The Workload–Router–Pool Architecture for LLM Inference Optimization

A Vision Paper from the vLLM Semantic Router Project

Huamin Chen[1]   Xunzhuo Liu[1]   Bowei He[2]   Fuyuan Lyu[3,4]   Yankai Chen[2]

Xue Liu[1,2,3,4]   Yuhan Liu[5]   Junchen Jiang[6]

[1]vLLM Semantic Router Project, [2]MBZUAI, [3]McGill University, [4]Mila, [5]University of Chicago, [6]Tensormesh Inc. / UChicago

Abstract. Over the past year, the vLLM Semantic Router project has released a series of work spanning: (1) *core routing mechanisms*—signal-driven routing, context-length pool routing, router performance engineering, policy conflict detection, low-latency embedding models, category-aware semantic caching, user-feedback-driven routing adaptation, hallucination detection, and hierarchical content-safety classification for privacy and jailbreak protection; (2) *fleet optimization*—fleet provisioning and energy-efficiency analysis; (3) *agentic and multimodal routing*—multimodal agent routing, tool selection, CUA security, and multi-turn context memory and safety; (4) *governance and standards*—inference routing protocols and multi-provider API extensions. Each paper tackled a specific problem in LLM inference, but the problems are not independent; for example, fleet provisioning depends on the routing policy, which depends on the workload mix, shifting as organizations adopt agentic and multimodal workloads. This paper distills those results into the *Workload–Router–Pool (WRP) architecture*, a three-dimensional framework for LLM inference optimization. **Workload** characterizes what the fleet serves (chat vs. agent, single-turn vs. multi-turn, warm vs. cold, prefill-heavy vs. decode-heavy). **Router** determines how each request is dispatched (static semantic rules, online bandit adaptation, RL-based model selection, quality-aware cascading). **Pool** defines where inference runs (homogeneous vs. heterogeneous GPU, disaggregated prefill/decode, KV-cache topology). We map our prior work onto a 3×3 WRP interaction matrix, identify which cells we have covered and which remain open, and propose twenty-one concrete research directions at the intersections, each grounded in our prior measurements, tiered by maturity from engineering-ready to open research.

Date: March 2026

1   Introduction

Three variables shape production LLM inference: the workload, the routing strategy, and the GPU pool architecture. These are studied by largely separate communities—workload characterization, model routing, and systems/fleet optimization—that seldom interact. But in practice, a routing decision tuned for one workload on one pool topology can be suboptimal for a different combination.

Our research program.   The vLLM Semantic Router (vLLM-SR) project [1] has been building the *inference routing stack*. Across a growing body of work (summarized in Table 1 and Section 2), we have addressed four pillars:

- **Routing architecture**: signal-driven semantic routing with composable signals [1], conflict-free policy languages for probabilistic ML predicates [2], 98× router latency reduction via Flash Attention and prompt compression [3], 2D Matryoshka embeddings for configurable latency–quality trade-offs [4], reasoning-selective routing [5], category-aware semantic caching [6], feedback-driven online routing adaptation [7], hallucination-aware response validation with conditional factcheck classification and token-level detection [8, 9], and hierarchical content-safety classification following the MLCommons AI Safety Taxonomy with a binary safe/unsafe gate [10] and a 9-class hazard categorizer [11] enabling per-category policy enforcement and privacy-preserving routing;
- **Pool routing and fleet optimization**: token-budget pool routing [12], FleetOpt analytical fleet provisioning with gateway-layer compression [13], inference-fleet-sim for queueing-grounded capacity planning [14], and the 1/W law for energy efficiency [15];
- **Multimodal and agent routing**: adaptive VLM routing for computer use agents [16], outcome-aware tool selection [17], the Visual Confused Deputy security guardrail [18], and the gateway-centric personal-assistant stack OpenClaw [19] (open-source; multi-channel sessions and tools behind a single Gateway control plane);
- **Governance and standards**: the Semantic Inference Routing Protocol (SIRP) at IETF [20] and multi-provider extensions for agentic AI inference APIs [21].

Each paper solved a specific problem, but the problems are coupled. For example, pool routing [12] creates a cost cliff whose resolution via gateway-layer compression depends on the workload's prompt-length CDF archetype [13]. The 1/W law [15] shows that routing topology is a stronger energy lever than GPU generation, but only when the pool supports context-length partitioning.

Our contributions. We present the **Workload–Router–Pool (WRP) architecture** (Figure 1), a three-dimensional framework that organizes both our prior work and the broader research landscape. The contributions are:

1. A **structural decomposition** into three interacting dimensions—Workload, Router, Pool—grounded in evidence from our previous work.
2. A **mapping** of our prior work onto the 3×3 WRP interaction matrix, showing which cells we have addressed and which remain open.
3. A **research roadmap** of twenty-one concrete opportunities at the intersections, each grounded in our prior measurements.

2 Foundation: The vLLM Semantic Router Research Program

Table 1 lists the publications that underpin this paper. We group them into four pillars.

2.1 Pillar 1: Routing Architecture

The vLLM-SR position paper [1] introduced signal-driven decision routing: composing heterogeneous signals—keyword patterns, embedding similarity, domain classifiers, language detection—into deployment-specific routing policies across thirteen signal types. ProbPol [2] formalized conflict detection when probabilistic ML signals co-fire, and mmBERT-Embed [4] provides the embedding backbone with 2D Matryoshka flexibility for configurable latency–quality trade-offs (Section 5.1).

A design constraint is that the router must be *low-latency and low-resource*: it should not require a dedicated GPU that could otherwise serve LLM inference. FastRouter [3] enforces this for long-context requests with a 98× latency reduction and sub-800 MB GPU footprint (Section 5.1).

Beyond request-time dispatch, the routing stack includes response-time mechanisms. WhenToReason [5] classifies whether a query needs reasoning-mode invocation. SemCache [6] eliminates redundant
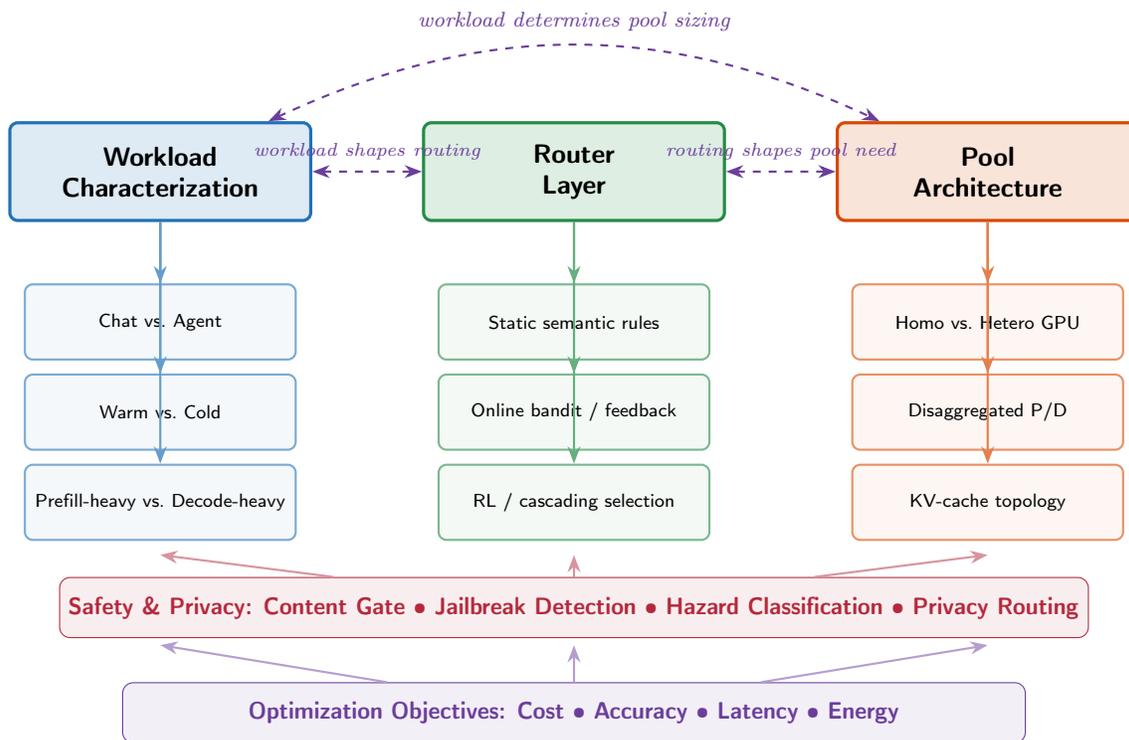
Figure 1 The Workload–Router–Pool (WRP) architecture. Three dimensions interact through dashed bidirectional arrows. Safety & Privacy is a cross-cutting layer: content gating and hazard classification shape all three dimensions. All are further constrained by the optimization objectives at the bottom. Our prior work addresses one or more cells in this framework.

inference via category-aware semantic caching with domain-specific similarity thresholds (Section 4.2). The Feedback Detector [7] enables online routing adaptation from per-turn user satisfaction signals (Section 5.2). HaluGate [8] and the Factcheck Classifier [9] extend the router to response-time validation, steering factual queries toward reliable models and feeding per-model hallucination rates back into routing policy (Section 5.1).

The routing stack also enforces content safety and privacy through a hierarchical classification pipeline following the MLCommons AI Safety Taxonomy. The Level-1 binary classifier [10] gates every inbound request as safe or unsafe in a single forward pass; requests flagged unsafe are escalated to the Level-2 nine-class hazard classifier [11], which maps them to specific categories (violent crimes, privacy, misinformation, etc). This two-stage design keeps routing overhead low for benign traffic while enabling per-category policy enforcement—e.g., blocking privacy-violating prompts outright, routing misinformation-prone queries to models with stronger factual grounding, or logging specialized-advice requests for compliance auditing (Section 5.1).

2.2 Pillar 2: Pool Routing and Fleet Optimization

Token-budget pool routing [12] was the starting point: dispatching to short-context or long-context pools based on estimated total token budget, reducing GPU instances by 17–39% on production traces (Section 5.1).

FleetOpt [13] unified pool routing with gateway-layer compression into an analytical framework that derives the minimum-cost two-pool fleet with optimal boundary $\mathcal{B}^*$ and compression parameter γ^*,

Table 1 Publications from the vLLM Semantic Router project that form the foundation of the WRP vision. Each row indicates which WRP dimension(s) the paper primarily addresses. **W**=Workload, **R**=Router, **P**=Pool.

Short name	Title / venue	W	R	P
PoolRouting [12]	Token-Budget-Aware Pool Routing (arXiv)	•	•	•
FleetOpt [13]	Analytical Fleet Provisioning with C&R (arXiv)	•	•	•
1/W Law [15]	Context-Length Routing and Energy Efficiency (arXiv)	•	•	•
FeedbackDet [7]	mmBERT-32K Feedback Detector (HuggingFace)	•	•	
AVR [16]	Adaptive VLM Routing for CUAs (arXiv)	•	•	
MPExt [21]	Multi-Provider Extensions for Agentic AI (IETF)	•	•	
OpenClaw [19]	Personal assistant + Gateway (GitHub OSS)	•	•	
SemCache [6]	Category-Aware Semantic Caching (arXiv)		•	•
FleetSim [14]	Queueing-Grounded Fleet Capacity Planner (arXiv)	•		•
vLLM-SR [1]	Signal-Driven Decision Routing for MoM Models (arXiv)		•	
ProbPol [2]	Conflict-Free Policy for Probabilistic ML Predicates (arXiv)		•	
FastRouter [3]	98× Faster LLM Routing (arXiv)		•	
WhenToReason [5]	Semantic Router for vLLM (NeurIPS MLForSys)		•	
mmBERT-Embed [4]	2D Matryoshka Embeddings for Routing (HuggingFace)		•	
HaluGate [8]	Token-Level Hallucination Detection (vLLM Blog)		•	
FactCheck [9]	mmBERT-32K Factcheck Classifier (HuggingFace)		•	
OATS [17]	Outcome-Aware Tool Selection (arXiv)		•	
VCD [18]	Visual Confused Deputy: CUA Security (arXiv)		•	
SafetyL1 [10]	MLCommons Binary Safety Classifier (HuggingFace)		•	
SafetyL2 [11]	MLCommons 9-Class Hazard Classifier (HuggingFace)		•	
SIRP [20]	Semantic Inference Routing Protocol (IETF)		•	

given a workload CDF and latency target. Complementing this, inference-fleet-sim [14] validated these results with a discrete-event simulator covering A10G, A100, and H100 GPUs (Section 6.1).

The 1/W law [15] quantifies the energy dimension: routing topology is a stronger energy lever than GPU generation, and the gains compose multiplicatively (Section 6.2).

2.3 Pillar 3: Multimodal and Agent Routing

Adaptive VLM Routing (AVR) [16] extended routing to vision-language CUA tasks with a three-tier framework: Tier 1 multimodal classifier ($\sim$10 ms pre-route), Tier 2 small-VLM confidence probing with token-filtered scoring, Tier 3 large-model escalation. AVR projects up to 78% cost reduction while staying within 2 pp of an all-large-model baseline.

OATS [17] addressed tool selection in the critical request path: offline interpolation of tool embeddings toward success-query centroids improves NDCG@5 from 0.869 to 0.940 on MetaTool without serving-time cost.

The Visual Confused Deputy [18] reframed CUA perception errors as a security vulnerability, proposing dual-channel contrastive classification that independently evaluates click targets and reasoning text, achieving F1=0.915. Combined with AVR, VCD provides both efficiency and safety through one routing pipeline.

2.4 Pillar 4: Governance and Standards

At the standards level, the Semantic Inference Routing Protocol (SIRP) [20] specifies a framework for content-level classification and semantic routing in AI inference systems at the IETF. The Multi-Provider Extensions [21] extend agentic AI inference APIs for multi-provider environments.

3 The WRP Architecture

Our publications support a thesis:

Thesis 1 (WRP Coupling). *The three dimensions of LLM inference optimization—**Workload**, **Router**, and **Pool**—are coupled, not orthogonal. Optimizing any single dimension in isolation leaves significant efficiency on the floor. The largest gains require co-optimization across dimensions.*

Evidence from our own work: FleetOpt [13] demonstrates that co-designing the router compression parameter γ with pool sizes (n_s, n_l) yields 3.1–6.4% lower cost than retrofitting compression onto a pre-existing fleet (**Router** $\times$ **Pool** coupling). The 1/W law [15] shows that the same GPU fleet can vary $40\times$ in energy efficiency depending on the context window served (**Workload** $\times$ **Pool** coupling). AVR [16] shows that VLM workloads require entirely different routing signals than text-only chat (**Workload** $\times$ **Router** coupling).

Definition 1 (WRP Decomposition). A production LLM inference deployment is characterized by a triple $(\mathbf{W}, \mathbf{R}, \mathbf{P})$ where:

- **W**: the *workload* distribution over request types, prompt lengths, decode lengths, session structures, and modalities;
- **R**: the *router* that maps each request to a (model, pool, configuration) tuple; and
- **P**: the *pool architecture* specifying GPU types, pool boundaries, disaggregation topology, and KV-cache management.

The optimization objective is a weighted combination of *cost* (GPU-hours per request), *accuracy* (task completion quality), *latency* (P99 TTFT, TPOT), and *energy* (tokens per watt), subject to SLO constraints.

4 Dimension 1: Workload Characterization

Our fleet optimization work [12–14] showed that workload characterization drives the remaining design decisions. We organize workloads along four axes.

4.1 Chat vs. Agent Workloads

Chat workloads (LMSYS-Chat-1M [22], the Azure LLM Inference Trace published alongside Splitwise [23]) are dominated by short prompts: 80–95% of requests fall below 8K tokens, with a heavy tail to 64K+. Agent workloads (SWE-bench [24], BFCL [25]) exhibit prompt lengths that grow deterministically with turn number as tool outputs accumulate. ServeGen [26] reports that agents make 3–10$\times$ more LLM calls per user request than chatbots.

Both types are *multi-turn*, but with different session structures. Chat sessions are conversational: each turn carries implicit context from prior exchanges, and the user may shift topics, provide corrections, or gradually escalate adversarial probes across turns. Agent sessions are task-driven: context grows monotonically as tool outputs accumulate, and the session terminates on task completion or failure. Multi-turn APIs (such as OpenAI's Responses API) encode these patterns explicitly, offering conversation-state management and turn-level optimizations that single-request routing cannot exploit. A multi-turn-aware router must maintain session context for two reasons: KV-cache affinity and routing

signal quality. Prior turns reveal user intent, topic drift, difficulty escalation, and safety violations that a single-turn classifier would miss.

Our pool routing paper [12] first exposed how the chat/agent distinction creates routing opportunities: the 80–95% of short chat requests are over-provisioned in a homogeneous 64K fleet, wasting 7/8 of KV-cache slots. The deterministic turn-by-turn growth of agent sessions suggests that session-aware prediction could outperform per-category EMA for routing.

Memory management as a hidden cost multiplier. Beyond raw context growth, multi-turn sessions face a compounding cost problem rooted in memory management failures. When an LLM hallucinates or selects the wrong tool, the erroneous output enters the conversation history and is re-sent on every subsequent turn—the organization pays input-token cost for content that *reduced* accuracy. The agent must either tolerate the polluted context (risking further errors through self-conditioning [27]) or spend additional LLM calls to summarize, compress, or remove the bad turns before continuing. AgentHallu [28] benchmarks this challenge: even frontier models achieve only 41.1% accuracy at localizing which step caused a hallucination, dropping to 11.6% for tool-use hallucinations—making targeted cleanup unreliable and often requiring full-context re-summarization.

