Trade-offs in Agentic AI: Benchmarking Accuracy, Latency, and Energy Across Multi-Agent LLM Systems

Michael Shen[1], Xiaoyue Zhou[1], Akiho Kawada[2], Ishan Patwardhan[1], Yueying Li[1], Leo Han[1], Udit Gupta[1]

[1]*Cornell Tech*
[2]*University of Tokyo*

Abstract

AI agents have emerged as a promising approach for extending the capabilities of LLMs in recent years. By combining specialized agents with external tools and data sources, these systems can tackle increasingly sophisticated tasks across a broader range of domains. However, translating these ideas into efficient real-world deployments remains far more challenging than early expectations suggested. In practice, naïve agentic designs can introduce substantial computational and orchestration overheads that, in some cases, outweigh the benefits they provide over simpler standalone LLM systems. Moreover, the large and highly dynamic design space of agentic architectures makes performance tradeoffs difficult to reason about and predict. In this paper, we characterize the underlying inefficiencies of agentic systems from a performance perspective and highlight the need for more deliberate systems-oriented optimization strategies when designing and deploying such systems.

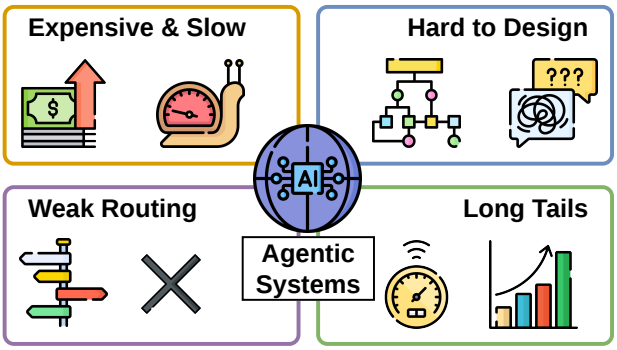

Figure 1: The design space of agentic systems naturally leads to configurations that are substantially more expensive than their non-agentic counterparts. These costs are further amplified by unreliable optimization techniques such as model routing, as well as long-tail latency effects, both of which make system-level optimization particularly challenging.

1. Introduction

Over the last couple of years, the machine learning and systems community has seen a rapid proliferation of Agentic AI workflows. By coupling autoregressive large language models (LLMs) with external tools, retrieval systems, and other LLMs, agentic pipelines now support a breadth of applications in software engineering, retrieval-augmented question answering, health care, document summarization, and customer-facing automation [51]. Fundamentally, these agentic workflows differ from monolithic LLMs. Rather than serving a single model in isolation, they orchestrate multiple specialized models and tools within a unified execution workflow and delegate subtasks across stages. This compositional design has driven their rapid adoption in both research and production.

Agentic systems are expected to deliver higher accuracy, automate complex multi-step tasks, and solve problems beyond the reach of a standalone model [65]. However, in practice, these gains come with rising computational, latency, and energy costs [34]. For example, a recent study [65] from Oppo AI Agent's team shows cutting-edge industry deployments of AI Agents (e.g., DeepResearch [27], and Manus [60]) yield impressive capabilities at exorbitant operating costs. In addition to the high cost of individual LLM calls, particularly for large and reasoning models, many agentic AI systems require hundreds of API calls exacerbating run-time costs and precluding wider adoption.

Given their high costs, emerging studies aim to understand and optimize the performance and efficiency of agents. A recent study studying accuracy-cost tradeoffs of different agentic workflows, demonstrates that while AI agents yield accuracy improvements with increased compute, the tradeoff rapidly comes at diminishing returns [34]. In addition to LLM costs being high for agents, studies have demonstrated external tool calls can also yield high overheads [53]. Autellix studies templates of prototypical agentic structures such as ReAct agents, chat-bots, map-reduce, and tree search to understand dependencies across each stage [43]. Finally, a research academic and industry survey quantifies the use of AI agents across 306 practitioners in 26 domains highlighting reliability is a fundamental challenge in AI agents driven primarily by a lack of benchmarks [51]. However, despite these studies, there is a lack of commensurate benchmarks that quantify accuracy, latency, and energy tradeoffs across a wide set of agentic workflows considering key design decisions when building agentic workflows such as model sizes, agentic structure, tail-latency, and routing decisions [51].

Commensurate benchmarking for agentic LLM workloads is challenging. LLM inference is volatile on its own. Identical queries can produce different latencies, token

counts, and correctness across repeated executions, which makes stable measurement hard. The design space is large and dynamic. It spans models, tools, datastores and retrieval parameters, the number of agents, and the number of turns; each axis reshapes the accuracy, latency, and energy tradeoff. Furthermore, there is a lack of consensus on what to measure. Mean and tail (p90/p99) latency, per-turn and end-to-end cost, time-to-first-token and total latency, and energy per query and energy per correct answer can lead to different system optimization outcomes. Existing inference benchmark suites, like MLPerf [54] are designed around single-model workloads and do not capture the multi-stage, tool-augmented, and iterative nature of agentic execution.

In this paper, we present a detailed characterization of agentic systems. We study 5 example workflows including a baseline LLM, retrieval augmented generation (RAG), ReAct, Debate, and Group Chat agents. All workflows are implemented on open-source models (e.g., Qwen [67], Gemma-3 [64], Llama 3.1 [19], Phi-4 [1], Nemotron [5], and Ministrial 3 [40]) and open-source datasets. All inference engines use vLLM or SGLang serving backends to enable studying state-of-the-art serving performance. We build a unified evaluation harness to jointly examine accuracy, performance, energy considering the impact of different models sizes, number of models and agentic workflow structure, model routing decisions, and tail effects of agentic AI. This paper makes the following contributions:

- We characterize the energy and latency implications of agentic systems by comparing them to iso-accuracy, non-agentic LLM counterparts, and find that agentic systems can often introduce substantial latency/energy overheads while providing limited to no accuracy gains if not carefully designed. Furthermore, due to the complexity of the design space associated with these agentic systems and interactions, finding optimal configurations is challenging.
- We examine the implications of LLM routers and selectors, and demonstrate why this methodology is currently impractical for systems-oriented optimization due to high inference variance causing up to 7.1× latency differences across identical calls, and routing decisions that can account 80% of an end-to-end agentic workflow latency.
- We highlight the substantial overhead introduced by consecutive LLM calls that can 35.8× the context length and 76.7× the latency overhead over a standalone LLM and argue that additional LLM calls should only be incorporated when they serve clear, targeted purposes.
- We characterize the dynamic behavior of agentic systems and show how variability between median (P50) and tail (P90) performance grows substantially across configurations, with context windows increasing by up to 10× in multi-turn agentic systems compared to single-LLM systems. Additionally, we demonstrate that Pareto frontiers across different performance percentiles are highly misaligned, with no single configuration dominating all regimes; in some cases, configurations can fall up to 133 seconds behind the frontier at another percentile.

To enable future work on end-to-end agentic system accuracy, performance, and energy optimization we intend to open-source all the agentic pipelines and their benchmarking methodologies, including evaluation harnesses, datastore generation scripts, and routers upon publication.

2. Background & Related Work

Agentic systems are modern inference workloads that move beyond monolithic LLMs by orchestrating multiple specialized models and external tools within a unified execution pipeline. These systems are motivated by the premise that specialization improves performance: models tailored to specific tasks can enhance overall coherence and accuracy, while access to external tools extends capabilities beyond what standalone LLMs can achieve.

There are two primary ways to design agentic systems: through static or dynamic execution flows. Static agentic systems rely on predefined, structured workflows in which LLMs follow fixed execution paths, offering high reliability and predictability for well-defined tasks. In contrast, dynamic agentic systems delegate planning to the LLM itself, allowing it to autonomously determine execution strategies, select tools, and adapt to changing environments, providing greater flexibility for complex or unpredictable tasks.

