SPECULATIVE ACTIONS: A LOSSLESS FRAMEWORK FOR FASTER AI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite growing interest in AI agents across industry and academia, their execution in an environment is often slow, hampering training, evaluation, and deployment. For example, letting two state-of-the-art agents play a game of chess may take hours. A key bottleneck is that agent behavior unfolds sequentially: each action requires an API call, and these calls can be time-consuming. Inspired by speculative execution in microprocessors and speculative decoding in LLM inference, we propose Speculative Actions, a lossless framework that predicts likely actions using faster models, enabling multiple steps to be executed in parallel. We evaluate this framework across four agentic environments: gaming, e-commerce, web search, and operating systems. In all cases, speculative actions yield substantial acceleration, with potential speedups of up to 30%. Moreover, performance can be further improved through stronger guessing models, top-K action prediction, multi-step speculation, and uncertainty aware optimization, opening a promising path toward real world, efficient deployment of AI agents.

1 INTRODUCTION

Large language model (LLM)–driven agents are shifting from single-shot predictions to processes that run inside rich environments: browsers, operating systems, game engines, e-commerce stacks, and human workflows. These environments are not incidental; they determine what the agent can observe and do, gate progress through interfaces and rate limits, and dominate end-to-end latency. In practice, the agent's behavior unfolds as a sequence of environment steps (tool calls, API or MCP server requests, human-in-the-loop queries, and further LLM invocations) each with nontrivial round-trip time and cost. As capabilities improve, a new bottleneck becomes visible: time-to-action in the environment. Even when accuracy is high, an agent that pauses too long between steps is impractical for interactive use or high-throughput automation.

OS Tasks (Abhyankar et al., 2025)	**Deep Research** (OpenAI, 2025)	**Data Pipeline** (Jin et al., 2025)	**Kaggle Chess Game** (Kaggle, 2025)
10–20 min	5–30 min	30–45 min	1 hour

Table 1: **Estimated time state-of-the-art AI agents spend on various tasks/environments.**

As shown in Table 1, AI agents may require tens of minutes to hours to complete a single run across different environments, a cost that grows significantly when hundreds or thousands of iterations are needed for reinforcement learning or prompt optimization (Agrawal et al., 2025).

Fundamentally, this inefficiency arises from the inherently sequential nature of API calls. Thus, we ask a simple question in this paper:

Must an agent's interaction with its environment proceed strictly in sequence?

Our answer is no. Inspired by speculative execution in microprocessors and speculative decoding for LLM inference, we propose *speculative actions*: a framework that allows agents to predict and tentatively pursue the most likely next actions using faster models, while slower ground-truth executors (powerful LLMs, external tools, or humans) catch up. In effect, the agent stages environment

interactions (prefetching data, launching safe parallel calls, and preparing reversible side effects) so that validation, not waiting, is the critical path. When those slower evaluators confirm the guesses, progress has already been made; when they disagree, we execute as usual. The result is an *as-if-sequential, lossless interface with parallel, opportunistic internals*.

Concretely, in such agents, speculative actions introduce two roles in the environment loop:

- *Actor(s)*: authoritative but slower executors (e.g., more capable LLMs, external APIs/tools, the environment's own responses, or humans) whose outputs materialize the ground truth for correctness and side effects.
- *Speculator(s)*: inexpensive, low-latency models that predict the next environment step, i.e., the action, its arguments, and the expected observation or state delta. Examples include smaller LLMs, simplified use of the same LLM with reduced prompts and reasoning steps, and domain heuristics.

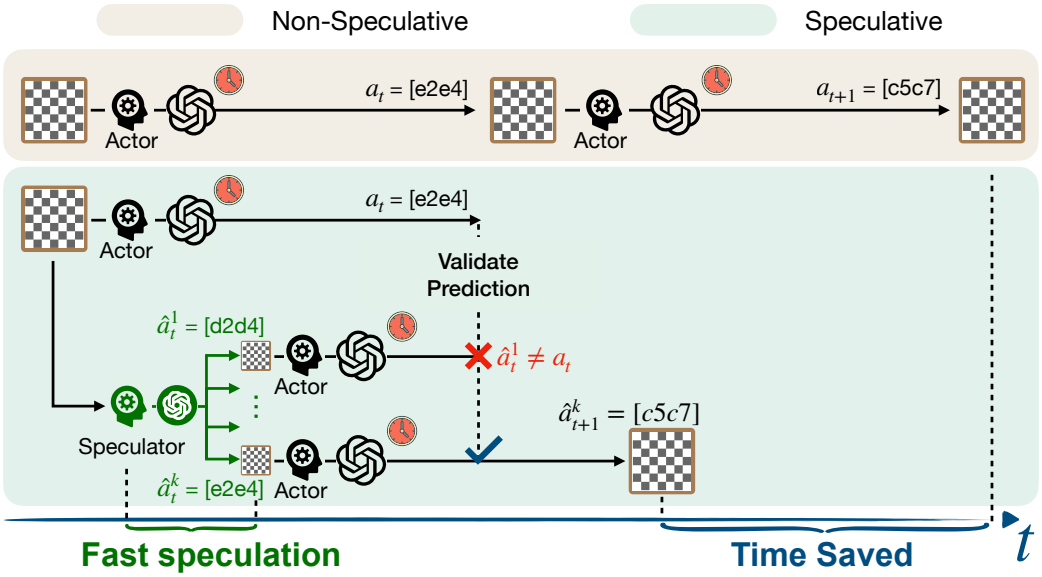


Figure 1: Illustration of our framework in a chess-playing environment. While the Actor issues an LLM call to decide the next move, the Speculator uses a faster model to guess it. These guesses enable parallel API calls for the next steps, and once a guess is verified, the system gains time through parallelization. The process runs in the backend, ensuring a lossless speedup for the user.

A key design goal is losslessness relative to the environment's baseline semantics: speculative actions should not degrade final outcomes versus a strictly sequential agent. We achieve this with (a) semantic guards (actors confirm equivalence of state transitions before commit), (b) safety envelopes (only idempotent, reversible, or sandboxed speculative side effects), and (c) repair paths (rollback or compensating actions when a guess is rejected). In many environments (e.g., web search pipelines, shopping carts before checkout, and OS-level operations in a sandbox) these patterns are natural and inexpensive to implement.

Can we guess the next API calls of agents? We show that, in practice, API intents can often be guessed with reasonable accuracy. In particular, we demonstrate speculative actions across four environments, each highlighting different aspects of agent latency:

- **Gameplay** (e.g., chess-like turn-taking): While waiting for a large model's long reasoning, we can perform move speculation and early lookahead. See Fig. 1.
- **E-commerce**: while waiting for a human response, Speculator can proactively infer intent (e.g., via follow-up questions) and safely trigger tool calls in advance (e.g., checking an order).
- **Web Search & Question Answering**: while waiting results from slow external calls, Speculator can guess answers from its knowledge base, enabling speculative query execution, snippet extraction and prefetching.

