Received: 15 June 2026, Revised: 03 November 2026, Accepted: 06 November 2026, Online: 19 November 2026

DOI: https://dx.doi.org/10.55708/js0511005

Scoring-Driven Context Memory and Token Optimization for Rapid Prototyping of Scalable Multimodal Intelligent Agents: The MERIT Framework

Muthukumarapandian Chandrasekaran*

LTM Limited, Texas, USA

*Corresponding author: Muthukumarapandian Chandrasekaran, LTM Limited, Texas, USA, Email: muthuchand2013@gmail.com

ABSTRACT: Frameworks such as AutoGen and CrewAI make it fast to assemble multi-agent systems built on large language models (LLMs), yet as a prototype scales the binding constraint shifts from reasoning quality to how context is selected, retained, and emitted under a finite token budget. This article extends a prior conference study that compared two prototyping frameworks and catalogued the open challenges of LLM-based agents. The conference work observed that coarse controls such as fixed conversation rounds and agent iteration limits bound context quantity without judging its value. We turn that observation into a design. We introduce MERIT, a scoring-driven framework with four components: Multi-Signal Value Scoring (MSVS), which scores each context item from recency, relevance, task criticality, provenance, and redundancy; Budget-Aware Retention Optimization (BARO), which selects a high-value subset under a token budget through a value-density knapsack; Stable-Prefix Emission (SPE), which orders retained items so that low-churn content forms a cache-reusable prefix; and a Prototype Scoring Loop (PSL), which ranks candidate configurations on a joint quality, cost, and latency objective using a successive-halving schedule. We provide the scoring model, pseudocode, complexity bounds, an integration design for existing substrates, an evaluation protocol, and a controlled synthetic study in which value-density retention holds about four times the ground-truth utility per token of recency truncation while stable-prefix emission recovers cross-turn cache reuse. MERIT replaces fixed round and iteration limits with a value-driven stopping rule and layers onto existing frameworks without changing their agent abstractions. Because MERIT reads each context item only through its token cost and its embedding, the same mechanisms apply to multimodal context such as images, audio, and document pages.

KEYWORDS: Multi-agent systems, Large language models, Context management, Token optimization, Key-value cache, Rapid prototyping, Agent orchestration, Value scoring

1. Introduction

This paper is an extension of work originally presented at the 2024 Artificial Intelligence for Business (AIxB) conference [1]. The conference contributions are summarized and rewritten in the foundational sections; the MERIT framework and all of its analysis, including its algorithms, complexity analysis, integration design, synthetic simulation study, and evaluation protocol, are new.

LLMs now act as controllers for autonomous agents that reason, call tools, and adapt to new observations across many domains [2, 3, 4, 5, 6]. A productive direction composes several such agents into a multi-agent system, where specialized roles collaborate through messages to decompose and solve tasks that a single agent handles poorly [7]. Prototyping frameworks make this composition fast: a developer defines roles, attaches tools, and wires a conversation or a task graph, and a working system follows in hours rather than weeks. The author has applied this paradigm to healthcare settings, including a supervisor-coordinated multi-agent architecture with ReAct reasoning and clinical-standard data integration [8] and an uncertainty-aware hierarchical multi-agent system for non-deterministic HealthTech workflows [9].

The author's earlier conference paper studied this setting in two ways [1]. It examined the foundations of multi-agent conversation frameworks, the conversable-agent abstraction and the programming of computation and control flow over conversations, and it set out the open challenges of LLM-based agents, namely role-playing fidelity, prompt robustness, hallucination, and the knowledge boundary problem. It then compared two representative frameworks, AutoGen and CrewAI, and identified a recurring weakness: the controls these frameworks expose for bounding context, a maximum number of conversation rounds in the first and a maximum number of agent iterations in the second, cap quantity without assessing the value of what is retained, and offer no principled way to set the cap for a given task.

This article extends that study by turning the identified weakness into a concrete mechanism. As a prototype grows, each agent turn carries a longer transcript of messages, tool outputs, and retrieved passages into a fixed context window, and every token of that context carries a compute and a monetary cost at inference. Models also do not use long contexts uniformly; information placed in the middle of a

long prompt is often underused, so accumulating history without judgment degrades both quality and cost [10]. The right response is to treat context as a budgeted memory whose contents are scored by value, and to treat the choice of prototype configuration as a scored search rather than manual trial and error.

We introduce MERIT, a framework with four components. MSVS scores each context item. BARO selects a value-maximizing subset under a token budget. SPE emits the selected items in an order that preserves a stable, cache-reusable prefix. PSL scores candidate agent configurations and concentrates evaluation effort on the most promising ones. The same value signals also yield a value-driven stopping rule that replaces fixed round and iteration limits.