Agentic AI Frameworks. To support increasingly complex agentic workloads and the coordination of interacting components, several frameworks have emerged, including AutoGen [66], Kairos [69], and ORLA [58]. Complementary efforts like AFlow [72] and automated agent design approaches help developers structure agent organization and inter-agent interactions [25]. This work builds on AutoGen [72] to systematically profile the exact latency, energy, and accuracy implications of different agents.

Serving Optimizations. Recent research on optimizing agentic systems spans serving, routing, and characterization. On the serving side, Parrot [39], InferCept [2], Autellix [43], ALTO [55], and Cortex [50] expose multi-stage workflow structure to the engine through semantic variables, intercept-aware KV-cache management, program-aware scheduling, and stage isolation, while Murakkab [7] and Asgar et al. [4] jointly map workflows to heterogeneous hardware under SLO and TCO constraints. On the application side, cascade routers [9] and workflow compilers [35] reduce cost by directing queries to cheaper models or parallelizing tool calls, but ignores system-side latency and energy cost. The closest prior work, Raj et al. [53], characterizes agentic workloads from a CPU-centric perspective instead of GPU or energy consumption. Therefore, the community requires a comprehensive characterization that jointly evaluates system-level metrics, such as inference latency and hardware energy, along with application-level task accuracy across varying agent workflows.

Model Routers. Routing queries across multiple LLMs to select the most appropriate model for efficiency and accuracy or determine when to invoke external tools has emerged as an active area of research. RouterArena [42] provides a benchmark for evaluating routing methods on accuracy and performance, while LLMSelector [8] explores dynamic routing to trade off latency and accuracy. Several

TABLE 1: Design dimensions of the five agentic systems we evaluate.

System	Benchmark	Models	Tool Calls	Multi-Turn	Multi-LLM	Parallel
Baseline	HumanEval	Qwen3 family [0.6B, 1.7B, 4B, 8B, 14B]	–	–	–	–
RAG	TriviaQA, HumanEval	Qwen3 family [0.6B, 1.7B, 4B, 8B, 14B]	Database Search	✗	✗	✗
ReAct	HumanEval	Qwen3 family [0.6B, 1.7B, 4B, 8B, 14B]	Code Execution	✓	✗	✗
Debate	HumanEval	Qwen3, Gemma3, Nemotron-H, Llama3.1, Ministral, Phi-4	–	✓	✓	✓
Group Chat	HumanEval	Qwen3 family [0.6B, 1.7B, 4B, 8B, 14B]	Code Execution, Web Search	✓	✓	✗

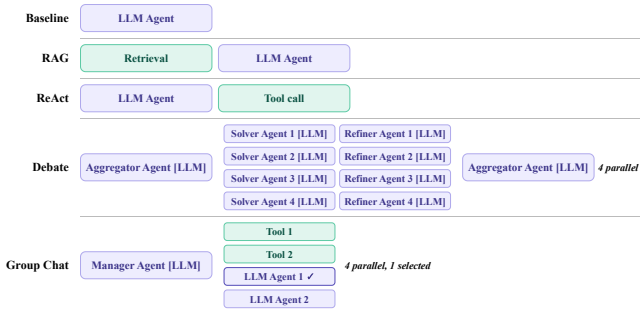

Figure 2: Workflow of the five systems we evaluate: a single-agent baseline and four multi-agent pipelines (RAG, ReAct, Debate, Group Chat). Purple boxes denote LLM agents; green boxes denote tool calls.

recent works propose hybrid or intelligent routing strategies for optimizing accuracy and task performance [11, 12, 17, 28, 46, 48, 61], including meta-LLM-based routers, embedding-driven selection, and RAGRouter [71], which selects the most suitable LLMs for RAG datastores.

However, the effectiveness of these routing techniques are limited, as extensive literature demonstrates that LLMs suffer from a measurable generation-discrimination gap. They are fundamentally poor at determining the correctness or confidence of their output [26, 29, 56]. Expensive LLM-based routers might introduce more latency and energy overhead than the efficiency gains (see Section 6)

3. Building Commensurate Agentic Pipelines

In this section we describe the evaluation harness developed to evaluate accuracy, latency, and energy of myriad agentic workflows.

3.1. Agentic Systems

We evaluate five representative agentic systems (see Figure 2 and Table 1). For serving and benchmarking, we adopt the AutoGen and AgentBench framework [41, 66] as our primary evaluation methodology.

Retrieval-Augmented Generation [37]: RAG models differ from traditional LLMs by incorporating a non-parametric datastore that is queried at runtime to retrieve relevant external context, which is combined with the user query to enhance the model's output. These datastores are typically constructed using vector databases designed to efficiently and effectively surface the most relevant context. We construct and evaluate multiple datastores using CompactDS [44], which is made up of 1.9B 256-word passages, and Sphere [52], which is made up of 100M 100-word passages. We leverage the FAISS [13, 32] library for implementing these datasets as vector indices.

ReAct [68] / Coding [57]: In ReAct workloads, an LLM is typically paired with an external tool that acts on the outputs of the LLM. For programming/coding tasks, this typically means integrating the model with a compiler or testbench to enable automated validation of the correctness and functionality of generated code. Unlike other systems which only evaluate once at the end, our coding agent invokes the Pytests at every turn, producing a per-turn correctness signal. We study two regimes for using this signal. In the naive regime, the agent ignores the signal except to terminate on success, holding the model fixed across turns. In the adaptive regime, we use the per-turn eval signal to compose heterogeneous model pipelines, such as escalating to stronger models when smaller ones fail.

Group Chat [38] / Debating [14]: Group chat and debate-based agentic systems both rely on multiple LLM agents collaborating to answer a query, but they differ in how that collaboration is structured. In a **group chat**, agents operate sequentially where each LLM receives the previous agent's response, builds on it, and passes its updated answer to the next agent in the chain. The orchestrator agent is responsible for selecting the agent for each step, determining when to terminate, and preparing the final answer in the end. In contrast, a **debate-based system** runs agents in parallel. Each agent independently generates an initial response, after which the agents exchange outputs, critique one another, and iteratively refine their answers. There is also an aggregator agent that constructs the final answer based on existing responses from all agents. This structure emphasizes cross-validation and improvement through comparison rather than strictly additive reasoning.

3.2. Metrics, Benchmarks, and Configurations

Benchmarks. We evaluate 5 systems on a mix of knowledge-intensive and coding benchmarks, including TriviaQA [33] and HumanEval [10]. The full benchmark, model, and workload matrix is given in Table 1.

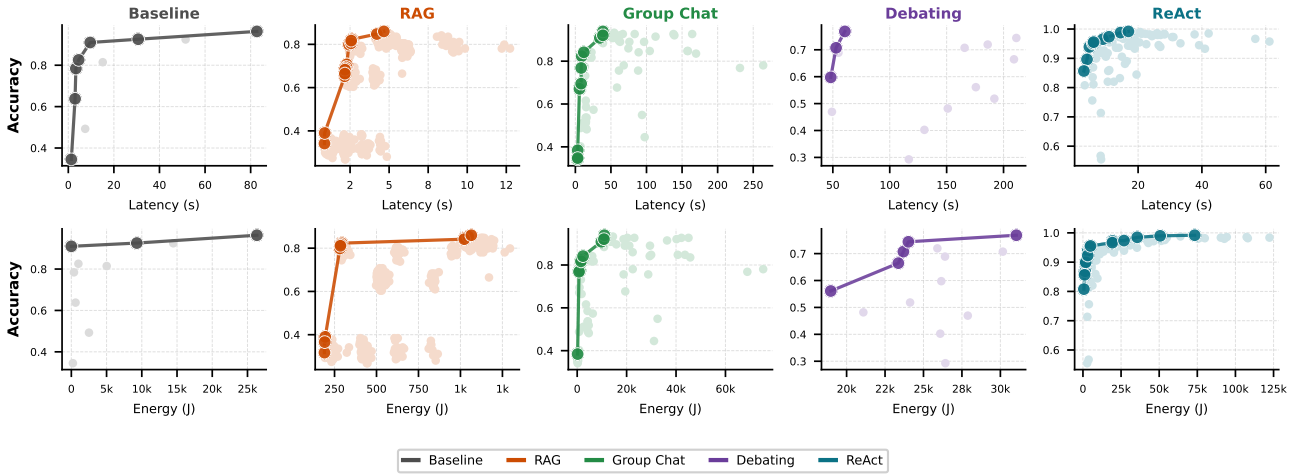

Figure 3: Accuracy-latency and accuracy-energy pareto frontiers for standalone LLMs and agentic systems show how some configurations (like RAG) can remain lightweight and outperform standalone LLMs and reduce latency to 0.86s while achieving up to 86.0% accuracy at only 1.06kJ. In contrast, complex agentic workflows such as debating and coding pipelines increase latency from 48–61s and energy consumption up to 73.3kJ with minimal accuracy gains.

