

SemServe: A Systematic View of Exploiting Application Semantics in Agentic Serving

Yaohua Xi

The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
yaohuaxi@link.cuhk.edu.cn

Minchen Yu*

The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
yumichen@cuhk.edu.cn

Abstract

Agent runtimes reason about workflow structure, validation, failure propagation, and join conditions, while model-serving systems schedule requests using model, worker, and cache state. The missing interface between these layers makes cross-layer decisions difficult to specify and verify. We present **SEMSErVE**, a prototype execution specification that carries explicit DAG semantics—including acceptance, join, failure, and criticality policies—from an agent runtime to a semantic router and an LLM serving stack. The same specification maintains a verifiable mapping from DAG nodes and attempts to requests, sessions, workers, and logical KV-cache allocations. This mapping enables two mechanisms: validator-triggered model escalation and policy-aware cancellation with early KV-cache reclamation. On a four-GPU prototype using Qwen3 models, our initial experiments show that criticality-driven escalation improves task correctness at an additional latency cost, while early-join cancellation substantially reduces wasted loser tokens and logical KV occupancy. These results motivate a cross-layer semantic control plane in which agent-level correctness conditions and serving-level resource events are represented by explicit, auditable specifications rather than ad hoc callbacks.

1 Introduction

Agentic applications are no longer isolated model invocations. A single user request may expand into multiple model turns, tool calls, validators, retries, and parallel branches whose execution spans a CPU-side runtime, model-routing services, gateways, and GPU workers. ReAct-style loops expose the interaction between reasoning and tools [19], while coding-agent traces show that real executions repeatedly cross these layers over long trajectories [22]. End-to-end behavior therefore depends not only on how an agent plans, but also on whether its execution intent can control the serving resources that realize the plan.

Existing systems provide important but separated pieces of this stack. LLMCompiler and GPTSwarm optimize tool or DAG [6, 23]; Agentix reconstructs dynamic program context to improve inference scheduling [10]; and Murakkab jointly configures workflows, models, and hardware under SLOs [1]. However, task-level control information that could enable parallel execution often remains implicit in the model-generated text and is not extracted for runtime scheduling. The runtime knows whether a branch is required, whether its output passed validation, and whether a join has already been satisfied. The serving stack instead knows the selected model and worker, active request state, and KV-cache ownership. Neither view alone is sufficient for verifiable cross-layer control.

This separation becomes visible during execution. A validation failure on a required branch may justify a stronger model, whereas the same failure on an optional branch may not. When a first-valid or top- k join becomes satisfied, remaining substitutable work may no longer have semantic value and its inference and cache state may be reclaimed. Yet a conventional model request does not express these conditions, and the runtime often cannot verify which model, worker, request, session, or logical KV allocation was actually affected.

We present **SemServe**, a systematic framework for exploiting application semantics in agentic serving. SemServe treats information that emerges during agent execution—including task structure, validation outcomes, completion requirements, branch criticality, and resource relevance—as first-class system signals rather than application-local metadata. It carries these semantics across the end-to-end serving path so that execution, model selection, and resource management can respond to the evolving state of an agent workflow. This unified semantic view provides a common basis for coordinating quality, latency, and resource efficiency while making cross-layer decisions explicit and verifiable.

SemServe uses an execution DAG as its structured substrate. DAG nodes and edges represent work and dependencies, while the semantic specification defines when a region is complete and which resource actions are authorized. This combination lets the system turn agent-level correctness conditions into concrete serving actions. Our goal is a complete semantic path from task execution to model and resource control.

This paper makes the following contributions:

- We formulate SemServe as a systematic view of agentic serving in which application semantics become first-class signals across tool execution, context structure, DAG design, resource management, and model routing.
- We define an explicit cross-layer semantic specification that separates graph dependencies from failure, join, acceptance, and criticality semantics, and links these semantics to observable execution and resource outcomes.
- We instantiate this view in an end-to-end prototype and evaluate two representative mechanisms—validator-triggered model escalation and early-join cancellation with asynchronous logical KV-cache reclamation.

2 Background

Agent systems span five recurring management dimensions: tool execution, context structure, DAG design, resource management, and model routing. Table 1 compares whether each system manages and optimizes these dimensions as first-class concerns.

*Corresponding author.