- **Operating Systems**: speculative, reversible actions react immediately to workload and environment changes, boosting end-to-end application performance while validators confirm.

Across these settings, we observe substantial acceleration—up to 50% for the accuracy in predicting the next API calls and 30% end-to-end lossless speedup. This is done with simple implementation, and performance can be further boosted with advanced techniques such as adaptive speculation. Qualitatively, users experience truly interactive agents that feel responsive even when authoritative components are slow.

1.1 Related Work

Speculation in Microprocessors and Thread-Level Speculation Speculation emerged in computer architecture to increase parallelism by executing instructions before their outcomes were resolved (Tomasulo, 1967), rolling back when predictions were wrong and became central to high-performance processors (Lam & Wilson, 1992). Thread-level speculation (TLS) extended this to whole program fragments, executing sequential regions in parallel and rolling back on conflicts, and exposing challenges like buffer overflows and rollback overhead (Estebanez et al., 2016).

Speculative decoding in LLM inference The same predict-verify pattern was recently applied to large language models. Speculative decoding accelerates autoregressive inference by using a small draft model to propose tokens that a larger target model verifies in batches, committing correct ones and regenerating failures (Leviathan et al., 2023; Zhang et al., 2024; Chen et al., 2023). At the reasoning level, speculation has also been used to accelerate chain-of-thought processes (Wang et al., 2025b;a; Fu et al., 2025). In all cases, speculation reduces latency in sequential pipelines by executing likely future steps in parallel with their validation.

From Token Speculation to Agent Speculation Speculation has also been applied in higher-level system contexts. For example, Speculator (Mambretti et al., 2019) let kernel processes bypass stalls, Speck (Nightingale et al., 2008) parallelized security checks, and AutoBash (Su et al., 2007) tested configuration fixes in isolation. More recently, hS (Liargkovas et al., 2023) used tracing and containment to speculatively execute shell scripts out of order. Farias et al. (2024) parallelized speculative actions in a supply chain system. Hua et al. (2024) and Guan et al. (2025) applied speculative planning to speedup LLM-based task planning.

*Unlike prior work, we propose a speculative framework for **entire agentic environments**, where all internal and external tool APIs, MCP-server APIs, LLM APIs, and even human responses can be speculated.* This enables a unified framework capable of achieving lossless speedups in agentic execution, leading to the efficient training/deployment of AI agents.

2 Framework

An agentic system is usually modeled as an Markov Decision Process (MDP) (s_t, a_t), where s_t denotes the state and a_t the agent's action at step t. The modeling of actions admits considerable flexibility: an action may represent a chatbot response, the choice of a tool to invoke, or a button clicked by a computer-use agent, among others.

From a systems perspective, we model each action in an agentic system as an **API call**, which may block execution until a response is returned. This abstraction offers two key advantages: (1) it precisely defines what constitutes an action, and (2) it provides a unified framework for optimizing system latency, as we will see shortly. Notably, this perspective aligns with the recent development of MCP servers for agentic systems (Anthropic, 2024).

Formally, at each step t, the policy π maps the current state s_t to an API call:

$$(h_t, q_t) \leftarrow \pi(s_t),$$

where h_t specifies the target API to invoke and q_t its associated parameters. The API then returns a response (or action), possibly after some delay:

$$a_t \leftarrow h_t(q_t).$$

The system subsequently transitions to the next state via a transition function f: $s_{t+1} \leftarrow f(s_t, a_t)$. As a concrete example, consider chess: the policy π determines how to construct the prompt based on the current board state, a_t corresponds to the move proposed by the LLM's response, and f updates the board configuration accordingly.

This formulation subsumes a broad range of realizations:

- **LLM calls:** each invocation of an LLM within the agent can be treated as an action.
- **Tool / MCP server calls:** Each actual call for internal/external tools is treated as an action: e.g., terminal access, web search, deep research APIs, weather APIs, or browser-use MCPs.
- **Human-as-an-API calls:** Futhermore, human responses themselves can be abstracted as API calls, often incurring even longer latencies than automated tools.

Given this abstraction, the fundamental bottleneck in executing agentic systems becomes apparent: each API call must complete before the next can be issued. To break this sequential dependency, we propose to **speculate a set of API responses** $\{\hat{a}_t\}$ using a faster model while waiting for the true response a_t. This enables speculative API calls for step $t+1$ to be launched in parallel. The speculative responses are cached, so if a true response matches a speculative one, the system can skip the actual invocation (see Algorithm 1).

Algorithm 1 Speculative Actions with k-way Parallel Next Calls

Require: Initial state s_0, horizon T, transition f, policy π, predictor $\hat{h}$, cache $\mathcal{C}$
for $t = 0$ to $T-1$ **do**
  Policy: $(h_t, q_t) \leftarrow \pi(s_t)$
  if $(h_t, q_t) \in \mathcal{C}$ **then** ▷ Cache hit
    $a_t \leftarrow \mathcal{C}[(h_t, q_t)]$
    $s_{t+1} \leftarrow f\big(s_t, a_t\big)$
    continue
  end if
  Actor: Issue real request (non-blocking): $a_t \leftarrow h_t(q_t)$
  Speculator: $\{\hat{a}_t^{(i)}\}_{i=1}^{k} \leftarrow \hat{h}(s_t, (h_t, q_t))$
  for $i = 1$ to k **do** ▷ One-step speculative rollout per guess
    $\hat{s}_{t+1}^{(i)} \leftarrow f\big(s_t, \hat{a}_t^{(i)}\big)$
    $(\hat{h}_{t+1}^{(i)}, \hat{q}_{t+1}^{(i)}) \leftarrow \pi(\hat{s}_{t+1}^{(i)})$
    Launch in parallel: $\hat{a}_{t+1}^{(i)} \leftarrow \hat{h}_{t+1}^{(i)}\big(\hat{q}_{t+1}^{(i)}\big)$
    On arrival, cache: $\mathcal{C}[(\hat{h}_{t+1}^{(i)}, \hat{q}_{t+1}^{(i)})] \leftarrow \hat{a}_{t+1}^{(i)}$
  end for
  Wait for a_t. When true response arrives or cache hit:
  $s_{t+1} \leftarrow f\big(s_t, a_t\big)$
end for

The resulting speedup relies on two key insights:

Assumption 1. *The speculative model is able to guess (h_{t+1}, q_{t+1}) with non-zero probability.*

As shown later, this often holds in practice because API behaviors are typically predictable.

Assumption 2. *Multiple API calls can be launched concurrently without side effects.*

In practice, this assumption is satisfied under modest traffic for many external APIs (e.g., web search, OpenAI LLM queries, email lookups). For self-hosted LLMs, concurrent calls also incur only minimal additional cost due to continuous batching.

We can then establish the following result (with proof deferred to the Appendix).