Metrics. We characterize agentic systems along three axes. The first is accuracy, computed via the LM Evaluation Harness [18] for the QA benchmarks and via the HumanEval test harness for coding. To support RAG with the LM-Eval-Harness, we intercept queries issued by the harness and augment each prompt with retrieved context before forwarding it to the model. The second is end-to-end latency, including mean, p50, p90, and p99. The third is energy consumption, measured as energy per task using PyNVML based on the Nvidia management library. Throughout the paper we compare configurations on the latency–accuracy and energy–accuracy Pareto frontiers (Figure 3).

Models and Serving. All inference is served with vLLM [36] or SGLang [74], which provide high-throughput, memory-efficient execution through optimized attention and KV-cache management. Our primary model families are Qwen3 [67] and Gemma3 [64], chosen to span reasoning and non-reasoning behavior across a wide size range from 0.6B to 14B parameters. This range lets us study model-size sweeps cleanly within consistent architecture families. The debate agent additionally draws on Llama 3.1 [19], Nemotron-H [5], Phi-4 [1], and Ministral [40] to introduce cross-family heterogeneity, since debate is most informative when participating agents exhibit genuinely distinct reasoning behavior. Our group-chat agent uses Qwen3-based selector, terminator, and finalizer components to coordinate a pool of agents and tools across iterative multi-round execution, including an LLM-based coder, a code-executing terminal, and an LLM-based web surfer. For all the workloads, we host each model on a separate GPU.

Hardware. Experiments are conducted on a heterogeneous hardware testbed to capture realistic deployment scenarios. CPU-based retrieval run on Intel Xeon Gold 6442Y, Intel Xeon Platinum 8380, while model inference is deployed using NVIDIA A6000 Ada GPUs. For LLM inference workloads that require multiple GPUs, devices are interconnected via PCIe Gen5.

TABLE 2: When naively deployed, most agentic configurations are less system-efficient than standalone LLMs. Across the agentic configurations tested, 44.7% fall below the standalone LLM pareto for latency and 91.7% for energy.

System	Latency vs Accuracy	Energy vs Accuracy
RAG	220 / 540 (40.7%)	540 / 540 (100.0%)
Group Chat	65 / 69 (94.2%)	64/69 (92.8%)
Debate	14 / 14 (100.0%)	14/14 (100.0%)
ReAct	15 / 79 (19.0%)	26 / 79 (32.9%)
TOTAL	**314 / 702 (44.7%)**	**644 / 702 (91.7%)**

4. Quantifying Accuracy, Performance, and Energy Trade-Offs in Agentic Systems

In this section, we examine how the dynamic behavior of agentic systems differs from standalone LLMs, showing how compounded execution overheads can limit their effectiveness. We further demonstrate that optimizing these systems requires jointly considering per-stage execution and hardware trade-offs, while the rapidly expanding configuration space of agentic pipelines makes balancing accuracy, latency, and energy increasingly challenging.

Takeaway 1: Agentic systems dramatically expand the accuracy, latency, and energy design space, with higher agentic complexity often yielding diminishing accuracy returns with orders of magnitude higher latency and energy overheads. For example, improving accuracy from 83% with RAG pipelines to 93% with group chat systems increases latency from 2.6s to more than 38.7s and energy consumption from 300J to over 11kJ.

In Figure 3, we present accuracy-latency and accuracy–energy Pareto frontiers for reasoning and non-reasoning inference across a baseline LLM system and several agentic configurations (including RAG, Group Chat, Debate, and ReAct/Coding). Compared to the standalone LLM, agentic AI workflows can yield better Pareto-optimal tradeoffs in terms of latency and accuracy, as well as energy and accuracy. However, mis-configuration of agentic workflow parameters can lead to lower accuracy than baseline models at higher latency and energy. For instance, up to 44.7% of the agentic configurations we tested fell below the baseline LLM Pareto frontier, with an even larger 91.7% of energy configurations underperforming; Table 2 shows the breakdowns for each of the four workflows compared to the baseline LLM. Among configurations that fall far from the Pareto frontier, latency can degrade by up to 3.2×, increasing from 82.7s for a standalone Qwen3 14B reasoning model to 264s for a 10-turn group-chat agentic system. Energy consumption can also increase by up to 4.7×, rising from 26kJ for the same 14B reasoning model to 122kJ for a 6-turn coding agentic system.

Even among Pareto-optimal configurations, small accuracy gains can incur substantial system costs; for example, improving accuracy in our ReAct system from 93% to 99% increases latency by 3.5× (4.74s to 16.97s) and energy consumption by 22.8× (3.2kJ to 73.2kJ). This indicates that a significant portion of agentic system designs, when naively composed, yield performance tradeoffs that are poorly aligned with marginal accuracy gains over standalone LLMs.

Takeaway 2: External tools add tight design spaces that require domain-specific knowledge to effectively optimize for. For example, RAG introduces non-intuitive tradeoffs to accuracy and cost where scaling a datastore from 10M to 1B passages increases latency and energy by nearly 100× with only modest accuracy gains.

In order to understand the overhead that can be associated with the integration of tools in agentic systems, we focus on the design space of RAG, as it is a widely used tool integration that is both well-established and extensively studied. The implementation and effective use of these vector databases with large language models often involve a complex design-space exploration challenge. RAG engines involve a wide range of design-time decisions that directly shape the tradeoffs between performance, energy efficiency, and retrieval accuracy, many of which are difficult to modify after deployment. Key design choices (including index structure, quantization strategy, and datastore size) can substantially influence system latency, accuracy, and energy consumption. Additional runtime parameters, such as the number of retrieved documents and search effort (nProbe), further expand the tradeoff space and often require careful task- and context-specific tuning. Recent ML studies claim augmenting LLMs with large datastores can reduce the size of LLM by up to 10× [6, 44]; however the cost retrieval and added input tokens from the retrieved context can often outweigh the reduction in model size [30, 59]. We highlight these configurations and tradeoffs in Figure 4, where we evaluate retrieval using indices constructed from 100 million passages from the Sphere dataset [52] and the CompactDS dataset [44]. Using TriviaQA [33] queries, we compare across a range of Qwen3 [67] models.

Figure 4 highlights the complex tradeoffs between accuracy, latency, and energy across both design-time and runtime RAG parameters. Higher-precision quantization schemes generally improve accuracy, but often provide only marginal gains while substantially increasing latency and energy consumption.

- **Quantization.** Although PQ64 achieves the lowest accuracy, it is up to 7.8× faster and 9.9× more energy efficient than SQ4, while higher-precision configurations differ by at most 1.1% in accuracy.
- **Datastore Scaling.** Increasing datastore size consistently improves accuracy, but incurs steep scalability costs. Expanding from 10M to 1B passages increases latency and energy consumption by roughly 100×, despite diminishing accuracy gains at larger scales.
- **Retrieved Documents.** Increasing the number of retrieved documents improves accuracy with diminishing returns, but significantly raises retrieval and inference overheads; compared to standalone generation, retrieving a single document can increase latency and energy by 3.9× and 6.2× respectively.
- **Search Depth Inefficiency.** Likewise, increasing search depth through parameters such as nProbe yields near-zero downstream accuracy improvements despite near-linear growth in latency and energy consumption.