Table 1: Management dimensions in agent systems. ✓ denotes native management or optimization, △ partial or application-defined support, and ✗ no native mechanism.

System	Tool Execution	Context Structure	DAG Design	Resource Management	Model Routing
SemServe	✓	✓	✓	✓	✓
Agentix	△	✓	✓	✓	✗
Murakkab	✓	△	✓	✓	△
LangGraph	✓	✓	✓	✗	△

Agentix uses program context and dynamic DAG structure to schedule LLM calls and serving resources, while tools remain external interrupts and routing is not based on request semantics [10]. Murakkab represents models and tools in a declarative DAG and jointly optimizes workflow and hardware choices, but its context and model selection are primarily profile- and SLO-driven rather than per-turn semantic control [1]. LangGraph natively manages tools, stateful context, and graph execution, while resource management and model selection remain application-defined [8]. SemServe connects all five dimensions through one runtime-to-serving control loop.

3 SemServe Overview

SemServe connects a custom DAG-based agent runtime to semantic routing and LLM serving. The runtime handles task decomposition, dependencies, validation, and joins; lower layers select models and workers, execute requests, and manage cache state. Four goals guide the design: explicit semantics separate readiness from parallel-region completion; cross-layer visibility binds branches and attempts to their model, request, session, worker, and logical KV state; verifiable feedback reports actual resource choices and lifecycle outcomes; and off-path cleanup reclaims unnecessary work without delaying eligible answers. These goals preserve distinctions among completed, accepted, critical, and surplus branches across layers, extending classical workflow patterns [15] and program-aware serving in Agentix and Murakkab [1, 10].

3.1 Architecture

SemServe is organized as four logical layers connected by an execution path, a control path, and a resource-feedback path.

The agent execution layer receives a user task, creates and updates the execution graph, schedules ready nodes, coordinates model and tool operations, validates branch outputs, evaluates completion policies, and manages the lifecycles of branches and attempts. Concurrent branches retain isolated histories, call identities, session state, and cancellation scope.

The semantic decision layer translates task properties and runtime state—such as domain, complexity, criticality, retry history, required capability, reasoning mode, and the remaining service objective—into an abstract execution requirement. It determines the capability and reasoning budget needed by a model turn without selecting a particular serving process.

The serving coordination layer resolves this abstract requirement to a compatible model pool and then to a concrete execution endpoint. It considers availability, load, and reusable state, binds the

request and session identities to the selected endpoint, and exposes lifecycle operations for active requests.

The inference execution layer performs model computation, emits text, tool-call, completion, failure, and cancellation events, and maintains the request-, session-, prefix-, and logical cache state associated with each model turn.

3.2 Runtime-to-Worker Control and Worker-to-Runtime Feedback

Below, we follow the direction of data flow to illustrate how SemServe coordinates agent execution with model serving: the runtime propagates task and control information toward LLM workers, while execution and resource feedback returns to support validation, scheduling, and lifecycle decisions.

Runtime → LLM worker. When a model node becomes ready, the runtime sends the normal Responses-style execution payload: branch-local instructions and input, tool schemas, prior tool results, and an output budget. A semantic envelope adds the information that cannot be inferred reliably from the prompt. Branch and attempt identifiers bind concurrent requests and retries to the correct DAG lineage; failure, join, acceptance, and criticality fields state the completion and validation semantics of the region; and required capability, reasoning mode, retry history, and remaining deadline guide model selection and execution budgeting. Request and session handles are retained so that a later policy decision can address the corresponding inference state for abort, session closure, or cache invalidation.

LLM worker → runtime. The return path carries the standard generated items, tool calls, usage, and terminal status, together with newly exposed execution evidence. The actual model, reasoning mode, and worker show whether the requested routing decision was enforced; request and session status correlate streaming and lifecycle events with the owning attempt; and logical cache and invalidation outcomes show whether reusable state was released. The runtime records this evidence in the corresponding DAG node before validation and join evaluation. It can then activate downstream work, create an independent retry or capability escalation, or launch policy-authorized cleanup. The upward path therefore reports verifiable outcomes; it does not give the serving layer control over the runtime.

