The Cartu Method: A Framework for Persistent Memory and Cost-Efficient Inference in Autonomous AI Agents

Anonymous*

February 2026
v1.0 — this is a living document; updates at https://github.com/jcartu/rasputin

Abstract

Autonomous AI agents that operate continuously face two compounding infrastructure problems: (1) context windows are finite, but operational memory must persist indefinitely, and (2) frontier models capable of high-quality reasoning are prohibitively expensive for the background tasks that constitute the majority of an agent's workload. We introduce **the Cartu Method**, a modular framework addressing both problems through complementary techniques. The framework currently comprises two components: *pre-compaction memory rescue*, which intercepts context compaction events and uses parallel fast-inference calls to extract and preserve critical memories before they are destroyed (achieving 100% fact retention vs. 5% for vanilla compaction); and *multi-model routing with quality gating*, which routes agent traffic across models of varying capability and cost while maintaining output quality through automated scoring and escalation (achieving 97.9% cost reduction with zero quality degradation in production). Both components have been running in production for 30+ days on an autonomous agent handling 1,400+ daily requests. The framework is designed to be extensible—new components addressing additional aspects of agent infrastructure can be integrated as they are developed. All components are open-source.

1 Introduction

The emergence of autonomous AI agents—systems that operate continuously, execute multi-step workflows, and maintain persistent state—has exposed critical infrastructure gaps that no single technique addresses. These agents face a constellation of interrelated problems:

- **Memory**: Context windows are finite; compaction destroys critical operational facts.
- **Cost**: Frontier models are expensive; background tasks don't need frontier reasoning.
- **Quality**: Cheaper models produce lower-quality output; quality must be guaranteed.
- **Safety**: Any model can generate dangerous commands; safety must be model-independent.
- **Reliability**: Providers go down; agents must keep running.

Each of these has been addressed individually in prior work, but autonomous agents need solutions that work *together* as a coherent stack. The Cartu Method is a framework that provides this integration, with modular components that can be adopted independently or composed.

This paper describes the first two components of the framework:

*Open-source implementation: https://github.com/jcartu/rasputin

1. **Memory Rescue** (§3): Pre-compaction extraction of critical facts into a persistent vector database using parallel fast-inference calls.

2. **Inference Routing** (§4): Multi-model request routing with automated quality gating, prompt compensation, and failover.

Both components are in production, and the framework is designed so that future components—retrieval optimization, tool-use safety analysis, multi-agent coordination, and others—can be added as they mature.

1.1 Design Principles

The Cartu Method is guided by several principles derived from production experience:

1. **No training required.** All components work with off-the-shelf models. No fine-tuning, no custom training data, no GPU-hours spent on adaptation.

2. **Model-agnostic.** Components work across providers (Anthropic, MiniMax, Cerebras, OpenAI, local models). Swapping models requires only configuration changes.

3. **Fail-safe by default.** If any component fails, the agent continues operating—with degraded performance, but without catastrophic failure.

4. **Measurable.** Every component exposes metrics (cost, latency, quality scores, pass rates) so operators can verify effectiveness.

5. **Composable.** Components can be used independently or together. Memory Rescue works without Inference Routing and vice versa.

2 Related Work

Context Management. RAG [Lewis et al., 2020] retrieves documents at query time but does not address context destruction during compaction. MemGPT [Packer et al., 2023] implements a virtual memory hierarchy with paging between main context and external storage. MemGPT's paging is reactive (evicting information to make room), whereas Memory Rescue is proactive (extracting information triggered by the compaction event itself). MemRL [Anonymous, 2025] applies reinforcement learning to memory retrieval, learning which memories to surface. MemRL addresses *retrieval selection* (what to recall), while Memory Rescue addresses *memory creation* (what to preserve). The two are complementary: Memory Rescue ensures facts survive compaction; MemRL-style approaches improve which of those facts are retrieved when needed.

