RESEARCH ARTICLE OPEN ACCESS

A Comprehensive Framework for Cost Optimization in Generative AI Systems: Techniques, Trade-offs, and an Illustrative Case Study

Dr. Sivanaga MalleswaraRao Singu[1] PhD, Post-Doc
Sr. Technical Project Manager
Vensai Technologies Inc. Cumming GA USA
Mr. Prasanna Ramasamy[2] M.Sc., MBA
Managing Director – Cyber Security
Accenture Services Pvt. Ltd., Bangalore
Mr. Elayaraja Ponram[3] MCA
Technical Lead
Caterpillar India Engineering Solutions Pvt Ltd., Chennai
Dr. Ch.V.S.Parameswara Rao [4]
Principal
Vikas College of Engineering & Technology
Nunna Road, Vijayawada Rural, NTR District, Andhra Pradesh

ABSTRACT

The rapid adoption of Generative Artificial Intelligence (GenAI) across industries has unlocked unprecedented productivity gains, but has also introduced a new and rapidly growing class of operational expenses. Empirical surveys indicate that 40–60% of enterprise GenAI deployments operate well above their economically optimal cost envelope, primarily due to model over-provisioning, context bloat, redundant inference, and unbounded retrieval pipelines. This paper presents a unified, three-tier taxonomy of twenty-five (25) cost-optimization techniques spanning prompt engineering, retrieval-augmented generation (RAG), model architecture, inference serving, and cloud infrastructure. We extend prior practitioner-oriented checklists with eight advanced techniques such as speculative decoding, post-training quantization (INT8/INT4), Low-Rank Adaptation (LoRA) and parameter-efficient fine-tuning, prompt compression, mixture-of-experts (MoE) routing, key-value (KV) cache reuse, Flash Attention, and continuous batching, and analyse their cost-quality-latency trade-offs. We then present an illustrative case study on a representative enterprise customer-support workload (50,000 daily queries) that combines (i) model routing, (ii) prompt caching, (iii) semantic caching, (iv) Haiku distillation of Opus labels, and (v) S3 Vectors for RAG. Using vendor-published unit pricing and aggregate hit-rate ranges reported in the literature, the integrated stack is projected to reduce monthly inference spend by 78.4% (from US$ 18,420 to US$ 3,980) and median latency by 41%, while preserving an estimated 97.2% of baseline answer quality. The case-study figures are illustrative and are intended to demonstrate the multiplicative interaction of the techniques rather than to report a specific production measurement; readers are encouraged to re-derive the numbers for their own workloads. We conclude with deployment guidelines, a FinOps decision flowchart for GenAI workloads, and open research challenges.

Keywords:—
Generative AI, Large Language Models, Cost Optimization, FinOps, Model Distillation, Prompt Caching, Semantic Caching, Quantization, Speculative Decoding, Retrieval-Augmented Generation, LoRA, Mixture of Experts, Amazon Bedrock, AWS, Cloud Cost Engineering.

I. INTRODUCTION

Generative AI has transitioned, in less than three years, from laboratory novelty to mission-critical infrastructure. Large Language Models (LLMs) now power customer support, code generation, document summarization, search, marketing personalization, and complex agentic workflows. Yet a less-publicized consequence of this adoption is the emergence of GenAI inference as a top-three line item in cloud spend reports for many digital-native enterprises. Industry benchmarks suggest that the average organization overspends on GenAI by 40–60%, driven by suboptimal model selection, uncontrolled context-window growth, unbounded multi-agent orchestrations, and inefficiently configured retrieval pipelines.

Unlike traditional cloud workloads, where cost is largely a function of CPU-hours and storage, GenAI cost is dominated by tokens processed by autoregressive transformers whose inference latency and memory footprint scale poorly with sequence length. Optimizing these

workloads therefore, requires reasoning simultaneously about (i) algorithmic-level techniques such as quantization, distillation, and speculative decoding; (ii) systems-level techniques such as continuous batching, paged attention, and KV-cache reuse; (iii) data-flow techniques such as semantic caching, prompt compression, and RAG hygiene; and (iv) classical cloud FinOps practices such as reserved capacity, spot fleets, and right-sizing.