Proposition 1. *Under Assumptions 1–2, suppose the speculative model $\hat{h}$ guesses (h_t, q_t) correctly with probability p for $t \in [T]$. Let the latency of $\hat{h}$ be $\mathrm{Exp}(\alpha)$ and the latency of the actual model h be $\mathrm{Exp}(\beta)$ with $\beta < \alpha$. All latencies and guesses occur independently. Assume the transition f*

and API parameter construction π are negligible. Then the ratio between $E[T_{\mathrm{s}}]$, the expected time of Algorithm 1, and $E[T_{\mathrm{seq}}]$, the expected time of sequential execution, is

$$\frac{E[T_{\mathrm{s}}]}{E[T_{\mathrm{seq}}]} \;=\; 1 - \Big(1 - \frac{1}{T}\Big)\,\frac{p\,\alpha}{2(\alpha+\beta)}.$$

Extension. In fact, Algorithm 1 is a parsimonious demonstration of the speculative action idea. For example, one could conduct multi-step speculation instead of 1-step speculation, or use adaptive speculation to prioritize outcomes with a higher probability of being correct. We defer a detailed discussion to Appendix, but despite its simplicity, the results from the four use cases in the following sections are already highly promising.

Cost-vs-Latency Tradeoff. More speculative API calls (e.g., increasing k) improve accuracy but also raise costs when pricing is based on the number of calls or tokens. Although cost is not the focus of this work, we provide an analysis of our experiments in the Appendix. For self-hosted LLMs, this increase is largely mitigated through batching.

Side Effects and Safety. Speculation executes a hypothesized next action $\hat{a}_{t+1}$ that may be wrong, so safety requires the ability to simulate first and then commit or roll back. In domains like chess, rollback is trivial; in others, overwrite is easy (e.g., OS tuning). But many systems involve irreversible or externally visible effects (e.g., deleting records, placing orders), where naive speculation is harmful. Thus, speculation need to be limited to cases where mispredictions are reversible, via fork, snapshot restoration, or roll-forward repair (e.g., refund/replace).

3 Environments

We now instantiate speculative actions in three environment-centered settings—chess, e-commerce dialogue, and HotpotQA—chosen to stress distinct latency bottlenecks (reasoning, tool/API round trips, and information retrieval). In each case we pair a fast Speculator with a slower Actor and implement Algorithm 1. We report prediction accuracy and end-to-end time saved.

3.1 Chess Environment

To demonstrate the effectiveness of our speculative action framework in competitive multi-agent gameplay, we evaluate it on chess—a turn-based game where traditional sequential move execution leads to substantial idle time. When two reasoning models compete, games can stretch for hours because each player only begins their analysis *after* the opponent has moved. Our framework breaks this strict serialization by enabling speculative parallel analysis, resulting in significant reductions in overall game duration.

3.1.1 Implementation

We build our framework upon TextArena Guertler et al. (2025), leveraging its standard gameplay interface.

Speculative Pipeline. Consider a game at turn t, the state s_t is simply the current board, and h_t is the API call made from the current in-turn player, and its associated parameter q_t is exactly the current state s_t with an extensive reasoning eliciting prompt. Player P is to move and player Q awaits. Our framework executes as follows

- Current in-turn player P: the player receives s_t, makes an API call h_t to the agent with parameter $q_t = (s_t, prompt)$. This API call returns the next move $a_t = h(q_t)$ with high latency due to deep and extensive reasoning.
- Other out-of-turn player Q:
  1. **Prediction phase**. The Speculator simultaneously receives the same current board state s_t. It makes an API call $\hat{h}_t$ with s_t and a reasoning efficient prompt, and obtains the top-k move predictions $\hat{a}_t^1, \hat{a}_t^2, \ldots, \hat{a}_t^k$ ordered by descending confidence.
  2. **Parallel Computation**. The out-of-turn player Q immediately launches k parallel processes, each analyzing a next move $\hat{a}_{t+1}^i = h_{t+1}(\hat{s}_{t+1}, prompt)$ for $i \in \{1, \ldots, k\}$, where $\hat{s}_{t+1}^i = f(s_t, \hat{a}_t^i)$ denotes the next state resulting from applying predicted action $\hat{a}_t^i$ to s_t.

3. **Validation**. When the current in-turn player P finishes reasoning and returns its move a_t, we immediately check if any of the predictions $\hat{a}_t^1, \hat{a}_t^2, \ldots, \hat{a}_t^k$ match this move exactly.

4. **Commit or Restart**. If there exists a match, we take the speculative action $\hat{a}_{t+1}^{match}$ to be a_{t+1} and terminate other threads. We commit the $s_{t+1} = \hat{s}_{t+1}^{match} = f(s_t, \hat{a}_t^{match})$, and the game advances two steps at a time. If there exist no match, we terminate all speculative threads, and proceed to the next turn where Q is in turn to compute $a_{t+1} = h_{t+1}(f(s_t, a_t), prompt)$.

This pipeline ensures *losslessness*, since the final trajectory remains identical to non-speculative play while time is saved from parallelizing agents thinking. Based on this framework, we record the time and number of tokens spent on each action, as well as the accuracy of the guesser.

Agent Configuration We find that employing the same Speculator model as the Actor model with different prompts maximizes the accuracy while maintaining a reasonably quick speculative time. Accordingly, in our experiments, the Actor is instantiated as GPT-5 with high reasoning effort, where each move is produced via an API call. The Speculator also employs GPT-5, but with low reasoning effort and a specialized system prompt designed for rapid move prediction rather than exhaustive analysis.

3.1.2 Results

For each game, we track the prediction accuracy, defined as the number of rounds when any prediction matches the actual move, and the time saved percentage $(T_{sequantial} - T_{speculative})/T_{sequential}$

Time Saved and Accuracy increases with more predictions. Figure 2 shows the results for 30 steps. We observe that our speculative framework consistently saves time compared to the sequential execution, with the percentage of time saved increasing as the number of predictions increases. For 30 steps of gameplay, across 5 runs, the average time saved with 3 predictions is 19.5%, and the average accuracy is 37.3%.

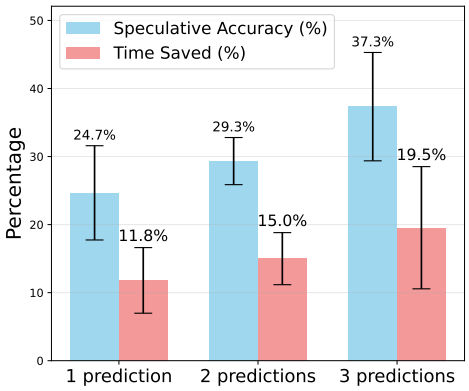


Figure 2: Percentage of time saved and percentage of correct predictions across 5 runs at 30 steps.