4 SemServe Design and Evaluation

SemServe designs and optimizes all five dimensions identified in Table 1: tool execution, context structure, DAG design, resource management, and model routing. Figure 1 maps the four logical layers to the prototype components used in evaluation. The following subsections present the corresponding insight, design, and prototype evidence for each dimension. Unless noted otherwise, experiments use four RTX 5090 GPUs (32 GB each), Qwen3-4B and Qwen3-8B, and the following layer instantiation: a Responses-style runtime (agent execution) → VSR (semantic decision) → SGLang gateway (serving coordination) → SGLang workers (inference execution).

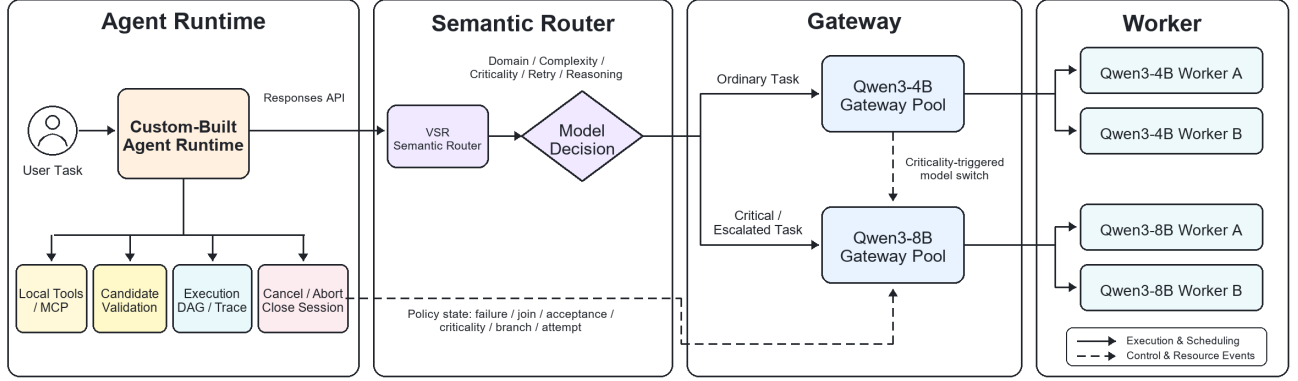


Figure 1: Prototype architecture used in evaluation. Solid arrows show execution and scheduling; dashed arrows show policy-derived control and resource-lifecycle events across the runtime, semantic router, gateway, and worker layers.

Table 2: Tool-execution results; E2E values are per-run means in seconds.

Workload	Runs	Baseline	SemServe	E2E ↓
Serial long task	30	11.669	8.118	30.43%
Four-branch DAG	10	6.848	5.809	15.17%

4.1 Tool Execution

Insight 1: Read-only external tools can occupy a substantial fraction of an agent loop’s latency, making tool execution a major opportunity for end-to-end performance optimization.

Design. An optional CPU-side Multinomial Naive Bayes classifier predicts and ranks the next tool action from runtime traces [4, 9, 12, 14]. Admission is limited to audited read-only, idempotent tools. A prediction executes against a temporary DAG overlay tagged with the base graph version; an exact match on the authoritative tool name and normalized arguments atomically commits and reuses the result.

Result. Table 2 summarizes latency across 30 serial long-task runs and ten four-branch DAG runs; all 40 final outputs passed validation. The parallel DAG also reduced mean TTFT by 1.006 s, with the smaller E2E gain consistent with the 2 s MCP branch remaining on the join-critical path. The controlled workload achieved 80/80 admitted-tool hits with no wrong speculation. CPU prediction cost was 0.3–0.44 ms for serial tasks and about 1.9 ms for the four-branch DAG. Because the plan extractor was near-oracle and delays were controlled, these results represent the mechanism’s upper bound rather than open-task accuracy.

4.2 Context Structure

Insight 2: Concurrent branches accumulate distinct histories and lifecycle state, making context structure an important optimization dimension for correctness, scalability, and execution efficiency.

Design. SemServe scopes execution context to a branch and its current attempt. Each child receives an isolated model history, tool observations, call identifiers, active transport state, serving session, and cancellation scope. A retry creates a new attempt context linked to the same branch rather than reusing the failed attempt’s mutable

provider state. Context updates are committed only to the owning branch and become visible to downstream nodes through explicit DAG dependencies.