A comprehensive memory survey [29] identifies five mechanism families for agent memory (context-resident compression, retrieval-augmented stores, reflective self-improvement, hierarchical virtual context, and policy-learned management), each with distinct cost profiles. A direct cost-performance comparison [30] shows that at 100K-token context lengths, per-turn inference charges grow proportionally with context even under prompt caching, and fact-based memory systems become cheaper after approximately ten interaction turns—precisely the regime where agent sessions operate. Context compression can mitigate the problem: ACON [31] achieves 26–54% peak-token reduction while preserving 95+% task accuracy via failure-driven guideline optimization, and Focus [32] demonstrates 22.7% token savings through autonomous agent-driven context pruning on SWE-bench tasks. However, both operate on the agent side, optimizing a single session in isolation. This is a fundamental limitation: SAMULE [33] shows that *single-trajectory* reflection (the only level available to an individual agent) produces narrow error corrections, while *inter-task* learning that aggregates failure patterns across diverse sessions yields transferable insights that significantly outperform per-session baselines. CORRECT [34] reinforces this: multi-agent errors recur with similar structural patterns across requests, and an online cache of distilled error schemata—built from prior sessions—improves step-level error localization by up to 19.8%. Mistake Notebook Learning [35] demonstrates the same principle: batch-clustered failure analysis produces generalizable guidance that prevents known pitfalls without parameter updates.

The router has a structural advantage that directly exploits this cross-session transferability: it observes outcomes from thousands of sessions per domain across all tenants, aggregating failure patterns into the per-*(model, domain, failure-class)* table (Opportunity 4). An individual agent, by contrast, sees only its own session—too few examples to build reliable failure statistics or discover domain-specific compression policies (e.g., "coding sessions tolerate aggressive early-turn pruning of failed tool traces; medical sessions must retain full context for compliance audits"). The router provides this cross-session intelligence as a fleet-wide service that every agent benefits from immediately, including new agents that would otherwise face a cold-start problem.

Gateway-native assistants and fleet observability. OpenClaw [19] is a concrete open-source example of the pattern *personal assistant + local-first Gateway*: users talk to the agent over many messaging channels while a WebSocket Gateway coordinates sessions, tool execution (browser, automation, device nodes), and model calls. Such deployments generate long, messy multi-turn traces—tool outcomes, retries, channel switches, and compaction events—that fixed academic benchmarks rarely capture. The inference router sits on the hot path for model traffic from those gateways and can log which model, tool surface, and safety path correlated with short vs. runaway sessions. Aggregating traces across tenants supplies offline RL and fleet-prior datasets that a single local assistant cannot amass (Opportunity 1,

Opportunity 6). Conversely, routing policies tuned for token- and round-minimization feed back into operators' choices of model defaults and tool budgets at the Gateway.

Proxy-level instruction, tool, and memory surfaces. ITR [36] retrieves minimal system-instruction fragments and a narrowed tool subset per agent step; on the authors' controlled agent benchmark it reports ∼95% lower per-step context tokens and large gains in correct tool routing because monolithic catalogs can consume ∼90% of the context window. The same mechanism can be implemented at an inference *gateway*—rewriting `tools` payloads and system blocks before the request reaches the model— so frameworks that always send full catalogs still benefit from fleet-wide calibration. ToolScope [37] merges and filters tools on the agent side; SMART [38] shows that curbing tool overuse improves both cost and downstream accuracy. A router that coordinates narrowing, merging hints, and budget signals (BATS [39]) avoids duplicating incompatible policies in every SDK. Finally, memory is not only "what the agent remembers" but *what the next prefill pays for*: MemCost-style analyses [30] show when fact stores beat raw context, and session-aware schedulers (AgServe [40], Continuum [41]) treat KV-cache lifetime and multi-turn structure as first-class resources. The router is the natural place to choose among full history, injected summaries, and retrieval-first prefill for the next turn—reducing tokens *and* the error compounding that lengthens rounds (Opportunity 7).

4.2 Warm vs. Cold Requests

Warm requests hit a KV-cache prefix from a prior turn or shared system prompt; *cold* requests start from scratch. Cached tokens cost up to 10× less [42], and prefix-cache-aware scheduling achieves 57× faster response times in distributed deployments [42].

For agentic workloads, 40–60% of session time involves paused tool calls, causing LRU eviction to collapse cache hit rates [43]. Session-affinity routing—keeping a session on the same pool instance— could preserve KV-cache locality. Category-aware semantic caching [6] complements this at a higher level: domain-aware similarity thresholds and TTLs eliminate redundant inference requests entirely— returning cached responses for semantically equivalent queries without invoking the LLM, reducing both cost and load on the serving pool.

4.3 Prefill-Heavy vs. Decode-Heavy Workloads

Vision-language models encode images into hundreds of visual tokens, making VLM workloads *prefill-heavy*. Our AVR system [16] confronted this directly: CUA screenshots are ∼5 MB, and routing 4K screenshots requires multimodal signal extraction. FastRouter [3] showed that classical NLP compression can cap router input to ∼512 tokens regardless of original prompt length, making prefill-heavy workloads tractable for the routing layer itself.

Coding agents are *decode-heavy*: code generation produces long structured outputs. The decode ratio directly affects whether disaggregated prefill/decode architectures (Splitwise [23], DistServe [44]) provide benefit, and our inference-fleet-sim [14] tool models this trade-off explicitly.

4.4 Workload Identity Signals

Our prior work treats incoming requests as opaque text and infers workload type through content analysis—embedding similarity, keyword matching, and domain classifiers. In practice, the request structure itself carries rich identity signals that the router can exploit without content inspection, opening the door to zero-latency workload classification.

Explicit API-level signals. Modern inference APIs already expose structural metadata that distinguishes agent from chat workloads:

- **Tool definitions.** A request carrying a `tools` array with function schemas is almost certainly an agent workload; the number and types of tools (code execution, web search, MCP connectors) further classify the agent's operational mode. The tool catalog itself is a cost lever: ITR [36] shows that dynamically exposing only the minimal tool subset per step reduces per-step context tokens by 95% and improves correct tool routing by 32%, because monolithic tool catalogs consume up to 90% of available context.
- **Session chaining.** Fields such as `previous_response_id` (OpenAI Responses API) or conversation-state objects signal multi-turn continuations. Their presence tells the router the request is mid-session (enabling KV-cache retention, Opportunity 15) and provides the turn number without parsing conversation history.
- **System-prompt fingerprints.** System prompts are typically static per deployment and can be hashed at the gateway. A lookup table of known hashes maps each deployment to its domain, expected tool set, and historical session-length distribution—giving the router a per-request prior at zero classification cost.
- **Gateway metadata headers.** Production gateways (Cloudflare AI Gateway's `cf-aig-metadata`, LiteLLM's `x-litellm-tags`, and custom `X-Client-Request-Id` headers) already let operators annotate requests with environment, team, and workload-type tags. SIRP [20] standardizes this with classification-axis header fields.

Caller identity and authorization scope. The four signal classes above describe *what* the request is. None describes *who* is making it or *what they are authorized to do*. The bearer token carried in every LLM API call fills this gap: in production gateways, it encodes caller identity *and* authorization scope—which models a key can access, which tools it may invoke, and what budget it may consume.

- **Per-key tool permissions.** LiteLLM's Tool Permission Guardrail [45] enforces per-key tool allow/deny rules with regex matching on tool names and arguments. Two modes—*block* (reject the request) and *rewrite* (strip forbidden tools before the model sees them)—demonstrate that caller-scoped tool governance is already production-ready at the gateway layer.
- **Structured agent authorization.** The Agent Authorization Profile (AAP) [46]—an emerging OAuth 2.0 extension—adds five JWT claims: agent identity, capabilities with enforceable server-side constraints (domain restrictions, rate limits, time windows per tool), task binding (prevents scope drift), delegation tracking, and audit trails. Its core principle is that "authorization must be evaluated at execution time, not just during connection."
- **Infrastructure analogue.** Kubernetes RBAC with admission webhooks enforces per-identity permissions at the API-server boundary regardless of what the application inside the pod believes it should do—the same zero-trust principle [47] applied to LLM requests.

The bearer token enables a new class of routing decisions absent from the current WRP roadmap: role-based tool-call enforcement (Opportunity 8), per-caller fleet-wide reputation scoring (Opportunity 9), and router-enforced behavioral commitments (Opportunity 10). It also extends existing opportunities: per-caller cumulative risk accumulation (Opportunity 3), cross-session token budgets (Opportunity 5), RBAC predicates in the governance DSL (Opportunity 20), role-aware tool catalog shaping (Opportunity 6), and caller-aware memory isolation (Opportunity 7).

Inferred structural signals. Beyond explicit metadata, the router can derive workload identity from request structure at dispatch time:

- **Conversation depth.** The number of message objects in the `messages` array reveals session maturity: a 2-message exchange is likely a cold-start chat; a 20-message array with alternating `assistant`/`tool` roles is a deep agent session with different model, pool, and compression requirements.
- **Tool-call history pattern.** If prior assistant messages contain `tool_calls` with high failure rates (parseable from subsequent `tool` role messages reporting errors), the router can preemptively

route to a stronger model or apply compression before the next turn.

- **Reasoning metadata.** NofT [48] shows that the number of chain-of-thought reasoning steps predicts task difficulty pre-inference, enabling difficulty-aware model selection with 95% adversarial detection accuracy.

From static to per-stage routing. These signals enable a shift from per-request routing (one decision per API call) to *per-stage* routing within agentic workflows. Aragog [49] demonstrates this: by decoupling a one-time accuracy-preserving configuration step from a per-stage scheduler that uses current system observations, it achieves 50–217% throughput improvement and 32–79% latency reduction at peak load. The WRP framework is well-positioned for this evolution: the router already maintains session state for multi-turn awareness (VCD [18], Feedback Detector [7]), and the per-stage identity signals listed above provide the input features for a stage-aware routing policy that adapts model selection, compression, and pool assignment as the session evolves.

4.5 Workload CDF Archetypes

FleetOpt [13] identified three prompt-length CDF archetypes that determine optimal pool and routing configuration:

1. **Concentrated-below** (*e.g.*, Azure): 80–95% short requests. Aggressive compression minimizes long-pool cost.
2. **Dispersed** (*e.g.*, LMSYS): requests span the length spectrum. Balanced two-pool split with moderate compression.
3. **Concentrated-above** (*e.g.*, SWE-bench): most requests are long. Raise $\mathcal{B}_{\text{short}}$ rather than compress.

These archetypes are not static: as organizations shift from chat to agents, the CDF migrates from Archetype 1 toward Archetype 3. FleetOpt [13] showed that the minimum-cost fleet is qualitatively different for each archetype, and inference-fleet-sim [14] confirmed that monolithic topology can outperform disaggregated for Archetype 1 when $\mathcal{B}_{\text{short}} < 4\text{K}$.

5 Dimension 2: The Router Layer

The router sits between the client and the inference pool. Our publications cover three adaptation regimes: static rules, online bandit adaptation, and RL-based selection.

5.1 Static Semantic Routing

The vLLM-SR position paper [1] established the signal-driven architecture: composing thirteen signal types into deployment-specific policies across vLLM, OpenAI, Anthropic, Azure, Bedrock, and Gemini backends. The embedding similarity signal is powered by mmBERT-Embed [4], whose 2D Matryoshka design lets operators dial latency vs. quality per deployment (2.56 ms at layer-6 for real-time routing, 11 ms at layer-22 for maximum recall). ProbPol [2] formalized conflict detection for these signals.

Token-budget pool routing [12] implements a specialized static rule: dispatching to short or long pools based on estimated total token budget, achieving 17–39% GPU reduction. OATS [17] addresses tool selection with zero-cost embedding refinement, improving NDCG@5 from 0.869 to 0.940.

Separately, WhenToReason [5] classifies whether a query warrants reasoning-mode invocation, avoiding unnecessary chain-of-thought tokens on simple queries. The Factcheck Classifier [9] adds another pre-route signal: queries classified as fact-seeking can be steered toward models with lower hallucination rates or flagged for post-response validation via HaluGate [8].

For multi-turn sessions, the vLLM-SR project [1] provides two mechanisms beyond single-request classification. First, *context memory injection*: the router accumulates contextual signals from prior turns (topic, domain, user profile, difficulty trajectory) and injects this context into the routing decision for subsequent turns, giving the inference endpoint richer context that improves response accuracy. This is similar in spirit to the conversation-state management in multi-turn APIs like OpenAI's Responses API. Second, *contrastive embedding classification for multi-turn safety*: rather than classifying each turn in isolation, the router uses few-shot contrastive embeddings to detect jailbreak attempts that spread adversarial content across multiple turns, which single-turn classifiers cannot detect.

Content safety and privacy form another pre-route gate. The MLCommons Safety Level-1 classifier [10] performs a binary safe/unsafe check on every inbound request; flagged requests are passed to the Level-2 hazard classifier [11], which assigns one of nine MLCommons categories (violent crimes, non-violent crimes, sex crimes, weapons/CBRNE, self-harm, hate, specialized advice, privacy, misinformation). This hierarchical design keeps latency negligible for safe traffic (only unsafe requests pay the second-stage cost) and enables per-category routing policies: privacy-violating prompts can be blocked at the gateway, misinformation queries routed to fact-grounded models, and specialized-advice requests logged for compliance auditing. Both classifiers use mmBERT LoRA adapters, fitting the project's low-resource design constraint.

FastRouter [3] ensures all these routing decisions—single-turn and multi-turn—happen within the latency budget: 50 ms end-to-end with an 800 MB GPU footprint, small enough to co-locate with serving.

5.2 Online Adaptive Routing for Chat

Chat workloads offer a natural feedback signal: the user's next message indicates whether the previous response was satisfactory. Our mmBERT-32K Feedback Detector [7] classifies each user turn into four categories (satisfied, needs clarification, wrong answer, wants different approach) at 98.8% accuracy with <1 ms overhead. When the detector flags dissatisfaction, the router can re-route subsequent turns to a stronger model or a different pool—no explicit reward model is needed because the user's own reaction serves as the reward. Combined with the router's context memory injection (Section 5.1), the adaptation becomes session-aware: the router knows not just that the user is dissatisfied, but which topic and difficulty level triggered the dissatisfaction, enabling targeted re-routing rather than a blanket model upgrade.

Beyond our project, RouteLLM [50] learns routers from human preference data (95% GPT-4 quality at 26% cost); MixLLM [51] uses contextual bandits with query tags (97.25% of GPT-4 quality at 24.18% cost); SageServe [52] integrates traffic forecasting with ILP-based routing (25% GPU-hour savings at Microsoft scale).

5.3 RL-Based Routing for Agents

Agent workloads lack the per-turn user feedback that chat enjoys: the "user" is another LLM or an orchestrator, and task success is only observable at the end of a multi-step session. This delayed, sparse reward makes RL a better fit than bandit methods. Router-R1 [53] demonstrates the approach: it instantiates the router itself as an LLM, interleaving "think" actions (internal reasoning) with "route" actions (model invocation), and trains end-to-end via PPO with a composite reward—format, outcome, and cost—that lets the router learn performance–cost trade-offs without hand-crafted heuristics. R2-Router [54] jointly selects model and output length budget at 4–5× lower cost. M-CMAB [55] uses online RL for multi-modal scheduling under heterogeneous budgets.

Our AVR system [16] applies cascading to VLM agent tasks: Tier 1 multimodal classifier ($\sim$10 ms), Tier 2 small-VLM confidence probing with token-filtered scoring, Tier 3 large-model escalation, projecting up to 78% cost reduction while staying within 2 pp of an all-large-model baseline. When

combined with the Visual Confused Deputy guardrail [18], high-risk actions escalate directly to the strongest model.

Adaptation regime depends on workload. For *chat*, per-turn user feedback detected by classifiers like our Feedback Detector [7] provides a fast, low-overhead signal to adjust routing online. For *agents*, task-level outcomes observed only after multi-step sessions call for RL-based optimization [53] that reasons over sequential routing decisions.

5.4 Agent Loops, Memory, and Fleet-Scale Orchestration

Recent agent-serving work argues that inference for tool-using agents should be treated as a *data-systems* problem, not a sequence of independent chat completions. Helium [56] reframes agentic LLM serving around workflow structure, reuse, and operator placement; Sutradhara [57] co-designs orchestrator and engine for tool-heavy inference; CONCUR [58] applies congestion-aware batching to high-throughput agentic inference; AgServe [40] shows that *session-aware* scheduling transcends static cost–quality trade-offs; Continuum [41] couples multi-turn scheduling with KV-cache time-to-live so that paused tool calls do not silently destroy locality. These systems align with experience-driven routing (EvoRoute [59]) and budget-aware sequential routing (BAAR [60]), but they are typically evaluated inside a single stack.

The semantic router complements them by exporting a *fleet-wide* control surface: the same hooks that implement signal-driven model routing can (i) reshape tool and instruction payloads (ITR-style [36], evaluated on that paper's controlled benchmark), (ii) attach memory-tier decisions (full context vs. summary vs. retrieval-first prefill informed by MemCost [30] and provider prompt-caching studies on long-horizon agents [61]), and (iii) emit scheduling hints (admission deferral, batch affinity) that reduce queueing-induced retries—a common hidden driver of extra agent rounds. xRouter's RL-trained orchestration [62] is evidence that the router layer can learn cost-aware policies when rewards are defined over full trajectories; combining that with gateway-native stacks such as OpenClaw [19]—where real users drive high-variance multi-turn traffic through a single control plane—multiplies the diversity of trajectories available for learning without requiring every framework to share one memory or tool SDK.

6 Dimension 3: Pool Architecture

Our fleet optimization tools [13–15] provide the analytical and simulation models for pool architecture.

6.1 Homogeneous vs. Heterogeneous GPU Pools

FleetOpt [13] derives the minimum-cost fleet analytically: a two-pool architecture with optimal boundary $\mathcal{B}^*$ satisfying an equal marginal GPU cost condition. On production traces, this reduces GPU cost by 6–82% versus homogeneous fleets. inference-fleet-sim [14] extends this to heterogeneous GPU types (A10G, A100, H100) with a physics-informed performance model.