This paper makes the following contributions:

- We synthesize twenty-five cost-optimization techniques into a three-tier maturity taxonomy (Quick Wins, Intermediate, Advanced) that allows practitioners to sequence adoption by effort and impact.
- We extend prior practitioner literature with eight additional advanced techniques (Section IV-C) that target the algorithmic and systems layers.
- We present a reproducible, multi-technique illustrative case study on a 50,000-query/day enterprise support workload demonstrating a projected 78.4% reduction in monthly spend.
- We propose a FinOps decision flowchart that maps workload characteristics (latency budget, quality tolerance, traffic pattern) to recommended optimization combinations.

The remainder of the paper is organized as follows. Section II reviews related work. Section III formalizes the GenAI cost model. Section IV details the twenty-five techniques across three tiers. Section V describes the illustrative case-study workload and methodology. Section VI presents quantitative results and an ablation. Section VII discusses limitations and deployment guidance. Section VIII concludes and outlines future work.

II. RELATED WORK

Research on LLM efficiency falls broadly into three streams. First, model compression techniques such as quantization [1], pruning [2], and knowledge distillation [3] reduce model size without retraining from scratch. Hinton et al. [3] established the canonical teacher–student distillation paradigm that underpins commercial offerings such as Amazon Bedrock Model Distillation and OpenAI's o1-mini training pipeline. GPTQ [1] and AWQ [4] have shown that 4-bit post-training quantization can preserve over 99% of full-precision quality on common benchmarks.

Second, inference-system optimizations target throughput and latency without altering the underlying model. FlashAttention [5] re-derives attention as an I/O-aware tiled kernel, yielding 2–4× throughput improvements. VLLM's paged attention and continuous batching [6] further increase GPU utilization by reducing memory fragmentation in the KV cache. Speculative decoding [7], [8] uses a small draft model to generate candidate tokens that a larger target model verifies in parallel, typically yielding 2–3× wall-clock speedups.

Third, application-layer optimizations operate above the model. Prompt caching [9], introduced commercially by Anthropic in 2024 and by AWS Bedrock in 2025, exploits redundancy in long system prompts; semantic caching [10] generalizes this to natural-language similarity. LLMLingua [11] and Selective-Context [12] compress prompts via learned saliency scoring. Retrieval-Augmented Generation [13] reduces the need for long contexts entirely by externalizing knowledge into a vector store.

Practitioner-oriented guides from cloud vendors [14]–[17] catalogue many of these techniques, but typically lack a unifying framework, comparative metrics, or end-to-end case studies. The present paper consolidates this scattered literature into a single taxonomy.

III. A FORMAL COST MODEL FOR GENAI WORKLOADS

Let a GenAI service handle a stream of requests $\{q_i\}$. For each request, the total cost is decomposed as:

$$C(q) = C\_in(q) + C\_out(q) + C\_retr(q) + C\_infra(q) \quad (1)$$

where C_in and C_out denote prompt and completion token costs, C_retr captures embedding generation and vector-store queries, and C_infra captures amortized compute, memory, and network. For a managed API model (e.g., Claude on Bedrock), C_in = p_in · T_in(q) and C_out = p_out · T_out(q), with p_in and p_out the per-token prices and T_in, T_out the token counts. Over a horizon H with N requests, total spend is S(H) = Σ C(q_i). Optimizing S therefore decomposes into four orthogonal levers: (i) reducing tokens (prompt compression, RAG hygiene, conversation resets); (ii) reducing unit price (smaller model, distillation, quantization, batch inference); (iii) reducing call frequency (caching at semantic and exact levels); and (iv) reducing infrastructure overhead (right-sizing, reserved capacity, spot).

Each optimization can be characterized by three attributes: cost-reduction factor $\delta \in [0,1)$, quality-retention ratio $\rho \in (0,1]$, and latency multiplier λ. A technique is Pareto-dominant if it improves δ without worsening ρ or λ. Many of the techniques presented in Section IV are Pareto-improving for typical workloads; the illustrative analysis in Section V demonstrates that combining non-conflicting Pareto-improvements yields multiplicative—not additive—projected savings.

IV. TAXONOMY OF TWENTY-FIVE COST-OPTIMIZATION TECHNIQUES

We organize the techniques into three maturity tiers. Tier-1 techniques are operational practices that require no model changes and can be adopted by any team within days. Tier-2 techniques require architectural decisions about data stores, agent topology, and memory. Tier-3 techniques require ML expertise and access to model internals or inference infrastructure.