Result. In the controlled four-branch all_terminal workflow, all ten runs completed and all 40 child branches reached the expected terminal state. Trace inspection preserved separate provider histories, call identifiers, and sessions for concurrent children, with no observed cross-branch cancellation or response-chain contamination. This is an isolation mechanism check rather than a general memory-quality evaluation.

4.3 DAG Design

Insight 3: DAG dependencies capture execution order, but parallel regions may differ in their completion, validation, and failure requirements, making DAG design a key optimization dimension for agent workflows.

Design. SemServe represents tasks, child agents, model turns, tool calls, validations, joins, final answers, and cleanup operations as typed DAG nodes. Nodes record dependencies, lifecycle state, validation and policy outcomes, model requirements, resource mappings, timestamps, and token counters. Required edge conditions activate ready nodes concurrently, while every retry creates a new attempt node rather than reviving a terminal node. This combines dataflow readiness [6, 11] with auditable multi-agent lineage [18]; semantic decomposition can likewise produce dependency-aware plans [24].

The semantic completion of each parallel region is declared by $\mathcal{P} = (F, J, A, C)$: failure, join, acceptance, and criticality. A compatibility checker rejects contradictory combinations before execution. The same DAG scheduler can therefore express all-required aggregation, first-valid races, top- k /quorum reduction, and critical fail-fast regions without changing graph dependencies. Edges define readiness, whereas the policy specification defines validation, reduction, retry, and when cancellation or reclamation becomes permissible [5, 15, 20]. Figure 2 shows representative compositions.

Result. The all-required specification completed 10/10 runs, correctly terminated all 40 child branches, and preserved every node, edge, attempt, and lifecycle event. The first-valid specification authorized cancellation only after a candidate passed validation. Thus, the same execution substrate changes completion behavior through

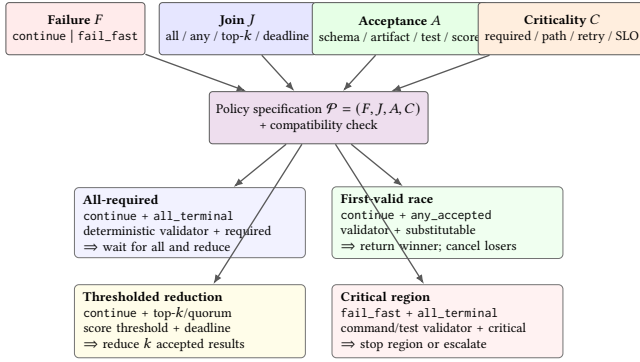


Figure 2: Orthogonal policy dimensions and representative valid compositions. Dimensions are declared independently, but compatibility is checked before execution.

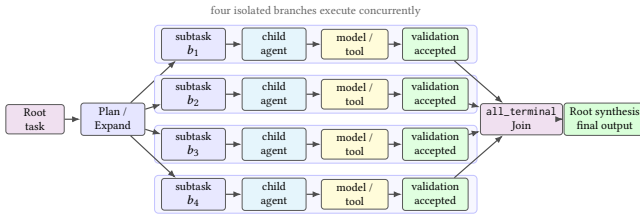


Figure 3: Runtime-built four-branch all_terminal DAG used for the mechanism check.

its policy specification; the resulting resource effects are evaluated next. Figure 3 shows the controlled four-branch execution.

4.4 Resource Management

Insight 4: Agent workflows may become semantically complete while serving work and cache state remain active, leaving significant room to optimize resource efficiency and response latency.

Design. After committing the policy-authorized result, SemServe launches one background cleanup chain per eligible branch: abort the request, close the session, invalidate its prefix, and release logical slots. Chains run concurrently and do not block answer delivery. The runtime maps each branch and attempt to its active request, session, worker, and logical cache allocation. Serving engines manage prefix reuse and KV allocation internally [2, 7, 21]; SemServe binds that lifecycle to agent-level join and failure semantics and records each observed stage as a DAG event. The prototype claims logical reuse only; physical CUDA VMM unmapping and an immediate decrease in process-visible HBM remain out of scope.