Randomness of agent call in gameplay The variance in Figure 2 reflects realistic gameplay dynamics from actual game runs with live API calls. Even when the Speculator correctly predicts the opponent's move, time savings vary dramatically. If the resulting position has an obvious response, the speedup is negligible since computation would be fast regardless. Substantial acceleration occurs only when successful guesses lead to positions requiring deeper analysis. This natural variance in response times shows that speculation's effectiveness depends on both prediction accuracy and computational complexity of the resulting positions.

3.2 Ecommerce Environment

Beyond competitive gameply, customer–agent interactions in the e-commerce domain provide a real-world setting where speculative actions can yield substantial impact. In a typical workflow, the customer submits a query through a chat interface and waits for the agent to respond. When the agent need to invoke multiple API calls sequentially (e.g., return all items in an order, which requires retrieving order information, validating return eligibility for each item, and initiating the return process), the resulting round-trip latency can significantly degrade user experience. By contrast, if some API calls are correctly speculated and executed in advance, once the user's query arrives, response latency is greatly reduced, making the interaction feel seamless and responsive. To demonstrate this setting, we test on τ-*bench*'s Yao et al. (2024) retail domain environment.

3.2.1 EXPERIMENTAL SETUP

Speculative Pipeline In this scenario, the current state s_t is defined as the conversation history between the user and Actor up to turn t, and h_t is the API calls (eg. get_user_details, get_order_details) that are needed to address the user's query. Our Speculator will predict

1. The user's query $\hat{a}_t$;
2. The target API calls and their corresponding parameters $(\hat{h}^i_{t+1}, \hat{q}^i_{t+1})$ for $i \in \{1, ..., k\}$, conditioned on the current state s_t and the predicted user's query from step 1. Since the number of API calls for each turn is not fixed, the Speculator also determines k.

Agent configuration We test various models as Speculator: OpenAI GPT models (gpt-5-nano, gpt-5-mini, gpt-5) and Google Gemini models (gemini-2.5-flash) with different reasoning budgets (1024, 2048, 4096 tokens). Prior work Jiang et al. (2023); Chen et al. (2025) shows that heterogeneous LLM ensembles often outperform individual models. Also, multiple models can execute in parallel under the same time budget. Motivated by these findings, we evaluate two Speculator configurations: (i) a *single-model* setting, where speculation relies on one model, and (ii) a *multi-model* setting, where models with comparable capacity and reasoning budgets run in parallel (i.e., gpt-5-nano with low-budget Gemini, gpt-5-mini with medium-budget Gemini, and gpt-5 with high-budget Gemini) and their predictions are aggregated into a candidate set of speculative actions. At runtime, when the user simulator provides the actual utterance, the Actor compares the speculative API calls with the ground-truth APIs. Correct predictions are committed immediately, absorbing latency, while incorrect predictions are safely discarded without affecting correctness.

Evaluation We evaluate performance using **APIs prediction accuracy**, defined as the fraction of speculative API calls that match the ground-truth APIs required to resolve the user's query. This metric directly reflects the proportion of turns in which the user receives an immediate response, without incurring the latency of waiting for API execution. In other words, higher prediction accuracy directly translates into greater time savings.

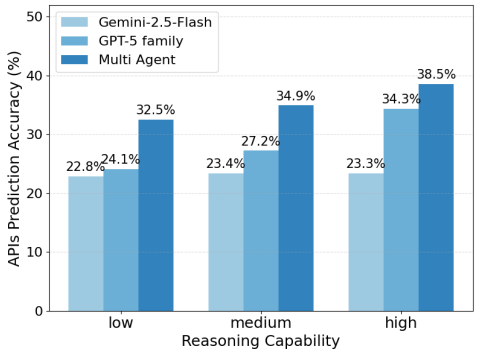


Figure 3: APIs prediction accuracy across different Speculator models with various reasoning capability.

3.2.2 RESULTS

Figure 3 shows that between 22% and 38% of API calls that would otherwise execute sequentially are correctly predicted by the Speculator. Accuracy improves with stronger models and the multi-agent configuration consistently outperforms single-model speculation. Importantly, the speculative time for models in the low group is only 2–3 seconds according to the LLM API providers leaderboard[1], which is well below the average user typing time of around 30 seconds (assuming 40 words per minute). This means that in around one-third of turns, the agent can provide a faster response without waiting for API execution. These results demonstrate that speculation can shift user experience from perceptibly delayed to effectively real-time in tool-heavy environments.

3.3 HOTPOTQA ENVIRONMENT

We further evaluate our framework on HotpotQA, a setting where the main performance bottleneck arises from information retrieval latency. In this example, the agent must answer multi-hop questions through sequential Wikipedia API calls Yang et al. (2018). This tool-calling pattern exemplifies real-world agentic workflows with high network latency per round-trip. We apply the framework from Section 2, where the Speculator predicts likely Wikipedia content while the actual API call executes. This parallelism enables the agent to continue reasoning on provisional information rather than blocking on API latency.

[1] https://artificialanalysis.ai/leaderboards/providers

3.3.1 Experimental Setup

We build our framework upon ReAct (Yao et al. (2023)), a structured paradigm for interleaving reasoning and acting.

Speculative Pipeline In this scenario, the state s_t consists of the entire history of reasoning and retrieved information (API responses). At each step, the Actor takes in the current state s_t, selects an API call $h_t \in \{\text{Search}(), \text{Lookup}(), \text{Finish}()\}$ and a corresponding parameter q_t, e.g. Search(entity). The call $h_t(q_t)$ returns a response a_t, typically providing information about the queried entity. Our speculative framework operates as follows:

1. Speculator predicts API call response $\hat{a}_t^i$, yielding predicted states$\hat{s}_{t+1}^i = f(s_t, \hat{a}_t^i)$, $i \in \{1, \ldots, k\}$.
2. Based on the states, the Actor generates reasoning traces and subsequently determines the next API decision $(\hat{h}_{t+1}^i, \hat{q}_{t+1}^i)$ for each predicted state.

Evaluation We evaluate the effectiveness of the speculative pipeline by the accuracy of the predicted API call decisions $(\hat{h}_{t+1}, \hat{q}_{t+1})$. Specifically, we compare the predicted call against the ground-truth call $(ht + 1, q_{t+1})$ obtained under the true response a_t. We employ a strict match criterion, counting a prediction as correct only when $\hat{h}_{t+1} = h_{t+1}$ and $\hat{q}_{t+1} = q_{t+1}$. This stringent criterion captures whether speculation enables meaningful progress, as even minor parameter differences (synonyms, word order) count as mismatches.

Agent configuration We evaluate speculative accuracy across three Speculator models: GPT-5-nano, GPT-4.1-nano and Gemini-2.5-flash. For each model, we measure the top-k prediction accuracy, with k $\in \{1,3\}$.

3.3.2 Results

Figure 4 shows that our Speculator successfully predicts the ground truth API call up to 46% of the time, despite our strict matching criterion. Accuracy improves significantly from top-1 to top-3 predictions, demonstrating that modest increases in speculation width yield substantial accuracy gains. Our speculation provides value by precomputing reasoning paths during otherwise idle API waiting time.

