OBSSPEC: ACCELERATING AGENTIC REINFORCEMENT LEARNING WITH OBSERVATION SPECULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents alternate between model generation and tool execution, waiting for each tool call to complete before starting the next generation. This sequential execution is costly during agentic reinforcement learning (RL), which requires large batches of multi-turn rollouts. We introduce OBSSPEC, an observation speculation pipeline for agentic RL that uses a small world model to predict tool outputs while the environment executes, allowing the policy to begin its next generation early. OBSSPEC retains speculative work only when the predicted and actual observations match exactly, making rollout collection lossless, while co-training keeps the world model up to date as the policy learns. On terminal-use tasks, predictions from a 0.6B world model exactly match actual observations in 66.4% of attempts and arrive roughly 16 times faster. Using these predictions, OBSSPEC reduces agent latency by 9.3% and 14.2% for 8B and 32B policies. Across controlled experiments, we characterize the regimes in which OBSSPEC reduces agent latency under shared policy resources. Finally, because OBSSPEC overlaps policy generation with tool execution, it is architecturally compatible with existing techniques for accelerating RL.

1 INTRODUCTION

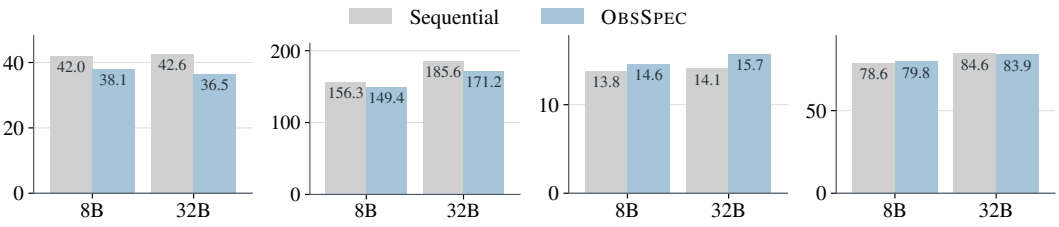
Tool-using agents solve complex tasks by alternating between model generation and tool execution (Yao et al., 2022; Schick et al., 2023; Qin et al., 2024). A terminal agent, for example, generates a shell command, waits for a sandbox to execute it, observes the output, and then generates its next command. In our experiments with terminal agents, tool execution makes up nearly half of the agent's latency. The cost is significant during agentic reinforcement learning (RL), where each policy update requires hundreds or even thousands of concurrent, multi-turn rollouts.

Observation speculation predicts a tool's output while the tool executes, allowing the policy to begin generating its next action before the actual observation arrives. Recent works apply this technique at inference time, substantially reducing agent latency (Ye et al., 2025; Cao et al., 2026; Saberi et al., 2026; Li et al., 2026; Chen et al., 2026a).

Applying observation speculation to RL introduces three requirements. First, rollout collection must be lossless: rollouts contain no mispredicted observations or actions generated from them. Second, speculation should accelerate rollout collection under shared policy resources. Because many rollouts run concurrently, speculative policy generations can slow other rollouts by competing for the same policy resources. Third, predictions must remain accurate as the policy learns. Policy updates can change the rollout distribution, so a frozen world model may become less accurate as training progresses.

We introduce OBSSPEC, an observation speculation pipeline for agentic RL designed around these requirements. A small world model predicts the next observation while the environment executes the corresponding action, allowing the policy to begin its next generation as soon as the prediction completes, without waiting for tool execution to finish. When the actual observation arrives, we retain the speculative generation only if the predicted and actual observations match exactly and discard it otherwise. This keeps rollout collection lossless. To limit contention with non-speculative policy generations, we bound how far ahead the pipeline speculates (depth) and how many alternative observations it predicts (breadth). We evaluate how these limits affect rollout latency under shared

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Agent latency (s) ↓. (b) Batch latency (s) ↓. (c) Async throughput (k rollouts/h) ↑. (d) Reward (%) ↑.

Figure 1: Across policy sizes, ObsSpec reduces latency and increases asynchronous rollout throughput with similar rewards.

policy resources. Finally, we co-train the world model with the policy, using actual observations as supervision to keep its predictions accurate as training proceeds.