Result. Among seven first-valid runs that cancelled two losers, moving permitted cleanup off the response path reduced mean E2E from 15.544 s to 12.881 s (17.13%). Across eight correct HumanEval/MATH race pairs, policy-authorized cancellation reduced output tokens by 55.51% and KV token-seconds by 62.21%, while instantaneous KV peak fell by 9.02% because branches overlapped before winner validation. At least 11 in-flight requests reached the worker abort path; two prefix invalidations returned `found=false`, exposing a remaining request-lifecycle race. In all_terminal runs

Table 3: Eight correct race pairs; values are per-run means.

Metric	LangGraph	SemServe	Difference
Output tokens	1654.9	736.3	-55.51%
Total tokens	9209.9	8634.3	-6.25%
KV peak (tokens)	2527.5	2299.6	-9.02%
KV token-seconds	12178.0	4602.6	-62.21%
Mean E2E (legacy)	12.48 s	15.19 s	+21.76%

no branch was reclaimable, so the runtime paid orchestration overhead without a resource advantage. The benefit is therefore specific to policy-proven redundant tails.

4.5 Model Routing

Insight 5: Agent tasks can require different model capabilities as execution unfolds, while prompt-only routing observes only static input, making model routing a key dimension for balancing quality and serving cost.

Design. For every model turn, the runtime sends VSR task domain, complexity, criticality, required capability, reasoning mode, retry history, and remaining SLO [13, 16]. VSR chooses the 4B or 8B pool, and the SGLang gateway chooses a worker within that pool. A failed critical 4B attempt increments the weak-model failure count and creates an independent retry attempt. The scheduler can increase capability (4B normal $\rightarrow$ 8B normal) or reasoning depth (4B normal $\rightarrow$ 4B thinking) [3, 17]; non-critical failures do not automatically consume the strong pool. Returned model, worker, request, and session IDs verify that the requested escalation actually occurred.

Result. Across 60 HumanEval runs (20 per policy), adaptive 4B $\rightarrow$ 8B achieved 14/20 correctness, compared with 12/20 for both fixed 4B and fixed 8B. Eight failed 4B attempts triggered 8B retries and two recovered (25%). The adaptive policy used 32 8B model turns, 60% fewer than the 80 turns under fixed 8B, with no routing violations. Its additional retry path increased mean E2E to 16.59 s, compared with 11.69 s for fixed 4B and 13.69 s for fixed 8B, demonstrating selective quality recovery at additional latency cost.

5 Conclusion and Future Work

SemServe connects agent-level execution semantics to model serving through a cross-layer control and feedback path. In our early controlled experiments, criticality-aware 4B $\rightarrow$ 8B escalation raised HumanEval artifact correctness from 60% to 70% while using 60% fewer 8B model turns than fixed 8B. In first-valid races, policy-authorized cancellation reduced output tokens by 55.51% and KV token-seconds by 62.21%; moving loser cleanup off the response path reduced triggered-race mean E2E by 17.13%. These results indicate that explicit runtime semantics can improve selective quality recovery and resource efficiency.

Future work spans two directions. *LLM engineering* should improve the robustness of planning, validation, and semantic routing across task domains, including better calibration of capability escalation and reasoning depth. Second, a *bidirectional semantic control plane* would extend the current downward control and upward evidence path into a closed loop: serving-side conditions such as queueing, cache pressure, worker availability, and SLO risk could influence runtime graph adaptation, while runtime policies continue to guide serving decisions.