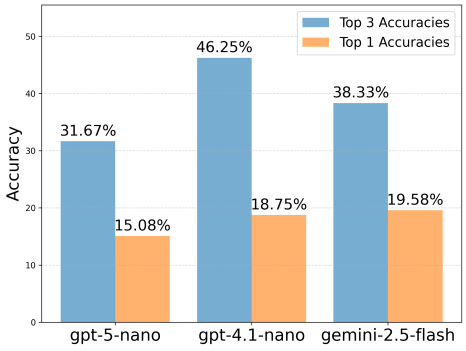
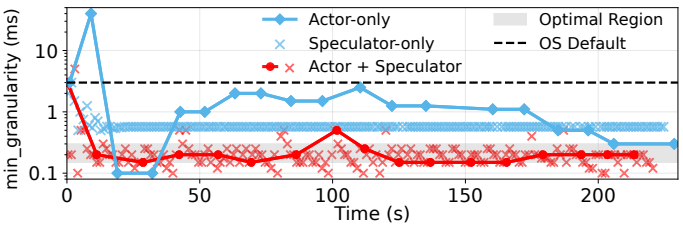


Figure 4: Accuracy with gemini-2.5-flash as the Actor. Speculating multiple actions (k=3) yields higher accuracy than predicting a single action.

Model Patterns We observe high variation of API decision across different Speculators. These are the result from phrasing discrepancies – some models phrase the calls concisely while some over-specify. Interestingly, stronger models often yield lower accuracy, as their more diverse and context-specific queries (e.g., "List of Nobel laureates in physics 1970s" vs. "1970s Nobel Prize Physics winners list") are penalized under strict matching. In contrast, weaker models tend to produce simpler, more predictable outputs.

4 Beyond Lossless Speculation: OS Hyperparameter Tuning Environment

Unlike the lossless speculation in preceding sections, we now demonstrate the benefits of a lossy approach where we relax the sequential step-by-step validation constraint. In latency-sensitive environments like an operating system, waiting for a powerful but slow Actor (10-15s deliberation) can leave the system in a poor state. Our framework uses a fast Speculator to apply immediate adjustments, improving performance in real-time while the Actor deliberates. This is made safe by a last-write-wins mechanism—the Actor's final decision simply overwrites any speculative action, re-



Configuration	Latency p95 (ms)
Untuned	102.97
Actor-only	54.00
Actor + Spec.	37.93

Figure 5: **(Left)** Comparison of **Speculator-Actor**, **Speculator-only**, and **Actor-only** convergence. The Speculator shortens time spent exploring poor settings. The Speculator-only agent stabilizes quickly but at a worse final value. **(Right)** Average p95 latency over a 20-second tuning experiment showing that rapid reaction offers immediate performance benefits (see §B.3.4). Lower is better.

moving the need for complex rollbacks. This method accelerates convergence and improves reaction time, which we test on `sysbench cpu`, a CPU-bound workload (Kopytov, 2020).

4.1 Experimental Setup

We tune a single parameter of Linux's Completely Fair Scheduler (CFS), `min_granularity`, which sets a task's minimum timeslice. This knob significantly influences scheduling behavior under our workload and past work by Liargkovas et al. (2025) has showed the potential of using an LLM agent to optimize it. Smaller timeslices typically improve latency but hurt throughput, creating a classic performance trade-off.

Our system consists of a fast Speculator and a slow Actor. The Speculator proposes a bounded parameter update each second using the latest performance metric. The Actor responds every 10–15 seconds after analyzing a compressed chronology of the Speculator's recent (measurement, action) pairs. When the Actor replies, its chosen setting is applied and its state updates the Speculator's context, ensuring subsequent fast steps proceed from a validated narrative rather than drifting.

Evaluation We evaluate our Speculator-Actor system against two baselines: an Actor-only agent (acting every 10–15s) and a Speculator-only agent (acting every 1s). Our evaluation shows that the Speculator: 1) improves reaction time, 2) accelerates convergence to the optimum, and 3) helps the system avoid the local minima that trap a Speculator-only agent.

4.2 Results

The Speculator dramatically improves reaction time, as observed Figure 5 (Right). During the recovery period, the full Speculator-Actor system maintained an average p95 latency of 37.93 ms. This is a substantial improvement over the Actor-only baseline, which was forced to endure the poor performance for much longer and averaged 54.00 ms. The Speculator's rapid correction prevents the system from lingering in a high-latency state (initially 102.97 ms), providing immediate benefits while the slower Actor deliberates (the full experiment is detailed in §B.3.4).

The Speculator's high-frequency updates also significantly accelerate convergence. In Figure 5 (Left), the Speculator-Actor system finds an optimal setting (e.g., 0.2 ms `min_granularity`) in approximately 10–15 seconds. In contrast, the Actor-only agent takes around 200 seconds—20x slowdown—and dwells for long periods in highly suboptimal states (e.g., latency $>$120 ms). The Speculator's rapid exploration provides the Actor a richer performance map, allowing it to steer the system away from these pathological regions more quickly.

Figure 5 also shows the Speculator-only agent settles quickly but sub-optimally, converging to a `min_granularity` of 0.55 ms (36.24 ms latency). This is significantly worse than the 0.2 ms (30.26 ms latency) achieved by the full Speculator-Actor system. Without the Actor's guidance, the Speculator lacks the reasoning depth to escape this local minimum.

Conclusion. We propose Speculative Actions, a lossless framework that parallelizes sequential decisions via fast top-K predictions. Evaluated across four settings, it yields consistent speedups and points toward promising optimization methods for more efficient real-world agentic systems.

Ethics Statement

We confirm that all authors have read and adhered to the ICLR Code of Ethics. Our work introduces a framework for accelerating AI agents, which has several ethical dimensions. The "lossy" OS tuning agent directly modifies a live system's parameters, which carries inherent risks if not properly managed. We address this by proposing safety mechanisms such as last-write-wins for easy rollbacks and discussing the importance of safety envelopes and repair paths, as detailed in our framework (§2) and the OS tuning case study (§4). While our e-commerce and human-expert experiments were conducted using simulators and consenting expert participants, the deployment of such agents in real-world human-in-the-loop systems would require further study into user experience and safety. The underlying LLMs may also inherit biases from their training data, and care must be taken to evaluate and mitigate these before deployment in sensitive, user-facing applications.

Reproducibility Statement

We are committed to ensuring the reproducibility of our work. To this end, all source code for our speculative action framework, experimental environments, and analysis scripts will be made publicly available upon acceptance. Our experiments are conducted on publicly available benchmarks, including TextArena for chess, τ-bench for e-commerce, and HotpotQA for web search, ensuring that other researchers can build upon our results. The OS tuning experiments use the open-source sysbench tool.