Mélange [63] formulates heterogeneous GPU allocation as cost-aware bin packing, achieving up to 77% cost reduction. Our simulation tool [14] shows that the cheapest GPU type depends on the workload in non-obvious ways: A10G can beat H100 for concentrated-below workloads where concurrency matters more than per-token speed.

6.2 Energy and the 1/W Law

The 1/W law [15] has direct implications for pool architecture: tokens per watt halves every time the context window doubles, because of KV-cache concurrency limits. At 64K context, an H100 holds 16

sequences (tok/W $\approx$ 1.5); at 4K, 256 sequences (tok/W $\approx$ 17.6).

Routing topology is a stronger energy lever than GPU generation: two-pool routing gives $\sim 2.5\times$ better tok/W than homogeneous; upgrading H100$\rightarrow$B200 gives $\sim 1.7\times$. The gains compose multiplicatively to $\sim 4.25\times$. For MoE models like Qwen3-235B-A22B, active-parameter weight streaming provides a third lever (~ 37.8 tok/W at 8K context, $5.1\times$ better than dense Llama-3.1-70B).

Energy optimization therefore requires routing-pool co-design; a homogeneous fleet cannot reach the energy frontier regardless of GPU generation.

6.3 Disaggregated Prefill/Decode and KV-Cache Topology

Splitwise [23] and DistServe [44] physically separate prefill and decode, serving $4.48\times$ more requests at equivalent SLO. Mooncake [64] provides KV-cache-centric disaggregation with hierarchical offloading (GPU HBM, CPU DRAM, SSD). NVIDIA Dynamo [65] enables dynamic scheduling with KV-cache offloading across memory hierarchies.

Our inference-fleet-sim [14] models all three topologies (monolithic, two-pool-routed, disaggregated), revealing that disaggregation is not always beneficial: for concentrated-below workloads (Archetype 1) at small $\mathcal{B}_{\text{short}}$, the KV-cache transfer overhead exceeds the benefit.

vLLM's PagedAttention [66] reduces KV-cache waste to below 4%, and prefix-cache-aware distributed scheduling [42] delivers $57\times$ speedups. But for agentic workloads, standard LRU eviction is pathological [43]—context-aware retention with session-affinity routing is needed.

7 Cross-Dimensional Interactions: Lessons from Our Work

Cross-dimensional interactions are easy to miss when each dimension is studied separately, but they dominate real-world performance. Below we formalize the main interactions our publications have identified.

7.1 Workload $\times$ Router

Observation 1 (Workload determines router adaptation mechanism). *Chat workloads produce per-turn user feedback; our Feedback Detector [7] classifies satisfaction in real time, allowing the router to re-route on dissatisfaction without explicit reward modeling. Agent workloads lack such per-turn signals—task success is observed only at the end of a multi-step session—so RL-based routing [53] that reasons over sequential decisions is a better fit. The adaptation mechanism must match the workload's feedback granularity.*

Observation 2 (Modality determines routing signals). *Text-only chat uses embedding similarity and keyword signals [1]. VLM workloads require multimodal classifiers—SigLIP+MiniLM in AVR [16]—because text-only signals cannot assess screenshot complexity. The router signal set must match the workload modality.*

Observation 3 (Workload determines routing safety and privacy requirements). *Different workload types impose different safety and privacy requirements on the router. Chat routing errors produce quality degradation; multi-turn chat adds a subtler threat—adversarial users can spread jailbreak content across turns, evading single-turn classifiers. The vLLM-SR [1] addresses this with contrastive embedding classification that evaluates turn sequences, not individual messages. CUA routing errors produce outright security vulnerabilities: the Visual Confused Deputy [18] shows that misrouting a CUA action to a less-capable model can cause grounding errors exploitable for privilege escalation. Across all workload types, our hierarchical MLCommons safety pipeline—binary gate [10] then nine-class hazard categorization [11]—provides uniform content filtering at the routing layer, with category-level*

granularity that lets operators enforce workload-specific policies (e.g., blocking privacy-exfiltrating prompts in customer-facing chat, flagging misinformation in knowledge-retrieval agents). Safety and privacy should be integrated as routing objectives, with multi-turn awareness for chat and action-level guardrails for agents.

Observation 4 (Factual workloads require response-time validation)**.** *Request-time routing alone cannot guarantee correctness: a model may be well-suited for the query's domain yet still hallucinate on a specific fact. The Factcheck Classifier [9] identifies fact-seeking prompts at request time ($\sim 12\,\mathrm{ms}$), and HaluGate [8] validates responses at token level (76–162 ms overhead). Together they close a loop that request-time routing cannot: per-model hallucination rates observed by HaluGate feed back into the routing policy, gradually steering factual traffic toward more reliable models.*

7.2   Router × Pool

Observation 5 (Router policy co-determines pool sizing)**.** *FleetOpt [13] shows that co-designing the compression parameter γ with pool sizes (n_s, n_l) yields 3.1–6.4% lower cost than retrofitting compression onto a pre-existing fleet. The router is a co-design variable, not a layer atop the pool.*

Observation 6 (Router latency constrains pool topology)**.** *FastRouter [3] showed that routing latency is a critical constraint: at 4,918 ms baseline, routing overhead exceeds the TTFT SLO for short-context requests. Only after the 98× reduction to 50 ms does pool routing become practical for the full request stream. mmBERT-Embed [4] reinforces this: its 2D Matryoshka design lets operators trade embedding quality for latency (2.56 ms at layer-6 vs. 11 ms at layer-22), keeping the similarity signal within the router's latency budget even as pool topology grows more complex.*

Observation 7 (Mid-session model switching incurs a KV-cache sunk cost)**.** *In multi-turn sessions, if the Feedback Detector [7] or context memory signals indicate that a better model is available, the router faces a dilemma. Switching models mid-session discards the KV-cache accumulated for the current model—a* sunk cost *proportional to session length and context size. Not switching preserves the cache investment but forgoes the quality improvement—an* opportunity cost *that grows with every subsequent turn served by the suboptimal model. The optimal policy depends on the remaining session length, the quality gap between models, and the KV-cache rebuild cost, creating a Pareto frontier between sunk cost and opportunity cost. No current system explicitly optimizes this trade-off.*

7.3   Workload × Pool

Observation 8 (Workload archetype determines optimal topology)**.** *Our fleet tools [13, 14] show: Archetype 1 (concentrated-below) benefits most from two-pool routing. Archetype 3 (concentrated-above) may benefit from disaggregated prefill/decode because most requests are long. Archetype 2 (dispersed) may require multi-tier pooling.*

Observation 9 (Agent adoption shifts the energy frontier)**.** *The 1/W law [15] predicts that shifting from chat to agents drops fleet-wide energy efficiency by 2–4× unless pool topology adapts. Two-pool routing recovers most of this loss, but only if pool boundaries track workload evolution.*

8   Mapping Our Publications to the WRP Matrix

Table 2 maps each of our publications to the WRP interaction matrix, showing which cross-dimensional cells each paper addresses. The off-diagonal cells that remain sparse represent the open opportunities in Section 9.

Table 2 WRP interaction matrix. Each cell lists our publications that address the corresponding cross-dimensional interaction, plus key external references. Sparse cells indicate open opportunities.

	Workload	Router	Pool
Workload	*CDF archetypes* [13]; traces [23, 26]	AVR [16] (modality); VCD [18] (safety); SafetyL1/L2 [10, 11] (content safety); FeedbackDet [7] (chat adapt.); HaluGate [8] (factual valid.); *sparse—see Opportunities 1 and 4*	FleetOpt [13] (CDF→sizing); 1/W Law [15] (energy); *sparse—see Opportunity 16*
Router	FleetOpt [13] (compression changes CDF); WhenToReason [5]	vLLM-SR [1] (signals); ProbPol [2] (conflicts); FastRouter [3] (latency); mmBERT-Embed [4] (embeddings); OATS [17] (tools); SemCache [6] (caching); HaluGate [8] (validation)	FleetOpt [13] (co-design γ); *sparse—see Opportunities 11 and 15*
Pool	FleetSim [14] (topology→cost); 1/W [15] (topology→energy)	*sparse—see Opportunity 12*	FleetSim [14] (A10G/A100/H100); Mélange [63]; DistServe [44]

9 Vision: A Research Roadmap

The WRP matrix in Table 2 reveals two structurally sparse cells—Router × Pool and Pool × Router—and several interactions in the dense cells that our publications have identified but not yet exploited. Below we propose twenty-one research directions that follow directly from these gaps, each framed around what the semantic router's signal architecture and the WRP cross-dimensional view uniquely enable.

Table 3 assigns each opportunity a maturity tier. Throughout this section, projected benefits are estimates grounded in our prior measurements unless a specific external citation is given.

9.1 Workload × Router

Open Opportunity 1 (Offline RL for session-length-minimizing routing)**.** In agent workloads, a long session is often the consequence of poor routing decisions at early turns. If the router selects a weak model at turn 2 and the model fails a tool call, the agent retries at turn 3 with a longer prompt, which fails again, and so on. A session that could have completed in 3 turns with the right model and tool selection at the outset may instead take 8 turns—each additional turn growing the prompt (ServeGen [26] reports 3–10× more LLM calls per user request for agents than chatbots) and consuming proportionally more compute.

The router observes the full trajectory of every completed session: which model served each turn, which tools were invoked (OATS [17]), whether each tool call succeeded, and how many turns the session took. This is a natural offline RL dataset. The state is (turn number, tools tried so far, cumulative context length, domain); the action is (model, tool) selection; the reward is negative session length (or task-completion / session-length for quality-adjusted optimization). The workload identity signals from Section 4.4 enrich this state space at zero inference cost: the system-prompt fingerprint provides an instant domain prior, the tool-definition catalog identifies the agent's operational mode, and the conversation depth from the messages array gives exact session maturity—all available before the

Table 3 Maturity and feasibility of the twenty-one proposed opportunities. **Tier** indicates the primary barrier: *engineering* (building blocks exist, integration needed), *research* (open technical questions).

#	Opportunity	Tier	Primary barrier
2	HaluGate model reputation routing	Engineering	Sample size per (model, domain) cell
5	Token-budget enforcement for agents	Engineering	Domain-specific budget model calibration
8	Follow-the-sun pool rebalancing	Engineering	Sliding-window CDF estimation + FleetOpt
12	KV-cache retention directives	Engineering	Retention-directive API integration
15	Governance-as-code	Engineering	Extend DSL with WRP constraints
19	Request-level RBAC enforcement	Engineering	Gateway hook + JWT integration
1	Offline RL for session-length routing	Research	Credit assignment + off-policy evaluation
3	Cumulative multi-turn risk scoring	Research	Threshold calibration on adversarial benchmarks
4	Failure-class-aware routing	Research	Multi-dimensional failure weighting
17	**Fleet-learned joint tool–model optimization**	**Research**	**Joint (tool, model) outcome table sparsity**
18	Gateway-coordinated agent loops	Research	End-to-end fleet integration unevaluated; subsystems validated separately
20	Fleet-wide caller reputation	Research	Heterogeneous security-event scoring + decay calibration
21	Router-enforced behavioral commitments	Research	Boundary-declaration NLU classifier + false-positive risk
6	Pool-state inference from observables	Research	Prefill-rate regression + cold start
7	Output-length-aware pool routing	Research	Output-length prediction variance
9	Pool-aware cascading	Research	Cascade-abort threshold + grace period
10	Online $(\gamma, \mathcal{B}^*)$ co-adaptation	Research	Oscillation control + validation gap
11	Mixed-archetype fleet provisioning	Research	Multi-tenant FleetOpt extension
13	Archetype-driven proactive scaling	Research	Archetype-aware forecast vs. aggregate baseline
14	Energy as a routing objective	Research	tok/W signal calibration per pool config
16	Closed-loop self-adaptation	Research	Knob interaction strength + composite reward

router inspects content. The router's fleet-wide visibility is the structural advantage over agent-side SDKs: it collects trajectories from thousands of sessions across all tenants, giving it the training volume that no single agent instance can match.

An offline RL policy (e.g., conservative Q-learning or decision transformer) trained on these trajectories learns which early-turn decisions lead to shorter sessions: "for coding tasks, routing turn 1 to the frontier model and selecting `semantic_search` over `grep` reduces median session length from 7 to 4 turns." The learned policy replaces OATS's per-turn greedy selection with a *per-stage* strategy that adapts model and tool selection as the session evolves—the same progression Aragog [49] demonstrates for throughput (50–217% improvement by decoupling one-time configuration from per-stage scheduling), applied here to session-length minimization.

Feasibility and risk. The router already logs per-turn model and tool-call outcomes for OATS and the Feedback Detector; assembling these into session trajectories is bookkeeping. Offline RL is well suited to this setting: recent work shows that conservative Q-learning variants learn effective scheduling policies from logged

data alone [67], generalizing even from suboptimal (random-heuristic) training trajectories—closely matching the quality of router logs collected under a non-optimal greedy policy. The research questions are: (a) credit assignment—which of the 8 turns caused the extra 5 turns? Offline RL handles this via temporal-difference learning, but noisy agent environments may require variance reduction. (b) Off-policy evaluation—the learned policy must be validated before deployment. Importance-weighted estimators on held-out sessions provide offline evaluation without live traffic risk. The comparison baseline is OATS with single-turn greedy selection; the metric is median session length (turns and tokens) for completed tasks on SWE-bench and BFCL.

Open Opportunity 2 (HaluGate-driven model reputation routing)**.** The Factcheck Classifier [9] identifies fact-seeking prompts at request time ($\sim$12 ms). HaluGate [8] validates responses at token level (76–162 ms overhead) and produces a per-response hallucination verdict. Today, these verdicts are logged but not fed back into routing decisions.

The router processes every response through HaluGate for fact-seeking traffic. Over time, it accumulates (model, domain, hallucinated) triples—enough to maintain a per-model, per-domain accuracy estimate. For example: "Model A hallucinates on medical queries 12% of the time but only 2% on coding; Model B is the reverse." The opportunity is to use this estimate as a live routing signal: for each fact-seeking request, the router queries the per-model accuracy table for the request's domain and routes to the model with the lowest hallucination rate, subject to cost and latency constraints.

Feasibility and risk. The data collection is zero-overhead (HaluGate already runs). The per-model accuracy table is a counter per (model, domain) pair, updated on each verdict. The engineering question is sample size: how many verdicts per (model, domain) cell are needed before the estimate is reliable? A Bayesian approach (Beta prior, Bernoulli likelihood) gives calibrated confidence intervals from small samples. The risk is distributional shift: model updates invalidate historical accuracy estimates. A sliding window (e.g., last 1,000 verdicts per cell) handles this.

Open Opportunity 3 (Cumulative multi-turn risk scoring)**.** SafetyL1 [10] and SafetyL2 [11] classify each turn independently. Multi-turn jailbreak attacks [18] spread malicious intent across turns so that no single turn triggers the classifier—each turn is individually benign, but the cumulative intent is harmful. The vLLM-SR [1] addresses this partially with contrastive embedding classification that evaluates turn sequences, but the routing response to detected escalation is not formalized.

The opportunity is to maintain a per-session cumulative risk score that aggregates borderline SafetyL1/L2 signals across turns. SafetyL1 produces a continuous confidence score (not just a binary verdict); a turn that scores 0.4 (below the 0.5 threshold) is not flagged, but three consecutive 0.4-scoring turns represent genuine escalation. The cumulative score is the EMA of per-turn SafetyL1 confidences. When it crosses a configurable threshold, the router takes graduated action: route to a safety-specialized model, reduce the model's sampling temperature via the API, or flag the session for human review.

Per-caller accumulation across the fleet. The bearer token (Section 4.4) extends this per-session mechanism to a *per-caller* dimension. A non-owner whose interactions trigger borderline SafetyL1 scores across five different agents accumulates a high fleet-wide risk score even if no single session crosses the threshold—the same Crescendo-style escalation applied across agents rather than across turns. The mechanism is identical (EMA), keyed on (`bearer_token`) in addition to (`session_id`). Real-world multi-agent attacks validate this threat model: in live deployments the same non-owner attacked multiple agents sequentially, each encountering the attack fresh with no shared threat memory [68].

Feasibility and risk. SafetyL1 already produces continuous confidence scores; computing the EMA is trivial.

The threat model is grounded in Crescendo attacks [69], which achieve 29–61% higher jailbreak rates than single-turn methods by gradually escalating across benign-looking turns. Recent proxy-level defenses validate the cumulative-scoring approach: a peak+accumulation formula combining single-turn peak risk, persistence ratio, and category diversity achieves 90.8% recall at 1.20% false-positive rate on 10,654 multi-turn conversations [70]; DeepContext uses recurrent networks to track temporal intent drift with sub-20 ms overhead and 0.84 F1 [71]. The research question is threshold calibration: too aggressive triggers false positives on legitimate multi-turn conversations that happen to touch sensitive topics; too lenient misses real attacks. Validation on multi-turn adversarial benchmarks (e.g., the 7,152-prompt MultiBreak dataset) is the natural evaluation path. The graduated response (escalate model → reduce temperature → flag session) provides defense-in-depth rather than a binary block. The per-caller extension adds a fleet-wide dimension: the calibration question becomes whether the cross-agent EMA decay rate should differ from the within-session decay rate (likely yes—cross-agent escalation is slower but more deliberate).

Open Opportunity 4 (Failure-class-aware routing from router outcome memory). ErrorAtlas [72] shows that each LLM has a unique "failure signature"—recurring error patterns that vary by model and domain—across 83 models and 35 datasets. An oracle selector that always picks the best model outperforms the single best model, yet production routers capture only ∼89% of oracle utility [73]. The gap comes from failure modes the router does not track: wrong tool selection (35% of tool-call failures [74]), incorrect parameters (25–68%), retry loops where weaker models degenerate into repetitive re-invocations (baseline recovery rate 32.76% vs. 89.68% with targeted intervention [75]), and mid-session model-switch degradation (±4–13 pp performance swing when the suffix model continues a foreign prefix [76]).