While these parameters already induce substantial variation in both the accuracy and performance of these serving configurations, they represent only a subset of the full RAG design space. More broadly, the breadth of this design space is expected to extend across a wide range of tool integrations, including compilers, search engines, and other forms of agentic actions as required by datasets like GAIA [45] and SWEBench [31].

5. Multi-LLM Call Agent Scaling Behavior

As shown in Section 1, the additional overhead introduced by agentic systems does not consistently yield performance improvements over a baseline LLM, largely due to the cost of extra LLM calls, which linearly increase token usage, latency, and energy consumption. To better understand when this overhead is justified, we analyze the magnitude of per-agent execution costs and examine conditions under which introducing additional agents becomes beneficial, particularly through model splitting strategies.

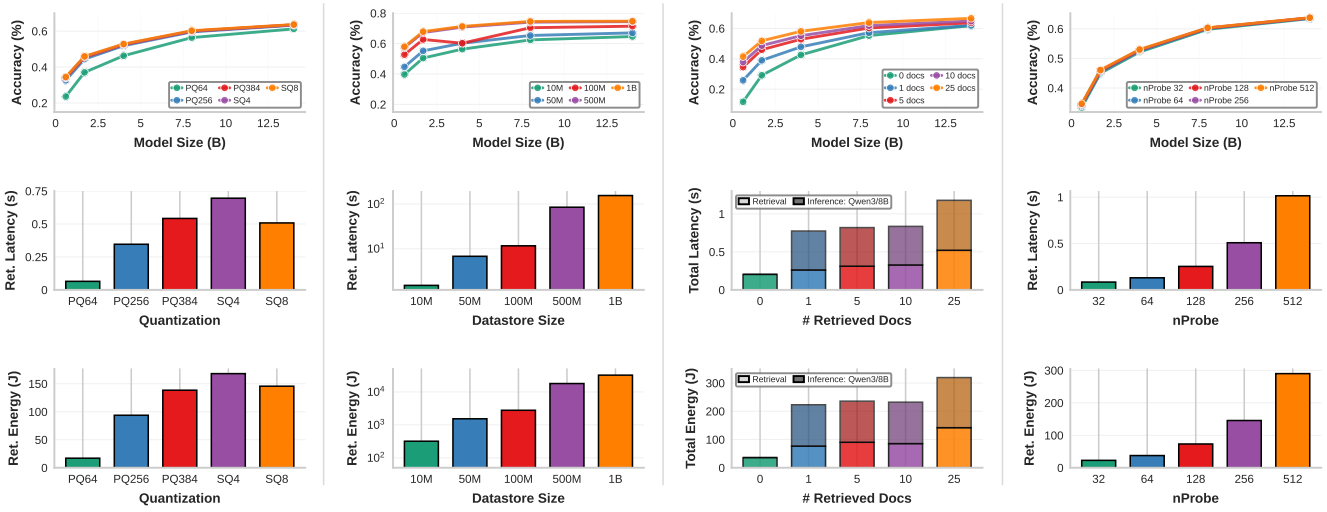

Figure 4: RAG expands the agentic design space through design time (quantization, datastore size) and runtime (number of retrieved documents, search depth/nProbe) parameters, that can induce tradeoffs in latency, accuracy, and energy. Columns correspond parameter sweeps and rows show accuracy versus model size (top), latency (middle), and energy (bottom).

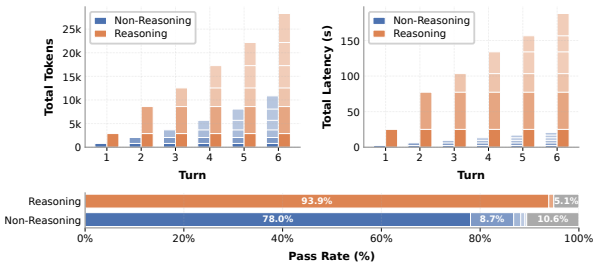

Figure 5: Additional LLM calls in agents dramatically increase token usage and end-to-end latency. Non-reasoning multi llm agentic systems scale from 794 tokens and 2.45s at one turn to 10.9k cumulative tokens and 20.8s by six turns, while reasoning systems can reach 28.3k cumulative tokens and 188s latency, all with diminishing accuracy gains.

Takeaway 3: Agentic systems must carefully limit the number of LLM calls, as multi-turn interactions lead to explosive growth in token generation and latency. In the case of coding agents, increasing LLM calls from one to six turns raises execution time by 13.7× and total token generation to 28.3k tokens, while improving reasoning accuracy by only 1%.

To study the effects of cascaded LLM calls and their impact on token overhead, we evaluate a coding ReAct-based system with a built-in call loop that repeatedly invokes the LLM until an iteration cap is reached or the query is successfully resolved, as determined by a compiler integrated into the execution pipeline (as seen in Figure 5). We find that, for non-reasoning traces, each additional turn produces progressively larger outputs, with generated tokens increasing from roughly 800 tokens in the first turn to nearly 2.8k tokens in the final turn, corresponding to an average growth of approximately 400 tokens per iteration. Across six turns, this compounds to a total of nearly 11k generated tokens for a single non-reasoning task. This increase in token generation also translates directly into higher inference costs, with per-turn inference time rising from 2.45s to 3.72s, with the cumulative end-to-end execution time across all six turns reaches 20.75s.

While the growth in token generation across turns for reasoning traces is less structured than non-reasoning traces, the token counts are substantially higher, ranging from 2.9k to 6.1k generated tokens per turn and compounding to a total of 28.3k tokens overall. This increase in token volume translates into significantly higher inference latency, with per-turn execution times ranging from 25s to 51s and a cumulative end-to-end execution time reaching 187s.

This explosive growth in token generation corresponds to a 13.7× and 9.8× increase over single-turn inference for non-reasoning and reasoning tasks, respectively, translating into 8.5× and 7.5× increases in end-to-end latency relative to baseline execution. Despite these substantial latency overheads, the resulting accuracy gains remain comparatively modest. Adding additional turns improves non-reasoning accuracy by only 8.7%, 1.4%, 0.8%, and 0.5% across successive stages, while incurring up to a 13.7× increase in latency. For reasoning traces, we see only a 1% improvement in accuracy despite a 9.8× increase in latency.

Takeaway 4: Choosing the number of stages in an agentic system, as well as the models assigned to each stage, is critical to preventing unbounded growth in the number of agentic calls. For example, we show by carefully balancing the number of stages and LLM size per stage for a coding agent can improve latency by up to 14× while improving accuracy by 1.3%.

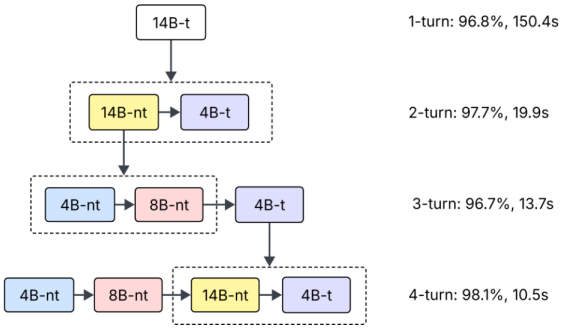

Figure 6: Iterative turn expansion [9] finds Pareto improvements over a single thinking model on HumanEval. Each row shows a ReAct configuration where boxes denote model and mode (e.g., 14B-t→Qwen3-14B Thinking). The 1 turn baseline uses a single thinking call with subsequent rows prepend cheaper non thinking models to the chain. Moving from the 1 turn baseline to a 4 turn system reduces latency by 14.3x (150.4s to 10.5s) while improving accuracy by 1.3%.