Details regarding the experimental setup, including hardware specifications, software versions, and specific configurations for each environment, are provided in the main text and extensively in §B. This includes the specific LLMs used (e.g., GPT and Gemini model families), reasoning budgets, and the exact prompts. The core logic of our framework is detailed in Algorithm 1. We believe these resources provide a clear path for reproducing our findings.

References

Reyna Abhyankar, Qi Qi, and Yiying Zhang. Osworld-human: Benchmarking the efficiency of computer-use agents. *arXiv preprint arXiv:2506.16042*, 2025.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, November 2024. Accessed: 2025-09-24.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Zhijun Chen, Jingzheng Li, Pengpeng Chen, Zhuoran Li, Kai Sun, Yuankai Luo, Qianren Mao, Ming Li, Likang Xiao, Dingqi Yang, Yikun Ban, Hailong Sun, and Philip S. Yu. Harnessing multiple large language models: A survey on llm ensemble, 2025. URL https://arxiv.org/abs/2502.18036.

Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pp. 1–14, July 2019. URL https://www.flux.utah.edu/paper/duplyakin-atc19.

Alvaro Estebanez, Diego R. Llanos, and Arturo Gonzalez-Escribano. A survey on thread-level speculation techniques. *ACM Comput. Surv.*, 49(2), June 2016. ISSN 0360-0300. doi: 10.1145/2938369. URL https://doi.org/10.1145/2938369.

Vivek Farias, Joren Gijsbrechts, Aryan Khojandi, Tianyi Peng, and Andrew Zheng. Speeding up policy simulation in supply chain rl. *arXiv preprint arXiv:2406.01939*, 2024.

Yichao Fu, Rui Ge, Zelei Shao, Zhijie Deng, and Hao Zhang. Scaling speculative decoding with lookahead reasoning. *arXiv preprint arXiv:2506.19830*, 2025.

Yilin Guan, Wenyue Hua, Qingfeng Lan, Sun Fei, Dujian Ding, Devang Acharya, Chi Wang, and William Yang Wang. Dynamic speculative agent planning, 2025. URL https://arxiv.org/abs/2509.01920.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. Textarena, 2025. URL https://arxiv.org/abs/2504.11442.

Wenyue Hua, Mengting Wan, Shashank Vadrevu, Ryan Nadel, Yongfeng Zhang, and Chi Wang. Interactive speculative planning: Enhance agent efficiency through co-design of system and user interface, 2024. URL https://arxiv.org/abs/2410.00079.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792/.

Tengjun Jin, Yuxuan Zhu, and Daniel Kang. Elt-bench: An end-to-end benchmark for evaluating ai agents on elt pipelines. *arXiv preprint arXiv:2504.04808*, 2025.

Kaggle. Game arena. https://www.kaggle.com/game-arena, 2025. Accessed: 2025-09-21.

Alexey Kopytov. Sysbench: Scriptable benchmark tool. https://github.com/akopytov/sysbench, 2020. Accessed: 2025-09-22.

Monica S Lam and Robert P Wilson. Limits of control flow on parallelism. *ACM SIGARCH Computer Architecture News*, 20(2):46–57, 1992.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Georgios Liargkovas, Konstantinos Kallas, Michael Greenberg, and Nikos Vasilakis. Executing shell scripts in the wrong order, correctly. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*, pp. 103–109, 2023.

Georgios Liargkovas, Vahab Jabrayilov, Hubertus Franke, and Kostis Kaffes. An expert in residence: Llm agents for always-on operating system tuning. In *Proceedings of the NeurIPS 2025 Workshop on Machine Learning for Systems (MLForSys)*, San Diego, CA, USA, December 2025. NeurIPS. Accepted paper.

Andrea Mambretti, Matthias Neugschwandtner, Alessandro Sorniotti, Engin Kirda, William Robertson, and Anil Kurmus. Speculator: a tool to analyze speculative execution attacks and mitigations. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 747–761, 2019.

Edmund B Nightingale, Daniel Peek, Peter M Chen, and Jason Flinn. Parallelizing security checks on commodity hardware. *ACM SIGARCH Computer Architecture News*, 36(1):308–318, 2008.

OpenAI. Introducing deep research. https://openai.com/index/introducing-deep-research/, 2025. Accessed: 2025-09-24.

Ya-Yunn Su, Mona Attariyan, and Jason Flinn. Autobash: improving configuration management with operating system causality analysis. *ACM SIGOPS Operating Systems Review*, 41(6):237–250, 2007.

Robert M Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of research and Development*, 11(1):25–33, 1967.

Jikai Wang, Juntao Li, Jianye Hou, Bowen Yan, Lijun Wu, and Min Zhang. Efficient reasoning for llms through speculative chain-of-thought, 2025a. URL https://arxiv.org/abs/2504.19095.

Zhihai Wang, Jie Wang, Jilai Pan, Xilin Xia, Huiling Zhen, Mingxuan Yuan, Jianye Hao, and Feng Wu. Accelerating large language model reasoning via speculative search, 2025b. URL https://arxiv.org/abs/2505.02865.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $$-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains, June 2024. URL http://arxiv.org/abs/2406.12045. arXiv:2406.12045.

Chen Zhang, Zhuorui Liu, and Dawei Song. Beyond the speculative game: A survey of speculative execution in large language models. *arXiv preprint arXiv:2404.14897*, 2024.

A Proof of Proposition 1

Proof. **Baseline.** In sequential execution, each of the T steps requires one call to the true model h with mean latency $1/\beta$. Therefore

$$E[T_{\text{seq}}] \;=\; \frac{T}{\beta}.$$

Block saving. Consider two consecutive steps $(t, t{+}1)$. In the baseline, the total completion time is $R = B + C$, where $B, C \sim \text{Exp}(\beta)$ are the latencies of step t and step $t{+}1$. With speculation, we launch $A \sim \text{Exp}(\alpha)$ during step t. If the guess is correct, the $(t{+}1)$ call C can be issued once either A or B finishes, so the block completes at

$$S \;=\; C + \min\{A, B\}.$$

Thus, the *block-level saving* (under a correct guess) is

$$R - S \;=\; (B - A)_+,$$

where $(x)_+ = \max\{x, 0\}$.