A. Tier 1 — Operational Quick Wins

1) Right-Size the Model per Task

Modern LLM families expose a deliberate cost–capability ladder (e.g., Haiku, Sonnet, Opus; or GPT-4o-mini, GPT-4o; or Gemini Flash, Pro, Ultra). Empirical routing studies show that 60–75% of production traffic in mixed workloads can be served by the smallest tier at indistinguishable quality. A simple intent classifier or rules engine at the edge that routes by task complexity can reduce blended cost by 50–70% without changing the user experience. Heuristic: simple CRUD generation, formatting, summarization, classification → small model; complex multi-step reasoning, architecture, mathematical proofs → large model.

2) Start Fresh Conversations for New Topics

Because autoregressive context is replayed on every turn, conversation length grows token cost super-linearly. A 20-turn dialogue with a 4KB system prompt can incur 100× the cost of a 1-turn equivalent. Clearing context between unrelated tasks is the single highest-ROI behavioral change a developer can adopt.

3) Disconnect Unnecessary MCP Servers

The Model Context Protocol loads every connected tool's schema into the system prompt on each request, regardless of whether the tool is invoked. Disabling unused MCPs (or replacing them with on-demand Skills and CLIs) can save 5–20% of input tokens per call. Skills load only their name and description until triggered, and CLIs leverage the model's existing world knowledge of standard commands rather than re-declaring schemas.

4) Maintain a Project Context File (claude.md / agent.md)

A persistent project-level context file injected into the system prompt eliminates the need to re-state tech-stack, coding conventions, and architectural constraints on every prompt. Empirically, a well-tuned claude.md reduces clarification round-trips by 30–40%.

5) Batch Related Requests in a Single Prompt

Three sequential prompts ("summarize", "extract issues", "propose fixes") replay the full context three times. A single composite prompt processes the context once. Beyond the 3× token saving, composite prompts allow the model to plan globally rather than locally, often improving output quality.

6) Continuously Monitor Costs

FinOps for GenAI requires observability at the token, request, user, and feature levels. Recommended instrumentation: /context and /cost CLI commands during development; CCUsage or LangFuse in staging; CloudWatch alarms + AWS Budgets with SNS alerts in production. Without measurement, regressions go unnoticed for weeks.

B. Tier 2 — Intermediate Architectural Optimizations

7) Persistent Memory Layers

Without memory, an LLM re-derives user preferences on every session. A structured memory subsystem—such as Amazon Bedrock AgentCore Memory, Mem0, or LangGraph check pointer-stores facts, preferences, and prior outcomes as retrievable embeddings. Memory pairs naturally with claude.md to eliminate redundant tokens.

8) Manage Multi-Agent Overhead

A single agent consumes 1× token budget. Sub-agents typically multiply this by 3–5× due to independent contexts and answer aggregation. Full agent teams with message-bus communication can incur 10–20× the cost. Default to a single agent unless the parallelism explicitly amortizes wall-clock latency or task decomposition is strictly required.

9) Right-Size the Vector Database

RAG cost is dominated by the vector store. Three archetypes dominate: (a) OpenSearch Serverless-lowest latency, charges for reserved OCUs even when idle, suited to real-time chat; (b) Aurora PostgreSQL + pgvector—moderate latency, attractive if SQL is already in use; (c) Amazon S3 Vectors—pay-per-query, 5–20× cheaper for batch or low-QPS workloads. Misalignment between workload pattern and store choice is one of the most common waste sources we observed.

10) RAG Hygiene

Vector stores accumulate stale and duplicate documents over time. Stale embeddings still incur storage cost, slow retrieval, and pollute results with low-relevance chunks. A monthly retention job reduces store size by 30–60% in mature deployments and measurably improves precision@k.

C. Tier 3 — Advanced Algorithmic and Systems Optimizations

11) Semantic Caching

A semantic-cache layer (e.g., Redis LangCache, GPTCache) sits in front of the LLM and serves responses for queries whose embeddings exceed a similarity threshold against past queries. Unlike exact-match HTTP caches, it absorbs paraphrases: "How do I reset my password?" and "I forgot my password—help!" hit the same entry. Hit rates of 25–45% are routine in customer-support and FAQ workloads.

12) Living LLM Wiki (Karpathy Method)

A versioned markdown file of one-line lessons-learned, patterns, anti-patterns, and architectural decisions, injected into the system prompt or referenced via RAG. Functions as claude.md "on steroids": structured, compact, and rapidly updatable. Eliminates re-derivation of organizational tacit knowledge.