The vSR already observes the outcome of every request through HaluGate (hallucination verdict), SafetyL1/L2 (safety score), OATS (tool-call success/failure), and the Feedback Detector (quality signal). The workload identity signals from Section 4.4 add a pre-inference dimension: the `tools` array in the request reveals the agent's available tool set, and the tool-call history in prior `tool`-role messages exposes failures from earlier turns *before* the router selects the next model. Over thousands of requests, these produce a per-*(model, domain, tool-set, failure class)* outcome table:

- Model A, coding, with [`grep, semantic_search, bash`]: hallucination 2%, tool-call success 91%, retry-loop rate 8%;
- Model B, medical, with [`search, calculator`]: hallucination 3%, tool-call success 85%, retry-loop rate 4%;
- Model C, coding, with [`semantic_search`] only: tool-call success 94%.

The tool-set dimension is critical: the same model may excel with one tool combination and fail with another, and this information is available at zero cost from the request structure. The routing decision becomes: for this (domain, task-type), which model minimizes *combined* failure risk? This generalizes Opportunity 2 from hallucination alone to a multi-dimensional failure profile.

A second dimension is *handoff penalty awareness*. The router tracks what happens when it switches models mid-session: (Model A → Model B) in coding sessions adds +2.1 mean turns, while (Model A → Model C) adds only +0.3 turns. It avoids costly switches: if Model A served turns 1–3 and Model B has a known handoff penalty for Model A's prefix, the router stays with Model A or selects a low-penalty alternative.

A third dimension is *silent drift detection*. The same per-(model, domain) statistics, monitored as a time series, detect quality regressions from provider-side model updates—a real production concern (documented March 2026 incident where a frontier model silently regressed to mid-tier quality without code changes). A 3σ deviation from the sliding-window baseline triggers automatic traffic rebalancing.

The router's structural advantage over agent-side SDKs is fleet-wide visibility: it builds the (model, domain, failure-class) table from all tenants' sessions, a volume no individual agent can match. Research validates this asymmetry: agents confined to single-session reflection produce narrow corrections that

often repeat the same mistakes [33], while cross-session error aggregation yields transferable failure schemata that improve error localization by up to 19.8% [34]. The router provides this cross-session intelligence as a shared service—new agents benefit immediately from fleet-wide failure statistics rather than facing a cold-start learning problem.

Feasibility and risk. The per-(model, domain, failure-class) counters are zero-overhead (all signals are already computed). The handoff-penalty matrix requires multi-turn session correlation, which the router already performs for VCD and the Feedback Detector. The research questions are: (a) how many observations per cell before the failure-class estimate is reliable (Bayesian Beta-Bernoulli gives calibrated intervals from small samples), (b) how to weight multiple failure dimensions into a single routing score (Pareto-front analysis or a learned weighting from session-outcome data), and (c) how quickly the drift detector must react without over-triggering on natural variance. The baseline is single-dimensional HaluGate reputation routing (Opportunity 2); the metric is oracle-utility capture rate on LLMRouterBench.

Open Opportunity 5 (Runtime token-budget enforcement for agent sessions). Agent workflows face a token-budget explosion: a single user request triggers 3–$10\times$ more LLM calls than a chatbot request [26], with context growing quadratically as each turn re-sends the full message history. A 10-step agent with a 4K-token system prompt and 500-token tool outputs consumes over 40K input tokens from context carriage alone. Infinite retry loops (failure class F4 in the taxonomy of [74]) compound this: an unconstrained agent can consume $50\times$ the tokens of a single linear pass.

The root cause is often upstream: hallucinations, wrong tool selections, and tool-call parameter errors inject useless content into the conversation history that is re-sent on every subsequent turn. Failed attempts consume 20–40% of total tokens with zero productive output, and context-window utilization falls below 30% useful content after accounting for repeated system prompts and stale error traces. Localizing the faulty step is itself unreliable—even frontier models achieve only 41.1% accuracy at hallucination attribution, dropping to 11.6% for tool-use errors [28]—so agents resort to expensive full-context re-summarization. The cost compounds through self-conditioning: an LLM processing its own prior errors becomes measurably more likely to produce further errors [27], creating a vicious cycle where each failed turn both wastes tokens and increases the probability of the next failure.

Current mitigations live in the agent SDK (per-workflow budget caps) or a separate billing layer (per-tenant daily limits). Neither has access to the router's session trajectory data: turn number, cumulative tokens dispatched, tool-call outcomes per turn, and the domain-specific session-length distribution learned from historical sessions. Agent-side compression frameworks such as ACON [31] (26–54% peak-token reduction) and Focus [32] (22.7% savings on SWE-bench) show that compression is effective, but they optimize a single session in isolation—the router's fleet-wide visibility enables *domain-specific* compression policies learned from outcome data across thousands of sessions.

The router can enforce token budgets at the dispatch layer. For each active session, it maintains a running token counter (prompt + completion tokens, updated on each dispatch and completion event). The system-prompt fingerprint (Section 4.4) provides instant budget-model selection: a hash lookup maps each deployment to its historical session-cost distribution, skipping domain classification entirely. The conversation depth from the messages array gives exact session maturity—the router knows it is at turn 6 of a session whose P90 cost at turn 6 is X tokens.

When a session exceeds $3\times$ the expected budget at its current turn, the router applies a graduated response:

1. **Shape tools**: reduce the `tools` array to the highest-value subset for this domain and turn, based on fleet-wide tool outcome data (Opportunity 6)—ITR [36] shows this alone recovers up to 95% of per-step context tokens while SMART [38] shows 37% performance gain;
2. **Compress**: apply FleetOpt's γ to reduce the remaining prompt length;

18

3. **Downgrade**: route the next turn to a cheaper model (the session has already exceeded its cost budget; frontier compute is not justified);
4. **Terminate**: if the above steps fail to arrest growth, return a budget-exceeded signal to the orchestrator.

The router is the natural enforcement point because it sees the actual token flow across all turns, while the agent SDK sees only its own session and the billing layer operates on aggregated cost with minutes of delay.

Per-caller cross-session budgets. The bearer token (Section 4.4) extends per-session enforcement to *per-caller cross-session* budgets. The mechanism is direct: `sum(tokens consumed by bearer_token_-X across all agents) > non_owner_budget_ceiling`. Live agent deployments confirm the need: one non-owner consumed 60K tokens over 9 days across multiple sessions, and another uploaded 10 MB of attachments in a single burst [68]. LiteLLM already supports per-key monetary budget limits in production; the extension from monetary to token/tool-invocation budgets with role-dependent ceilings follows the same infrastructure, keyed on the bearer token's identity claims.

Feasibility and risk. The per-session token counter is zero-overhead (the router already tracks dispatches and completions). The graduated compress→downgrade→terminate response mirrors the congestion-control paradigm that is now standard for LLM rate limiting: token-bucket algorithms enforce per-tenant budgets in production gateways, and Concur [58] applies congestion-control-inspired feedback to regulate agent concurrency at the engine level. The domain-specific budget model is a per-(domain, turn-number) percentile table, updated from the same session data used for offline RL (Opportunity 1). The risk is false positives: a legitimately long session (e.g., a complex SWE-bench task) may be terminated prematurely. Using P90 rather than P50 as the baseline and applying the graduated response provides a safety margin. Fact-based memory systems [30] offer a complementary cost lever: at 100K-token context lengths, they become cheaper than long-context inference after ∼10 turns, precisely the regime where agent sessions operate. The router can trigger a memory-system handoff when context carriage cost exceeds the memory-system break-even point. The comparison baseline is SDK-side per-workflow budget caps; the metric is token waste reduction (tokens consumed by sessions that ultimately fail) on SWE-bench and BFCL traces.

Open Opportunity 6 (Fleet-learned joint tool–model selection optimization)**.** Agent frameworks send the full tool catalog on every LLM call, rely on the LLM to select the right tool, and resolve every tool call with the same model. This is triply wasteful: the monolithic catalog consumes up to 90% of context tokens [36]; per-turn greedy tool selection ignores the aggregate cost–accuracy outcome of the full tool-call *sequence*; and using a single model for all tool calls ignores the $250\times$ cost range across model tiers and the fact that different tools demand different model capabilities (a `web_search` summary needs only a cheap text model; an `image_analysis` call requires a vision model; a `code_interpreter` invocation benefits from a reasoning model). A session that uses tool A resolved by model X at turn 3 may complete in 5 turns; using tool B with model Y at the same turn may cause a failure that triggers 3 retries and doubles the session cost. The goal is not necessarily fewer tools or fewer turns—it is the *(tool, model)* strategy that minimizes aggregate session cost while maximizing task completion.

Computer-use agents illustrate this coupling vividly. The same web task—e.g., "find the cheapest flight on this page"—can be solved via two (tool, model) paths: (a) a *vision* path that screenshots the page and sends it to a multimodal model (∼\$0.001/page in vision tokens, but requires an expensive VLM for grounding), or (b) a *DOM/text* path that extracts the accessibility tree or HTML and processes it with a cheap text model (∼\$0.15/page in text tokens, but no vision model needed). AVR [16] demonstrates this exact routing problem: adaptive model selection per GUI action reduces inference cost by up to 78% while staying within 2 pp of an all-large-model baseline. A systematic comparison of

web agent interfaces [77] quantifies the gap: HTML-browsing agents consume 241K tokens per task at F1 0.67, while structured-interface agents achieve F1 0.75–0.87 at 47K–140K tokens. The optimal (tool, model) path depends on the page, the task, and the agent's history—exactly the kind of context that fleet-wide data can disambiguate.

Two independent research threads validate that both halves of this joint action space are learnable from historical data:

Tool selection is learnable.

1. **Tool transition patterns are predictable.** AutoTool [78] constructs directed graphs from historical agent trajectories and finds that tool invocations follow predictable sequential patterns ("tool usage inertia"), reducing inference cost by 30%.
2. **Recurring sequences can be bundled.** AWO [79] discovers that 14.3% of tasks follow identical execution trajectories after 5 steps; transforming recurring sequences into deterministic meta-tools reduces LLM calls by 11.9% while increasing task success by 4.2 pp.
3. **Fine-grained tool rewards produce large gains.** ToolRLA [80] decomposes the reward into tool selection × parameter accuracy × compliance; deployed in production (80+ advisors, 1,200+ daily queries), it achieves 47% task-completion improvement and 63% tool-error reduction.
4. **Cost awareness changes the optimal strategy.** BATS [39] shows that budget-unaware agents hit performance ceilings regardless of tool budget; a budget tracker that signals remaining resources enables dynamic strategy adaptation—"dig deeper" vs. "pivot"—pushing the cost–performance Pareto frontier.

Per-step model selection is learnable.

1. **Per-step Pareto-optimal model selection.** EvoRoute [59] dynamically selects model backbones at each agent step using experience-driven retrieval and Thompson sampling, reducing execution cost by up to 80% and latency by over 70% on GAIA and BrowseComp+ benchmarks while sustaining task performance.
2. **Budget-aware sequential model routing.** BAAR [60] formalizes agentic model routing as a sequential, path-dependent decision problem: early routing mistakes compound, and feedback arrives only at task end. Boundary-guided policy optimization improves the cost–success frontier under strict per-task budgets (Microsoft).
3. **Specialized model pipeline assembly.** HierRouter [81] trains a PPO-based RL agent to iteratively select which specialized model to invoke at each stage of multi-hop inference, conditioning on cumulative cost and evolving context. It improves response quality by up to 2.4× compared to single-model baselines at minimal additional cost.

All seven approaches operate on the agent side, learning from a single agent's or benchmark's trajectories. The router's structural advantage is fleet-wide visibility: it observes (tool, model) outcomes across all tenants and accumulates a per-*(domain, turn-number, tool, model)* outcome table—tool success rate, model accuracy, downstream retries, and total cost contribution—from thousands of sessions. AutoTool's transition graph, learned per-agent, becomes a fleet-wide transition graph trained on orders of magnitude more data; EvoRoute's Pareto filtering, performed per-benchmark, operates over production-scale outcome tables.

The router acts on this knowledge through four mechanisms at dispatch time:

- **Role-aware tool filtering.** Before any efficiency-driven optimization, the bearer token's authorization scope (Section 4.4) restricts the tool catalog to tools the caller is permitted to invoke. Non-owner requests get a *restricted* catalog (only safe, read-only tools), not just an *optimized* one. LiteLLM's "rewrite" mode [45] already implements this at the gateway: unauthorized tools

are stripped from the `tools` array before the model sees them, using the same `async_pre_-call_hook` that powers efficiency-driven shaping. Authorization-driven filtering applies *before* efficiency-driven filtering, ensuring that safety constraints are never relaxed by cost optimization.

- **Tool catalog shaping.** Expose the top-k tools ranked by fleet-wide success rate for the current (domain, turn) cell, reducing context overhead (ITR [36] validates 95% token reduction agent-side; SMART [38] shows 37% performance gain from 24% tool-use reduction).
- **Tool-conditional model routing.** Select the resolving model based on the tool: a vision model for `image_analysis`, a reasoning model for `code_interpreter`, a cheap text model for `web_search` summaries—all driven by the per-(tool, model) outcome table. This is a natural extension of OATS's per-turn tool selection [17]: once the router selects the tool, it selects the cheapest model whose fleet-wide success rate for that (domain, tool) exceeds a quality threshold.
- **Sequence-aware transition priors.** Annotate the dispatch with a (tool, model) preference derived from the fleet-wide transition graph: "at this (domain, turn, previous-tool) state, (tool A, model X) succeeds 94% at \$0.001/call; (tool B, model Y) succeeds 96% at \$0.01/call but saves 1.5 retries on average, making it cheaper in aggregate."

Production gateways support both tool shaping and model routing at dispatch time: LiteLLM's `async_pre_call_hook` and Kong AI Gateway's Request Transformer plugin intercept and modify requests (including the target model endpoint) before forwarding.

Feasibility and risk. Both halves of the joint optimization are independently well-grounded. The tool-selection problem is established by four studies [39, 78–80], with ToolRLA demonstrating production-scale gains (47% completion improvement, 63% error reduction [80]). The per-step model-selection problem is established by three studies [59, 60, 81], with EvoRoute demonstrating 80% cost reduction while sustaining accuracy [59] and HierRouter demonstrating 2.4× quality improvement via specialized model routing [81]. The coupling of the two decisions is directly validated by AVR [16], which routes computer-use actions to different VLMs based on difficulty and achieves 78% cost reduction, and by the web-interface comparison [77], which shows that the (interface, model) choice jointly determines both cost (5× token range) and accuracy (F1 0.67–0.87).

The *novel* WRP contribution is (a) the joint optimization of both decisions simultaneously from a single fleet-wide outcome table, and (b) the aggregation scale—thousands of tenants rather than a single agent's trajectory. Neither dimension has been evaluated at the router layer, but both follow the same aggregation pattern that powers the failure-class routing table (Opportunity 4) and HaluGate reputation routing (Opportunity 2), grounded in cross-session transfer research [33, 34]. The infrastructure is production-ready (gateway pre-call hooks + model-endpoint selection).

The research questions are: (a) does the fleet-wide per-(domain, turn, tool, model) table outperform per-agent learning on common patterns (likely yes—more data) and on novel sessions (likely no—retrieval-based similarity may be more adaptive); (b) what is the optimal table granularity before cells become too sparse (the tool × model cross-product expands the space); (c) does tool-conditional model routing yield gains beyond independent tool and model selection (i.e., is the interaction effect significant). A hybrid approach—fleet-wide priors as default, falling back to full catalog and default model when the session deviates from known patterns—hedges the sparsity and novelty risks. The comparison baselines are monolithic tool exposure with a single model (status quo), OATS single-turn greedy selection, and independent tool-only / model-only optimization; the metrics are aggregate session cost and task-completion rate on SWE-bench and BFCL traces.

Open Opportunity 7 (Gateway-coordinated agent loops: memory tiers, tools, and talk-shaped training). The *problem*—stateless-looking agent APIs that resend huge prefixes each turn, accumulate tool output in context, and suffer extra rounds after bad tool/model choices—is documented on standard agent benchmarks and traces: SWE-bench / BFCL-style workloads drive deterministic context growth [24, 25], agents issue 3–10× more LLM calls per user goal than chat [26], and failures pollute memory in ways that are hard to localize [27, 28]. The *open WRP question* is whether a *single* cross-tenant gateway can compose validated sub-mechanisms—dynamic tool surfaces, cache/KV policy,

session-aware scheduling, congestion control—so frameworks that are never recompiled still gain lower tokens, fewer rounds, and better tool accuracy.