New material relative to the conference paper. This article (1) revises the title, abstract, introduction, and figures; (2) rewrites and consolidates the foundational study of the conference paper as motivation, and folds its framework comparison and the shared fixed-cap weakness into the introduction; and adds, as new contributions, (3) a composite value model for heterogeneous context items reduced to value per token; (4) a budgeted retention rule with a known approximation guarantee; (5) a stable-prefix emission policy that ties retention to key-value cache reuse; (6) a prototype scoring loop with a value-driven stopping rule; (7) a substrate-agnostic integration design; (8) a complexity analysis and an evaluation protocol; and (9) a controlled synthetic study isolating the contribution of each mechanism.

2. Foundations of Multi-Agent Conversation Frameworks

A multi-agent conversation framework consolidates complex LLM applications around a single principle: workflows are expressed as conversations among agents, and agents are made reusable so that the same components serve many applications. Two ideas anchor this principle, the conversable agent and conversation programming.

2.1. Conversable Agents

A conversable agent is an entity with an assigned role that exchanges messages to send and receive information, and that maintains its own context as those messages accumulate. The behavior of an agent is determined by the backends it is configured with. An LLM backend supplies reasoning, role adoption, state inference, and learning from conversation history, together with inference-time features such as result caching and error handling. A human backend admits human participation at configurable points. A tool backend executes code or function calls, for example a proxy agent that runs code or a function suggested by an LLM.

These backends compose. A common pattern places a foundational abstraction, here called the conversable-agent class, beneath two specializations: an assistant role that reasons with an LLM, and a user-proxy role that mediates human input and executes code or function calls with help from humans or tools. Specialized agents are then created by extending these templates, which keeps construction fast and the resulting agents easy to maintain. Conversable agents are the building blocks, but to make them advance a task a developer must also define how their conversations are structured, which is the role of conversation programming. Figure 1 sketches a supervisor that routes work among role agents.

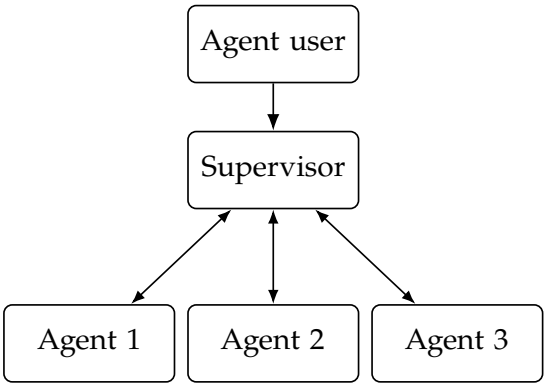


Figure 1: A supervisor agent routes work among role-specialized agents, each of which may use LLM, human, or tool backends.

2.2. Conversation Programming

Conversation programming organizes two concerns: computation, the actions an agent takes to produce a response, and control flow, the order and conditions under which those actions occur. Both are expressed in terms of conversations. An agent acts on the conversation it is part of, and its action passes a message that seeds the next step unless a termination condition is met. The control flow follows from the agents' decisions about whom to address next, so a developer can reason about a workflow as an interaction between agent actions and message passing.

The paradigm supports static flows with a predefined sequence and dynamic flows decided at run time. A custom reply function lets an agent manage the current exchange while triggering interactions with other agents based on the message and the surrounding context, and an LLM can decide whether to invoke a function depending on the state of the conversation, which brings further agents into the exchange. A group-chat manager generalizes this to many participants by selecting the next speaker and broadcasting each response, and recent additions such as finite-state-machine graphs let a designer specify permitted and forbidden transitions [7].

3. Challenges in LLM-Based Autonomous Agents

Although LLM-based agents have shown clear successes, the area is young and several challenges remain open. We summarize four that bear directly on prototyping at scale.

3.1. Role-Playing Fidelity

Unlike a plain LLM, an agent often must adopt a specific role such as coder, researcher, or domain expert. LLMs handle common roles that appear widely in training data but struggle with uncommon or emerging roles, and they do not reliably model human cognitive traits, which can leave an agent lacking self-awareness in conversation [11]. Two remedies have been proposed: fine-tuning on data for the target role or trait, which raises role-specific performance at some risk to common roles, and designing

tailored prompts and architectures, which is effective but searches a large design space [12].