Inference Cost Optimization. FrugalGPT [Chen et al., 2023] proposes LLM cascading where cheaper models attempt tasks first with expensive fallback. Martian's Model Router dynamically selects models per-request based on predicted difficulty. Inference Routing differs in two ways: it uses *task-type* routing (all scheduled tasks to cheap models, interactive to expensive) rather than per-request difficulty estimation, and it applies *post-generation quality scoring* rather than pre-generation difficulty prediction. Prompt caching [Anthropic, 2024] and speculative decoding [Leviathan et al., 2023] optimize within a single model; Inference Routing optimizes *across* models.

Agent Safety. Existing work on LLM safety focuses on alignment during training [Bai et al., 2022]. The Cartu Method's safety layer operates at the infrastructure level—blocking dangerous commands regardless of which model generates them—complementing rather than replacing alignment-based approaches.

3 Component 1: Memory Rescue

3.1 Problem: Compaction Amnesia

When an agent's accumulated context approaches its window limit (typically 128K–200K tokens), the framework triggers *compaction*: summarizing the full context into a shorter representation. This summarization is lossy by design—a 200K-token context compacted to 10K tokens discards ∼95% of content.

We term the resulting failure mode **compaction amnesia**: the agent functions normally, undergoes compaction, and immediately loses access to facts it had seconds earlier. The summarization model systematically drops:

- Specific numerical values (port numbers, thresholds, IDs)
- Configuration decisions and their rationale
- Temporal sequences and causal chains
- Entity relationships and biographical details

This is distinct from retrieval failures (information exists but isn't found) and hallucination (information is fabricated). Compaction amnesia is *structural*—the information is genuinely destroyed.

3.2 Architecture

Memory Rescue operates as middleware between the agent framework and the LLM provider. When compaction is triggered, it executes a parallel extraction pipeline *before* the context is summarized.

Algorithm 1 Memory Rescue: Pre-Compaction Extraction

Require: Context window C, vector database V, fast model M_f
Agent framework signals compaction of C
$S \leftarrow \text{serialize}(C)$
parallel {
  $F \leftarrow M_f(\text{“Extract critical facts from: ”} + S)$
  $E \leftarrow M_f(\text{“Extract entity relationships from: ”} + S)$
  $D \leftarrow M_f(\text{“Extract decisions and rationale from: ”} + S)$
}
memories $\leftarrow \text{deduplicate}(F \cup E \cup D)$
for m **in** memories **do**
  $\mathbf{v} \leftarrow \text{embed}(m)$
  $V.\text{upsert}(m, \mathbf{v}, \text{metadata}(m))$
end for
Allow compaction to proceed on C

Design decisions.

- **Parallel extraction**: Three specialized prompts run concurrently, targeting facts, relationships, and decisions separately. Total extraction time: ∼4 seconds on Cerebras (vs. ∼30s sequential).
- **Fast-inference models**: We use Cerebras GLM-4.7 (>2000 tok/s, free tier). The model extracts and reformats rather than reasons, so frontier quality is unnecessary.
- **Pre-compaction timing**: Extraction occurs while the full context is still available. Post-compaction rescue would operate on already-lossy output.
- **Hybrid retrieval**: Dense embeddings (nomic-embed-text-v1.5, 768d) combined with BM25 sparse retrieval and cross-encoder reranking (bge-reranker-v2-m3).

3.3 Vector Database

We use Qdrant as the persistent store:

- 762,051 memories accumulated over 90+ days of continuous operation
- Embeddings generated on local GPU (nomic-embed-text-v1.5, 768 dimensions)
- HNSW index with 8 segments, 1.5M indexed vectors (including payload indices)
- Cross-encoder reranking on local GPU (bge-reranker-v2-m3)

3.4 Evaluation

3.4.1 Fact Retention

We evaluate retention using 20 synthetic operational facts (port numbers, configuration values, entity names, temporal details) representative of information typically lost during compaction.

Table 1: Fact retention under compaction ($n = 20$ operational facts)

Method	Preserved	Retention Rate	Overhead
Vanilla compaction	1/20	5.0%	0s
Memory Rescue	20/20	100.0%	∼4s

Vanilla compaction retains only 1 of 20 specific facts (5%), preserving only the most general information. Memory Rescue achieves perfect retention by individually committing each fact to the vector database before compaction occurs.