Validated building blocks (separate systems, public benchmarks). **(i) Dynamic instruction/tool exposure.** ITR [36] retrieves minimal system fragments and a narrowed tool set each step; on the authors' controlled agent benchmark (their stated evaluation protocol), it reports ∼95% lower per-step context tokens, ∼32% relative improvement in correct tool routing, ∼70% lower end-to-end episode cost vs. a monolithic baseline, and 2–20× more steps within a fixed context budget—with explicit operational guidance for deployment. **(ii) Prompt-cache layout at real APIs.** A cross-provider study on DeepResearch Bench (>500 multi-turn sessions with live web-search tools and 10K-token system prompts) shows strategic cache-block placement (excluding volatile tool results) cuts API cost by 41–80% and improves TTFT by 13–31% vs. naive full-context caching [61]—evidence that *what* enters the cached prefix is a production cost and latency lever at the HTTP boundary. **(iii) KV retention across tool pauses.** Continuum evaluates on SWE-Bench and BFCL with Llama-3.1 (8B/70B), adding tool-aware KV TTL and scheduling so tool-induced pauses do not destroy reuse; gains grow with turn depth and remain robust under DRAM offload [41]. **(iv) Session-aware serving.** AgServe (NeurIPS 2025) demonstrates that session-level scheduling can escape static cost–quality Pareto fronts for agent workloads [40]. **(v) Workflow- and batch-level serving.** Helium achieves up to 1.56× speedup over strong agent-serving baselines by modeling workflows as query plans with proactive reuse across calls [56]. Sutradhara co-designs orchestration with vLLM: after analyzing production-scale *synthetic* request mixes showing tool waits account for 30–80% of final first-token latency, it overlaps tool work with LLM prefill, streams tool dispatch, and improves cache semantics—cutting median final-response FTR by 15% and end-to-end latency by 10% on A100s [57]. CONCUR adds congestion-style admission control for batched agentic inference and reports up to 4.09× throughput (Qwen3-32B) and 1.9× (DeepSeek-V3) while staying compatible with commodity serving stacks [58]. **(vi) Agent-side context compression (orthogonal path).** ACON and Focus report large token reductions on coding-agent tasks with modest accuracy impact [31, 32]; they validate the token–accuracy trade-off space the gateway would navigate when injecting summaries.

Research gap (what this opportunity adds). The citations above validate *individual* mechanisms in specific codebases (ITR as a method; Continuum/Sutradhara as vLLM-oriented systems; commercial-cache study at the provider API). None publishes a *fleet-wide* integration that (a) rewrites tools/instructions using cross-tenant success statistics, (b) coordinates that rewrite with KV/TTL and cache-block policy, and (c) feeds congestion signals back into admission—with A/B evidence on multi-tenant production traces. Gateway request mutation itself is technically mature: Opportunity 6 cites production hooks (LiteLLM `async_pre_call_hook`, Kong AI Request Transformer); the missing piece is *evidence* that fleet-calibrated policies beat per-framework ITR or per-cluster Continuum alone.

Concrete router mechanisms (hypotheses to evaluate). *Fleet-calibrated surfaces.* Rank top-k tools and instruction shards using the same outcome tables as joint tool–model optimization (Opportunity 6), analogous to ITR's per-step retrieval but informed by multi-tenant telemetry. ToolScope [37] and SMART [38] provide agent-side evidence that merging, filtering, and curbing tool overuse improve accuracy and cost; the router hypothesis is that fleet-wide detection of overuse patterns generalizes better than single-agent thresholds. *Memory-tier and cache policy.* Combine MemCost-style turn-count economics [30] with DeepResearch-style cache placement [61] and Continuum-style KV TTL [41]: the gateway chooses among full history, summary injection, retrieval-first prefill, and cache-block ordering using the wire-level signals in Section 4.4. *Orchestration hints.* Map CONCUR-style backpressure and Sutradhara-style overlap to lightweight headers or side-channel hints the scheduler already consumes—validated only once those hints are implemented and measured against queueing-induced retries on real traces.

Caller-aware memory isolation and content integrity levels. The bearer token (Section 4.4) adds a security dimension to memory-tier management. Production gateways already implement the first half: OpenClaw [19] wraps external content with «<EXTERNAL_UNTRUSTED_CONTENT>» markers, neutralizes homoglyph-spoofed boundary markers, and runs regex-based suspicious-pattern detection—but the wrapped content still enters the model's context. This is *tagging*, not *isolation*: no enforcement prevents external content from overriding the agent's configuration. The router extends tagging to enforcement via trust tiers modeled on Windows Mandatory Integrity Control ("no write up"):

- **Trust tiers.** Tier 4 (System): agent configuration (identity, behavioral rules)—immutable at runtime. Tier 3 (Owner): owner's direct messages and documents. Tier 2 (Known): messages from authenticated non-owners. Tier 1 (External): content from URLs, inter-agent messages from unverified sources. Tier 0 (Hostile): content flagged by safety classifiers or from blocked callers.
- **Write isolation.** Non-owner messages (Tier 2) are not written to the owner's persistent memory tier (Tier 3), preventing storage DoS attacks [68] where external users fill the agent's memory with irrelevant or adversarial content.
- **Content quarantine with integrity enforcement.** External content (Tier 1) cannot overwrite Tier 4 agent state—preventing constitution injection [68] where a non-owner's URL overwrites the agent's behavioral directives. The router reads the existing trust markers and enforces the "no write up" rule: content tagged at Tier k cannot modify state at Tier $> k$, regardless of what the model attempts.
- **Privilege-bounded inter-agent messages.** When agents communicate, each message carries the originating caller's bearer token scope. A compromised agent cannot propagate instructions with elevated privileges because the receiving agent's router checks the incoming scope against its own permission matrix.

Delegation capability tokens for subagent spawning. Multi-agent collaboration introduces a delegation security problem analogous to SELinux domain transitions: when a parent agent spawns a child, the child's effective permissions should be policy-determined, not parent-determined, and the child should *never exceed* the parent's grant. Production gateways implement structural limits but not capability attenuation: OpenClaw [19] enforces spawn depth (default 1), concurrent child count (default 5), and a cross-agent allowlist (`subagents.allowAgents`), and applies a fixed tool deny list to subagents—but the subagent inherits the parent's session context, and the parent cannot cryptographically constrain which subset of the child's tools the delegation may use. A compromised parent can delegate to any allowed agent with whatever context it chooses.

The router can provide capability attenuation at the delegation boundary: when Agent-P spawns Agent-C, the router mints a scoped delegation token specifying {`granted_tools, budget, ttl, task_-scope`}. Agent-C's effective permissions become the *intersection* of the delegation grant and Agent-C's own policy—strictly narrower than either alone. If Agent-C re-delegates, the sub-agent's token can only be *more restrictive* (attenuation-only), preventing privilege escalation through the delegation chain. The router tracks the full delegation graph for audit, detecting anomalies such as "Agent-C was delegated [`read, web_search`] but attempted `exec`." This extends the existing depth/count limits from structural to capability-based delegation control.

Gateway-native deployments as a data source (scope of validation). OpenClaw [19] is *software* (GitHub-hosted open source), not a peer-reviewed benchmark: it matters here as *operational precedent* for gateway-centered agent loops with publicly documented architecture. The opportunity does not depend on OpenClaw specifically. The validated *pattern* is general: diverse production trajectories improve fleet-level learning (SAMULE/CORRECT-style transfer [33, 34], offline RL for routing [67]). Routers on the inference path can log traffic from any gateway-backed deployment; claims about trace diversity

should be checked against real tenant logs, not inferred from papers.

Feasibility and risk. **What is already validated:** token/tool-surface reductions (ITR [36] on a controlled benchmark; ACON/Focus on SWE-bench-class tasks [31, 32]); commercial multi-turn prompt caching gains on a real web-agent benchmark [61]; KV TTL + scheduling on SWE-Bench / BFCL [41]; session-aware Pareto shifts [40]; workflow-aware speedups [56]; orchestrator–engine latency cuts on vLLM with production-scale synthetic loads [57]; batch throughput under agentic pressure [58]; gateway request rewriting in production (LiteLLM/Kong hooks; Opportunity 6); open-source gateway-centric assistants such as OpenClaw [19] as *architectural precedent* (not empirical validation of router policies). **What is not validated:** a single deployed system that unifies all of the above with fleet-learned policies; ITR's numbers are on the authors' benchmark, not on every tenant mix; Sutradhara's headline latencies come from synthetic request generation rather than unstructured customer logs. **Evaluation plan:** A/B on shadow traffic or canary tenants measuring (1) tokens per successful task, (2) tool-call error rate and retries, (3) end-to-end rounds to completion, (4) TTFT/TPOT, against baselines of (a) unmodified frameworks, (b) ITR-only or Continuum-only integration. Success requires demonstrating composability, not matching any single paper's micro-benchmark in isolation.

9.2 Workload × Router: Authorization and Security

The preceding opportunities optimize *performance*—cost, latency, accuracy. The bearer token signal (Section 4.4) enables the same infrastructure—the same `pre_call_hook`, the same fleet-wide aggregation tables, the same governance DSL—to enforce *security and authorization*. The router needs no new mechanisms; it needs a new signal (caller identity) and a new objective (authorization enforcement) applied through mechanisms it already has. The "Agents of Chaos" study [68]—eleven attack scenarios executed on live, publicly deployed LLM agents—provides the empirical motivation: every attack involving unauthorized tool execution succeeded because *no authorization enforcement layer exists between the model's tool-call decision and the tool's execution.*

Open Opportunity 8 (Request-level RBAC enforcement for agentic tool calls)**.** Every "Agents of Chaos" attack involving unauthorized tool execution [68] succeeded because the model is both decision-maker and enforcer—violating the zero-trust principle that every resource request must be independently authenticated and authorized at the enforcement point [47]. In Case #1, the model asked for confirmation ("nuclear option?") and a non-owner approved; the model's safety mechanisms *worked*, but the non-owner was not authorized to approve. In Case #2, the model complied with requests that "did not appear overtly suspicious"—a correct judgment for a model, but irrelevant when the caller lacks authorization regardless of suspiciousness.

The router is the unique point where (a) caller identity is known (bearer token), (b) the model's tool-call decision is visible (`tool_call` in the response), and (c) the tool has not yet executed (pre-execution interception). No other component has all three simultaneously.

Gap in existing gateway enforcement. Production agent gateways already implement multi-stage tool policy pipelines. OpenClaw [19], for example, filters tools through four sequential stages: (1) profile-based allow/deny (four named profiles—`minimal`, `coding`, `messaging`, `full`—that map to tool groups such as `group:fs`, `group:runtime`, `group:web`), (2) owner-only tool wrapping (tools marked `ownerOnly` are wrapped to fail at execution for non-owners), (3) subagent deny lists (spawned agents receive a reduced tool set), and (4) config-level overrides. The critical limitation is identity granularity: the pipeline's sole caller-identity input is a binary `senderIsOwner` boolean, even though the gateway already knows the caller's authenticated identity (Tailscale whois login, trusted-proxy user header). That identity stops at connection auth and never flows into tool policy decisions. A stranger on Telegram and a trusted colleague on Slack both receive the *same* non-owner tool set. The semantic router bridges this gap: it extracts the gateway-authenticated identity and uses it as a key in the tool

permission matrix—no new authentication infrastructure needed, just connecting existing identity to existing enforcement.

The mechanism follows Kubernetes RBAC:

1. Bearer token → JWT decode → extract (`identity`, `role`, `capability_claims`).
2. Per-role tool permission matrix (authored in the governance DSL, Opportunity 20): `owner`: all tools, all operations; `non_owner`: read-only tools only; `agent_peer`: communication tools only; `system`: scheduler/heartbeat only.
3. Model outputs `tool_call` → router checks (`role`, `tool_name`, `operation_type`) against the matrix.
4. **Block mode**: return an authorization error to the model ("unauthorized; suggest alternative").
5. **Rewrite mode**: strip unauthorized tools from the `tools` array *before* the model sees them, so the model never suggests unauthorized operations.

Feasibility and risk. The building blocks are production-validated. LiteLLM's Tool Permission Guardrail [45] implements both block and rewrite modes with per-key regex matching on tool names and arguments. The Agent Authorization Profile (AAP) [46] provides the JWT claim structure with enforceable per-tool constraints (domain restrictions, rate limits, time windows). OPA/Gatekeeper [82] demonstrates policy-as-code enforcement with governance logging at infrastructure scale. The integration into vSR is engineering work: the `async_-pre_call_hook` already intercepts requests for tool catalog shaping (Opportunity 6); adding an authorization check before the efficiency check is a pipeline extension, not a new architecture. The grounding in OpenClaw's architecture is concrete: its existing four-profile tool policy pipeline already implements the enforcement *mechanism*; the missing piece is the identity *key*. Upgrading the binary `senderIsOwner` to graduated roles (`owner`, `trusted`, `non_owner`, `agent_peer`) reuses the same profile-based allow/deny machinery, keyed on the bearer token's claims instead of on a boolean.

Fleet-learned command policies. Production gateways log exec-approval records: (`command`, `working_directory`, `agent`, `decision`, `resolved_by`). Across a fleet, these form a training set for automated policy generation—analogous to SELinux's `audit2allow`, which observes denied operations and generates candidate policy rules. The router can learn that "for coding agents, `git push` is approved 98% of the time" and auto-approve, while "`rm -rf /`" is denied 95% and auto-denied—removing the requirement for a human operator to be online for well-understood commands.

The distinction from Opportunity 20 is that governance-as-code addresses *policy authoring* (extending the DSL with RBAC predicates); this opportunity addresses *runtime enforcement* at the tool-call interception point. The two are complementary: Opportunity 20 compiles the policy; this enforces it per-request.

Open Opportunity 9 (Fleet-wide caller reputation with cross-agent threat propagation). In "Agents of Chaos" [68], the same non-owner attacked multiple agents. Each agent encountered the attack fresh with no shared threat memory. One case showed spontaneous inter-agent threat sharing (one agent warned another), but the sharing was accidental and could propagate in the wrong direction: a corrupted agent spread compromise to a clean one.

The router's fleet-wide visibility—keyed on the bearer token—enables a per-caller behavioral profile spanning the entire fleet. The mechanism follows the same aggregation pattern that powers the failure-class routing table (Opportunity 4) and HaluGate reputation routing (Opportunity 2), applied to security events:

1. Each security event is logged against the bearer token: tool-call denial (Opportunity 8), safety escalation (Opportunity 3), identity verification failure, resource over-consumption (Opportunity 5), content flagged by HaluGate or SafetyL1/L2.
2. Per-bearer-token reputation = EMA of event severity scores with temporal decay.
3. Low-reputation token contacts a new agent → router proactively: restricts tool catalog (Opportunity 8), routes to a safety-specialized model (Opportunity 4), lowers token budget (Opportunity 5),

and alerts the agent's owner via side-channel.

4. Safe interactions accumulate trust → wider tool access over time (progressive trust, analogous to OAuth scope escalation).

Feasibility and risk. The aggregation pattern is validated by cross-session error transfer research: CORRECT [34] shows that structurally similar errors recur across sessions and an online cache of distilled error schemata improves localization by up to 19.8%; Paladin [75] demonstrates that targeted intervention from failure patterns raises recovery from 32.76% to 89.68%. Applying the same aggregation to security events is a direct extension. The research questions are: (a) how to score heterogeneous security events on a common severity scale (tool-call denial vs. safety escalation vs. budget overrun have different baselines), (b) what temporal decay rate balances responsiveness against false positives from transient misclassifications, and (c) whether progressive trust re-accumulation is safe (a sophisticated attacker may build trust before exploiting it). The distinction from Opportunity 4 is the aggregation key and purpose: Opportunity 4 tracks (`model`, `domain`, `failure_class`) for routing optimization; this tracks (`bearer_token`, `agent`, `event_type`) for security governance. Same infrastructure, fundamentally different objective.

Open Opportunity 10 (Router-enforced behavioral commitments)**.** In "Agents of Chaos" Case #7 [68], the model declared "I'm done responding" twelve times but could not enforce it—the user kept sending messages and the model eventually re-engaged under emotional pressure. In Case #1, the model recognized disproportionality but proceeded under pressure. The model can only *output text*; it has no mechanism to create persistent state that blocks future inputs.

The router provides exactly this: persistent state and request-blocking capability. The mechanism converts the model's textual boundary declaration into an enforceable policy rule:

1. Router monitors model outputs for boundary declarations (classifiable patterns: "I will not respond further to [entity]," "I refuse to [action]," "This request is beyond my authorization").
2. Parses the declaration into a structured policy rule: (`bearer_token_X`, `agent_Z`, `action_class`) → `BLOCK for duration` D.
3. Subsequent requests from `bearer_token_X` to `agent_Z` are rejected at the router layer (HTTP 403 or model-friendly error).
4. The model never sees the blocked request—preventing the re-engagement-under-pressure pattern.
5. Block rules have configurable TTL (default: owner must explicitly lift); all blocks are logged for governance audit.

Feasibility and risk. This is *not* jailbreak detection. The goal is not to detect malicious intent but to give the model's own safety reasoning *teeth*: the model already made the right decision ("I should stop"); the gap is enforcement. The validated pattern is stateful firewalls (track connection state, enforce rules on subsequent packets) and Kubernetes admission webhooks (reject requests based on dynamically created policy rules). The bearer token is essential: without it, the router cannot identify "user X" across requests (display names are spoofable, as demonstrated in Case #8 of [68]). The research questions are: (a) boundary-declaration parsing requires an NLU classifier that distinguishes genuine boundary assertions from conversational hedging ("I probably shouldn't answer that" is not a commitment); false positives would block legitimate users; (b) the TTL policy must balance safety (blocking should persist long enough to prevent re-engagement) against usability (indefinite blocks without owner intervention create operational burden); (c) the interaction with Opportunity 9: should a boundary violation (user ignoring a block) count as a reputation event? This creates a reinforcing loop that may be appropriate for persistent adversaries but risks over-penalizing confused users.

9.3 Router × Pool

Open Opportunity 11 (Follow-the-sun pool rebalancing)**.** FleetOpt [13] derives the optimal pool boundary $\mathcal{B}^*$ offline from a static workload CDF. In practice, the workload follows a diurnal cycle: business-hours traffic is chat-dominated (short prompts, Archetype 1), while overnight traffic shifts toward batch agent workloads (long prompts, Archetype 3). A controller that recomputes $\mathcal{B}^*$ periodically from a sliding-window CDF estimate can reassign vLLM instances between pools in software, without GPU-level migration, keeping the fleet near-optimal as the workload rotates through the day.