3.2. Prompt Robustness

Planning and memory modules are often bolted onto an LLM to keep an agent's behavior coherent, and these additions require more elaborate prompts. Prior studies report that LLM prompts are brittle, in that small wording changes can produce very different outputs [13, 14]. The effect compounds in an agent that spans several prompt-driven modules, because a prompt written for one module can affect others, and prompt structure can shift sharply between models. Building a prompt scheme that holds across models remains unsolved. Two practical approaches are to generate prompts automatically with an LLM or to refine prompt elements by iterative trial and error.

3.3. Hallucination

LLMs can state incorrect information with high confidence, and this carries into agents [15]. In code generation, for instance, an agent given terse instructions may produce misleading or insecure code, raising correctness and security concerns. Incorporating human corrective feedback into the iterative loop of human-agent communication has been proposed as a mitigation [16].

3.4. Knowledge Boundary

Faithful simulation of human behavior requires that an agent reason only from what the simulated person would plausibly know. Because LLMs are trained on broad web corpora, they often hold knowledge beyond an average person's, which can leak into a simulation. When modeling a user choosing among films, for example, an agent may already know facts a real user would not, distorting the decision. Constraining the knowledge an agent uses to what the role would possess is therefore important for realistic simulation [17].

4. The MERIT Framework

4.1. Related Approaches to Context and Token Management

Several lines of work address parts of this problem. Retrieval augmented generation injects external passages selected by similarity to the query, which focuses the prompt but does not manage the running agent transcript [18]. MemGPT treats the context window as a tiered memory with paging between an in-context working set and external storage, which motivates explicit retention policies for agents [19]. At the token level, attention-based methods such as StreamingLLM and Heavy-Hitter Oracle keep a small set of influential tokens and discard the rest to bound the key-value cache [20, 21]. Prompt compression methods such as LLMLingua shorten prompts while preserving task signal [22]. MERIT differs by scoring heterogeneous items, system instructions, tool schemas, messages, tool outputs, and retrieved passages, on a common value scale, by selecting under an explicit token budget, and by ordering the result for cache stability, while reusing the same scoring discipline to choose among prototype configurations.

4.2. Overview

MERIT (Multi-signal value scoring for Efficient Retention and Iterative prototyping with Token-budget control) sits between an agent's context pool and the LLM call, as shown in Figure 2. At each turn the candidate items are scored by MSVS, a value-maximizing subset is chosen by BARO under the token budget, and SPE orders that subset so stable items lead. The same value signals drive PSL, which scores whole configurations during prototyping and decides when a configuration is good enough to stop refining.

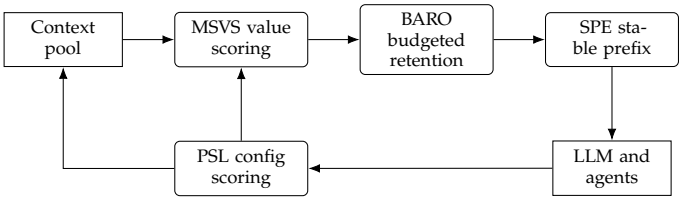


Figure 2: MERIT pipeline. Items flow from the context pool through value scoring, budgeted retention, and stable-prefix emission into the agent call. The prototype scoring loop reuses the value signals to rank configurations and to tune retention parameters.

We write the context pool at turn t as a set $\mathcal{C} = \{c_1, \ldots, c_n\}$. Each item c_i has a token length τ_i, a timestamp t_i, an embedding $\mathbf{e}_i$, a class label such as system instruction, tool schema, message, tool output, or retrieved passage, and a provenance tag. The current query or task is q with embedding $\mathbf{e}_q$. Nothing in this model is specific to text. An image, an audio segment, or a rendered document page enters the pool as an item like any other, provided its token cost τ_i can be estimated for the target model and an embedding $\mathbf{e}_i$ is available in a space shared with the query, for example from a multimodal encoder. The scoring, retention, and emission steps below then apply unchanged, which is what makes MERIT suitable for multimodal agents. The simulation study in this article uses text-like items; per-modality estimation of cost and relevance is discussed as future work.

4.3. Multi-Signal Value Scoring (MSVS)

MSVS assigns each item a value that combines five signals:

$$V(c_i) = \omega_R R(c_i) + \omega_S S(c_i, q) + \omega_T T(c_i) + \omega_P P(c_i) - \omega_D D(c_i), \tag{1}$$

where the nonnegative weights $\omega_\bullet$ sum to one over the additive terms.