Expected block saving. By independence of A, B,

$$\mathbb{E}[(B - A)_+] = \int_0^\infty \int_0^b (b - a)\, \alpha e^{-\alpha a}\, \beta e^{-\beta b}\, da\, db = \frac{\alpha}{\beta(\alpha + \beta)}.$$

Per-boundary allocation. This saving spans a 2-step block. To avoid double-counting across successive blocks, we allocate the benefit symmetrically, assigning half to each adjacent boundary. Hence, the expected saving *per boundary, conditional on a correct guess*, is

$$\Delta_{\text{corr}} \;=\; \tfrac{1}{2}\, \frac{\alpha}{\beta(\alpha + \beta)}.$$

Total saving. With correctness probability p, the expected per-boundary saving is

$$\Delta = p\, \Delta_{\text{corr}} = \frac{p\, \alpha}{2\beta(\alpha + \beta)}.$$

There are $T - 1$ improvable boundaries, so

$$E[T_{\text{s}}] = \frac{T}{\beta} - (T - 1)\, \Delta = \frac{T}{\beta} - (T - 1)\, \frac{p\, \alpha}{2\beta(\alpha + \beta)}.$$

Final ratio. Dividing by $E[T_{\text{seq}}] = T/\beta$ gives

$$\frac{E[T_{\text{s}}]}{E[T_{\text{seq}}]} = 1 - \left(1 - \tfrac{1}{T}\right) \frac{p\, \alpha}{2(\alpha + \beta)}.$$

□

B Additional Environment Details

B.1 Chess

Trade-off between Time Saved and Token Cost. In addition to time savings, we also measure the additional token cost of parallel speculation. We track this using the metric **extra token percentage**: $(M_{sequantial} - M_{speculative})/M_{sequential}$. Figure 6 shows the trade-off between time saved and token cost for each number of predictions. We observe that the extra token percentage increases as the number of predictions increases, while the time saved percentage also increases, creating a clear trade-off.

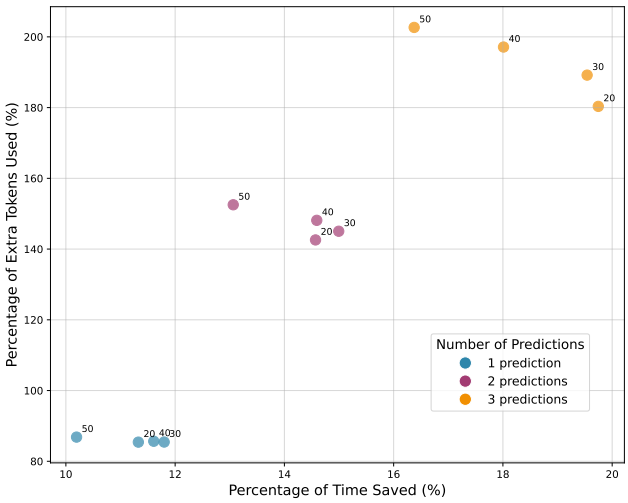
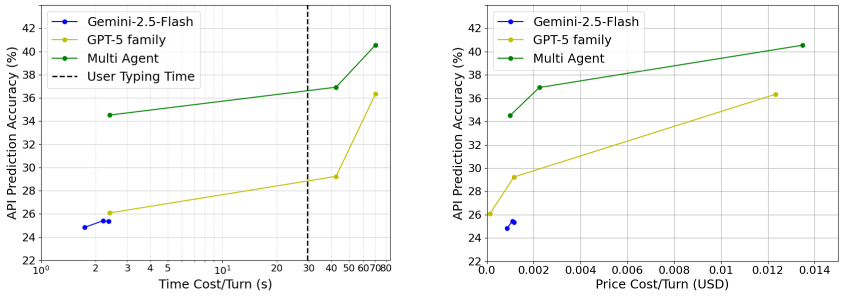


Figure 6: Percentage of extra tokens used against percentage of time saved for different numbers of predictions at different numbers of steps, averaged across 5 runs.

B.2 ECOMMERCE

τ-bench: A benchmark designed for dynamic task-oriented dialogues between a user (simulated by language models) and an API-augmented agent. The benchmark spans two domains — retail and airline, with structured databases, domain-specific tools. We focus on the retail domain, where the agent assists users with operations such as canceling or modifying pending orders, initiating returns or exchanges, or providing product and order information. The benchmark defines 115 tasks with 15 APIs (7 write, 8 read-only).

Trade-off between Prediction Accuracy and Cost. The time cost in Figure 7a consists of latency (Time to First Token) and output response time. The dashed vertical line represents the average user typing time, estimated at 40 words per minute. At this threshold, the multi-agent setting achieves approximately 34% prediction accuracy, meaning that in over one-third of cases the agent can return an immediate response without waiting for API execution. This demonstrates that speculation can transform user experience from perceptibly laggy to effectively real-time in tool-heavy environments.



(a) (b)

Figure 7: **Prediction Accuracy against Speculator's Cost across different models.** (a) Accuracy–Speculator time cost trade-off across models. The dashed line shows average user typing time. (d) Accuracy–Speculator price trade-off across models, reflecting the monetary cost of speculative execution.

B.3 Operating System Tuning

B.3.1 Experimental Setup and Implementation Details

System and Workload Configuration All experiments were conducted on a dedicated machine with 2× Intel Xeon Silver 4114 10-core CPUs at 2.20 GHz, 192 GB DDR4 RAM, and a 1 TB NVMe SSD running Ubuntu 22.04 with Linux Kernel 5.15, hosted on Cloudlab (Duplyakin et al., 2019).

We run `sysbench cpu` (Kopytov, 2020), a CPU-bound benchmark that repeatedly calculates a large prime number sequence. The benchmark reports several performance metrics every second. We run sysbench on 16 concurrent threads pinned on two CPU cores.

Tuner Implementation Details The system consists of two agents, a fast Speculator and a slow Actor, which collaborate to minimize the p95 latency of the workload. At each step, the tuner proposes a new configuration, which is applied to the live system. Applying the proposed parameters is a near-instant operation.

CFS Parameter Details The Completely Fair Scheduler (CFS) is a CPU scheduler for Linux that aims to give every task a fair share of CPU time. It exposes various hyperparameters that allow administrators to adjust its behavior. We tuned `min_granularity_ns`, which enforces a minimum timeslice a task will receive. The prompt templates guided the agents to explore a range from 50,000 to 50,000,000 nanoseconds (0.05 ms to 50 ms). The default value on Kernel 5.15 is 3 ms. Lower values for this parameter are expected to increase responsiveness at the cost of higher context-switching overhead, while higher values improve throughput but can worsen latency.

History Compression History compression is managed via distinct prompt structures for the two agents. When the slower Actor is invoked, its prompt context contains a fully compressed summary of all actions taken during its deliberation window. Each action from the Speculator is listed as a concise (parameter, result) pair. In contrast, the faster Speculator receives a hybrid context: it sees the same compressed history from the last Actor cycle, supplemented by the full, verbose replies from its own most recent actions. This dual-context mechanism allows the Actor to analyze long-term trends from a compact summary, while the Speculator retains immediate, detailed context for its rapid, reactive decisions.

B.3.2 Prompt Engineering for Multi-Agent Optimization

The following are the prompt templates used to guide the two LLM agents.

Initial System Prompt for Actor and Speculator

You are a Linux kernel scheduler tuning expert with deep knowledge of the Completely Fair Scheduler (CFS).