Experiments on terminal-use tasks show that ObsSpec predicts tool outputs quickly and accurately. For example, predictions from a 0.6B world model exactly match actual observations in 66.4% of attempts and arrive roughly 16 times faster than tool execution. ObsSpec reduces agent latency by 9.3% and 14.2% for 8B and 32B policies, respectively, and increases asynchronous rollout throughput by up to 11.5%. By overlapping policy generation with tool execution, ObsSpec is architecturally compatible with other RL acceleration techniques, including fully asynchronous rollout scheduling (Fu et al., 2026; Piché et al., 2025), sample-efficient RL algorithms (Chen et al., 2025; Gao et al., 2025a; Shrivastava et al., 2026), low-precision inference (Xiao et al., 2023; Qiu et al., 2026), speculative decoding (Leviathan et al., 2023; Chen et al., 2026b; Pham et al., 2026), and multi-token prediction (Gloeckle et al., 2024; Wang et al., 2026).

In short, we make three contributions.

- We introduce ObsSpec, a lossless observation speculation pipeline for agentic RL that co-trains a lightweight world model with the policy.
- We show that tool outputs can often be predicted accurately and substantially faster than the corresponding tool calls can be executed. ObsSpec translates this opportunity into up to 14.2% lower agent latency and 11.5% higher asynchronous rollout throughput on terminal-use tasks.
- We show how world model size, speculation depth, and speculation breadth affect rollout latency under shared policy resources, and how co-training helps maintain prediction accuracy as the policy learns.

2 Approach

2.1 Problem Setup

To solve a task x, a tool-using agent takes a sequence of actions toward the task's goal. Each action is a tool call, and its output is an observation. At time step t, the agent conditions on the interaction history $h_t = (x, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$ and samples an action a_t. The environment $\mathcal{E}$ then executes a_t, updates its state from s_t to s_{t+1}, and returns an observation o_t:

$$a_t \sim \pi_\theta(\cdot \mid h_t), \qquad (s_{t+1}, o_t) \leftarrow \mathcal{E}(s_t, a_t). \tag{1}$$

For instance, in a terminal environment, a_t is a tool call containing a shell command, s_t includes the current filesystem and process state, and o_t contains the stdout and stderr returned by the sandbox. A complete sequence $\tau = (x, a_1, o_1, \ldots, a_T, o_T)$ constitutes a rollout and receives a reward $R(\tau)$ from a verifier. During RL, the current policy π_{θ_k} collects a batch of rollouts and uses their rewards in a policy update.

In the standard sequential pipeline shown in Figure 2, the policy cannot generate a_{t+1} until the tool has executed a_t and returned o_t, so policy generation and tool execution alternate on the critical

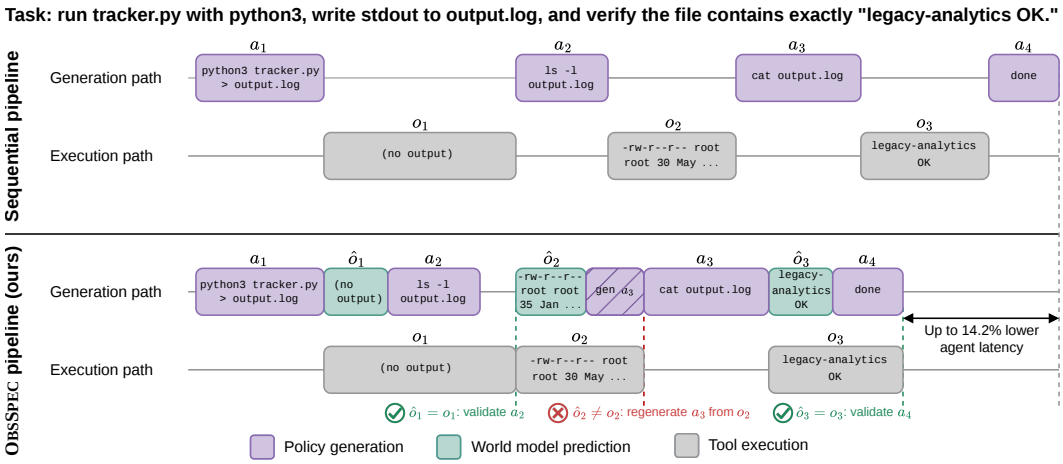


Figure 2: Comparison of the sequential and OBSSPEC pipelines on a terminal task. In the sequential pipeline (top), policy generation and tool execution alternate, so generation of each action cannot begin until the previous action finishes. In our OBSSPEC pipeline (bottom), a world model predicts each observation while the corresponding action executes, allowing the policy to generate the next action before the execution of the previous action finishes. The first predicted observation $\hat{o}_1$ matches the actual observation o_1, whereas $\hat{o}_2$ differs from o_2. Consequently, the in-progress generation of a_3 is canceled and restarted from the actual observation o_2. By overlapping policy generation with tool execution, OBSSPEC reduces agent latency by up to 14.2%.

path. Often, the time spent waiting on tool execution is significant. For example, on terminal-use tasks from Endless Terminals (Gandhi et al., 2026), nearly half of agent latency (47.1%) is spent waiting for sandbox execution.