Recency decays with elapsed turns or seconds $\Delta t_i = t_{\text{now}} - t_i$:

$$R(c_i) = \exp(-\lambda\, \Delta t_i), \qquad \lambda > 0. \tag{2}$$

Relevance is the clipped cosine similarity to the active query:

$$S(c_i, q) = \max\big(0,\ \cos(\mathbf{e}_i, \mathbf{e}_q)\big). \tag{3}$$

Task criticality $T(c_i) \in [0, 1]$ is a class prior that elevates items the task cannot proceed without, such as the system instruction and active tool schemas. Items with $T(c_i) = 1$ are marked pinned and are always retained, which gives

a principled answer to the conference-paper difficulty of specifying role and persona reliably, since those items are never silently dropped.

Provenance $P(c_i) \in [0,1]$ is a trust prior that ranks verified tool output and validated facts above unverified model-generated text, which lowers the chance that a hallucinated item displaces a grounded one [15]. Treating source trust as an explicit signal is consistent with uncertainty-aware multi-agent designs that reason about non-determinism rather than ignore it [9].

Redundancy penalizes overlap with content already chosen, in the spirit of maximal marginal relevance [23]:

$$D(c_i) = \max_{j \in \mathcal{K}} \cos(\mathbf{e}_i, \mathbf{e}_j), \tag{4}$$

where $\mathcal{K}$ is the set retained so far. Because D depends on $\mathcal{K}$, MSVS is evaluated jointly with retention rather than once in isolation. The value density of an item is

$$\rho_i = \frac{V(c_i)}{\tau_i}, \tag{5}$$

which expresses value per token and is the quantity retention maximizes.

4.4. Budget-Aware Retention Optimization (BARO)

Let B be the token budget for context, set below the model window to leave room for the response. Retention selects $x_i \in \{0,1\}$ to

$$\max_{x} \sum_i V(c_i)\, x_i \quad \text{s.t.} \quad \sum_i \tau_i x_i \le B, \qquad x_i = 1 \;\; \forall i \in \mathcal{P}, \tag{6}$$

where $\mathcal{P}$ is the pinned set. This is a 0/1 knapsack with forced items. After reserving budget $B_{\mathcal{P}} = \sum_{i \in \mathcal{P}} \tau_i$ for pinned items, BARO fills the remaining budget $B - B_{\mathcal{P}}$ by a value-density greedy: it sorts non-pinned items by ρ_i and admits them in order while they fit, recomputing the redundancy term in (1) as each item joins $\mathcal{K}$. Taking the better of this greedy solution and the single highest-value item that fits yields the standard one-half approximation guarantee for 0/1 knapsack, and the density ordering is optimal for the fractional relaxation. Pseudocode appears in Algorithm 1.

Algorithm 1: BARO with online redundancy

Input: items C, budget B, pinned set $\mathcal{P}$
Output: retained set $\mathcal{K}$
$\mathcal{K} \leftarrow \mathcal{P}$; $b \leftarrow B - \sum_{i \in \mathcal{P}} \tau_i$;
sort $C \setminus \mathcal{P}$ by initial density ρ_i descending;
foreach *item* c_i *in sorted order* **do**
  recompute $D(c_i)$ against $\mathcal{K}$; update $V(c_i)$, ρ_i;
  if $\tau_i \le b$ ***and*** $\rho_i > 0$ **then**
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{c_i\}$; $b \leftarrow b - \tau_i$;
  end
end
return $\mathcal{K}$;

4.5. Stable-Prefix Emission (SPE)

Given the retained set $\mathcal{K}$, the order in which items enter the prompt determines how much key-value cache can be reused from the previous turn [24]. SPE assigns each item a churn estimate

$$\chi_i = \Pr[c_i \text{ is evicted or mutated at turn } t+1] \in [0,1], \tag{7}$$

estimated from item class and from observed retention history, and partitions $\mathcal{K}$ into a stable core $\mathcal{S} = \{i : \chi_i \le \theta\}$ and a volatile tail $\mathcal{V} = \mathcal{K} \setminus \mathcal{S}$. The prompt emits $\mathcal{S}$ first, ordered by a fixed key so the ordering itself does not change between turns, then emits $\mathcal{V}$ ordered by value. If the leading L_p tokens match the previous turn, the cache reuse fraction is

$$\eta = \frac{L_p}{\sum_{i \in \mathcal{K}} \tau_i}, \tag{8}$$

and prefix prefill recomputation is avoided in proportion to η. Pinned items, which by construction persist, anchor the stable core, so SPE tends to grow L_p across a session. Pseudocode appears in Algorithm 2.