Given the high cost of large LLMs, another strategy to reduce the cost of serving while maintaining high accuracy is cascading models of varying sizes. Model cascading is commonly found in large-scale recommendation engines where initial lightweight models filter relevant candidates and subsequent heavyweight models conduct fine-grained ranking [20–22, 47]. Building on multi-turn agents, a similar model-cascading paradigm can be applied to agents. However, given the overheads associated with end-to-end LLM serving, additional agent calls should only be introduced when they provide meaningful improvements in system performance or accuracy without disproportionately increasing overall cost. Consistent with our findings in multi-turn LLM's, we observe that repeatedly invoking the same model does not necessarily translate into proportional system-level accuracy gains.

Figure 6 shows an example cascading architecture of the ReAct coding agent. Coding agents verify the end-output generated by evaluating against the provided test cases (e.g., call execution tool); however, this paradigm can be extended to other scenarios with LLM-as-a-judge [73] or external verification engines to determine when outputs from early stages fails. Beginning with a single 14B parameter reasoning model as baseline, we illustrate the benefits of introducing additional stages, maintaining similar end-to-end accuracy at lower latency. By analyzing LLM-to-LLM interactions and identifying smaller model pairings capable of achieving comparable accuracy with significantly lower latency and cost, we can decompose larger models into more efficient specialized components until reaching the practical limit of achievable accuracy and latency improvements. We transform a single-turn LLM solution into a four-turn workflow that reduces overall overhead by 14.3× while still improving accuracy by 1.3%.

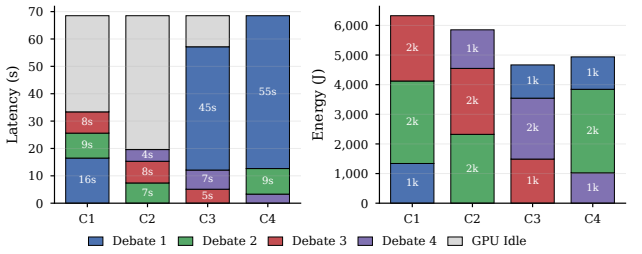

Figure 7: Parallel debate agent traces induce substantial GPU idling on the non-critical paths, reaching 51.1%, 71.6%, and 16.6% for C1, C2, and C3 respectively. This inefficiency compounds across agents, leading to an overall energy overhead of 4.4× a non-parallel system.

Takeaway 5: Parallelism introduces inefficiencies that lead to GPU idling where faster agents wait on slower ones (resulting in 34.8% GPU idle time) and up to 4.4× higher energy usage in parallel execution.

In our debate-based agentic system, we observe that heterogeneous LLM assignments produce highly variable runtime distributions. The combination of differing model capacities and progressively expanding context windows leads to increasingly unstable latencies, which are further amplified by long-tail P90 behavior (Discussed in Section 7) in certain LLM executions. As a result, the system exhibits wide variance in per-agent execution times and near-inevitable GPU underutilization. We see this in debate track C2 where this imbalance leads to substantial idle time, with the GPU remaining idle for approximately 71.5% of the total execution track duration.

Inefficiency in parallel agents is reflected in elevated GPU idle time and increased energy overhead. Specifically, energy consumption rises to 22 kJ over 80 seconds, which is approximately 4.4× higher than the 5 kJ that would be required by an equivalent non-parallel baseline configuration. The benefits of this parallelism from consecutive LLM calls are not clearly commensurate with its cost, as comparable or higher accuracy can often be achieved using RAG- or coding-based configurations (Figure 3, Takeaway 1). For instance, a RAG-based agent achieves 81% accuracy at 285J, whereas the debate-based structure reaches only 76.8% accuracy while consuming 31kJ of energy. Importantly, this estimate is conservative, as it does not account for additional energy wasted during GPU idle periods, which further exacerbates the true efficiency gap.

6. Challenges in Model Selection & Routing

One way to navigate the accuracy, performance, and energy tradeoffs in Agentic AI workflows (as seen in Section 4) is to intelligently route queries to the appropriate model based on input and task complexity. Given the dynamic nature of these workloads and the unpredictability of model performance, identifying the optimal models for a system's accuracy and performance at different stages of an agentic

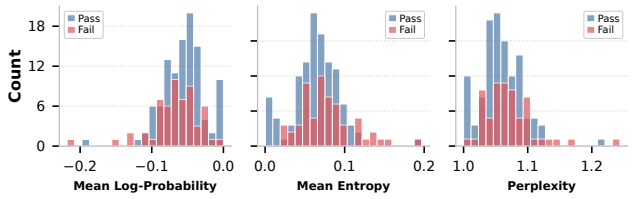

Figure 8: Distributions of mean log probability, mean entropy, and mean perplexity over 164 HumanEval tasks generated by Qwen3-1.7B baseline with thinking enabled, split by pass and fail. Pass and fail distributions are nearly indistinguishable across all three signals, indicating that these per-token confidence measures are not reliable predictors of task success and are unsuitable as routing signals in agentic pipelines.

pipeline cannot be determined a priori. Instead, model selection and system configurations must be dynamically tuned during workload execution. Traditionally, this burden has fallen on model routers [3, 8, 11, 15, 16, 48, 71], which determine at runtime which models should be used to satisfy a system's accuracy or performance requirements.

Routers can generally be categorized into three classes: LLM-based routers, MLP-based routers, and signal-based routers that rely on metrics such as cosine similarity or logits. However, as we demonstrate, none of these approaches are currently reliable enough to be used for producing justifiable improvements in system performance. In many cases, routers must also be more capable, as systems such as RouteLLM [48] are primarily designed to compare between only two LLMs, whereas agentic frameworks such as debate systems and group chats will require selecting among a much larger pool of candidate models.

Takeaway 6: The stochastic nature of LLM inference means that model routers are often trained on noisy and unreliable signals, including internal metrics such as logits, entropy, and perplexity, as well as output token counts (that can vary by as much as 3.8× for identical queries). Consequently, these routers frequently struggle to reliably distinguish between low- and high-complexity queries for fine-grained routing decisions.

Currently available open-source routers used for model routing in compound AI systems are fundamentally limited by the training data on which they are built, as the dynamic nature of LLM inference and the inconsistent performance characteristics of models make routing behavior inherently difficult to predict and train off reliably. For example, routers designed to make decisions based on LLMs' uncertainty and confidence remain unreliable [63]. In Figure 8 we examine three high-level confidence-related metrics commonly used during token generation (perplexity, log-prob distributions, and entropy distributions), we observe that the values corresponding to correct and incorrect responses overlap almost entirely. Even for metrics such as log probability and

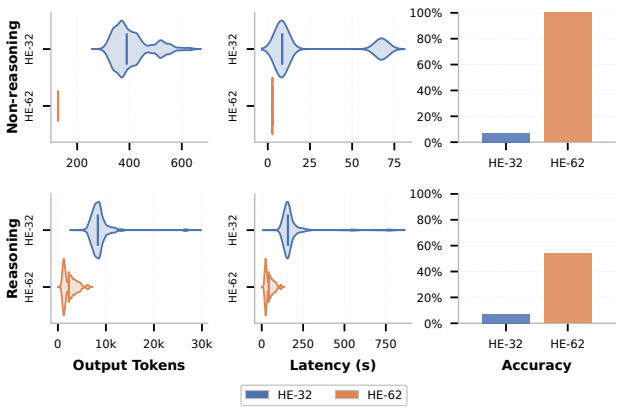

Figure 9: Latency and accuracy are not reproducible properties of a query and unreliable training metrics for model routers. Two tasks (HE-32, HE-62) executed 500 times on Qwen3-8B baseline with (top) and without (bottom) thinking. For reasoning, output length spans 5.7k to 26.6k tokens and latency spans 107s to 763s on identical inputs. Accuracy is also not perfectly stable with reasoning queries passing inconsistently 7% and 52% of the time.

entropy, where confidence is represented as a distribution rather than a single scalar value, the distributions associated with passing and failing queries still overlap almost entirely. Furthermore, query complexity analyzers such as NVIDIA's NeMo Curator [23, 24, 49], which estimate task complexity across dimensions such as creativity, domain knowledge, and reasoning, similarly struggle to reliably select the appropriate LLM. This substantial overlap further highlights the difficulty of using these signals to reliably distinguish successful generations from failures at inference time.

