to reduce memory and inference time while maintaining comparable performance in long-document and conversation tasks [4]. These methods demonstrate that many prompt tokens can be removed without proportional quality loss. However, compression methods alone do not decide whether a business should delete a component, cache it, retrieve it more selectively, or preserve it because it reduces risk. MVT addresses that decision layer.

2.3 Prompt Caching and Reusable Context

Prompt Cache formalizes modular attention reuse for low-latency inference by reusing computation associated with recurring prompt segments [5]. In business workflows, recurring components include system instructions, output schemas, tool definitions, policy manuals, and stable style rules. A reusable component should not be classified as low-value simply because it is long. If it is important and repeated, the appropriate action may be stable-prefix layout and caching rather than deletion. MVT therefore distinguishes low-value tokens from reusable tokens.

2.4 Retrieval-Augmented Generation and Evaluation

Retrieval-augmented generation (RAG) grounds model outputs in external knowledge sources [6]. RAG is common in business settings because enterprise knowledge changes frequently and cannot always be encoded in model weights. At the same time, retrieval can introduce irrelevant or

conflicting context. RAGAS introduced metrics for evaluating generated answers with respect to retrieved context, including faithfulness and context relevance [7]. HELM argues for multi-metric language-model evaluation rather than single-score reporting [8]. MVT adopts a similar philosophy: prompt components should be evaluated by quality, cost, latency, groundedness, format compliance, and failure behavior.

2.5 Long-Context Utilization

Long context windows do not guarantee effective context use. Liu *et al.* show that language models can be less effective when relevant information appears in the middle of long contexts [9]. This finding is important for business prompt design. Adding more context can increase cost while also making relevant information less salient. MVT explicitly includes negative-value components to account for cases in which additional tokens reduce utility.

3 Prompt Components in Business Workflows

3.1 Definition

A *prompt component* is a functional segment of a prompt that can be identified, versioned, and experimentally manipulated. Components may be contiguous text blocks or structured objects such as messages, tool definitions, examples, schemas, retrieved passages, or summaries of prior state. The component is the correct unit of analysis because isolated tokens rarely have stable semantic value in a business application. A retrieved paragraph, an invoice-extraction schema, or a refund-policy clause is meaningful as a component even though it contains many tokens.

Let a prompt for task instance i be represented as an ordered sequence of K components:

$$P_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,K}]. \tag{1}$$

Each component has attributes such as token count, source, owner, update frequency, cacheability, risk level, and role in the workflow.

3.2 Common Business Use Cases

Table 1 summarizes representative business workloads. The list is not exhaustive, but it captures common operational uses of LLMs across industries. These workloads differ in the type of utility that matters: support systems emphasize correctness and groundedness, summarization emphasizes coverage and factuality, and extraction emphasizes schema compliance and field-level accuracy.

Table 1: Representative business LLM workloads and evaluation signals.

Workload	Example industries	Typical use cases	Primary evaluation signals
Customer support and policy Q&A	SaaS, ecommerce, insurance, banking, healthcare administration, education	Refund eligibility, billing explanations, account troubleshooting, claims rules, appointment policies, product-support answers	Correctness, groundedness, unsupported-claim rate, refusal appropriateness, latency
Document summarization	Legal, finance, HR, real estate, manufacturing, healthcare administration	Contract briefs, policy memo summaries, meeting-note synthesis, incident summaries, audit summaries, case-file digests	Coverage, factuality, omission rate, concision, decision usefulness
Structured extraction	Finance, procurement, insurance, logistics, HR, sales operations	Invoice fields, purchase-order fields, contract clauses, shipping notices, resumes, CRM notes, support tickets	JSON validity, field-level accuracy, type correctness, missing-field rate, cost per record
Workflow assistance	Legal, procurement, operations, software support, sales	Drafting responses, classifying tickets, suggesting next steps, producing task checklists	Task completion, format compliance, human-edit distance, escalation rate

3.3 Component Taxonomy

Table 2 defines the component taxonomy used in the MVT framework. The expected class is not assumed; it is a hypothesis to be measured. For example, few-shot examples can be high-value for structured extraction on a smaller model but redundant for a stronger model. Retrieved context can be high-value when relevant and negative-value when irrelevant.