Algorithm 2: SPE stable-prefix ordering

Input: retained set $\mathcal{K}$, churn threshold θ
Output: ordered prompt sequence
$\mathcal{S} \leftarrow \{i \in \mathcal{K} : \chi_i \le \theta\}$; $\mathcal{V} \leftarrow \mathcal{K} \setminus \mathcal{S}$;
sort $\mathcal{S}$ by fixed stable key (class, provenance, id);
sort $\mathcal{V}$ by value $V(c_i)$ descending;
return *concatenation* $[\,\mathcal{S}\,;\,\mathcal{V}\,]$;

4.6. Prototype Scoring Loop (PSL)

A prototype configuration $g \in \mathcal{G}$ bundles an agent topology, a role and tool assignment, and the memory policy parameters $(\omega, \lambda, B, \theta)$. PSL scores a configuration on a joint objective

$$\Phi(g) = \alpha\, \hat{Q}(g) - \beta\, \hat{C}(g) - \gamma\, \hat{L}(g), \tag{9}$$

where $\hat{Q}$ is estimated task quality from an evaluation suite, $\hat{C}$ is expected token cost per task, and $\hat{L}$ is expected latency. Rather than evaluate every configuration fully, PSL uses a successive-halving schedule: it gives each candidate a small evaluation budget, keeps the top fraction by Φ, increases the per-candidate budget, and repeats until one configuration remains or the budget is spent [25]. Cheap self-evaluation signals can supply the quality estimate during early rounds [26]. The output is the surviving configuration together with the quality and cost frontier of the candidates, which exposes the trade-off to the developer instead of hiding it behind a single fixed limit.

4.7. Value-Driven Stopping

The conference paper criticized fixed round and iteration limits for capping turns without judging value. MERIT replaces the limit with a stopping rule. Let ΔV_t be the additional retained value a new turn is expected to contribute and let κ be the token cost of that turn, priced in value units through β in (9). The loop continues while

$$\Delta V_t > \beta\, \kappa, \tag{10}$$

and stops when the marginal value of another turn no longer covers its cost. A hard cap remains available as a safety bound but is no longer the primary control.

5. Complexity Analysis

Let n be the number of candidate items, d the embedding dimension, $k = |\mathcal{K}|$ the retained count, and $m = |\mathcal{G}|$ the number of configurations. With item embeddings precomputed and cached, MSVS computes recency, criticality, and provenance in $O(n)$ and the similarity terms in $O(nd)$. The redundancy term, evaluated online during BARO, costs $O(nk)$ in the worst case and can be bounded by capping the comparison set or by approximate nearest neighbor lookup. BARO sorts in $O(n \log n)$ and admits in $O(n)$. SPE partitions in $O(n)$ and sorts the core in $O(k \log k)$. The per-turn cost is therefore $O(n \log n + nd + nk)$, dominated in practice by the similarity and redundancy terms, both of which parallelize. PSL, under successive halving with total evaluation budget spread across m candidates, performs $O(m \log m)$ candidate evaluations in the standard analysis rather than the $O(m)$ full evaluations of an exhaustive sweep, while spending most effort on survivors [25]. None of the per-turn steps changes the asymptotic cost of the underlying agent framework, so MERIT layers on without altering its abstractions.

6. Integration with Rapid Prototyping Workflows

MERIT is substrate agnostic. On an AutoGen-style substrate, the context pool is the running group-chat transcript plus tool outputs and any retrieved passages; MSVS, BARO, and SPE run in the reply-generation hook before each model call, and value-driven stopping replaces the maximum round count [7]. On a CrewAI-style substrate, the context pool is the accumulated task output and shared memory; the same hooks run before each agent step, and PSL tunes the maximum iteration count and the process type rather than leaving them to manual choice. Tool schemas and system instructions are pinned through task criticality, so the role and persona content the conference paper found hard to specify reliably is never silently dropped. The same hooks apply to supervisor-coordinated architectures with external-standard data integration, such as a clinical telemedicine system where the supervisor routes work among role agents and pins the data-access schema [8]. Table 1 contrasts MERIT with common context controls.

Table 1: Context control strategies compared on four properties

Strategy	Value aware	Token budget	Cache stable	Proto. search
Fixed round or iter limit	no	no	no	no
Recency truncation	no	yes	no	no
Sliding window	no	yes	partial	no
RAG only	partial	partial	no	no
Attention eviction	token only	yes	no	no
MERIT	yes	yes	yes	yes

7. Evaluation Methodology

We describe a protocol rather than report measured numbers, so the framework can be assessed reproducibly on a chosen task suite.

7.1. Metrics

Quality is the task success rate on a held-out suite. Cost is mean prompt tokens per task and mean total tokens per task. Efficiency is the cache reuse fraction η averaged over turns. Latency is wall-clock time to completion. The headline efficiency measure is value retained per thousand prompt tokens, that is mean $\sum_{i \in \mathcal{K}} V(c_i)$ divided by emitted tokens.