3.4.2 Retrieval Accuracy

We evaluate retrieval on 20 diverse queries against the full 762K-point corpus:

Table 2: Retrieval accuracy on production corpus

Metric	Value
Corpus size	762,051 memories
Recall@5	55.0% (11/20)
Mean precision	66.25%
Mean latency	700 ms
Median latency	786 ms
P95 latency	926 ms

The 55% Recall@5 reflects query difficulty: ambiguous queries ("What happened Feb 19?") and queries requiring inference ("Dashboard URL") score lower, while queries with clear keyword overlap achieve >90% precision. Sub-second median latency is acceptable for agent workflows where retrieval occurs at session start and periodically during operation.

4 Component 2: Inference Routing

4.1 Problem: Cost of Uniform Model Selection

An autonomous agent running 25+ scheduled background tasks (monitoring, reporting, scanning, data aggregation) alongside interactive chat faces a cost disparity: frontier models like Claude Opus 4.6 ($15/$75 per MTok input/output) deliver quality unnecessary for routine background work. Using a single expensive model for all traffic wastes resources on tasks that don't require frontier reasoning.

4.2 Architecture

Inference Routing is implemented as an Anthropic-compatible HTTP proxy that routes requests across multiple model "heads":

Table 3: Model head configuration

Head	Role	Input $/MTok	Output $/MTok
Claude Opus 4.6	Interactive reasoning	$15.00	$75.00
MiniMax M2.5-Highspeed	Background tasks	$1.00	$4.00
Cerebras GLM-4.7	Compaction & extraction	$0.00	$0.00
OpenCode Zen	Free Opus-equivalent fallback	$0.00	$0.00
Anthropic Direct	Paid last-resort fallback	$15.00	$75.00

The proxy implements four layers:

1. Task-Based Routing. Requests are classified by origin: interactive chat (user messages) routes to Opus; scheduled background tasks route to MiniMax. Classification is based on the request metadata, not content analysis.

2. Quality Gate. Every response from a budget model is scored on a 0.0–1.0 scale using a lightweight, rule-based scorer (no additional LLM call):

- **XML hallucination** (-0.4): Detects spurious tags (`<thinking>`, `<observation>`) that some models hallucinate when imitating chain-of-thought.
- **Formatting compliance** (-0.15 each): Checks for bold text and emoji per communication standards.
- **System message handling** (-0.5): Verifies null responses for system-only messages.

Responses below threshold (0.5) are automatically escalated to the next stronger model.

3. Prompt Compensation. A model-specific suffix (897 characters) is injected into the system prompt for budget models, encoding formatting rules, XML avoidance, and style requirements. This compensates for known weaknesses without requiring fine-tuning.

4. Safety Layer. Independent of model selection, a blocklist-based safety layer provides:

- 13 dangerous command patterns (e.g., `rm -rf`, `chmod 777`)
- 7 protected configuration file paths
- 8 API key and credential patterns (scrubbing)
- 5 XML hallucination cleanup patterns

5. Failover Chain. If the primary provider is unavailable, requests cascade: Opus → OpenCode Zen (free) → Anthropic Direct (paid) → error.

4.3 Evaluation

4.3.1 Cost Reduction

Production data from 24 hours of operation (1,454 total requests):

Table 4: Cost comparison — background task traffic only (production, 24h)

Configuration	Daily Cost	Monthly (proj.)	Savings
All-Opus baseline	\$35.24	\$1,057	—
Inference Routing	\$0.73	\$21.90	97.9%

The \$0.73 daily cost comprises MiniMax M2.5-Highspeed (\$0.73) and Cerebras (\$0.00). Interactive chat continues on Opus (separate budget, excluded from comparison).

4.3.2 Quality Preservation

Table 5: Quality gate performance

Metric	Test Suite	Production
Pass rate	85.7% (6/7)	100% (27/27)
Escalations	1/7	0/27
Quality threshold	0.5	
Prompt suffix	897 characters	

The higher production pass rate (100% vs. 86% test suite) reflects that the test suite includes intentionally adversarial cases. In practice, prompt compensation eliminates the formatting issues that would trigger escalation.