Table 2: Prompt components and measurement questions.

Component	Examples	Possible class	Measurement question
System instruction	Operating rules, role, boundaries, safety constraints	High-value or reusable	Does it reduce task failure, unsafe behavior, or format drift?
Task instruction	The immediate instruction to answer, summarize, extract, classify, or transform	High-value	Does removal or shortening degrade task success?
Business policy or domain rule	Refund policy, claims logic, procurement limits, HR policy, legal clause definitions	High-value or reusable	Is the rule necessary for correctness or risk control?
Retrieved context	Knowledge-base chunks, contract passages, prior tickets, source documents	High-value, low-value, or negative	Which passages improve groundedness and which distract?
Few-shot examples	Demonstrations of desired answers, JSON outputs, or citation style	Model-dependent or reusable	Do examples improve consistency enough to justify their length?
Chat history	Prior user turns, system actions, resolved state, unresolved requests	Mixed	Can raw history be summarized without losing task state?
Output schema	JSON schema, field descriptions, citation format, maximum length	High-value or reusable	Does it reduce invalid outputs and downstream parsing cost?
Tool definition	Function names, parameters, descriptions, tool-selection rules	Reusable	Can stable tools be shortened or cached without harming tool accuracy?
Boilerplate	Generic politeness, redundant reminders, decorative style instructions	Low-value	Can it be deleted or moved outside the prompt?
Irrelevant or stale context	Wrong policy, unrelated document, obsolete rule, noisy history	Negative-value	Does removal improve accuracy or reduce hallucination?

4 Marginal Value of Tokens Framework

4.1 Business Utility

For task instance i, model or configuration m, and prompt variant r, define business utility as

$$U_{i,m,r} = Q_{i,m,r} - \alpha C_{i,m,r} - \beta L_{i,m,r} - \gamma F_{i,m,r} - \eta R_{i,m,r}, \tag{2}$$

where Q is task quality, C is monetary cost, L is latency, F is a task-failure penalty, R is a risk penalty, and $\alpha, \beta, \gamma, \eta \geq 0$ are application-specific weights. The terms should be scaled so that utility differences are interpretable. For example, Q may be a 1–5 rubric score, while F may be a binary penalty for invalid JSON, unsupported policy claims, or refusal errors.

The utility function is intentionally flexible. A real-time customer-service assistant may place high weight on latency and unsupported claims. A batch invoice-extraction process may place higher weight on field accuracy and cost per record. A legal summarization workflow may place high weight on factuality and omission penalties.

4.2 Cost and Token Accounting

Let $T^{in}_{i,m,r}$ and $T^{out}_{i,m,r}$ denote input and output tokens. Let $C_{i,m,r}$ denote measured or estimated cost. A simple token-cost model is

$$C_{i,m,r} = p^{in}_m T^{in}_{i,m,r} + p^{out}_m T^{out}_{i,m,r} + C^{aux}_{i,m,r}, \tag{3}$$

where p^{in}_m and p^{out}_m are input and output prices and C^{aux} captures auxiliary charges such as retrieval, tool use, moderation, or evaluation. If caching is available, cost can be decomposed as

$$C_{i,m,r} = p^{u}_m T^{u}_{i,m,r} + p^{cr}_m T^{cr}_{i,m,r} + p^{cw}_m T^{cw}_{i,m,r} + p^{out}_m T^{out}_{i,m,r} + C^{aux}_{i,m,r}, \tag{4}$$

where T^u denotes uncached input tokens, T^{cr} cache-read tokens, and T^{cw} cache-write tokens. This decomposition matters because reusable components should be judged by realized cost, not raw token length.

4.3 Latency Accounting

Latency can be represented as a weighted combination of time-to-first-token and end-to-end completion time:

$$L_{i,m,r} = \lambda_1 \mathrm{TTFT}_{i,m,r} + \lambda_2 \mathrm{E2E}_{i,m,r}. \tag{5}$$

Time-to-first-token is important in interactive applications, while end-to-end time is important for batch throughput and downstream automation. Input length primarily affects prefill and context processing, while output length affects decoding time. Therefore, an output constraint may improve latency even if the input prompt is unchanged.