7.2. Baselines and Ablations

Baselines are the fixed-limit control native to the substrate, recency truncation, a sliding window, and a retrieval-only prompt. Ablations remove one MERIT component at a time: MSVS with uniform weights, BARO replaced by greedy recency, SPE replaced by value ordering, and PSL replaced by a fixed configuration. Each ablation isolates the contribution of one mechanism to quality, cost, and cache reuse.

7.3. Illustrative Walkthrough

The following is a constructed example; the numbers are stated assumptions used to trace the mechanics, not measured results. Assume a budget $B = 8000$ tokens, a pool of $n = 60$ items, a pinned set of a system instruction and two tool schemas totaling $B_{\mathcal{P}} = 1200$ tokens, and a remaining budget of 6800 tokens. Suppose density ordering admits 22 non-pinned items before the budget fills and that redundancy pruning drops 5 near-duplicate items that recency alone would have admitted. SPE then finds the pinned items plus 9 low-churn retained items, 3100 tokens in total, unchanged from the previous turn, giving a leading stable prefix and a cache reuse fraction $\eta = 3100/8000 \approx 0.39$ for that turn. PSL, sweeping six candidate configurations under successive halving, would complete on the order of twelve partial evaluations rather than the full evaluations an exhaustive sweep at full budget would require, and would surface the quality and cost frontier across the six. The walkthrough shows how the components compose; the actual values depend on the task and the model and must be measured.

8. Synthetic Simulation Study

To show how the mechanisms behave when composed, we ran a controlled synthetic study. The numbers below are outputs of the simulation described here, not measurements on real tasks; they are meant to illustrate the mechanism and are fully reproducible from the stated setup.

8.1. Setup

Each run simulates a session of $T = 20$ turns with a candidate pool of $n = 60$ items per turn, a token budget $B = 8000$, and embeddings of dimension $d = 64$. A fixed hidden task vector defines a ground-truth utility u_i for every item as the nonnegative alignment of its embedding with the task plus a criticality bonus, and the per-turn

query is a noisy version of the task. A persistent subset of fifteen items, three pinned and twelve long-lived, recurs across turns, while the rest are transient and refresh each turn. Three of the persistent items are constructed as near-duplicates of another to exercise the redundancy term. Token lengths are drawn between 40 and 400. Results are averaged over eight seeds and over turns after the first, since cache reuse needs a prior turn. The ground-truth utility is hidden from the scorer, so utility per token is a fair measure of how much true value each policy retains.

8.2. Conditions

We compare three policies on the same pools: recency truncation, which fills the budget with the newest items and emits them chronologically; MERIT retention with plain value ordering, which applies MSVS and BARO but emits by value and omits SPE; and full MERIT, which adds SPE. The second and third differ only in emission order, which isolates the cache contribution of SPE.

8.3. Results

Table 2 reports retained ground-truth utility per thousand emitted tokens, mean prompt tokens, and the cross-turn cache reuse fraction η. MSVS and BARO raise retained utility per thousand tokens from 0.28 under recency truncation to 1.16, about a fourfold gain, while using slightly fewer tokens, because the scorer keeps high-value items and the redundancy term drops near-duplicates that recency would have admitted. SPE leaves retention unchanged and raises cache reuse from 0.5 percent to 11.7 percent by holding low-churn items in a fixed leading order, whereas value ordering reshuffles the prefix each turn and reuses almost nothing. The cache figure is modest here because redundancy pruning can still move which persistent duplicate is kept, which shows a real tension between aggressive deduplication and prefix stability that a deployment can tune through θ and the redundancy weight.

Table 2: Synthetic study, averaged over eight seeds. Values are simulation outputs, not real-task measurements.

Policy	Util. per 1k tok	Mean tokens	Cache reuse
Recency truncation	0.28	7975	0.0%
MERIT, value order (no SPE)	1.16	7897	0.5%
MERIT (full)	**1.16**	**7897**	**11.7%**

The study is deliberately simple and synthetic. It supports two claims that follow from the design: value scoring with a budgeted knapsack retains more true utility per token than recency, and stable-prefix emission recovers cache reuse that value ordering discards. The per-turn view in Figure 3 shows both effects are stable across the session rather than artifacts of a single turn. It does not stand in for evaluation on real task suites, which remains future work under the protocol of the previous section.