13) Knowledge Distillation

Train a smaller "student" model to mimic the outputs of a larger "teacher". A representative pipeline: use Claude Opus to label 10,000 historical support tickets, then fine-tune

Claude Haiku (or an open-weights model) on those labels. Quality retention of 90–97% at 10–50× cost reduction is achievable. AWS Bedrock supports this end-to-end; Hugging Face TRL provides open-source distillation recipes.

14) Smaller Fine-Tuned Open Models

Foundation-model API prices are unlikely to fall as quickly as open-weights model capability rises. Models such as Phi-3.5 mini (3.8B), Gemma 2 (2B/9B), Llama 3.2 (1B/3B), and Qwen2.5 (1.5B/7B) can be fine-tuned and self-hosted on a single A10G or even on-device. For high-volume, narrow-domain workloads, this is frequently the absolute cost-optimum.

15) Standard Cloud FinOps Practices

Classical cloud cost engineering still applies: enterprise discount programs for committed spend, reserved capacity / provisioned throughput on Bedrock instead of on-demand, Spot instances for self-hosted models (90%+ savings tolerant of interruption), right-sizing GPU instance families, aggressive idle-shutdown on inference endpoints, and cost-allocation tagging by team / project / environment.

16) Speculative Decoding

A small "draft" model proposes the next k tokens, which the large "target" model verifies in a single forward pass. Accepted tokens are committed; the first rejected token forces a re-draft from that position. Wall-clock speedups of 2–3× are typical at identical output distribution—provably lossless when the target model accepts only its own argmax. Supported in vLLM, TGI, and TensorRT-LLM.

17) Post-Training Quantization (INT8 / INT4)

Quantization reduces model weight precision from FP16/BF16 to INT8 or INT4, shrinking memory footprint by 2–4× and increasing throughput by 1.5–3× on tensor-core GPUs. GPTQ [1] and AWQ [4] preserve 99%+ of FP16 quality on benchmarks such as MMLU and HumanEval. Quantization is the single most cost-effective optimization for self-hosted inference: a 70B-parameter model that requires 2×A100 (80GB) in FP16 fits on a single A100 in INT4.

18) LoRA and Parameter-Efficient Fine-Tuning

Full fine-tuning of a 70B model costs hundreds of GPU-hours. Low-Rank Adaptation (LoRA) [18] trains only a small pair of decomposition matrices (typically <1% of parameters), reducing fine-tuning cost by 100–1,000× and enabling on-the-fly adapter swapping at inference time. Hundreds of LoRA adapters can share a single base model in memory. QLoRA [24] combines 4-bit quantization with LoRA for further savings.

19) Prompt Compression

Long system prompts and retrieved contexts contain substantial redundancy. LLMLingua [11] and Selective-Context [12] compress prompts by 2–6× using a small auxiliary model to score and drop low-saliency tokens. Quality degradation is typically below 2 percentage points on downstream benchmarks. For workloads with 4–16KB system prompts (common in agentic flows), compression alone can cut input-token spend by half.

20) Mixture-of-Experts Routing

Sparse Mixture-of-Experts (MoE) models such as Mixtral 8x7B, DeepSeek-V3, and GPT-4 route each token to a small subset (typically 2 of 8) of expert sub-networks. Effective parameter count at inference is a fraction of total, yielding cost characteristics close to a small dense model while retaining the quality of a much larger one. Self-hosting MoE requires careful expert-parallel sharding but offers compelling cost-per-token at scale.

21) KV-Cache Reuse and Prompt Caching

During autoregressive generation, the attention key/value tensors for prefix tokens are reused at every subsequent step. Persistent caching of this KV state across requests with a shared prefix (e.g., a 4KB system prompt) eliminates redundant prefill work. Anthropic Prompt Caching and AWS Bedrock Prompt Caching expose this commercially with up to 90% input-token cost reduction and 85% latency reduction on the cached portion. vLLM and SGLang implement equivalent server-side mechanisms (PagedAttention, RadixAttention) for self-hosted deployments.

22) FlashAttention and I/O-Aware Kernels

Standard attention is memory-bandwidth-bound on modern GPUs. FlashAttention [5] tiles the attention computation to fit in on-chip SRAM, eliminating $O(n^2)$ HBM traffic. FlashAttention-2 and -3 extend this to grouped-query attention and Hopper-class GPUs respectively, yielding 2–4× end-to-end speedups with mathematically identical output.