2.2 SPECULATING TOOL OUTPUTS

Tool outputs are often highly predictable. In a terminal setting, for example, many commands such as changing directories return no output, while others repeat information already available in the history, such as the contents of a previously viewed file. If an observation can be predicted before the corresponding tool call finishes executing, the policy could begin generating its next action without waiting for the tool output. Recent works exploit this opportunity during inference through observation speculation: they predict tool outputs and start the next policy generation before the tool call finishes executing (Ye et al., 2025; Cao et al., 2026; Saberi et al., 2026; Li et al., 2026; Chen et al., 2026a). We study how to adapt this inference-time technique to an RL setting.

Applying observation speculation to RL introduces three requirements. First, lossless rollout collection requires that a rollout contain no mispredicted observations or actions generated from them. Existing inference-time methods for observation speculation may accept semantically similar observations (Saberi et al., 2026) or reuse speculative tool execution when the next action matches (Ye et al., 2025; Li et al., 2026), which can violate this requirement and corrupt training. Second, speculation should accelerate rollout collection under shared policy resources. Existing methods assume request latencies are independent, and evaluate proprietary models such as GPT-5, where large deployments absorb speculative generations without noticeably slowing other requests (Ye et al., 2025; Li et al., 2026), or estimate latency savings through simulation or trajectory replay (Saberi et al., 2026; Chen et al., 2026a). In contrast, RL systems generate many rollouts concurrently to make full use of the allocated policy resources (Sheng et al., 2025; Fu et al., 2026; Piché et al., 2025). Consequently, speculative requests compete with non-speculative requests for the same resources, so the gains from speculation must offset the slowdown in policy generation. Third, predictions must remain accurate as the policy learns. Policy updates can change the tool calls the agent makes, the states it visits, and the observations it receives. A world model fit to an earlier distribution may consequently become less accurate, making speculation less effective.

To address these challenges, we propose OBSSPEC, an observation speculation pipeline shown in Figure 2. After the policy samples a_t, the pipeline starts executing it in the environment and simultaneously asks a small world model q_ϕ to predict the resulting observation $\widehat{o}_t \sim q_\phi(\cdot \mid h_t, a_t)$. If tool execution finishes first, we cancel the prediction, append o_t to the history, and sample $a_{t+1} \sim \pi_\theta(\cdot \mid h_{t+1})$, where $h_{t+1} = (h_t, a_t, o_t)$. If the prediction finishes first, we form the speculative history $\widehat{h}_{t+1} = (h_t, a_t, \widehat{o}_t)$ and begin sampling $a_{t+1} \sim \pi_\theta(\cdot \mid \widehat{h}_{t+1})$ while the tool continues executing.

When the actual observation arrives, we compare it with the prediction. If $\widehat{o}_t = o_t$, we retain the speculative generation of a_{t+1}, waiting for it to finish if necessary. Otherwise, we discard that generation and regenerate a_{t+1} from the actual history h_{t+1}. Because actions can mutate the environment state, we execute them sequentially, checking each prediction before executing the next action. This procedure ensures that rollout collection is lossless: the rollout contains no mispredicted observations or actions generated from them. A wrong prediction therefore wastes speculative computation without corrupting the rollout. Pseudocode for OBSSPEC is in Algorithm 1.

We use speculation depth and breadth to limit how much speculative work the pipeline generates. These limits matter because concurrent RL rollouts share a finite amount of policy resources (Sheng et al., 2025; Fu et al., 2026; Piché et al., 2025). Speculation depth D allows speculative work to advance by alternating predicted observations and conditional policy actions, with at most D actions generated ahead of actual execution (Saberi et al., 2026). Speculation breadth K instead produces up to K candidate predicted observations and launches a speculative policy generation from each distinct candidate (Ye et al., 2025; Li et al., 2026; Chen et al., 2026a). Increasing depth or breadth creates more opportunities for overlap, but also requires more policy generations, which can slow other rollouts even when predictions are correct. We evaluate this tradeoff in Section 3.5.