4.4 Marginal Value of Tokens

For component j, let $r = 0$ denote the baseline prompt and $r = -j$ denote the prompt with component j removed or replaced by a reduced version. The average utility contribution of component j is

$$\widehat{\Delta U}_{j,m} = \frac{1}{N} \sum_{i=1}^{N} \left(U_{i,m,0} - U_{i,m,-j} \right). \tag{6}$$

The token-normalized marginal value is

$$\widehat{\mathrm{MVT}}^{tok}_{j,m} = \frac{\widehat{\Delta U}_{j,m}}{\frac{1}{N} \sum_{i=1}^{N} \left(T^{in}_{i,m,0} - T^{in}_{i,m,-j} \right)}. \tag{7}$$

The cost-normalized form is

$$\widehat{\mathrm{MVT}}^{cost}_{j,m} = \frac{\widehat{\Delta U}_{j,m}}{\frac{1}{N} \sum_{i=1}^{N} \left(C_{i,m,0} - C_{i,m,-j} \right)}. \tag{8}$$

A high positive value indicates that the component provides substantial utility relative to its cost. A near-zero value indicates a candidate for removal or compression. A negative value indicates that the component may be harming the task.

5 Estimating Component Value

5.1 Paired Ablation

The simplest estimator is paired leave-one-component-out ablation. Each task instance is evaluated with the full prompt and with one component removed. Pairing is important because task difficulty varies across examples. The component effect is measured within the same task instance, reducing variance relative to comparing unrelated prompts.

Ablation should be performed carefully. Removing a component must not produce an invalid prompt structure. For example, removing few-shot examples should preserve the output schema. Removing retrieved context should preserve the instruction to cite sources only if sources remain available; otherwise the instruction should be adjusted consistently. The goal is to isolate the

functional contribution of a component, not to create artificial prompt defects.

5.2 Compression and Replacement Ablations

Not every component is either present or absent. Chat history may be replaced by a state summary. Retrieved context may be reduced from top-k chunks to top-$k/2$ chunks. A long policy may be replaced by a compressed rule list. These replacement ablations estimate the value of a representation rather than the value of raw text. Let $r = j \to j'$ denote replacing component j with compressed component j'. The replacement value is

$$\widehat{\Delta U}_{j\to j',m} = \frac{1}{N}\sum_{i=1}^{N}\left(U_{i,m,0} - U_{i,m,j\to j'}\right). \tag{9}$$

If utility loss is within an acceptable margin and cost decreases, replacement is preferred to deletion.

5.3 Randomized Coalition Attribution

Prompt components interact. To estimate interactions, sample subsets $S \subseteq \{1,\ldots,K\}$ and evaluate prompts containing only components in S. A linear-interaction model can be fitted:

$$U_{i,m,S} = \theta_0 + \sum_j \theta_j z_{j,S} + \sum_{j<k} \theta_{jk} z_{j,S} z_{k,S} + \varepsilon, \tag{10}$$

where $z_{j,S} = 1$ if component j is present. The coefficient θ_j estimates average component contribution within the sampled design, while θ_{jk} captures interaction effects. A Shapley-style approximation can also be used by sampling random component orderings and measuring each component's incremental contribution when added to a partial prompt.

5.4 Non-Inferiority Constraint

Cost reduction is useful only if quality remains acceptable. Let δ_Q denote the maximum acceptable quality loss. A reduced prompt variant r is deployable if

$$\mathrm{LCB}_{95}(Q_r - Q_0) > -\delta_Q \tag{11}$$

and either cost or latency decreases by a practically meaningful margin. This rule prevents teams from accepting prompt reductions based on visible token savings while ignoring small but systematic quality degradation. Non-inferiority thresholds should be defined before the experiment and should differ by workload risk.

5.5 Classification Rules

Table 3 defines operational classes. Classification should use both statistical evidence and deployment context. A component with small average utility may still be retained if it prevents rare but severe failures.

Table 3: Component classification rules.