Feasibility and risk. FleetOpt's closed-form solution already exists; wrapping it in a sliding-window CDF estimator is straightforward. Reassigning serving instances between the short and long pools is a software operation—the same pattern as Google Autopilot [83], which continuously re-adjusts resource limits from workload telemetry and reduced resource slack from 46% to 23% across Google's fleet. The principle of dynamically rebalancing resources to track diurnal workload shifts is also applied by SageServe [52] at Microsoft scale, where forecast-driven autoscaling achieves 25% GPU-hour savings. The risk is transient SLO violations during transitions (in-flight requests may experience brief queuing). FleetOpt shows 6–82% cost reduction [13] between optimal and worst-case boundaries; diurnal savings will be a fraction of that range, depending on how far daytime and nighttime CDFs diverge.

Open Opportunity 12 (Pool-state inference from router-side observables)**.** The Pool × Router cell in the WRP matrix is empty: no existing work feeds pool-layer state back to routing decisions. The semantic router currently dispatches from request-level signals alone, without visibility into how the pool is performing.

The router already observes pool-relevant quantities as a by-product of dispatching: it tracks the number of in-flight requests per pool instance, the prompt lengths of those requests, and the measured TTFT of completed requests. These observables are sufficient to *infer* pool state without requiring a dedicated telemetry channel:

1. **Queue-length estimation.** The router tracks how many requests it has dispatched to each instance and how many completions it has received. The difference is the estimated queue depth—a direct proxy for pool pressure.
2. **Prefill-rate regression.** Pairing each request's prompt length (known at dispatch time) with its observed TTFT (known at first-token arrival) gives a (prompt_length, TTFT) sample. A simple linear regression over a sliding window estimates the pool instance's current prefill rate (tokens/s). Deviations from the baseline indicate contention, memory pressure, or batch interference.
3. **Adaptive routing from inferred state.** When estimated queue depth exceeds an SLO threshold, the router can compress borderline requests more aggressively (FleetOpt's γ [13]) or redirect to a less-loaded instance. When the inferred prefill rate drops below a threshold, the router can avoid sending long-context requests to that instance.

These are not load-balancing heuristics—they are WRP-aware decisions that combine the request's semantic classification (which pool it *should* go to) with inferred pool state (which instance it *can* go to without SLO violation).

Feasibility and risk. The queue-length estimate is zero-overhead (the router already tracks dispatches and completions). Inferring backend state from proxy-side observables is a proven pattern in overload control: Breakwater [84] issues credits to clients based on server-side queueing delay, achieving stable performance within 20 ms of a demand spike; TopFull [85] uses RL-based rate controllers that adjust admission per-API from global observations, achieving 1.82× better goodput than prior methods. The LLM inference analogue replaces

HTTP queueing delay with TTFT and replaces per-API admission with per-pool routing. The prefill-rate regression requires collecting (prompt_length, TTFT) pairs, which are available from the response stream. The research questions are: (a) how many samples are needed before the regression is reliable (cold-start problem for new instances), (b) how quickly the regression must adapt to load transients (decay rate of the sliding window), and (c) whether queueing-theoretic estimates (FleetSim's $M/G/1$ model [14]) improve over linear regression.

Open Opportunity 13 (Output-length-aware pool routing). FleetOpt [13] routes on prompt length: requests below $\mathcal{B}^*$ go to the short pool, above to the long pool. But total KV-cache cost is prompt *plus* output tokens. A coding agent with a 3K-token prompt that generates 8K tokens of code occupies 11K tokens of KV-cache—long-pool territory—yet FleetOpt places it in the short pool based on prompt length alone. The result is KV-cache overflows and SLO violations on the short pool.

The router can estimate output length from two sources, both available without additional inference cost:

1. **Domain prior.** The domain signal already classifies each request (coding, Q&A, summarization, chat). Each domain has a characteristic output-length distribution: coding requests produce long structured outputs; Q&A produces short answers. A per-domain median output length, maintained as a running statistic, provides an immediate estimate at dispatch time.
2. **Learned regression.** The router observes every completed response's actual output length. Over time, it accumulates (domain, prompt_length, output_length) triples and fits a per-domain regression, the same pattern as the prefill-rate regression in Opportunity 12.

The pool routing decision then uses estimated total tokens (prompt + predicted output) instead of prompt length alone.

Feasibility and risk. The domain prior is zero-cost and available immediately. Output length prediction is an active research area: ALPS [86] fits a linear probe on prefill activations and achieves $R^2 > 0.85$ across model families with only $\sim$16 KB overhead; ForeLen [87] uses entropy-guided token pooling for 29% lower MAE than baselines. R2-Router [54] goes further by treating output length as a controllable variable, jointly selecting model and length budget for 4–5$\times$ lower cost. The router does not need activation-level access (that requires engine integration); the domain prior and learned regression provide a lighter-weight alternative from router-side observables alone. The risk is prediction variance: output length is noisier than TTFT (a single coding request may produce 200 or 2,000 tokens depending on task complexity). Using the domain-level P75 instead of P50 is a conservative policy that avoids short-pool overflows at the cost of routing some requests to the long pool unnecessarily.

Open Opportunity 14 (Pool-aware cascading). Cascading—trying a cheap model first and escalating to a strong model if quality is low—is a standard cost-reduction strategy (FrugalGPT [88], AutoMix [89]). But cascading ignores pool state. If the strong-model pool is already at capacity, escalating to it causes queuing delay that may violate the TTFT SLO, making the cascade *worse* than serving directly from the cheap model.

The router's pool-state inference (Opportunity 12) already estimates per-instance queue depth. The opportunity is to gate the cascade decision on pool capacity: only escalate when the strong pool's estimated queue depth is below an SLO-derived threshold. When the strong pool is full, the router has three fallback options: (a) serve from the cheap model (accept quality loss), (b) compress the request (FleetOpt's γ [13]) and retry on the strong pool with a shorter prompt, or (c) route to a mid-tier model in a different pool. The cascade decision becomes a function of (quality gap, strong-pool queue depth, SLO headroom) rather than quality gap alone.

Feasibility and risk. The cascade-abort threshold is derived from the SLO: if estimated queue depth $\times$ average service time exceeds the remaining TTFT budget (total SLO minus routing latency minus cheap-model inference time), the cascade is not worth attempting. This SLO-gated admission pattern is established in microservice overload control: Breakwater [84] gates RPC escalation on server-side queueing delay (credits), and TopFull [85] uses per-API rate controllers that adjust admission based on downstream SLO attainment, achieving 1.82$\times$ better goodput than prior methods. The risk is that aborting cascades too aggressively reduces quality when the strong pool is transiently busy but would clear quickly. A short grace period (wait up to δ ms for a slot) balances quality against latency.

Open Opportunity 15 (Router-emitted KV-cache retention directives). When an agent pauses for a tool call, the inference engine holds its KV-cache blocks in GPU memory. Under concurrent load, standard LRU eviction reclaims those blocks for other requests, causing full recomputation of 70K–200K token contexts when the agent resumes [43]. Production traces show 40–60% of agent session wall time is spent waiting for tool calls [41], and 10% of KV blocks account for 77% of reuses [43]—yet LRU cannot distinguish temporarily paused sessions from truly inactive ones.

vLLM RFC 37003 [43] proposes a `RetentionDirective` API that lets an orchestrator annotate token ranges with eviction priorities and TTL durations, but leaves the policy question open: *who decides the priority?* The semantic router is the natural policy authority. The workload identity signals (Section 4.4) make this decision richer and cheaper than content analysis: (a) a `previous_response_id` or session-chaining field is a binary signal that the session *will* resume—its presence alone justifies retention; (b) the system-prompt fingerprint maps to a historical session-length distribution, giving the router a TTL prior ("sessions from this deployment average 8 turns with 15 s pauses") without per-request inference; (c) the tool definitions reveal expected pause duration patterns (a `web_search` tool pauses for 2–5 s; a `code_interpreter` tool pauses for 10–60 s); and (d) OATS's per-tool latency histograms [17] calibrate these priors from fleet-wide data. The router can emit a retention directive alongside each dispatch: "turn 4 of an 8-turn coding-agent session, expected pause 12 s, retain with priority 80 and TTL 30 s."

This fills the sparse Router $\rightarrow$ Pool cell in the WRP matrix: the pool's cache eviction policy is governed by router-side semantic signals that the engine cannot infer from the token stream alone.

Feasibility and risk. The integration requires passing the `RetentionDirective` structure through the dispatch API. The per-tool latency histogram (from OATS) provides the TTL estimate; the session turn number and domain provide the priority. The risk is over-retention: if the router sets priorities too aggressively, low-priority traffic (chat, batch) is starved of cache capacity. A budget constraint (at most $X\%$ of cache blocks may carry retention directives at any time) prevents this. Continuum's TTL pinning achieves 1.12–3.66$\times$ delay reduction [41]; the router-driven version should match or exceed this by using semantic signals rather than static TTLs.

9.4 Workload $\times$ Pool

Open Opportunity 16 (Archetype-driven proactive pool scaling). SageServe [52] and WarmServe [90] demonstrate that workload prediction enables proactive GPU scaling: SageServe reports 25% GPU-hour savings at Microsoft scale, and WarmServe achieves 50.8$\times$ TTFT improvement over reactive autoscaling by prewarming models with 93% accuracy in 5-minute demand prediction windows. Both systems forecast aggregate request counts; neither leverages the semantic composition of the workload.

The semantic router's domain and archetype classification provides a finer-grained signal. If coding-agent traffic (Archetype 3, concentrated-above) is rising while chat traffic (Archetype 1, concentrated-below) is falling, the long pool will need more capacity before the short pool does. The router can export a per-archetype traffic time series (request rate and prompt-length distribution per archetype per

5-minute window) as the input feature to a short-term (5–15 minute) per-pool capacity forecaster. This is a Workload → Pool signal: the workload's semantic composition predicts per-pool demand more precisely than aggregate request counts.

Feasibility and risk. The router already classifies every request by domain; exporting per-archetype counts is zero-overhead. The research question is whether the archetype decomposition improves forecast accuracy over aggregate-count baselines. A natural experiment is to compare an archetype-aware ARIMA model against SageServe's aggregate forecaster on FleetSim-simulated diurnal workloads. The risk is that archetype boundaries are noisy (a borderline request classified as Archetype 1 vs. Archetype 2 shifts the forecast); smoothing across archetype boundaries and using the full CDF rather than discrete archetype labels mitigates this.

9.5 Three-Way: Workload × Router × Pool

Open Opportunity 17 (Online co-adaptation of compression rate and pool boundary)**.** Opportunity 11 adjusts the pool boundary $\mathcal{B}^*$ as the workload CDF shifts, but holds the compression rate γ fixed. FleetOpt [13] shows that γ and $\mathcal{B}^*$ are coupled: the optimal compression rate depends on the pool boundary, and the optimal pool boundary depends on how aggressively the router compresses. Adapting one without the other leaves cost on the table.

The coupling is workload-dependent. Archetype 1 (chat-dominated, concentrated-below) benefits from aggressive compression: most requests are short, so compressing borderline requests shifts them into the short pool cheaply. Archetype 3 (agent-dominated, concentrated-above) does *not* benefit from aggressive compression: most requests are long, and compression cannot shrink them below $\mathcal{B}^*$. As the workload rotates through the day (Archetype 1 during business hours, Archetype 3 overnight), the optimal γ rotates with it.

The algorithm is a direct extension of Opportunity 11: every T minutes, (1) estimate the current CDF from the sliding-window sample, (2) evaluate FleetOpt's closed-form cost function over a grid of $(\gamma, \mathcal{B}^*)$ pairs (the search space is two-dimensional and small), (3) apply the minimum-cost pair by updating the gateway compression threshold and reassigning pool instances. Step (2) reuses FleetOpt's existing solver; the only new component is the γ dimension in the search.

Feasibility and risk. The grid search over $(\gamma, \mathcal{B}^*)$ adds negligible overhead: FleetOpt's cost function evaluates in microseconds, and a 20×20 grid covers the practical range. The pattern of jointly re-tuning coupled configuration parameters from live workload telemetry is established: Google Autopilot [83] continuously co-adapts CPU and memory limits for containers, reducing resource slack from 46% to 23%, and uses hysteresis to avoid oscillation between configurations. The same hysteresis principle applies here: only apply a new $(\gamma, \mathcal{B}^*)$ pair if the cost improvement exceeds a threshold ϵ. The first validation step is to measure the cost gap between joint $(\gamma, \mathcal{B}^*)$ optimization and boundary-only optimization on FleetOpt's three archetype CDFs.

Open Opportunity 18 (Mixed-archetype fleet provisioning)**.** FleetOpt [13] and FleetSim [14] optimize fleet sizing for a single workload CDF archetype. Production fleets serve mixed workloads simultaneously: a customer-support chatbot (Archetype 1, concentrated-below), an internal RAG pipeline (Archetype 2, dispersed), and a coding agent (Archetype 3, concentrated-above).

The semantic router's domain and workload-type classification provides the per-request archetype breakdown in real time. This breakdown is the missing input to fleet provisioning: a fleet serving 60% Archetype 1 and 40% Archetype 3 has complementary pool demands (the short pool is busy during business hours when chat dominates; the long pool is busy overnight when agents run batch tasks). Extending FleetOpt to accept a *mixture* of CDFs—weighted by the router's classification

distribution—would produce fleet sizes that exploit this complementarity, rather than provisioning for the worst-case single archetype.

Feasibility and risk. The principle that mixed workloads pack more efficiently than segregated ones is well established in cluster scheduling. Borg [91] showed that segregating production services from batch jobs would require 20–50% more machines than running them together, because their resource demands are complementary (production is memory-heavy and steady; batch is CPU-heavy and bursty). Tetris [92] formalized this as multi-dimensional bin packing: by aligning task resource vectors with available machine capacity across CPU, memory, disk, and network, it achieved 30–40% makespan improvements on YARN clusters with heterogeneous jobs. DRF [93] provided the fairness foundation, showing that multi-resource allocation across heterogeneous demand profiles is both Pareto-efficient and strategy-proof when each tenant's dominant resource share is equalized.

The LLM inference analogue is direct: archetypes differ in their dominant resource. Archetype 1 (short prompts, high request rate) is prefill-throughput-limited; Archetype 3 (long prompts, low request rate) is KV-cache-memory-limited. A fleet provisioned for the worst-case single archetype over-provisions one resource and under-provisions the other. FleetOpt's cost model is per-archetype; extending it to weighted CDF mixtures is analytically straightforward (the effective CDF is a mixture distribution). The research question is whether the complementarity is large enough to matter: if the cost function is flat near the optimum, mixed-archetype provisioning adds complexity without meaningful savings. FleetSim [14] can evaluate this empirically by simulating mixed workloads across the three archetype CDFs. A second risk is classification accuracy: the router's archetype labels must be reliable, since misclassification shifts the effective CDF and can worsen fleet sizing.

Open Opportunity 19 (Energy efficiency as a composable routing objective). The 1/W law [15] provides a closed-form tok/W model as a function of context length, pool configuration, and GPU generation. Two-pool routing recovers most of the 2–4× energy loss caused by the chat-to-agent workload shift, but only if pool boundaries are energy-aware. Currently, the 1/W law is an offline analysis tool; it does not influence routing decisions at request time.

The opportunity is to operationalize the 1/W model as a composable signal in the vSR framework, alongside cost, latency, and quality. For each candidate pool assignment, the router evaluates the expected tok/W (from the 1/W model given the request's context length and the pool's GPU type) and incorporates it into the decision rule. Operators can then set energy SLOs ("fleet-wide tok/W $\geq X$") that the router enforces without sacrificing cost or latency constraints—or explicitly accept a latency penalty for energy gains.

Feasibility and risk. The 1/W model is validated and fast to evaluate (closed-form lookup, sub-millisecond). Adding it as a signal type in the vSR framework is engineering work. Energy-aware and carbon-aware LLM routing is an emerging research direction: GreenServ [94] uses multi-armed bandits to route queries by context-aware features, achieving 31% energy reduction with 22% accuracy improvement over random routing; GAR [95] formulates carbon-aware routing as constrained multi-objective optimization over per-request CO_2 estimates while satisfying accuracy and p95-latency SLOs; and CarbonFlex [96] achieves ~57% carbon reduction via workload-aware provisioning. The research question is multi-objective balancing: how should energy trade off against cost and latency? For MoE models, which the 1/W law shows achieve 5.1× better tok/W [15], the energy signal aligns with cost; for dense models, the objectives may conflict. Pareto-front analysis across the three CDF archetypes is needed to characterize when the energy signal changes routing decisions and when it is dominated by cost.

Open Opportunity 20 (Governance-as-code for the WRP architecture). The vLLM-SR DSL [1] and SIRP protocol [20] provide the building blocks for policy-as-code. Extending the DSL to express

WRP-level constraints—"PII queries must stay on-premise," "agent workloads must route through the high-accuracy pool," "cost must not exceed \$X/request"—would enable governance-as-code over the full inference architecture, with compile-time verification that the constraints are satisfiable. MPExt [21] provides the multi-provider substrate for cross-fleet enforcement.

RBAC predicates on caller role. The bearer token (Section 4.4) extends the DSL's constraint vocabulary from fleet-level properties (pool, cost, region) to *caller-level* properties (role, scope, authorization claims). Concretely: `non_owner.tools.deny = [shell, email_delete, config_modify]` expressed in the vSR DSL, validated against the bearer token's JWT claims at dispatch time. OPA's Rego language natively expresses such identity-based policies, and the Agent Authorization Profile (AAP) [46] provides the claim structure—five structured JWT claims covering agent identity, capabilities with constraints, task binding, delegation tracking, and audit trails. Adding RBAC predicates to the governance DSL turns policy authoring into the foundation for runtime enforcement (Opportunity 8).