2.3 CO-TRAINING THE POLICY AND WORLD MODEL

To keep predictions accurate as the policy learns, we co-train the policy and world model on the same rollouts. Each rollout provides rewards for the policy update (e.g., via GRPO (Shao et al., 2024; Yu et al., 2026a)) and actual observations for world model training. Given a rollout tokenized as $z_{1:N}$, we train the world model with the masked cross-entropy loss

$$\mathcal{L}_{\mathrm{WM}}(\phi) = -\frac{1}{\sum_i m_i} \sum_{i=1}^{N} m_i \log q_\phi(z_i \mid z_{<i}). \tag{2}$$

Here, m_i is a mask indicating whether z_i is an observation token. We include the end-of-observation token (e.g., `</tool_response>`) in the mask so that it serves as the stop token during world model generation.

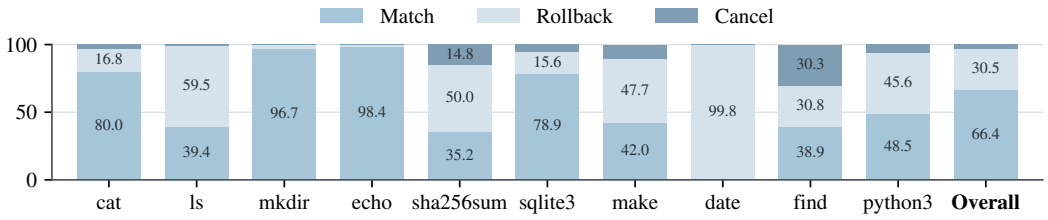
At training iteration k, we collect a batch of rollouts using policy π_{θ_k} and world model q_{ϕ_k}. Using this batch, the policy and world model are updated concurrently: rollout rewards update the policy to $\pi_{\theta_{k+1}}$, while the actual observations update the world model to $q_{\phi_{k+1}}$ using $\mathcal{L}_{\mathrm{WM}}$. Co-training lets the world model track the policy's changing rollout distribution.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Agent. Each task provides the agent with a natural-language instruction and two tools: `bash` and `done`. The `bash` tool executes a shell command in a stateful sandbox and returns its stdout, stderr, and exit code. When the agent believes the task is complete, it calls `done`.

Tasks. Our main experiments evaluate OBSSPEC on long-horizon terminal-use tasks from Endless Terminals (Gandhi et al., 2026). For example, the task in Figure 2 asks the agent to execute a Python program, redirect its output to a file, and verify the file's contents. Solving such tasks may require inspecting existing files, executing programs, transforming their outputs, and checking the resulting state across multiple shell commands. Each task includes test cases that inspect the final sandbox state and award a binary reward.



(a) Outcomes (%) by command.



(b) Latency (s) by command (log scale).

Figure 3: Observation prediction outcomes (top) and latency (bottom) for common commands.

Policy and World Models. We use Qwen3-8B and Qwen3-32B as policies and Qwen3-0.6B as the world model (Yang et al., 2025). Unless otherwise noted, policies are trained for 500 steps with GRPO (Shao et al., 2024; Yu et al., 2026a). Specifically, we train each policy and save its rollouts, and then replay the rollouts in the same order to train world models offline. This procedure produces the same artifacts as the co-training procedure outlined in Section 2.3, but allows us to compare speculation configurations under the same fixed policy, isolating the effect of speculation from differences in policy behavior.

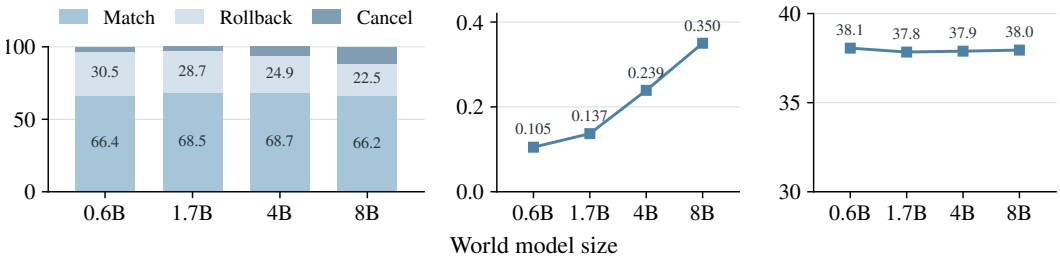
Speculation configuration. Unless otherwise specified, OBSSPEC uses depth $D = 1$ and breadth $K = 1$. The sequential baseline uses the same policy, sampling parameters, evaluation prompts, and policy inference resources, but waits for every tool output before beginning the next policy generation.