4.3.3 Latency

Table 6: Inference latency (identical prompt, short response)

Model	Mean Latency	Output Tokens
MiniMax M2.5-Highspeed	2.40s	128
Claude Opus 4.6	3.79s	108
Speedup	**1.58×**	

Budget routing is both cheaper *and* faster, producing more tokens in less time.

4.3.4 Robustness

Table 7: Adversarial robustness testing

Test Suite	Tests	Pass Rate	Focus
General gauntlet	20	80%	Broad capability
Adversarial ("murder suite")	40	88%	Edge cases, failures
Safety layer	43	100%	Command/secret blocking

4.3.5 Concurrency Scaling

Table 8: Concurrent request scaling (MiniMax M2.5-Highspeed)

Concurrent Requests	Failures	Throughput
1–20	0	60–450 tok/s
50–100	0	550–594 tok/s
120+	Rate limited	—

5 Combined Operation

When both components operate together, the agent achieves:

1. **Persistent memory across compactions**: Facts extracted by Memory Rescue (using Cerebras at $0) survive indefinitely in the vector database.
2. **Cost-efficient background processing**: Scheduled tasks run on MiniMax at 97.9% lower cost with quality guaranteed by the gate.
3. **Frontier quality for interactive work**: User-facing chat remains on Opus, unaffected by budget routing.
4. **Safety independent of model**: The blocklist-based safety layer catches dangerous commands regardless of which model generates them.

In production, this combination has operated for 30+ days handling 1,400+ daily requests, 25+ scheduled tasks, and 762K+ persistent memories with zero unrecovered failures.

6 Future Components

The Cartu Method is designed as an extensible framework. Components under development or consideration include:

- **Retrieval Optimization**: RL-based memory selection (inspired by MemRL) to improve Recall@5 beyond the current 55%.
- **Best-of-N Consensus**: Running N parallel inference calls on budget models and selecting the best response by majority voting, trading latency for quality.
- **Semantic Quality Scoring**: Replacing or augmenting the rule-based quality gate with an LLM-based semantic scorer that can assess factual correctness.
- **Multi-Agent Coordination**: Memory sharing and task routing across multiple specialized agents operating on shared infrastructure.
- **Adaptive Model Selection**: Learning which model to use per-task based on historical quality scores, moving beyond static task-type routing.

Updates will be published to the repository as components mature.

7 Limitations

- **Retrieval accuracy**: 55% Recall@5 on a 762K corpus is insufficient for safety-critical fact lookup. Improving retrieval quality is the highest-priority future component.
- **Quality gate scope**: The rule-based scorer catches formatting and hallucination issues but cannot assess semantic correctness. A well-formatted but factually wrong response will pass.
- **Evaluation scale**: Some benchmarks use small sample sizes ($n = 2$ for latency, $n = 27$ for quality gate). Larger-scale evaluation would strengthen confidence.

- **Single-agent validation**: All results are from one production agent. Generalization to different architectures, tasks, and domains requires further study.
- **Compaction simulation**: The vanilla compaction baseline uses a manually written summary rather than an actual LLM-generated compaction, which may understate vanilla performance.

8 Conclusion

The Cartu Method provides a practical, production-tested framework for two critical problems in autonomous AI agent infrastructure: memory persistence across context compaction and cost-efficient multi-model inference routing. By treating these as components of a unified framework rather than isolated solutions, the method enables agents to operate continuously with persistent memory, safe command execution, and costs reduced by 97.9%—without sacrificing output quality.

The framework is open-source, requires no model training, and is designed so that new components can be added as they are developed. We encourage the community to contribute additional components, evaluation benchmarks, and integration guides.

Code and documentation: https://github.com/jcartu/rasputin

References

Y. Bai, S. Kadavath, S. Kundu, A. Askell, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

L. Chen, M. Zaharia, and J. Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

P. Lewis, E. Perez, A. Piktus, F. Petroni, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, 2020.

Anonymous. MemRL: Memory-augmented reinforcement learning for language agent memory management. *arXiv preprint arXiv:2601.03192*, 2025.

C. Packer, S. Wooders, K. Lin, V. Fang, et al. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Anthropic. Prompt caching. https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching, 2024.