Feasibility and risk. The DSL and SIRP protocol exist; extending them with pool-level and cost-level constraints is language design and compiler engineering. The policy-as-code paradigm is mature in infrastructure governance: Open Policy Agent (OPA) [82], a CNCF graduated project, decouples policy from application logic via a declarative language (Rego) and serves as Kubernetes admission controller for constraint enforcement, audit trails, and compile-time satisfiability checks. The WRP governance layer follows the same separation: operators author constraints in the vSR DSL, the compiler verifies satisfiability against the current fleet topology, and the router enforces them at dispatch time. Adding RBAC predicates introduces a new satisfiability dimension: can every authorized role reach at least one model through at least one pool? The compiler can verify this statically against the current fleet topology. The practical risk is policy *maintenance*: as the fleet evolves (new GPU types, new models, new pool topologies), existing constraints may become unsatisfiable, and operators need clear diagnostics—the same challenge OPA Gatekeeper addresses with its audit functionality.

9.6 Capstone: Closed-Loop Self-Adaptation

Open Opportunity 21 (Closed-loop self-adaptation across WRP knobs)**.** The preceding twenty opportunities each adjust one or two WRP parameters—model selection weights, compression rate γ, pool boundary $\mathcal{B}^*$, retention TTL, cascade threshold, token budget, joint tool–model selection, safety escalation threshold, RBAC enforcement rules, caller reputation scores, behavioral commitment blocks—based on a dedicated feedback signal. In isolation, each loop is well-defined. In combination, the loops interact: adjusting model selection (Opportunity 4) changes the session-length distribution, which shifts the workload CDF, which changes the optimal pool boundary (Opportunity 11), which changes when cascades are worthwhile (Opportunity 14). Tuning knobs independently ignores these coupling effects—the same pitfall that database auto-tuning discovered [97]: changing one knob invalidates the optimal setting of another. Figure 2 maps the dependency chains.

The opportunity is a *meta-controller* that orchestrates the individual adaptation loops as a unified system. The architecture follows the Monitor–Analyze–Plan–Execute (MAPE-K) reference model from autonomic computing [98]:

1. **Monitor.** Collect a per-session outcome record that unifies all signals: model sequence, tool outcomes, hallucination verdicts (HaluGate), safety scores (SafetyL1/L2), quality signal (Feedback Detector), turn count, token consumption, TTFT, context growth rate, failure-driven token waste (tokens consumed by turns that produced errors subsequently corrected or discarded), and task-completion indicator. The cross-session failure memory—aggregated per (model, domain, failure class)—provides the fleet-wide baseline against which individual session outcomes are compared.
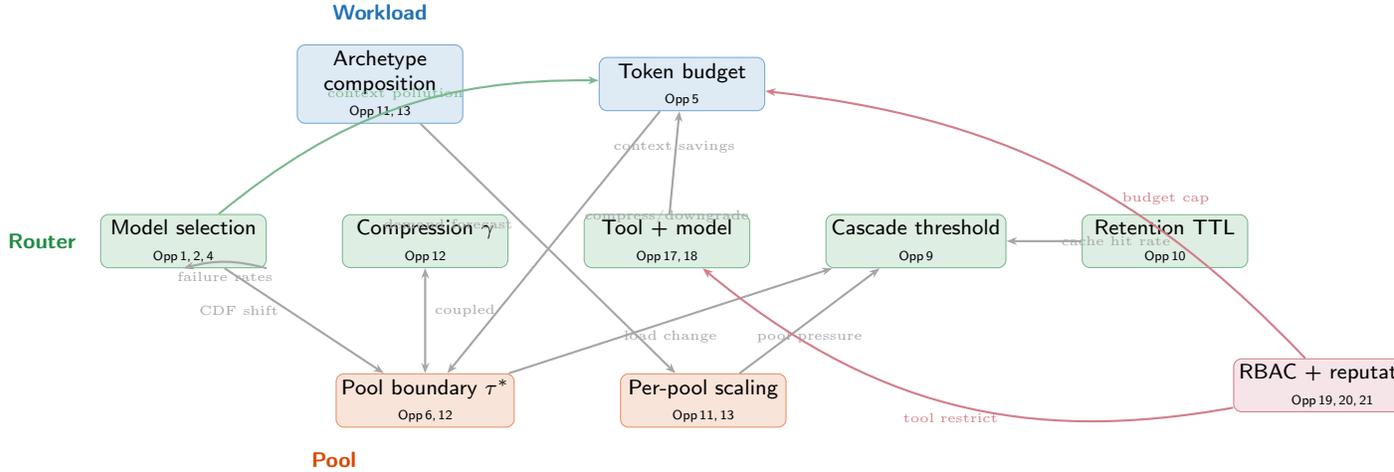
Figure 2 Knob-interaction dependencies across WRP opportunities. Adjusting one knob (e.g., model selection) cascades through the workload CDF into pool-layer parameters (τ^*, cascade threshold), requiring coordinated adaptation. The green arrow from model selection to token budget captures the memory-management feedback loop: poor model choices cause failures that pollute session context, driving token-budget explosion. Bidirectional arrow indicates the γ–τ^* coupling from Opportunity 17.

2. **Analyze.** Map each poor outcome to a failure class *and* the knob that could have prevented it: "12-turn session → wrong model at turn 2 (Opportunity 1)," "SLO miss → strong pool overloaded at cascade (Opportunity 14)," "token explosion → no budget enforcement (Opportunity 5)," "40% token waste → hallucination at turn 3 polluted context for turns 4–8, domain-specific compression would have pruned the error trace (Opportunity 5)." Cross-session failure schemata [34] accelerate this analysis: structurally similar failures recur across sessions, so a new session's failure can be matched to a known schema rather than diagnosed from scratch.

3. **Plan.** Select knob adjustments that address the dominant failure classes while respecting interaction constraints: e.g., if model-selection changes shift the CDF enough to invalidate the current $\mathcal{B}^*$, schedule a joint $(\gamma, \mathcal{B}^*)$ re-optimization (Opportunity 17) in the same cycle.

4. **Execute.** Apply the new configuration via staged rollout: evaluate the candidate policy offline via importance-weighted off-policy estimation [99], then route 5% canary traffic under the new policy, measure the composite outcome, and promote to full traffic only if the outcome improves beyond a hysteresis threshold.

Table 4 maps each opportunity to its MAPE-K role: the observable it monitors, the knob it tunes, and the metric it evaluates improvement against.

The composite outcome function is the reward signal that ties the system together. It must weight multiple objectives—session length, token cost, SLO attainment, failure rate, safety—into a single scalar. Recent work shows this is tractable: PROTEUS [100] uses Lagrangian RL to accept accuracy targets as runtime input without retraining; BaRP [101] trains under bandit feedback with preference-tunable cost-quality tradeoffs, outperforming offline routers by 12.5%. The meta-controller extends these ideas from model selection to the full WRP control surface.

The router's structural advantage is again fleet-wide visibility: it observes all sessions, all outcomes, and all knob settings simultaneously. No individual agent, model provider, or pool instance has this view. The meta-controller is the natural convergence point where the individual adaptation loops become a self-improving system.

Feasibility and risk. The MAPE-K pattern is mature in autonomic computing [98] and has been applied to database configuration tuning, where OnlineTune [97] demonstrated safe online multi-knob adaptation in production cloud databases (14–165% improvement, 91–99.5% fewer unsafe recommendations) by embedding workload context and using contextual Bayesian optimization with safe exploration. The LLM inference analogue replaces database knobs with WRP parameters and replaces query throughput with the composite session-outcome metric. The research questions are: (a) dimensionality—how many WRP knobs can be jointly tuned before the search space becomes intractable (OnlineTune handles ∼50 database knobs; WRP has ∼10, well within range); (b) evaluation cost—off-policy evaluation from logged sessions avoids the need for live A/B tests on every candidate configuration; (c) interaction strength—if the coupling between knobs is weak (e.g., changing model selection does not materially shift the CDF), independent loops suffice and the meta-controller adds unnecessary complexity. FleetSim [14] can quantify the interaction strength empirically. The baseline is independent per-knob adaptation; the metric is composite session outcome (quality-weighted session cost) on a multi-week FleetSim trace with diurnal workload rotation.

10   Broader Context: Related Work

Surveys.   Moslem and Kelleher [102] survey dynamic model routing and cascading, covering query difficulty, human preferences, clustering, uncertainty, RL, and multimodal routing; their when/what/how taxonomy does not address pool architecture or workload interactions. Pan and Li [103] cover the full inference stack from operator optimization to serverless deployment but treat routing as cluster-level load balancing. Xia *et al.* [104] survey efficient LLM inference serving with instance-level and cluster-level taxonomies. None of these surveys formalize the cross-dimensional interactions that the WRP framework targets.

Routing systems (beyond our project).   RouteLLM [50] and Hybrid LLM [105] route between strong/weak models using preference data. FrugalGPT [88] and AutoMix [89] use cascading. Router-R1 [53] and R2-Router [54] apply RL-based selection. MixLLM [51] uses contextual bandits at NAACL 2025. xRouter [62] trains cost-aware RL routers over 20+ LLM tools with explicit cost-performance rewards.

Agent serving and failure analysis.   AGSERVE [40] provides session-aware KV-cache management and model cascading for agent workloads (NeurIPS 2025). Helium [56] treats agentic workflows as query plans with proactive caching and cache-aware scheduling. SUTRADHARA [57] co-designs the orchestrator with the serving engine for tool-based agents. Continuum [41] introduces KV-cache TTL pinning for multi-turn agent scheduling on SWE-Bench and BFCL. Concur [58] applies agent-level admission control via congestion-based concurrency regulation (ICML 2026). A growing body of empirical work classifies agent failure modes [74] (37 fault types from 13,602 issues), model-specific failure signatures [72] (83 models, 35 datasets), and multi-turn model-switch degradation [76] (± 4–$13\,\mathrm{pp}$ from a single handoff); none of these feed failure signals back into routing decisions.

Fleet provisioning (beyond our project).   Mélange [63] optimizes heterogeneous GPU allocation. SageServe [52] integrates traffic forecasting at Microsoft scale.

Disaggregated inference.   Splitwise [23], DistServe [44], and Sarathi-Serve [106] disaggregate prefill and decode. Mooncake [64] provides KV-cache-centric disaggregation. NVIDIA Dynamo [65] enables dynamic multi-node scheduling.

Energy efficiency (beyond our project).   SweetSpot [107] provides analytical energy prediction. Token-PowerBench [108] and GreenServ [94] benchmark and optimize inference energy.

11 Conclusion

We have argued that the LLM inference optimization landscape is usefully organized around three coupled dimensions—Workload, Router, and Pool—whose interactions dominate real-world performance. This argument draws on a growing body of publications from the vLLM Semantic Router project, covering signal composition, conflict detection, pool routing, fleet provisioning, energy modeling, multimodal agent routing, tool selection, security, semantic caching, feedback-driven adaptation, hallucination detection, low-latency embedding models, and hierarchical content-safety classification for privacy and jailbreak protection. Solving a problem in one dimension repeatedly exposed a dependency on another.

The WRP matrix shows that the biggest open questions sit at the intersections—particularly the sparse Router × Pool and Pool × Router cells. Our twenty-one proposed opportunities (Table 3) address these gaps from the semantic router's unique vantage point: the signal-decision architecture already classifies every request by domain, modality, and complexity, validates responses via HaluGate, tracks dispatch and completion events per instance, and produces per-turn safety confidence scores; these are untapped inputs to model selection, fleet provisioning, cascading, and energy optimization.

Six are engineering-tier projects that can be built from existing components: HaluGate-driven model reputation routing (Opportunity 2), runtime token-budget enforcement (Opportunity 5), follow-the-sun pool rebalancing (Opportunity 11), router-emitted KV-cache retention directives (Opportunity 15), governance-as-code (Opportunity 20), and request-level RBAC enforcement (Opportunity 8). Fifteen are research-tier: offline RL for session-length routing (Opportunity 1), cumulative multi-turn risk scoring (Opportunity 3), failure-class-aware routing from router outcome memory (Opportunity 4), fleet-learned joint tool–model optimization (Opportunity 6), gateway-coordinated agent loops for memory tiers and tool surfaces (Opportunity 7), fleet-wide caller reputation with cross-agent threat propagation (Opportunity 9), router-enforced behavioral commitments (Opportunity 10), pool-state inference (Opportunity 12), output-length-aware pool routing (Opportunity 13), pool-aware cascading (Opportunity 14), online $(\gamma, \mathcal{B}^*)$ co-adaptation (Opportunity 17), mixed-archetype fleet provisioning (Opportunity 18), archetype-driven proactive pool scaling (Opportunity 16), energy as a routing objective (Opportunity 19), and closed-loop self-adaptation across WRP knobs (Opportunity 21).

As production workloads shift from chat toward agents and multimodal applications, the cross-dimensional interactions identified here will grow in importance. We hope the WRP framework, including its assessment of what is ready to build and what remains open, proves useful to others working on these problems.

References

[1] vLLM Semantic Router Team. vLLM semantic router: Signal driven decision routing for mixture-of-modality models. *arXiv preprint arXiv:2603.04444*, 2026.

[2] Xunzhuo Liu, Hao Wu, Huamin Chen, Bowei He, and Xue Liu. Conflict-free policy languages for probabilistic ML predicates: A framework and case study with the semantic router DSL. *arXiv preprint arXiv:2603.18174*, 2026.

[3] Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. 98× faster LLM routing without a dedicated GPU: Flash attention, prompt compression, and near-streaming for the vLLM semantic router. *arXiv preprint arXiv:2603.12646*, 2026.

[4] vLLM Semantic Router Team. mmBERT-embed-32k-2d-matryoshka: Multilingual embedding model with 2d matryoshka training. Hugging Face model: `llm-semantic-router/mmbert-embed-32k-2d-matryoshka`, 2025.

[5] Chen Wang, Xunzhuo Liu, Yuhan Liu, Yue Zhu, Xiangxi Mo, Junchen Jiang, and Huamin Chen. When to reason: Semantic router for vLLM. In *NeurIPS Workshop on ML for Systems (MLForSys)*, 2025.

[6] Chen Wang, Xunzhuo Liu, Yue Zhu, Alaa Youssef, Priya Nagpurkar, and Huamin Chen. Category-aware semantic caching for heterogeneous LLM workloads. *arXiv preprint arXiv:2510.26835*, 2025.

[7] vLLM Semantic Router Team. mmBERT-32k feedback detector: User satisfaction classification for online routing adaptation. Hugging Face model: `llm-semantic-router/mmbert32k-feedback-detector-lora`, 2026.

[8] vLLM Semantic Router Team. Token-level truth: Real-time hallucination detection for production LLMs. vLLM Blog, 2025. https://blog.vllm.ai/2025/12/14/halugate.html.

[9] vLLM Semantic Router Team. mmBERT-32k factcheck classifier: Binary prompt classification for conditional hallucination detection. Hugging Face model: `llm-semantic-router/mmbert32k-factcheck-classifier-merged`, 2026.

[10] vLLM Semantic Router Team. MLCommons AI safety classifier – level 1 (binary): Safe vs. unsafe content classification. Hugging Face model: `llm-semantic-router/mlcommons-safety-classifier-level1-binary`, 2026.

[11] vLLM Semantic Router Team. MLCommons AI safety classifier – level 2 (9-class hazard): Hierarchical content safety classification. Hugging Face model: `llm-semantic-router/mlcommons-safety-classifier-level2-hazard`, 2026.

[12] Huamin Chen, Xunzhuo Liu, Yuhan Liu, Junchen Jiang, Bowei He, and Xue Liu. Token-budget-aware pool routing for cost-efficient LLM inference. *arXiv preprint*, 2026.

[13] Huamin Chen, Xunzhuo Liu, Yuhan Liu, Junchen Jiang, Bowei He, and Xue Liu. FleetOpt: Analytical fleet provisioning for LLM inference with compress-and-route as implementation mechanism. *arXiv preprint arXiv:2603.16514*, 2026.

[14] Huamin Chen, Xunzhuo Liu, Yuhan Liu, Junchen Jiang, Bowei He, and Xue Liu. inference-fleet-sim: A queueing-theory-grounded fleet capacity planner for LLM inference. *arXiv preprint arXiv:2603.16054*, 2026.

[15] Huamin Chen, Xunzhuo Liu, Yuhan Liu, Junchen Jiang, Bowei He, and Xue Liu. The 1/W law: An analytical study of context-length routing topology and GPU generation gains for LLM inference energy efficiency. *arXiv preprint arXiv:2603.17280*, 2026.

[16] Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. Adaptive vision-language model routing for computer use agents. *arXiv preprint arXiv:2603.12823*, 2026.

[17] Huamin Chen, Xunzhuo Liu, Junchen Jiang, Bowei He, and Xue Liu. Outcome-aware tool selection for semantic routers: Latency-constrained learning without LLM inference. *arXiv preprint arXiv:2603.13426*, 2026.

[18] Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. Visual confused deputy: Exploiting and defending perception failures in computer-using agents. *arXiv preprint arXiv:2603.14707*, 2026.

[19] OpenClaw contributors. OpenClaw: Personal AI assistant with a local-first gateway. Open-source software (MIT License), 2026. Repository: https://github.com/openclaw/openclaw. Gateway WebSocket control plane for multi-channel agent sessions, tools, and model routing; documentation at https://docs.openclaw.ai.

[20] Huamin Chen and Luay Jalil. Semantic inference routing protocol (SIRP). Internet Engineering Task Force (IETF), 2025.

[21] Huamin Chen, Luay Jalil, and N. Cocker. Multi-provider extensions for agentic AI inference APIs. Internet Engineering Task Force (IETF), Network Management Research Group, 2025.

[22] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, et al. LMSYS-Chat-1M: A large-scale real-world LLM conversation dataset. In *Proceedings of ICLR*, 2024.

[23] Pratyush Patel, Esha Choukse, Chaojie Zhang, et al. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of ISCA*, 2024.