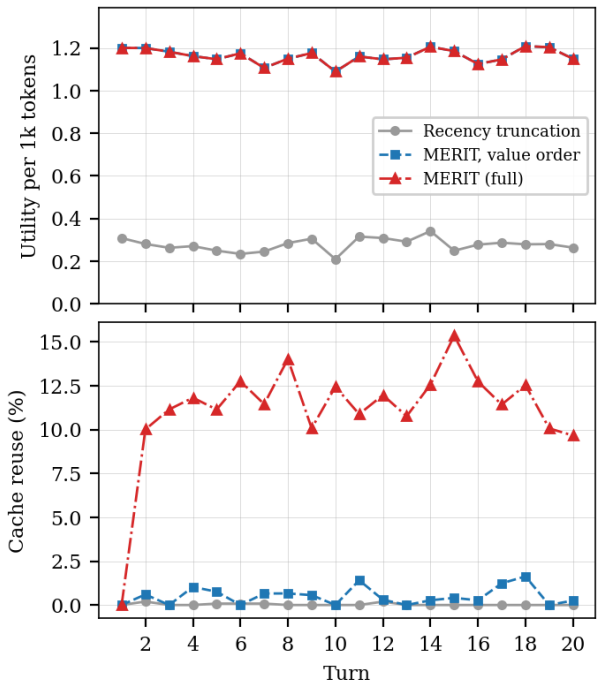


Figure 3: Per-turn results averaged over eight seeds. Top: retained ground-truth utility per thousand emitted tokens. Both MERIT variants share the same retention and overlap above recency truncation. Bottom: cross-turn cache reuse. Only full MERIT with stable-prefix emission recovers reuse, since recency and plain value ordering both reshuffle the leading tokens each turn. Values are simulation outputs, not real-task measurements.

9. Discussion

MERIT introduces parameters, the weights $\boldsymbol{\omega}$, the decay λ, the budget B, the churn threshold θ, and the objective weights α, β, γ. PSL is the intended way to set them, which turns parameter choice from manual tuning into a scored search but adds an initial cost paid once per task family. The value model depends on embedding quality for its relevance and redundancy terms, so a weak embedding model degrades scoring; provenance priors mitigate this by protecting grounded items. SPE assumes the serving stack reuses prefix cache, which holds for systems that support prefix sharing but yields no benefit where it is absent [24]. Value-driven stopping prices value against cost through a single coefficient, a deliberate simplification; tasks with sharp quality cliffs may still warrant a conservative hard cap as a guard.

10. Conclusion

This article extended a prior conference study of rapid prototyping for multi-agent LLM systems. The conference work compared two prototyping frameworks and identified that their fixed round and iteration limits bound context without judging its value. We turned that observation into MERIT, a scoring-driven framework that scores context items by a composite value, retains a value-maximizing subset under a token budget, emits that subset in a cache-stable order, and applies the same scoring to rank prototype configurations and to decide when to stop. The design replaces fixed limits with a value-driven stopping rule, layers onto existing substrates without changing their abstractions, and comes with pseudocode, complexity bounds, a

synthetic simulation that isolates each mechanism, and an evaluation protocol. The next step is empirical validation across task suites and serving stacks, and an extension of the value model to multimodal items whose token cost and relevance are estimated per modality.

Conflict of Interest The author declares no conflict of interest.

References

[1] M. Chandrasekaran, "Enhancing efficiency and flexibility of rapid prototyping for scalable multimodal intelligent agents", "2024 Artificial Intelligence for Business (AIxB)", pp. 66–71, IEEE, Laguna Hills, CA, USA, 2024, doi:10.1109/AIxB62249.2024.00019.

[2] T. B. Brown, *et al.*, "Language models are few-shot learners", "Advances in Neural Information Processing Systems", vol. 33, pp. 1877–1901, 2020.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, "LLaMA: Open and efficient foundation language models", arXiv preprint arXiv:2302.13971, 2023, doi:10.48550/arXiv.2302.13971.

[4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, Y. Cao, "ReAct: Synergizing reasoning and acting in language models", arXiv preprint arXiv:2210.03629, 2023, doi:10.48550/arXiv.2210.03629.

[5] Z. Xi, *et al.*, "The rise and potential of large language model based agents: A survey", *Science China Information Sciences*, vol. 68, no. 2, 2025, doi:10.1007/s11432-024-4222-0, article 121101.

[6] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, J. Wen, "A survey on large language model based autonomous agents", *Frontiers of Computer Science*, vol. 18, no. 6, 2024, doi:10.1007/s11704-024-40231-1, article 186345.

[7] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, C. Wang, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation", arXiv preprint arXiv:2308.08155, 2023, doi:10.48550/arXiv.2308.08155.

[8] M. Chandrasekaran, "A supervisor-coordinated multi-agent architecture with ReAct reasoning and FHIR integration, achieving 41% diagnostic time reduction in telemedicine workflows", "2026 6th Middle East and North Africa Communications Conference (MENACOMM)", pp. 1–8, IEEE, Hammamet, Tunisia, 2026, doi:10.1109/MENACOMM69507.2026.11532940.