In addition to entropy and confidence based routing, we show the inherent noise in token distributions and LLM latency as poor prediction signals; such routers are used in making coarse decisions in run systems scheduling [62]. In Figure 9, we examine the latency distributions and accuracy characteristics of two queries from the HumanEval dataset by executing each query 500 times. For non-reasoning workloads, we observe substantial variability in performance: the gap between the p5 and p95 latencies can reach up to 1.8×, while the queried model answers correctly only 7% of the time across executions. For reasoning workloads, this latency gap grows to as much as 2.4×, while accuracy remains similarly unpredictable, with some queries still being answered correctly only 7% of the time across executions. However, this level of dynamism is not universal. As demonstrated by HE-62, certain queries exhibit highly stable and predictable performance behavior despite repeated executions. Nevertheless, the former highlights a key challenge for router training: models trained on performance characteristics that are themselves highly unstable may struggle to produce reliable or consistently expected routing decisions at inference time.

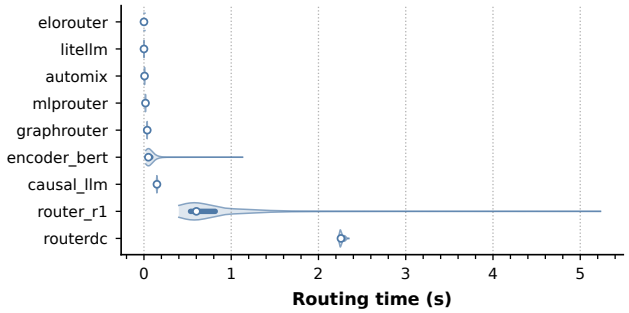

Figure 10: Routing overhead varies by orders of magnitude across methods, ranging from near-zero latency to over 2.26s per decision, with some approaches reaching 5.23s in the worst case. These overheads can quickly compound and can make up as much as 82% of an end-to-end agentic system's performance.

Takeaway 7: LLM routers can introduce substantial overhead. State-of-the-art routers can take 5 seconds per routing decision and account for as much as 82% of the total latency in agentic workflows such as RAG.

Beyond challenges in making high fidelity routing decisions, we find many router architectures can themselves introduce substantial inference overhead. Figure 10 shows the latency cost of 9 different model routers which varies by orders of magnitude [3, 11, 15, 48, 70]. Some routers, such as EloRouter [48] which scores each query based on offline training data, achieve low performance overhead (less than 100ms). However, these routing methodologies often fall short in agentic systems, where routing decisions may require selecting among many candidate models rather than performing simple pairwise comparisons.

In contrast, routers that explicitly support multi-model candidate selection, such as RouterDC [11] and RouterR1 [70], can incur routing decision latencies of up to 2.3s and 5.2s, respectively. While this overhead may appear modest depending on the surrounding system, agentic pipelines are frequently proposed as architectures built around many smaller specialized models rather than a single large model. Consequently, routing overhead can compound across every stage of the pipeline. In systems such as the RAG pipeline proposed in Section 4, routing overhead can therefore constitute up to 81.5% of the end to end inference time. Across a system like debating when agents are continuously called and require multiple routing decisions, the routing overhead can still makes up 8.9% of the end to end latency overhead.

7. Understanding Tail Performance

Traditional LLMs already exhibit long-tail latency behavior that requires careful scheduling and optimization to mitigate and hide overheads. Agentic systems dramatically exacerbate this issue. Since each LLM has its own P90 latency distribution, cascading multiple across a single system (with tool calls) causes tail latencies to compound. Moreover, different agentic configurations may operate on distinct Pareto frontiers, where the optimal serving configuration may shift depending on incoming queries. As a result, the ideal deployment setup can vary dynamically with workload characteristics within the same system.

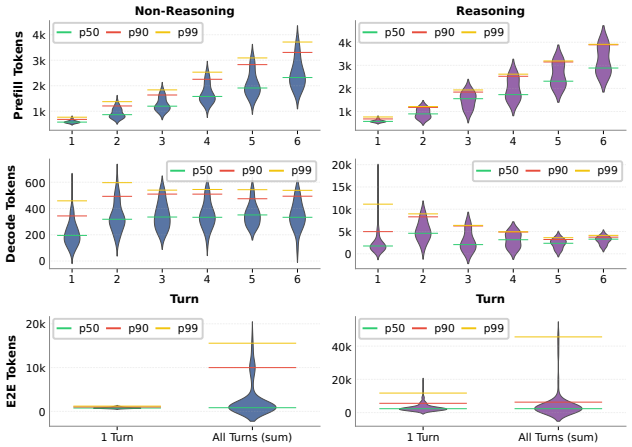

Figure 11: Distributions of prefill tokens (top), decode tokens (middle), and end to end tokens (bottom) for Qwen3-4B running ReAct on HumanEval, split by non-reasoning (left) and reasoning (right) modes. Horizontal lines mark p50, p90, and p99 across tasks. Prefill grows roughly linearly with turn count as prior context is reincluded, while decode per turn stays comparatively stable. Aggregated across all turns, the p99 to p50 ratio reaches 18.68x for non reasoning and 19.41x for reasoning traces, indicating that tail behavior dominates worst case cost in multi turn agentic execution.

Takeaway 8: Consecutive, multi-turn LLM calls in agentic systems can significantly amplify tail context sizes. For example, we find that compared to individual LLM invocations, the gap between p90 and p50 output lengths is 10× higher.

In Figure 11, we analyze how P50, P90, and P99 latencies evolve as additional turns are introduced into the agentic system, and show how the compounding of LLM calls causes P99 latency to increase sharply relative to a single-turn baseline. As the number of turns increases, we observe that P90 and P99 latency grow faster than P50 (due to the compounding context length from repeated LLM calls), consistent with accumulating inference overheads across multi-turn execution. On average, we observe that P99 input token counts grow from roughly 780 tokens in the single-turn setting to 3.8k tokens by six turns for both reasoning and non-reasoning traces. This growth remains relatively consistent because input token accumulation is largely deterministic, driven by prompt structure and the iterative reuse of prior-turn context rather than newly

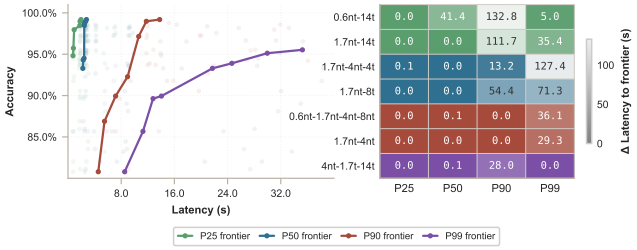

Figure 12: Pareto frontiers of agentic configurations shift substantially across latency percentiles, with frontier latencies increasing from 0.8–1.9s at P25 to 5.1–35.2s at P99. Moreover, as latency distributions shift, configurations that are Pareto-optimal at one percentile (e.g., a 0.6B non-reasoning model followed by a 14B reasoning model on the P25 coding frontier) can become up to 132s slower relative to the Pareto frontier at another percentile such as P75.

generated outputs. In contrast, P99 decode token growth is substantially more variable, ranging from 458–598 tokens for non-reasoning traces and 4.1k–11.1k tokens for reasoning traces. Although the per-turn gap between the P50 and P99 distributions appears relatively stable, the cumulative effect of carrying context forward across turns compounds these differences significantly over the full execution trace.

When aggregating tokens across all LLM calls in a trace, we observe strong compounding effects as the number of turns increases. For non-reasoning workloads, total end-to-end token counts grow from 575 tokens in the single-turn case to 10.4k tokens by six turns at P50 (18× growth), while P99 increases from 1.2k to 16.6k tokens (14× growth). Reasoning traces are substantially more expensive overall, with P50 token counts increasing from 2.3k to 29k tokens (12.6×) and P99 growing from 11.7k to 51.2k tokens (4.4×).