Class	Rule and action
High-value	Removal or compression causes meaningful utility loss. Preserve; improve wording or retrieval quality if needed.
Low-value	Removal is non-inferior and reduces cost or latency. Delete, shorten, or move outside the prompt.
Negative-value	Removal improves quality, groundedness, or failure rate. Prune and debug retrieval, memory, or source selection.
Reusable	Component improves utility and repeats across calls. Preserve but place in a stable prefix and cache when possible.
Model-dependent	Component helps one model tier but not another. Include conditionally based on routing.
Workflow-dependent	Component matters only under certain risk, domain, or task conditions. Include based on workload policy.

6 Experimental Methodology

6.1 Task Set

The evaluation design uses 60 prompts across three workloads: 20 customer-support or policy question-answering tasks, 20 document summarization tasks, and 20 structured extraction tasks. Each workload should span multiple industries. The task set is designed to be small enough for careful human review and large enough to expose differences across prompt components.

For policy question answering, each task includes a user question, a relevant source policy, and an expected answer. For summarization, each task includes a source document and a checklist of facts that should be covered or avoided. For extraction, each task includes unstructured text and a gold structured record.

6.2 Prompt Variants

Each task instance is evaluated under a baseline prompt and seven controlled variants, as shown in Table 4. The baseline is the production-style prompt containing all relevant components. Variants remove, compress, restructure, or perturb specific components.

Table 4: Prompt-component experimental conditions.

Condition	Manipulation	Diagnostic purpose
C0 Full baseline	Include system instruction, task instruction, business rules, retrieved context, examples, history, and output constraints where applicable.	Establish reference quality, cost, latency, and failure behavior.
C1 Boilerplate removal	Remove generic politeness, repeated reminders, and non-functional style text.	Detect low-value wording.
C2 Few-shot removal	Remove examples while preserving task instruction and output schema.	Estimate demonstration value.
C3 Context reduction	Reduce retrieved context by chunk score, source filter, or top-k reduction.	Measure marginal value of retrieved evidence.
C4 History compression	Replace raw chat history with a structured state summary.	Test whether state can be preserved with fewer tokens.
C5 Output constraint	Add concise answer, bullet limit, JSON-only, or maximum-field response constraint.	Measure output-token reduction and format compliance.
C6 Reusable-prefix layout	Place stable instructions, policy, schemas, and tools before dynamic content.	Measure reusable-token economics and prefix stability.
C7 Irrelevant-context control	Add plausible but unrelated retrieved context.	Quantify negative value from noisy retrieval.

6.3 Metrics

For every request, the measurement log records: model/configuration identifier, prompt variant, component inventory, input tokens, output tokens, total tokens, estimated cost, time-to-first-token, end-to-end latency, raw output, error status, quality score, task success, format compliance, groundedness, and failure labels. Failure labels should include unsupported claim, wrong answer, missing required field, invalid JSON, hallucinated citation, refusal error, timeout, and policy violation.

Quality metrics differ by workload. Structured extraction uses exact field match, type match,

field-level precision and recall, and schema validity. Policy question answering uses answer correctness, evidence support, refusal appropriateness, and unsupported-claim rate. Summarization uses factuality, coverage, omission rate, concision, and decision usefulness. When automated evaluators are used, a subset of outputs is human-reviewed to calibrate judge reliability.

6.4 Scoring Rubric

A 1–5 quality scale is suitable for many business evaluations:

- 5: fully correct, complete, grounded, and format-compliant;
- 4: correct with minor omissions or style defects;
- 3: partially correct but missing important information;
- 2: materially flawed, unsupported, or incomplete;
- 1: incorrect, unsafe, invalid, or unusable.

For extraction tasks, field-level deterministic scoring overrides broad rubric scores. For RAG tasks, the evaluator separately scores answer quality and evidence support so that fluent unsupported answers are penalized.

6.5 Execution Protocol

The experiment proceeds as follows.

Step 1: Select task instances. Choose 60 tasks across the three workloads and multiple industries. Avoid tasks that are trivial or impossible.

Step 2: Create gold references. Prepare expected answers, source passages, required summary facts, or gold structured records.