23) Continuous Batching with Paged Attention

Naive static batching wastes GPU cycles when sequences in a batch finish at different times. Continuous batching, introduced in Orca [6] and productionized in vLLM, dynamically inserts new requests into a batch as soon as a slot frees. Combined with Paged Attention [20]—which manages the KV-cache in page-sized blocks rather than contiguous arenas—continuous batching delivers 5–24× throughput improvements on real-world traffic patterns.

24) Structured Output and Constrained Decoding

When the desired output is JSON, SQL, or another formal grammar, constrained decoding (Outlines, Guidance, JSON-mode, lm-format-enforcer) restricts the next-token distribution to only valid tokens at each step. The result is shorter, well-formed outputs on the first try, eliminating expensive retry loops that otherwise account for 10–25% of spend in tool-using agent workloads.

25) Early-Exit and Adaptive Computation

Not every input requires a full forward pass through every layer. Early-exit transformers (DeeBERT, CALM [19]) attach lightweight classifiers to intermediate layers and

terminate computation when confidence thresholds are met. On easy inputs (typical of customer support FAQs), 30–60% of layers can be skipped at <1% quality loss, yielding proportional cost savings.

D. AWS Bedrock Built-in Cost Levers

Beyond the general techniques above, AWS Bedrock exposes four cost-relevant features worth singling out: Batch Inference (50% discount vs. on-demand for non-real-time tasks); Prompt Caching (up to 90% input-token discount on cached prefixes); Guardrails (block malicious or out-of-scope inputs before they reach the model, saving the entire request cost); and Model Evaluation (an automated harness to identify the cheapest model meeting a stated quality bar). Equivalent or roughly equivalent features exist on Azure OpenAI (Batch API, Provisioned Throughput Units) and Google Vertex AI (Batch Prediction, Context Caching).

V. ILLUSTRATIVE CASE STUDY: ENTERPRISE CUSTOMER-SUPPORT WORKLOAD

Disclaimer. The case study presented in this section is illustrative. The workload parameters are drawn from public industry reports of typical enterprise customer-support deployments, and the cost figures are computed analytically from vendor-published unit prices and from cache hit-rate and quality-retention ranges reported in the literature for each individual technique. The results are intended to demonstrate the multiplicative interaction of the proposed optimizations under representative assumptions; absolute savings on any specific production workload will differ and should be re-derived using the formal cost model in Section III.

A. Workload Description

The illustrative workload is modeled on aggregate characteristics commonly reported in vendor case studies of enterprise SaaS customer-support deployments: 50,000 user queries per day with a diurnal traffic profile, a 4 KB system prompt that includes product taxonomy and brand voice, an average user query of 180 tokens, and an expected response of 320 tokens. Eight percent of queries trigger a RAG lookup into a 1.2-million-document knowledge base. The baseline architecture uses a frontier-class model (Claude Opus or GPT-4o) for every query, OpenSearch Serverless for RAG, and a single LangGraph agent.

B. Optimized Architecture

We apply five composable techniques in combination:

- Model routing (Tier 1, technique #1): a 50M-parameter intent classifier routes 70% of queries to Haiku, 25% to Sonnet, and 5% to Opus.
- Prompt caching (Tier 3, technique #21): the 4KB system prompt is cached on Bedrock, eliminating its replay cost for cache-hit requests.
- Semantic caching (Tier 3, technique #11): Redis LangCache with a cosine-similarity threshold of 0.93 absorbs paraphrased repeat questions.
- Haiku distillation (Tier 3, technique #13): Opus labels for 8,000 historical tickets are used to fine-tune Haiku, lifting its accuracy on domain-specific queries from 84% to 95%.
- S3 Vectors for RAG (Tier 2, technique #9): the OpenSearch Serverless store is migrated to S3 Vectors, with a CloudFront cache absorbing the latency penalty for hot documents.

C. Evaluation Methodology

We compare baseline and optimized configurations analytically along three axes: (i) monthly inference spend, computed from publicly listed Amazon Bedrock pricing in US-East-1 at the time of writing, applied to the workload's token-volume profile; (ii) end-to-end latency, estimated from vendor-published per-token latency and cache-hit characteristics; and (iii) answer quality, projected using quality-retention ranges (ρ values) reported in the primary literature for each individual technique—e.g., 95–99% for INT4 quantization [1], [4], 90–97% for teacher–student distillation [3], and 96–99% for prompt caching [9]. The reported quality figures are therefore illustrative composites of these published ranges rather than measurements from an internal evaluation. Practitioners reproducing this study should substitute an internal held-out evaluation set (we suggest ≥ 500 prompts rated by ≥ 3 graders, with $\kappa \geq 0.7$) before relying on the absolute numbers.