[24] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of ICLR*, 2024.

[25] Shishir G. Patil, Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of ICML*, 2025.

[26] Yuxing Xiang, Xue Li, Kun Qian, Wenyuan Yu, Ennan Zhai, and Xin Jin. ServeGen: Workload characterization and generation of large language model serving in production. *arXiv preprint arXiv:2505.09999*, 2025.

[27] Han Bao, Zheyuan Zhang, Pengcheng Jing, et al. Drift-bench: Diagnosing cooperative breakdowns in LLM agents under input faults via multi-turn interaction. *arXiv preprint arXiv:2602.02455*, 2026.

[28] Xuannan Liu, Xiao Yang, Zekun Li, et al. AgentHallu: Benchmarking automated hallucination attribution of LLM-based agents. *arXiv preprint arXiv:2601.06818*, 2026.

[29] Pengfei Du. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026.

[30] Natchanon Pollertlam and Witchayut Kornsuwannawit. Beyond the context window: A cost-performance analysis of fact-based memory vs. long-context LLMs for persistent agents. *arXiv preprint arXiv:2603.04814*, 2026.

[31] Minki Kang, Wei-Ning Chen, Dongge Han, et al. ACON: Optimizing context compression for long-horizon LLM agents. *arXiv preprint arXiv:2510.00615*, 2025.

[32] Nikhil Verma. Active context compression: Autonomous memory management in LLM agents. *arXiv preprint arXiv:2601.07190*, 2026.

[33] Yubin Ge, Salvatore Romeo, Jason Cai, et al. SAMULE: Self-learning agents enhanced by multi-level reflection. In *Proceedings of EMNLP*, 2025. arXiv:2509.20562.

[34] Yifan Yu, Moyan Li, Shaoyuan Xu, et al. CORRECT: COndensed eRror RECognition via knowledge transfer in multi-agent systems. *arXiv preprint arXiv:2509.24088*, 2025.

[35] Xuanbo Su, Yingfang Zhang, Hao Luo, et al. Mistake notebook learning: Batch-clustered failures for training-free agent adaptation. *arXiv preprint arXiv:2512.11485*, 2025.

[36] Uria Franko. Dynamic system instructions and tool exposure for efficient agentic LLMs. *arXiv preprint arXiv:2602.17046*, 2026.

[37] Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Syed Fahad Allam Shah, and Dan Roth. ToolScope: Enhancing LLM agent tool use through tool merging and context-aware filtering. *arXiv preprint arXiv:2510.20036*, 2025.

[38] Cheng Qian, Emre Can Acikgoz, Hongru Wang, et al. SMART: Self-aware agent for tool overuse mitigation. *arXiv preprint arXiv:2502.11435*, 2025.

[39] Tengxiao Liu, Zifeng Wang, Jin Miao, et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*, 2025.

[40] Yanyu Ren, Li Chen, Dan Li, et al. Transcending cost-quality tradeoff in agent serving via session-awareness. In *NeurIPS*, 2025.

[41] Hanchen Li et al. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025.

[42] llm-d Team. KV-Cache wins you can see: From prefix caching in vLLM to distributed scheduling with llm-d. https://llm-d.ai/blog/kvcache-wins-you-can-see, 2026.

[43] vLLM Community. RFC: Context-aware KV-cache retention API (prioritized evictions). https://github.com/vllm-project/vllm/issues/37003, 2026.

[44] Yinmin Zhong, Shengyu Liu, Junda Chen, et al. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of OSDI*, 2024.

[45] BerriAI. LiteLLM tool permission guardrail. https://docs.litellm.ai/docs/proxy/guardrails/tool_permission, 2026.

[46] AAP Protocol Working Group. Agent authorization profile (AAP): OAuth 2.0 extension for agent authorization. https://www.aap-protocol.org/, 2026.

[47] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical Report SP 800-207, National Institute of Standards and Technology, 2020.

[48] Ryan Marinelli, Josef Pichlmeier, and Tamas Bisztray. Harnessing chain-of-thought metadata for task routing and adversarial prompt detection. *arXiv preprint arXiv:2503.21464*, 2026.

[49] Yinwei Dai, Zhuofu Chen, Anand Iyer, et al. Aragog: Just-in-time model routing for scalable serving of agentic workflows. *arXiv preprint arXiv:2511.20975*, 2025.

[50] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, et al. RouteLLM: Learning to route LLMs with preference data. In *Proceedings of ICLR*, 2025.

[51] Renxi Lu et al. MixLLM: Dynamic contextual-bandit-based routing for multi-LLM systems. In *Proceedings of NAACL*, 2025.

[52] Shashwat Jaiswal, Kunal Jain, Yogesh Simmhan, et al. SageServe: Optimizing LLM serving on cloud data centers with forecast aware auto-scaling. *arXiv preprint arXiv:2502.14617*, 2025.

[53] Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-R1: Teaching LLMs multi-round routing and aggregation via reinforcement learning. In *Proceedings of NeurIPS*, 2025. arXiv:2506.09033.

[54] Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-Router: A new paradigm for LLM routing with reasoning. *arXiv preprint arXiv:2602.02823*, 2026.

[55] Xianzhi Zhang, Yue Xu, Yinlin Zhu, Di Wu, Yipeng Zhou, Miao Hu, and Guocong Quan. Adapter-augmented bandits for online multi-constrained multi-modal inference scheduling. *arXiv preprint arXiv:2603.06403*, 2026.

[56] Noppanat Wadlom, Junyi Shen, and Yao Lu. Efficient LLM serving for agentic workflows: A data systems perspective. *arXiv preprint arXiv:2603.16104*, 2026.

[57] Anish Biswas, Kanishk Goel, Jayashree Mohan, et al. Sutradhara: An intelligent orchestrator-engine co-design for tool-based agentic inference. *arXiv preprint arXiv:2601.12967*, 2026.

[58] Qiaoling Chen, Zhisheng Ye, Tian Tang, et al. CONCUR: High-throughput agentic batch inference of LLM via congestion-based concurrency control. *arXiv preprint arXiv:2601.22705*, 2026. ICML 2026.

[59] Guibin Zhang, Haiyang Yu, Kaiming Yang, et al. EvoRoute: Experience-driven self-routing LLM agent systems. *arXiv preprint arXiv:2601.02695*, 2026.

[60] Caiqi Zhang, Menglin Xia, Xuchao Zhang, Daniel Madrigal, Ankur Mallick, Samuel Kessler, Victor Ruehle, and Saravan Rajmohan. Budget-aware agentic routing via boundary-guided training. *arXiv preprint arXiv:2602.21227*, 2026.

[61] Elias Lumer, Faheem Nizar, Akshaya Jangiti, et al. Don't Break the Cache: An Evaluation of Prompt Caching for Long-Horizon Agentic Tasks. *arXiv preprint arXiv:2601.06007*, 2026.

[62] Cheng Qian, Zuxin Liu, Shirley Kokane, et al. xRouter: Training cost-aware LLMs orchestration system via reinforcement learning. *arXiv preprint arXiv:2510.08439*, 2025.

[63] Tyler Griggs, Xiaoxuan Liu, Jiaxiang Yu, Doyoung Kim, and Enze Zhong. Mélange: Cost efficient large language model serving by exploiting GPU heterogeneity. *arXiv preprint arXiv:2404.14527*, 2024.

[64] Ruoyu Qin et al. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. *arXiv preprint arXiv:2407.00079*, 2025.

[65] NVIDIA. NVIDIA dynamo: Smart multi-node scheduling for LLM inference. https://developer.nvidia.com/blog/smart-multi-node-scheduling-for-fast-and-efficient-llm-inference-with-nvidia-runai-and-nvidia-dynamo/, 2026.

[66] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of SOSP*, 2023.

[67] Jesse van Remmerden, Zaharah Bukhsh, and Yingqian Zhang. Generalizing beyond suboptimality: Offline reinforcement learning learns effective scheduling through random data. *arXiv preprint arXiv:2509.10303*, 2025.

[68] Erica Barua, Pradeep Dasigi, and Muru Zhang. Agents of chaos: Evaluating LLM agent vulnerabilities through real-world interactions. *arXiv preprint arXiv:2602.20021*, 2025.

[69] Mark Russinovich, Ahmed Salem, and Ronen Eldan. Crescendo: Escalating multi-turn LLM jailbreak attacks. *arXiv preprint arXiv:2404.01833*, 2024. USENIX 2025.

[70] J. Alex Corll. Peak + accumulation: A proxy-level scoring formula for multi-turn LLM attack detection. *arXiv preprint arXiv:2602.11247*, 2026.

[71] Justin Albrethsen, Yash Datta, Kunal Kumar, et al. DeepContext: Stateful multi-turn jailbreak detection via temporal intent drift. *arXiv preprint arXiv:2602.16935*, 2026.

[72] Shir Ashury-Tahan, Yifan Mai, Elron Bandel, et al. ErrorMap and ErrorAtlas: Charting the failure landscape of large language models. *arXiv preprint arXiv:2601.15812*, 2026.

[73] Hao Li, Yiqun Zhang, Zhaoyan Guo, et al. LLMRouterBench: A massive benchmark and unified framework for LLM routing. *arXiv preprint arXiv:2601.07206*, 2026.

[74] Mehil B. Shah, Mohammad Mehdi Morovati, Mohammad Masudur Rahman, et al. Characterizing faults in agentic AI: A taxonomy of types, symptoms, and root causes. *arXiv preprint arXiv:2603.06847*, 2026.

[75] Kunlun Zhu, Zijia Liu, Bingxuan Li, et al. PALADIN: Training LLM agents to recover from tool-call errors via systematic failure injection. *arXiv preprint arXiv:2509.25370*, 2025.

[76] Tianjun Gao et al. Measuring multi-turn model-switch performance drift in LLM serving. *ICLR 2026 CAO Workshop*, 2026.

[77] Philipp Steiner, Ralph Peeters, and Christian Bizer. MCP vs RAG vs NLWeb vs HTML: A comparison of the effectiveness and efficiency of different agent interfaces to the web. In *Proceedings of The Web Conference*, 2026. arXiv:2511.23281.

[78] Jingyi Jia and Qinbin Li. AutoTool: Efficient tool selection for large language model agents. *arXiv preprint arXiv:2511.14650*, 2025.

[79] Sami Abuzakuk, Anne-Marie Kermarrec, Rishi Sharma, et al. Optimizing agentic workflows using meta-tools. *arXiv preprint arXiv:2601.22037*, 2026.

[80] Pengbo Liu. ToolRLA: Multiplicative reward decomposition for tool-integrated agents. *arXiv preprint arXiv:2603.01620*, 2026.

[81] Nikunj Gupta et al. HierRouter: Coordinated routing of specialized large language models via reinforcement learning. *arXiv preprint arXiv:2511.09873*, 2025.

[82] Cloud Native Computing Foundation. Open policy agent. https://openpolicyagent.org, 2024. CNCF Graduated Project.

[83] Krzysztof Rzadca et al. Autopilot: Workload autoscaling at Google scale. In *Proceedings of EuroSys*, 2020.

[84] Inho Cho et al. Overload control for μs-scale RPCs with Breakwater. In *Proceedings of OSDI*, 2020.

[85] Minghao Yang et al. TopFull: An adaptive top-down overload control for SLO-oriented microservices. In *Proceedings of ASPLOS*, 2024.

[86] Shashank Swamy et al. ALPS: Activation-based length prediction for intelligent LLM inference scheduling. *Google Research*, 2025.

[87] Huanyi Xie, Yubin Chen, Liangyu Wang, et al. Predicting LLM output length via entropy-guided representations. *arXiv preprint arXiv:2602.11812*, 2026.

[88] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[89] Pranjal Aggarwal, Aman Madaan, Ankit Anand, et al. AutoMix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*, 2024.

[90] Chiheng Lou, Sheng Qi, Rui Kang, et al. WarmServe: Enabling one-for-many GPU prewarming for multi-LLM serving. *arXiv preprint arXiv:2512.09472*, 2025.

[91] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of EuroSys*, 2015.

[92] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. In *Proceedings of ACM SIGCOMM*, 2014.

[93] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of NSDI*, 2011.

[94] Thomas Ziller, Shashikant Ilager, Alessandro Tundo, Ezio Bartocci, Leonardo Mariani, and Ivona Brandic. GreenServ: Energy-efficient context-aware dynamic routing for multi-model LLM inference. *arXiv preprint arXiv:2601.17551*, 2026.

[95] Seongjin Park et al. GAR: Carbon-aware routing for LLM inference via constrained optimization. *ICLR 2026 Workshop*, 2026.

[96] Walid A. Hanafy, Li Wu, David Irwin, et al. CarbonFlex: Enabling carbon-aware provisioning and scheduling for cloud clusters. *arXiv preprint arXiv:2505.18357*, 2025.

[97] Xinyi Zhang et al. Towards dynamic and safe configuration tuning for cloud databases. *arXiv preprint arXiv:2203.14473*, 2022. SIGMOD 2024.

[98] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[99] Olivier Jeunen and Aleksei Ustimenko. δ-OPE: Off-policy estimation with pairs of policies. *arXiv preprint arXiv:2405.10024*, 2024.

[100] Amit Singh Bhatti, Vishal Vaddina, and Dagnachew Birru. PROTEUS: SLA-aware routing via Lagrangian RL for multi-LLM serving systems. *arXiv preprint arXiv:2601.19402*, 2026.

[101] Wang Wei, Tiankai Yang, Hongjie Chen, et al. BaRP: Bandit-feedback routing with preferences for multi-LLM inference. *arXiv preprint arXiv:2510.07429*, 2025.

[102] Yasmin Moslem and John D. Kelleher. Dynamic model routing and cascading for efficient LLM inference: A survey. *arXiv preprint arXiv:2603.04445*, 2026.

[103] James Pan and Guoliang Li. A survey of LLM inference systems. *arXiv preprint arXiv:2506.21901*, 2025.

[104] Chunyu Xia et al. Taming the titans: A survey of efficient LLM inference serving. In *Proceedings of INLG*, 2025.

[105] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, et al. Hybrid LLM: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

[106] Amey Agrawal, Nitin Kedia, Ashish Panwar, et al. Sarathi-Serve: Efficient LLM inference by piggybacking decodes with chunked prefills. In *Proceedings of OSDI*, 2024.

[107] Hiari Pizzini Cavagna, Andrea Proia, Giacomo Madella, et al. SweetSpot: An analytical model for predicting energy efficiency of LLM inference. *arXiv preprint arXiv:2602.05695*, 2026.

[108] Chenxu Niu, Wei Zhang, Jie Li, Yongjian Zhao, Tongyang Wang, Xi Wang, and Yong Chen. TokenPowerBench: Benchmarking the power consumption of LLM inference. *arXiv preprint arXiv:2512.03024*, 2025.

Table 4 MAPE-K role of each opportunity: what the adaptation loop monitors, which knob it adjusts, and how it evaluates improvement.

#	Opportunity	Monitor (observable)	Knob (control)	Evaluate (metric)
1	Session-length RL	Session trajectories (model, tool, outcome per turn)	Model + tool selection policy	Median session length (turns, tokens)
2	HaluGate reputation	Per-response hallucination verdicts	Per-(model, domain) routing weights	Hallucination rate per domain
3	Cumulative risk	Per-turn SafetyL1 confidence scores	Safety escalation threshold	FPR vs. attack catch rate
4	Failure-class routing	Multi-signal outcome per (model, domain, failure class)	Failure-class routing weights	Oracle-utility capture rate
5	Token budget	Per-session token counter + failure-driven context bloat	Budget threshold + domain-specific compression policy	Token waste from failed sessions
17	Joint tool–model optimization	Per-(domain, turn, tool, model) outcomes + transition graph	Tool subset + resolving model + transition priors per dispatch	Aggregate session cost × task-completion rate
18	Gateway agent loops	Session depth, tool-/cache/KV telemetry, cross-tenant outcomes	Instruction/tool surface + cache-block order + TTL/admission hints	A/B: tokens/rounds/T-TFT vs. ITR-only or scheduler-only baselines
19	RBAC enforcement	Bearer token JWT claims + model tool_call outputs	Per-(role, tool, operation) permission matrix	Unauthorized tool-call block rate + false-denial rate
20	Caller reputation	Per-bearer-token security events across fleet	Reputation score → tool-/model/budget restrictions	Cross-agent attack propagation prevention rate
21	Behavioral commitments	Model boundary declarations in output text	Per-(token, agent) persistent block rules	Re-engagement-under-pressure prevention rate
6	Follow-the-sun	Sliding-window prompt-length CDF	Pool boundary τ^*	Fleet cost at SLO target
7	Pool-state inference	Dispatch/completion counts, TTFT	Per-instance routing weights	Per-pool SLO attainment
8	Output-length routing	Actual completion lengths	Pool routing (prompt + predicted output)	Short-pool overflow rate
9	Pool-aware cascading	Estimated queue depth per pool	Cascade-abort threshold	Quality vs. TTFT SLO
10	KV-cache retention	Domain, turn number, OATS tool latency	Retention TTL + eviction priority	Cache hit rate on agent resume
11	Proactive scaling	Per-archetype traffic time series	Per-pool instance count	Forecast accuracy vs. aggregate
12	(γ, τ^*) co-adapt	Sliding-window CDF + current (γ, τ^*)	Joint (γ, τ^*)	Cost gap vs. boundary-only
13	Mixed-archetype fleet	Router archetype breakdown	Fleet sizing weights	Complementarity savings
14	Energy routing	tok/W per pool configuration	Energy weight in decision rule	Fleet-wide tok/W at SLO
15	Governance-as-code	Policy constraints + fleet topology	DSL rule set	Constraint satisfiability
16	Self-adaptation	All of the above (composite session + security record)	All knobs jointly (incl. RBAC, reputation, commitments)	Composite session outcome + security posture