MULTI-AGENT ROLE: You are part of a MULTI-AGENT System.
[**For Actor**] You are the Actor. Your role is to provide thoughtful, well-analyzed parameter recommendations. You work alongside a Speculator that explores the parameter space rapidly. You will receive accumulated results from multiple agent calls to perform deeper analysis and identify trends.

[**For Speculator**] You are the Speculator. Your role is to provide immediate, intuitive parameter recommendations for each window. You work alongside an Actor that performs deeper analysis.

Your goal is to MINIMIZE p95 latency for a CPU-bound workload. The workload performance metrics might be NOISY, so look for consistent trends across configurations.

Tunable CFS parameter:

- `min_granularity_ns`: Minimum time slice before preemption. Lower values increase responsiveness but also overhead. Higher values improve throughput but can worsen latency.

Parameter Range:

- `min_granularity_ns`: 50,000 to 50,000,000 nanoseconds

Performance data will be provided in future calls. Respond ONLY in the format shown below:

```
Analysis: <Your one or two-sentence decision reasoning>
Config: { "min_granularity_ns": <int> }
```

Update for Speculator

[*Context includes the compressed history for calls 1-10 and the raw Speculator responses for iterations 11-18*]

CURRENT BEST: p95 latency=[value] at call #[value]

Latest Result for call #19:
Config: "min_granularity_ns": [value] → p95 latency=[value]

Please provide your analysis and the next configuration for iteration #20.

Update for Actor

[*Context includes the compressed history for calls 1-10*]

CURRENT BEST: p95 latency=[value] at call #[value]

RESULT for call #11 [SPECULATOR]: min_granularity_ns=[value] → p95 latency=[value]
RESULT for call #12 [SPECULATOR]: min_granularity_ns=[value] → p95 latency=[value]
...
RESULT for call #19 [SPECULATOR]: min_granularity_ns=[value] → p95 latency=[value]

Please provide your analysis of the trend and the next configuration for call #20.

Sample Agent Response

Analysis: The performance peaked at 300,000 ns, suggesting the optimal value is likely in that region. I will narrow the search around that peak.
Config: { "min_granularity_ns": 250000 }

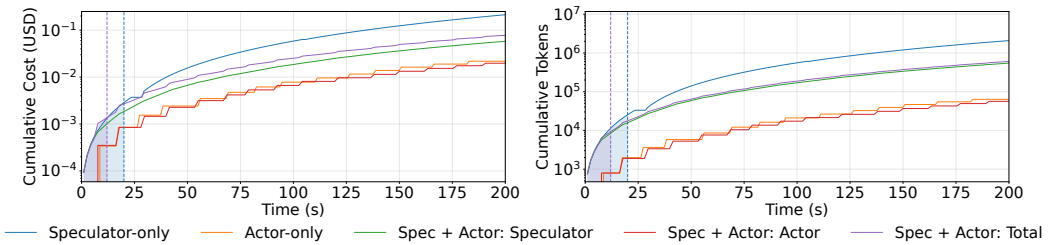


Figure 8: **Cumulative token usage and cost over time.** The left and right plots show the cumulative cost (USD) and total tokens used, respectively, for all three configurations. The vertical lines mark the observed convergence point for each system.

Table 2: Cumulative tokens and cost (in cents) at selected time marks.

	Actor-only		Speculator-only		Actor+Speculator (Total)	
Elapsed Time	**Tokens**	**Cost (cents)**	**Tokens**	**Cost (cents)**	**Tokens**	**Cost (cents)**
Base	744	0.02	690	0.01	690	0.03
10s	790	0.03	9,539	0.11	8,548	0.13
30s	3,631	0.15	45,768	0.57	32,459	0.48
60s	8,581	0.35	205,794	2.24	84,568	1.18
120s	26,398	0.96	778,253	8.12	261,855	3.53
200s	63,376	2.18	2,099,894	21.5	607,877	7.83

B.3.3 Token Usage and Costs

As illustrated in Figure 8 and detailed in Table 2, the high frequency of the Speculator leads to rapid growth in token consumption and cost. In practice, however, this growth is bounded by the system's fast convergence. The combined Actor-Speculator system converges in approximately 15 seconds, while the Speculator-only system converges in 20 seconds. The Actor-only system converges after 200 seconds. Once an optimal state is reached, the tuning process concludes, rendering the potential for long-term exponential cost negligible in this context. Several optimization strategies, like truncating the context to a fixed window or disabling exploration after convergence, could further mitigate token growth but are left for future work.

B.3.4 Speculative Reaction Time Benefits

To provide a targeted example of how speculation mitigates transient performance loss, we conducted a controlled experiment. In this scenario, the system is deliberately perturbed at time t_0 by setting the `min_granularity` parameter to a highly suboptimal value (10 ms). We then compare the system's recovery under two configurations: the Actor-Speculator system and an Actor-Only baseline, which replays only the actions proposed by the Actor from the full Actor-Speculator trace.

As shown in Figure 9, the Actor-Speculator system reacts almost instantly. The fast Speculator, seeing the immediate performance degradation, applies a corrective action that brings the system back to an efficient state in about one second. In contrast, the Actor-Only system is forced to endure the poor performance for over 10 seconds, as it must wait for the slower Actor to complete its deliberation cycle before it can act. The performance gap shown in the plot is quantified in the main text (Figure 5, Right).

C Multi-Step and Adaptive Speculation Extension

We can naturally extend Algorithm 1 to *multi-step speculation*, where the Speculator predicts not only the next step but up to s steps ahead. This yields a tree-search structure and requires sufficiently high success probability to justify deeper rollouts.

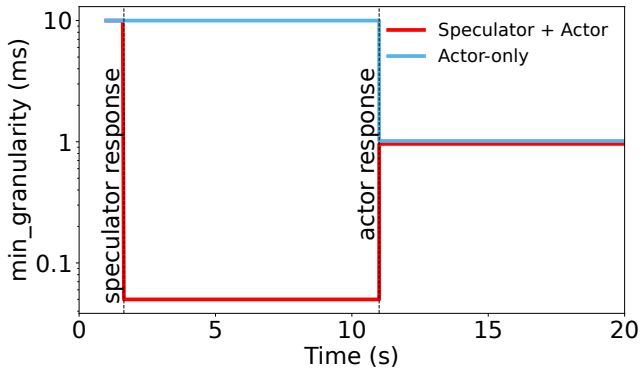


Figure 9: A controlled experiment showing the system's step response after a manual perturbation at $t = 0$. The **Actor-Speculator** system corrects the poor setting within a second, while the **Actor-only** system must wait over 10 seconds for its next decision cycle. The quantitative results of this experiment are summarized in Figure 5 (Right) in the main text.

This can be further combined with *adaptive speculation*: instead of generating k guesses for a_t uniformly, the Speculator also estimates confidence for each guess (e.g., via prompting LLMs or uncertainty-quantification methods). The most promising branches can then be expanded in a beam-search–like manner. Together, these ideas highlight the richness of speculative actions, which we leave for future work.