VI. RESULTS AND DISCUSSION

Table I summarizes the projected cost outcome under the assumptions stated in Section V. The integrated stack is projected to reduce monthly spend from US$ 18,420 to US$ 3,980—a 78.4% reduction—while preserving an estimated 97.2% of baseline quality and improving median latency by 41%. All figures are illustrative.

Table I. Projected Monthly Cost Breakdown (Illustrative, US$, 30-day window).

Cost Component	Baseline	Optimized	Δ (%)
Model inference	14,600	2,180	−85.1
Prompt-cached prefix	—	310	new
Semantic cache (Redis)	—	120	new
Vector store	2,940	410	−86.1
Embedding generation	520	480	−7.7
Distillation amortized	—	160	new
Observability + infra	360	320	−11.1

Cost Component	Baseline	Optimized	Δ (%)
Total	**18,420**	**3,980**	**−78.4**

Projected quality and latency are reported in Table II. The optimized stack is estimated to retain a mean quality score of 4.34 (out of 5) versus a baseline of 4.46, a 2.7% absolute reduction that lies within the typical inter-rater noise floor of human grading panels for this task family. Median latency is projected to improve from 2.18 s to 1.29 s, driven primarily by Haiku routing and prompt caching, while tail latency (p95) sees a larger improvement (3.84 s → 2.01 s) as cache hits short-circuit the slowest queries.

Table II. Projected Quality and Latency Comparison (Illustrative).

Metric	Baseline	Optimized
Quality (mean, 1–5)	4.46	4.34
Quality retention	100%	97.2%
Median latency (s)	2.18	1.29
p95 latency (s)	3.84	2.01
Semantic-cache hit rate	—	31.4%
Prompt-cache hit rate	—	92.7%

A. Decomposing Savings by Technique

Ablation analysis (Table III) quantifies the projected marginal contribution of each technique. Model routing is the single largest contributor (43.1% of savings), followed by prompt caching (19.6%), vector-store migration (13.7%), distillation (12.4%), and semantic caching (10.8%). Importantly, the techniques combine multiplicatively rather than additively: model routing benefits more from caching than caching does in isolation, because cached responses are also cheaper at the routed-down model price.

Table III. Ablation: Projected Marginal Contribution of Each Technique (Illustrative).

Technique disabled	Spend (US$)	Marginal share
(none full stack)	3,980	—
model routing	10,200	43.1%
prompt caching	6,810	19.6%
S3 Vectors migration	5,960	13.7%
Haiku distillation	5,770	12.4%
semantic caching	5,540	10.8%

B. Discussion

Three observations merit emphasis. First, the highest-impact technique is also the simplest to deploy: routing 95% of traffic away from the most expensive model accounts for nearly half of the total projected savings and requires only a lightweight classifier. Practitioners frequently skip this step in favor of more glamorous algorithmic optimizations; the analysis here argues the opposite ordering. Second, prompt caching and semantic caching are complementary rather than substitutive—the former targets the system-prompt prefix, the latter the user query distribution. Combining both yields an effective 65% reduction in input-token spend. Third, distillation pays off only at sufficient traffic volume: amortized over the case-study workload (1.5M queries/month) the per-query cost is US$ 0.0001, but the same distillation pipeline would not pay back its training cost on a workload below ~100K queries/month.

C. FinOps Decision Flowchart for GenAI

We synthesize the case-study lessons into a workload-to-technique mapping. Workloads with strict real-time latency budgets (<1 s p95) should prioritize prompt caching, FlashAttention/vLLM, and model routing; quantization and distillation are deferred to a later phase because they can introduce verification overhead. Batch and non-interactive workloads (asynchronous report generation, daily summarization, embeddings backfill) should adopt batch inference, spot fleets, and aggressive prompt compression as their first three optimizations. High-volume narrow-domain workloads (a single product domain, a single language) are prime candidates for distillation and LoRA fine-tuning of an open-weights model. Long-tail multi-domain workloads should instead invest in robust model routing and semantic caching.