Takeaway 9: Optimal agentic configurations differ across latency percentile distributions, with no single configuration consistently optimizing performance across all latency distributions.

In Figure 12 we examine how the pareto frontiers for our coding-React agentic system shifts across different percentiles of our serving distribution. Notably, as latency percentile increases, the Pareto frontier shifts substantially to the right: compared to the P25 frontier (0.8s - 1.9s for 93.5–98.8% accuracy), the P50 frontier shifts by roughly 1.4×, the P90 frontier by 4 – 8×, and the P99 frontier by up to 16 – 18 ×. Moreover, the frontier heatmap shows that Pareto-optimal configurations under one latency percentile can become dramatically suboptimal under another. For example, while 0.6nt-4t lies on the P25 frontier, it incurs a 133s latency overhead relative to the P99 frontier. In contrast, configurations such as 1nt-1.7t-14t, which are optimal at P99, are roughly 28s less competitive at P90, but can also remain close to optimal at P25 and P50. This indicates a challenging design space, where the notion of an "ideal"

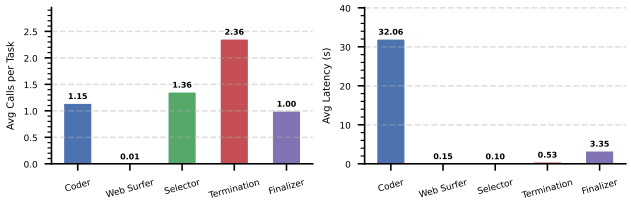

Figure 13: Calls per task and latency for dynamic execution path agentic systems show that while the termination and selector modules are invoked most frequently (2.36× and 1.36× per task), the coder dominates end-to-end latency with an average execution time of 32.1s per call.

configuration is already unstable due to complex LLM–LLM interactions, and is further complicated by sensitivity to latency percentiles in the serving distribution.

Takeaway 10: In agentic systems with dynamic execution paths, agent utilization is highly skewed, with orchestration agents invoked 2.36× while task-specific agents like the web surfer are nearly unused (0.01 calls per task). Despite this imbalance, a single compute-heavy stage (coding) dominates 88.5% of total latency.

In Figure 13, we profile the per-agent invocation frequency and latency of our group chat system, which uses an LLM-based selector to dynamically route queries across a pool of specialized agents. The termination checker and selector are the most frequently invoked agents at 2.36 and 1.36 calls per task, respectively, yet contribute minimally to end-to-end latency (0.53s and 0.10s per call). In contrast, the coder agent is called only 1.15 times per task on average but accounts for 32.1s per invocation, making it the dominant latency bottleneck. The web surfer agent is effectively unused at 0.01 calls per task. These results suggest that dynamic agent selection does not guarantee balanced utilization and that provisioning resources uniformly across agents leads to substantial underutilization. More importantly, optimizing end-to-end performance in such systems requires identifying and targeting the latency-dominant stage rather than reducing the invocation frequency of lightweight orchestration components.

8. Conclusion

In this work, we demonstrate across a wide range of workflows (including RAG, ReAct, Debate, and Group Chat), that agentic systems introduce a highly dynamic and challenging design space where additional agents, tools, do not consistently translate into proportional gains in accuracy. We also show why approaches such as routing are limited and may not be the most effective way to design agentic systems from a systems-efficiency perspective. Finally, we showed that scaling agentic systems can explosively increase token generation and system overheads, significantly amplifying end-to-end latency, particularly at the tail. Together, these findings highlight that building

efficient agentic systems requires carefully balancing specialization, routing overhead, and execution structure, rather than assuming that adding more agents, tools, or parallelism will inherently improve system performance.

References

[1] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*, 2024.

[2] R. Abhyankar, Z. He, V. Srivatsa, H. Zhang, and Y. Zhang, "Infercept: Efficient intercept support for augmented large language model inference," *arXiv preprint arXiv:2402.01869*, 2024.

[3] P. Aggarwal, A. Madaan, A. Anand, S. P. Potharaju, S. Mishra, P. Zhou, A. Gupta, D. Rajagopal, K. Kappaganthu, Y. Yang *et al.*, "Automix: Automatically mixing language models," *arXiv preprint arXiv:2310.12963*, 2023.

[4] Z. Asgar, M. Nguyen, and S. Katti, "Efficient and scalable agentic ai with heterogeneous systems," *arXiv preprint arXiv:2507.19635*, 2025.

[5] A. Blakeman, A. Basant, A. Khattar, A. Renduchintala, A. Bercovich, A. Ficek, A. Bjorlin, A. Taghibakhshi, A. S. Deshmukh, A. S. Mahabaleshwarkar *et al.*, "Nemotron-h: A family of accurate and efficient hybrid mamba-transformer models," *arXiv preprint arXiv:2504.03624*, 2025.

[6] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*. PMLR, 2022, pp. 2206–2240.

[7] G. I. Chaudhry *et al.*, "Murakkab: Resource-efficient agentic workflow orchestration in cloud platforms," *arXiv preprint arXiv:2508.18298*, 2025.

[8] L. Chen, J. Q. Davis, B. Hanin, P. Bailis, J. Zou, M. Zaharia, and I. Stoica, "Llmselector: Learning to select models in compound ai systems," in *ICML 2025 Workshop on Collaborative and Federated Agentic Workflows*.

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

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021.

[11] S. Chen, W. Jiang, B. Lin, J. Kwok, and Y. Zhang, "Routerdc: Query-based router by dual contrastive learning for assembling large language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 66 305–66 328, 2024.

[12] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Ruhle, L. V. S. Lakshmanan, and A. H. Awadallah, "Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing," Apr. 2024, arXiv:2404.14618 [cs]. [Online]. Available: http://arxiv.org/abs/2404.14618

[13] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," 2024.

[14] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," in *Forty-first international conference on machine learning*, 2024.

[15] T. Feng, Y. Shen, and J. You, "Graphrouter: A graph-based router for llm selections," in *International Conference on Learning Representations*, vol. 2025, 2025, pp. 26 186–26 203.

[16] T. Feng, H. Zhang, Z. Lei, P. Han, M. Patwary, M. Shoeybi, B. Catanzaro, and J. You, "Fusionfactory: Fusing llm capabilities with multi-llm log data," *arXiv preprint arXiv:2507.10540*, 2025.

[17] N. Fernandez, B. Kveton, R. A. Rossi, A. S. Lan, and Z. Wang, "RADAR: Reasoning-Ability and Difficulty-Aware Routing for Reasoning LLMs," Oct. 2025, arXiv:2509.25426 [cs]. [Online]. Available: http://arxiv.org/abs/2509.25426

[18] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "The language model evaluation harness," 07 2024. [Online]. Available: https://zenodo.org/records/12608602

[19] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[20] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 982–995.

[21] U. Gupta, S. Hsia, J. Zhang, M. Wilkening, J. Pombra, H.-H. S. Lee, G.-Y. Wei, C.-J. Wu, and D. Brooks, "Recpipe: Co-designing models and hardware to jointly optimize recommendation quality and performance," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 870–884.

[22] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia *et al.*, "The architectural implications of facebook's dnn-based personalized recommendation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 488–501.

[23] P. He, J. Gao, and W. Chen, "Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing," *arXiv preprint arXiv:2111.09543*, 2021.

[24] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[25] S. Hu, C. Lu, and J. Clune, "Automated design of agentic systems," *arXiv preprint arXiv:2408.08435*, 2024.

[26] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. Yu, X. Song, and D. Zhou, "Large language models cannot self-correct reasoning yet," in *International conference on learning representations*, vol. 2024, 2024, pp. 32 808–32 824.

[27] Y. Huang, Y. Chen, H. Zhang, K. Li, H. Zhou, M. Fang, L. Yang, X. Li, L. Shang, S. Xu *et al.*, "Deep research agents: A systematic examination and roadmap," *arXiv preprint arXiv:2506.18096*, 2025.