[9] M. Chandrasekaran, "AURELIUS: Agentic uncertainty-reasoning and learning with hierarchical bayesian multi-agents unified system for handling non-determinism in HealthTech", "2025 4th International Conference on Computing, Management and Telecommunications (ComManTel)", pp. 52–57, IEEE, Madrid, Spain, 2025, doi:10.1109/ComManTel68363.2025.11368546.

[10] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, 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, doi:10.1162/tacl_a_00638.

[11] K. A. Fischer, "Reflective linguistic programming (RLP): A stepping stone in socially-aware AGI (SocialAGI)", arXiv preprint arXiv:2305.12647, 2023, doi:10.48550/arXiv.2305.12647.

[12] C. Li, J. Wang, Y. Zhang, K. Zhu, W. Hou, J. Lian, F. Luo, Q. Yang, X. Xie, "Large language models understand and can be enhanced by emotional stimuli", arXiv preprint arXiv:2307.11760, 2023, doi:10.48550/arXiv.2307.11760.

[13] T. Y. Zhuo, Z. Li, Y. Huang, F. Shiri, W. Wang, G. Haffari, Y.-F. Li, "On robustness of prompt-based semantic parsing with large pre-trained language model: An empirical study on Codex", "Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics", pp. 1090–1102, Association for Computational Linguistics, Dubrovnik, Croatia, 2023, doi:10.18653/v1/2023.eacl-main.77.

[14] Z. Gekhman, N. Oved, O. Keller, I. Szpektor, R. Reichart, "On the robustness of dialogue history representation in conversational question answering: A comprehensive study and a new prompt-based method", *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 351–366, 2023, doi:10.1162/tacl_a_00549.

[15] Z. Ji, *et al.*, "Survey of hallucination in natural language generation", *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023, doi:10.1145/3571730, article 248.

[16] Y. Dong, X. Jiang, Z. Jin, G. Li, "Self-collaboration code generation via ChatGPT", *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 7, pp. 1–38, 2024, doi:10.1145/3672459, article 189.

[17] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior", "Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology", pp. 1–22, ACM, 2023, doi:10.1145/3586183.3606763, article 2.

[18] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks", "Advances in Neural Information Processing Systems", vol. 33, pp. 9459–9474, 2020.

[19] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, J. E. Gonzalez, "MemGPT: Towards LLMs as operating systems", arXiv preprint arXiv:2310.08560, 2023, doi:10.48550/arXiv.2310.08560.

[20] G. Xiao, Y. Tian, B. Chen, S. Han, M. Lewis, "Efficient streaming language models with attention sinks", "International Conference on Learning Representations (ICLR)", 2024.

[21] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, B. Chen, "H2O: Heavy-hitter oracle for efficient generative inference of large language models", "Advances in Neural Information Processing Systems", vol. 36, 2023, doi:10.52202/075280-1506.

[22] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models", "Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing", pp. 13358–13376, Association for Computational Linguistics, Singapore, 2023, doi:10.18653/v1/2023.emnlp-main.825.

[23] J. G. Carbonell, J. Goldstein, "The use of MMR, diversity-based reranking for reordering documents and producing summaries", "Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval", pp. 335–336, ACM, 1998, doi:10.1145/290941.291025.

[24] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, I. Stoica, "Efficient memory management for large language model serving with PagedAttention", "Proceedings of the 29th ACM Symposium on Operating Systems Principles", pp. 611–626, ACM, 2023, doi:10.1145/3600006.3613165.

[25] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization", *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[26] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, S. Yao, "Reflexion: Language agents with verbal reinforcement learning", "Advances in Neural Information Processing Systems", vol. 36, 2023, doi:10.52202/075280-0377.

Copyright: This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY-SA) license (https://creativecommons.org/licenses/by-sa/4.0/).



MUTHUKUMARAPANDIAN CHANDRASEKARAN heads a business AI practice at LTM Limited covering healthcare, life sciences, and travel and hospitality, where he developed the dynamic context packaging framework. He has more than 21 years of experience in AI and data architecture across healthcare, financial services, and enterprise technology.

He was previously Chief AI Architect and Vice President at CitiusTech, where he led Knewron, an agentic AI platform for healthcare, and created the AURELIUS framework. He has also held roles at Ascendion, Cognizant, and USAA. He holds U.S. Patent 12,475,154 B1, has served as a technical program committee reviewer for several IEEE conferences, and was named Professional of the Year 2024 in Data and Analytics/AI at OnConferences.