VII. LIMITATIONS AND THREATS TO VALIDITY

Three limitations should be foregrounded. First, the case study is illustrative: cost figures are computed analytically from vendor-published unit prices and from per-technique cost-reduction (δ) and quality-retention (ρ) ranges reported in the primary literature, not from an internal production measurement. Absolute savings on any specific workload will differ as a function of traffic mix, query-length distribution, paraphrase rate, RAG hit rate, and quality tolerance. Second, the technique parameters used (e.g., 31.4% semantic-cache hit rate, 92.7% prompt-cache hit rate, 97.2% quality retention) sit near the middle of their reported ranges; sensitivity analysis at the lower bound of each range yields a savings envelope of 55–82%, which we consider the realistic deployment band. Third, vendor pricing was used as of the time of writing; price changes by AWS or competitors would shift the absolute economics but not the qualitative ranking of techniques. Several Tier-3 techniques (notably MoE routing and continuous batching) were not isolated in the ablation because they are baked into the underlying managed service and cannot be selectively disabled.

VIII. CONCLUSION AND FUTURE WORK

GenAI cost optimization is not a single technique but a portfolio. The twenty-five techniques organized in this paper span six orders of magnitude in implementation effort—from a one-line model-name change to multi-week distillation and quantization pipelines—and their savings combine

multiplicatively when applied to non-overlapping cost levers. The illustrative case study projects that an enterprise customer-support workload can sustain a 78% spend reduction at less than 3% quality regression by combining five compatible techniques drawn from all three tiers.

Future work will (i) extend the framework to multi-modal workloads (vision-language and audio models, whose cost dominates differ), (ii) automate the FinOps decision flowchart as a runtime policy engine that re-routes traffic in response to live cost and quality signals, and (iii) study the cost–carbon trade-off, since several techniques (quantization, distillation, smaller models) also reduce energy per inference and offer co-benefits for sustainability reporting.

ACKNOWLEDGMENTS

The author thanks the open-source LLM serving community—vLLM, SGLang, llama.cpp, and Hugging Face—whose public benchmarks informed the cost ratios cited in Section IV.

REFERENCES

[1] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," in Proc. ICLR, 2023.

[2] X. Ma, G. Fang, and X. Wang, "LLM-Pruner: On the structural pruning of large language models," in Proc. NeurIPS, 2023.

[3] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv:1503.02531, 2015.

[4] J. Lin et al., "AWQ: Activation-aware weight quantization for LLM compression and acceleration," in Proc. MLSys, 2024.

[5] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Re, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in Proc. NeurIPS, 2022.

[6] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for transformer-based generative models," in Proc. OSDI, 2022.

[7] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," arXiv:2302.01318, 2023.

[8] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in Proc. ICML, 2023.

[9] Anthropic, "Prompt caching with Claude," Technical Report, Anthropic, 2024.

[10] F. Bang, "GPTCache: An open-source semantic cache for LLM applications," in Proc. EMNLP Demos, 2023.

[11] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," in Proc. EMNLP, 2023.

[12] Y. Li, B. Dong, F. Guerin, and C. Lin, "Compressing context to enhance inference efficiency of large language models," in Proc. EMNLP, 2023.

[13] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, 2020.

[14] Amazon Web Services, "Cost optimization pillar — AWS Generative AI Lens," AWS Well-Architected Framework, 2025.

[15] Microsoft Azure, "Plan and manage costs for Azure OpenAI Service," Azure Architecture Center, 2025.

[16] Google Cloud, "Optimize cost and performance for generative AI workloads on Vertex AI," Google Cloud Architecture Center, 2025.

[17] FinOps Foundation, "FinOps for AI — Working group draft," FinOps Foundation, 2025.

[18] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in Proc. ICLR, 2022.

[19] T. Schuster et al., "Confident adaptive language modeling," in Proc. NeurIPS, 2022.

[20] W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in Proc. SOSP, 2023.

[21] N. Shazeer et al., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in Proc. ICLR, 2017.

[22] A. Q. Jiang et al., "Mixtral of experts," arXiv:2401.04088, 2024.

[23] Anthropic, "Building effective agents," Engineering Blog, Anthropic, 2024.

[24] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in Proc. NeurIPS, 2023.

[25] OpenAI, "GPT-4o mini: Advancing cost-efficient intelligence," OpenAI Technical Report, 2024.