References

- [1] Gohar Irfan Chaudhry, Esha Choukse, Haoran Qiu, Inigo Goiri, Rodrigo Fonseca, Adam Belay, and Riccardo Bianchini. 2026. Murakkab: Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. USENIX Association, Seattle, WA, 567–587. <https://www.usenix.org/conference/osdi26/presentation/chaudhry>
- [2] Kaiwen Chen, Xin Tan, Minchen Yu, Jingzong Li, and Hong Xu. 2026. ReasonCache: Accelerating Large Reasoning Model Serving through KV Cache Sharing. In *IEEE/ACM International Symposium on Quality of Service*. <https://arxiv.org/abs/2507.21433>
- [3] Kaiyu Huang, Xingyu Wang, Mingze Kong, Zhuo Shi, Yuqian Hou, Hong Xu, Zhongxiang Dai, Minchen Yu, and Qingjiang Shi. 2026. UniScale: Adaptive Unified Inference Scaling via Online Joint Optimization of Model Routing and Test-Time Scaling. [arXiv:2605.30898 \[cs.LG\]](https://arxiv.org/abs/2605.30898) <https://arxiv.org/abs/2605.30898>
- [4] Kaiyu Huang, Hao Wu, Zhuo Shi, Han Zou, Minchen Yu, and Qingjiang Shi. 2026. AdaSpec: Adaptive Speculative Decoding for Fast, SLO-Aware Large Language Model Serving. In *Proceedings of the ACM Symposium on Cloud Computing*. 361–374. doi:10.1145/3772052.3772239
- [5] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Toronto, Canada, 14165–14178. doi:10.18653/v1/2023.acl-long.792
- [6] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An LLM Compiler for Parallel Function Calling. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, Vienna, Austria, 24370–24391. <https://proceedings.mlr.press/v235/kim24y.html>
- [7] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PageAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165
- [8] LangChain. 2026. LangGraph: A Low-Level Orchestration Framework for Stateful Agents. <https://docs.langchain.com/oss/python/langgraph/overview>. Accessed: 2026-08-30.
- [9] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, Honolulu, Hawaii, USA, 19274–19286. <https://proceedings.mlr.press/v202/leviathan23a.html>
- [10] Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. 2026. Agentix: An Efficient Serving Engine for LLM Agents as General Programs. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association, Renton, WA, 2443–2459. <https://www.usenix.org/conference/nsdi26/presentation/luo>
- [11] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 135–146. doi:10.1145/1807167.1807184
- [12] Daniel Nichols, Prajwal Singhania, Charles Jekel, Abhinav Bhatele, and Harshitha Menon. 2025. Optimizing Agentic Language Model Inference via Speculative Tool Calls. [arXiv preprint arXiv:2512.15834](https://arxiv.org/abs/2512.15834). <https://arxiv.org/abs/2512.15834>
- [13] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs from Preference Data. The Thirteenth International Conference on Learning Representations. https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html
- [14] Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, and Yuqing Yang. 2026. Act While Thinking: Accelerating LLM Agents via Pattern-Aware Speculative Tool Execution. [arXiv preprint arXiv:2603.18897](https://arxiv.org/abs/2603.18897). <https://arxiv.org/abs/2603.18897>
- [15] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. 2003. Workflow Patterns. *Distributed and Parallel Databases* 14, 1 (2003), 5–51. doi:10.1023/A:1022883727209
- [16] vLLM Semantic Router Team. 2025. vLLM Semantic Router. Open-source software. <https://github.com/vllm-project/semantic-router>
- [17] Chen Wang, Xunzhuo Liu, Yuhao Liu, Yue Zhu, Xiangxi Mo, Junchen Jiang, and Huamin Chen. 2025. When to Reason: Semantic Router for vLLM. *NeurIPS 2025 Workshop on Machine Learning for Systems*. <https://arxiv.org/abs/2510.08731>
- [18] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *Conference on Language Modeling (COLM)*. <https://arxiv.org/abs/2308.08155>
- [19] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=WE_vluYUL-X
- [20] Xing Zhang, Yanwei Cui, Guanghui Wang, Qucy Wei Qiu, Ziyuan Li, Fangwei Han, Yajing Huang, Hengzhi Qiu, Bin Zhu, and Peiyang He. 2026. Verified Multi-Agent Orchestration: A Plan-Execute-Verify-Replan Framework for Complex Query Resolution. *ICLR 2026 Workshop on MALGAI*. <https://arxiv.org/abs/2603.11445>
- [21] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. *Advances in Neural Information Processing Systems*. <https://arxiv.org/abs/2312.07104>
- [22] Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. 2026. TraceLab: Characterizing Coding Agent Workloads for LLM Serving. [arXiv:2606.30560](https://arxiv.org/abs/2606.30560). <https://arxiv.org/abs/2606.30560>
- [23] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language Agents as Optimizable Graphs. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, Vienna, Austria, 62743–62767.
- [24] Wenhao Zou, Zhijie Cai, Minchen Yu, Zongshuai Zhang, and Guangxu Zhu. 2026. Think in Cloud, Look at Edges: Semantic-Driven Query Decomposition for Efficient Video Reasoning. In *Proceedings of the 43rd International Conference on Machine Learning*. Spotlight. <https://openreview.net/forum?id=9HyDFcKhLV>