Step 3: Define components. Split each prompt into named components with token counts, source labels, cacheability flags, and risk labels.

Step 4: Generate variants. Produce C0–C7 variants using deterministic templates. Keep user input constant across variants.

Step 5: Freeze settings. Fix temperature, maximum output tokens, tool settings, retrieval settings, and model configuration.

Step 6: Randomize execution. Randomize call order across task instances and variants to reduce transient system effects.

Step 7: Log raw measurements. Store prompts, outputs, usage fields, timestamps, latency, errors, and metadata without overwriting failures.

Step 8: Score outputs. Run deterministic validators first, then rubric-based evaluation. Blind human raters to the condition label where feasible.

Step 9: Estimate effects. Compute paired utility deltas, token deltas, cost deltas, latency deltas, confidence intervals, and non-inferiority decisions.

Step 10: Classify and act. Assign each component to a value class and translate the classification into a deployment decision.

6.6 Statistical Analysis

For scalar metrics such as quality, cost, and latency, compute paired differences between baseline and each variant. Bootstrap confidence intervals are computed over task instances. For binary outcomes such as task success or JSON validity, paired proportions and confidence intervals are computed. For multiple components and workloads, both aggregate and workload-specific effects are reported. Average quality alone is insufficient; a prompt variant that improves average cost but increases severe failures may be unacceptable.

Sensitivity analysis varies the utility weights in (2). A component may be low-value under a low-risk utility function and high-value under a compliance-sensitive utility function. This is not a flaw; it reflects the fact that business value is context-dependent.

7 Interpreting MVT Measurements

7.1 Component-Level Decisions

MVT is intended to produce deployment decisions, not merely diagnostic scores. Table 5 maps measurement outcomes to actions.

Table 5: Deployment actions based on measured component value.

Measured pattern	Interpretation	Action	Example
Large positive utility, large cost	Expensive but necessary component	Preserve; compress only with non-inferiority testing	Policy clauses required for correct refund decisions
Near-zero utility, positive cost	Redundant or decorative component	Remove or shorten	Repeated role boilerplate and generic politeness
Negative utility	Component distracts or misleads the model	Remove; debug source or retrieval process	Unrelated retrieved passage causing unsupported answer
Positive utility, repeated frequently	Valuable reusable component	Preserve as stable prefix; cache when feasible	Tool schema or output contract used in every call
Positive only for some models	Model-dependent component	Include conditionally by model route	Few-shot examples needed for cheaper model only
Positive only for some workloads	Workflow-dependent component	Include by policy or risk tier	Legal disclaimer relevant to regulated advice but not internal tagging

7.2 Input Tokens Versus Output Tokens

Prompt optimization often focuses on input length, but output tokens can dominate both cost and latency. Output constraints should be evaluated as first-class prompt components. In structured extraction, JSON-only instructions and explicit schemas can reduce invalid outputs and downstream parsing failures. In summarization, aggressive length constraints may reduce cost but increase omission risk. The marginal value of output constraints should therefore be measured by utility, not by token reduction alone.

7.3 Retrieval as Economic Selection

RAG systems should not treat all retrieved chunks as equally valuable. Each chunk introduces cost and potential distraction. Retrieved context should be evaluated by source relevance, support contribution, redundancy, and conflict with other context. Negative-control irrelevant context is useful because it estimates the harm of retrieval noise. If noisy context reduces quality, the correct optimization may be better retrieval and reranking rather than a larger model.

7.4 Reusable Tokens Are Not Low-Value Tokens

A stable output schema or tool definition may contain many tokens, but its marginal value may be high because it reduces invalid outputs. If the component repeats across many calls, caching or stable-prefix layout can reduce its realized cost. This distinction prevents a common error: deleting useful long instructions simply because they are visible in token counts.

8 Business Deployment Guidelines

8.1 Maintain a Prompt-Component Ledger

Organizations should treat prompt components as versioned artifacts. Each component should have an identifier, owner, purpose, token count, source, update frequency, cacheability, risk level, last evaluation date, and current value class. This creates auditability and prevents prompt changes from becoming untracked operational risk.