[28] K. Jain, A. Parayil, A. Mallick, E. Choukse, X. Qin, J. Zhang, Goiri, R. Wang, C. Bansal, V. Rühle, A. Kulkarni, S. Kofsky, and S. Rajmohan, "Intelligent Router for LLM Workloads: Improving Performance Through Workload-Aware Load Balancing," Jan. 2025, arXiv:2408.13510 [cs]. [Online]. Available: http://arxiv.org/abs/2408.13510

[29] D. Jiang, J. Zhang, O. Weller, N. Weir, B. Van Durme, and D. Khashabi, "SELF-[IN]CORRECT: LLMs struggle with discriminating self-generated responses," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 23, pp. 24 266–24 275, 2025. [Online]. Available: https://doi.org/10.1609/aaai.v39i23.34603

[30] W. Jiang, S. Subramanian, C. Graves, G. Alonso, A. Yazdanbakhsh, and V. Dadu, "Rago: Systematic performance optimization for retrieval-augmented generation serving," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 974–989.

[31] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?" in *International Conference on Learning Representations*, vol. 2024, 2024, pp. 54 107–54 157.

[32] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE transactions on big data*, vol. 7, no. 3, pp. 535–547, 2019.

[33] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[34] J. Kim, B. Shin, J. Chung, and M. Rhu, "The cost of dynamic reasoning: Demystifying ai agents and test-time scaling from an ai infrastructure perspective," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–16.

[35] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. Mahoney, K. Keutzer, and A. Gholami, "An llm compiler for parallel function calling," *arXiv*, 2023.

[36] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[37] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[38] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for" mind" exploration of large language model society," *Advances in neural information processing systems*, vol. 36, pp. 51 991–52 008, 2023.

[39] C. Lin, Z. Han, C. Zhang, Y. Yang, F. Yang, C. Chen, and L. Qiu, "Parrot: Efficient serving of {LLM-based} applications with semantic variable," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 929–945.

[40] A. H. Liu, K. Khandelwal, S. Subramanian, V. Jouault, A. Rastogi, A. Sadé, A. Jeffares, A. Jiang, A. Cahill, A. Gavaudan *et al.*, "Ministral 3," *arXiv preprint arXiv:2601.08584*, 2026.

[41] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang *et al.*, "Agentbench: Evaluating llms as agents," *arXiv preprint arXiv:2308.03688*, 2023.

[42] Y. Lu, R. Liu, J. Yuan, X. Cui, S. Zhang, H. Liu, and J. Xing, "Routerarena: An open platform for comprehensive comparison of llm routers," *arXiv preprint arXiv:2510.00202*, 2025.

[43] M. Luo *et al.*, "Autellix: An efficient serving engine for llm agents as general programs," *arXiv preprint arXiv:2502.13965*, 2025.

[44] X. Lyu, M. Duan, R. Shao, P. W. Koh, and S. Min, "Frustratingly simple retrieval improves challenging, reasoning-intensive benchmarks," *arXiv preprint arXiv:2507.01297*, 2025.

[45] G. Mialon, C. Fourrier, T. Wolf, Y. LeCun, and T. Scialom, "Gaia: a benchmark for general ai assistants," in *International Conference on Learning Representations*, vol. 2024, 2024, pp. 9025–9049.

[46] N. Muennighoff, S. Hongjin, L. Wang, N. Yang, F. Wei, T. Yu, A. Singh, and D. Kiela, "Generative representational instruction tuning," in *The Thirteenth International Conference on Learning Representations*, 2024.

[47] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[48] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, "RouteLLM: Learning to Route LLMs with Preference Data," Feb. 2025, arXiv:2406.18665 [cs]. [Online]. Available: http://arxiv.org/abs/2406.18665

[49] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[50] N. Pagonas, Y. Chung, K. Kaffes, and A. Krishnamurthy, "Cortex: Workflow-aware resource pooling and scheduling for agentic serving," *arXiv preprint arXiv:2510.14126*, 2025.

[51] M. Z. Pan, N. Arabzadeh, R. Cogo, Y. Zhu, A. Xiong, L. A. Agrawal, H. Mao, E. Shen, S. Pallerla, L. Patel *et al.*, "Measuring agents in production," *arXiv preprint arXiv:2512.04123*, 2025.

[52] A. Piktus, F. Petroni, V. Karpukhin, D. Okhonko, S. Broscheit, G. Izacard, P. Lewis, B. Oğuz, E. Grave, W.-t. Yih *et al.*, "The web is your oyster-knowledge-intensive nlp against a very large web corpus," *arXiv preprint arXiv:2112.09924*, 2021.

[53] R. Raj, H. Wang, and T. Krishna, "A cpu-centric perspective on agentic ai," *arXiv preprint arXiv:2511.00739*, 2025.

[54] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[55] K. Santhanam *et al.*, "Alto: An efficient network orchestrator for compound ai systems," in *Proceedings of the 4th Workshop on Machine Learning and Systems*, 2024, pp. 117–125.

[56] W. Saunders, C. Yeh, J. Wu, S. Bills, O. Long, J. Ward, and J. Leike, "Self-critiquing models for assisting human evaluators," *ArXiv*, vol. abs/2206.05802, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249626555

[57] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *Advances in neural information processing systems*, vol. 36, pp. 68 539–68 551, 2023.

[58] R. Shahout, H. Tirmazi, M. Yu, and M. Mitzenmacher, "Orla: A library for serving llm-based multi-agent systems," *arXiv preprint arXiv:2603.13605*, 2026.

[59] M. Shen, M. Umar, K. Maeng, G. E. Suh, and U. Gupta, "Hermes: Algorithm-system co-design for efficient retrieval-augmented generation at-scale," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 958–973.

[60] M. Shen, Y. Li, L. Chen, Z. Fan, Y. Li, and Q. Yang, "From mind to machine: The rise of manus ai as a fully autonomous digital agent," *arXiv preprint arXiv:2505.02024*, 2025.

[61] W. Song, Z. Huang, C. Cheng, W. Gao, B. Xu, G. Zhao, F. Wang, and R. Wu, "Irt-router: Effective and interpretable multi-llm routing via item response theory," *arXiv preprint arXiv:2506.01048*, 2025.

[62] J. Stojkovic, C. Zhang, Í. Goiri, J. Torrellas, and E. Choukse, "Dynamollm: Designing llm inference clusters for performance and energy efficiency," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 1348–1362.

[63] J. Su, F. Lin, Z. Feng, H. Zheng, T. Wang, Z. Xiao, X. Zhao, Z. Liu, L. Cheng, and H. Wang, "Cp-router: An uncertainty-aware router between llm and lrm," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 40, no. 39, 2026, pp. 33 065–33 073.

[64] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[65] N. Wang, X. Hu, P. Liu, H. Zhu, Y. Hou, H. Huang, S. Zhang, J. Yang, J. Liu, G. Zhang *et al.*, "Efficient agents: Building effective agents while reducing cost," *arXiv preprint arXiv:2508.02694*, 2025.

[66] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation," 2023. [Online]. Available: https://arxiv.org/abs/2308.08155

[67] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[68] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[69] Y. Yuan, M. Chowdhury, and N. Talati, "Kairos: Stateful, context-aware power-efficient agentic inference serving," *arXiv preprint arXiv:2604.16682*, 2026.

[70] H. Zhang, T. Feng, and J. You, "Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[71] J. Zhang, X. Liu, Y. Hu, C. Niu, F. Wu, and G. Chen, "Ragrouter: Learning to route queries to multiple retrieval-augmented language models," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[72] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang *et al.*, "Aflow: Automating agentic workflow generation," *arXiv preprint arXiv:2410.10762*, 2024.

[73] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in neural information processing systems*, vol. 36, pp. 46 595–46 623, 2023.

[74] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, "Sglang: Efficient execution of structured language model programs," *Advances in neural information processing systems*, vol. 37, pp. 62 557–62 583, 2024.