8.2 Use Non-Inferiority Gates

A prompt reduction should not be deployed merely because it reduces token count. It should pass a predeclared non-inferiority gate for quality and failure rate. High-risk workflows should use narrower margins and more human review.

8.3 Separate Stable and Dynamic Content

Stable instructions, schemas, tools, and reusable policy text should be separated from dynamic user content, retrieved snippets, and time-sensitive state. This improves maintainability and allows reuse or caching where the serving environment supports it.

8.4 Measure Retrieval Noise

Retrieval pipelines should be evaluated not only for recall but also for harm caused by irrelevant context. The negative-control condition should be part of routine evaluation because it reveals whether additional context improves grounding or merely increases distraction.

8.5 Make Prompt Design Conditional

A single prompt template is rarely optimal across all models, tasks, and risk levels. Few-shot examples, detailed schemas, and long policy sections may be included only for certain routes or

workloads. MVT supports conditional prompt assembly based on measured component value.

8.6 Evaluate Cost, Latency, and Quality Together

Cost-only optimization can degrade service quality. Quality-only optimization can produce economically unsustainable systems. Latency-only optimization can remove necessary evidence. Production prompt design should evaluate these dimensions jointly, with explicit penalties for failures and risk.

9 Limitations

MVT is a measurement framework rather than a new compression algorithm or language model. Several limitations therefore apply. First, component effects may drift as model behavior changes. Second, automated quality evaluators can be biased or inconsistent, especially for nuanced business decisions. Third, component interactions can make simple leave-one-out ablation incomplete; coalition analysis is more reliable but more expensive. Fourth, utility weights are business-specific and may not transfer across organizations. Fifth, rare but severe failures may be underrepresented in a 60-prompt evaluation. Sixth, tokenization, pricing, latency, and caching behavior differ across serving environments. Finally, business utility may include legal, reputational, or operational risk that is difficult to reduce to a scalar score. These limitations do not negate the framework; they define the conditions under which measurement should be interpreted cautiously.

10 Conclusion

This paper introduced Marginal Value of Tokens as a framework for measuring the contribution of prompt components in business LLM applications. The central claim is that prompt tokens should not be treated as homogeneous units of text. Some tokens encode essential business rules, some provide evidence, some enforce output contracts, some repeat across calls and should be reused, and some distract the model or increase failure risk. The goal is therefore not blind token minimization. The goal is to maximize useful business output per token, per dollar, and per second.

By decomposing prompts into components, measuring paired utility differences, accounting for cost and latency, applying non-inferiority constraints, and classifying components into operational value classes, organizations can convert prompt design from informal editing into an empirical optimization process. This framework is most useful where LLM applications are operationally important: support, summarization, extraction, policy assistance, and document-heavy business workflows. Treating prompts as economic objects gives applied AI teams a disciplined way to reduce cost and latency while preserving the quality and reliability required for business adoption.

References

[1] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," *Trans. Mach. Learn. Res.*, 2024.

[2] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," in *Proc. Conf. Empirical Methods Natural Language Processing (EMNLP)*, 2023, pp. 13358–13376.

[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," in *Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2024, pp. 1658–1677.

[4] Y. Li, B. Dong, C. Lin, and F. Guerin, "Compressing context to enhance inference efficiency of large language models," in *Proc. Conf. Empirical Methods Natural Language Processing (EMNLP)*, 2023, pp. 6342–6353.

[5] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong, "Prompt Cache: Modular attention reuse for low-latency inference," *arXiv preprint arXiv:2311.04934*, 2023.

[6] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 9459–9474.

[7] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated evaluation of retrieval augmented generation," in *Proc. 18th Conf. European Chapter Assoc. Comput. Linguistics: System Demonstrations*, 2024, pp. 150–158.

[8] P. Liang *et al.*, "Holistic evaluation of language models," *Ann. New York Acad. Sci.*, vol. 1525, no. 1, pp. 140–146, 2023.

[9] 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," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 157–173, 2024.

[10] T. B. Brown *et al.*, "Language models are few-shot learners," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 1877–1901.

[11] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.