Evaluation protocol and metrics. Our standard protocol mimics synchronous RL rollout generation with 4,096 rollouts partitioned into 16 batches. Each batch contains 16 held-out prompts and 16 sampled rollouts per prompt; batch assignments are identical across all experiments. The primary metric in these experiments is agent latency: wall-clock time spent generating policy actions and executing tool calls, excluding sandbox setup and final verification. Batch latency and asynchronous rollout throughput are measured on a larger set of 65,536 rollouts, and unlike agent latency, both include sandbox setup and verification time. We also measure the mean reward obtained by the policy. We classify each speculation attempt into three mutually exclusive outcomes: *match*, when the predicted observation exactly equals the actual observation; *rollback*, when the observations differ; and *cancel*, when the tool finishes before the world model. To quantify policy inference contention and wasted computation, we report per-request decode rate and discarded tokens for the policy. For each metric, we report the mean and standard error in Appendix C.

Serving and sandbox infrastructure. We use VERL to orchestrate training and evaluation workloads (Sheng et al., 2025). Policies and world models are served asynchronously with vLLM (Kwon et al., 2023). The 8B and 32B policies are allocated eight and sixteen GPUs, respectively, and world models are allocated their own eight GPUs. Most experiments use our default sandbox service. We additionally evaluate Modal and Daytona in Section 3.7.

3.2 OBSERVATION PREDICTIONS ARE FAST AND OFTEN MATCH

Observation speculation can help only when tool outputs are both predictable and predicted before tool execution finishes. We first evaluate these two prerequisites using the 8B policy and 0.6B world model. Across 47,762 speculation attempts, 66.4% match exactly, 30.5% roll back, and only



(a) Prediction outcomes. (b) WM latency (s). (c) Agent latency (s).

Figure 4: Effect of world model size with the 8B policy at $D = K = 1$. Larger models reduce rollbacks but take longer to predict and do not reduce agent latency.

3.1% are canceled because execution finishes first. Thus, the world model finishes before the tool in 96.9% of attempts and exactly matches the actual observation in approximately two thirds of attempts. Among completed predictions, mean world model latency is 0.10 seconds, compared with 1.69 seconds for tool execution. These results establish that many tool outputs can be predicted accurately and substantially faster than the corresponding tool calls can be executed.

Figure 3 demonstrates this across a selection of common shell commands. `cat`, which accounts for 40.9% of tool calls, has an 80.0% match rate because files are often created or inspected earlier in the rollout. Moreover, outputs from commands which normally return no output, such as `mkdir`, are highly predictable and much faster to predict than to obtain through tool execution. On the other hand, outputs from commands such as `date` are nearly impossible to predict. However, predictability alone does not determine speculation value. Python commands have a match rate of only 48.5%, but the gap between execution latency and prediction latency is significant, leaving more time to generate the next action while the tool executes. Conversely, a command can produce a highly predictable output but be too fast or too close to the end of a rollout to materially affect latency.

3.3 ObsSpec Improves Rollout Latency and Throughput

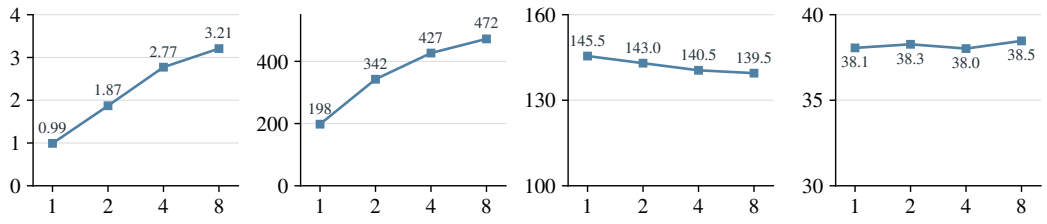
We next evaluate whether ObsSpec converts fast, accurate observation predictions into lower agent latency and higher rollout throughput. Figure 1 summarizes the results using a small 0.6B world model. With the 8B policy, ObsSpec reduces agent latency by 9.3% and synchronous batch latency by 4.4%, and increases asynchronous rollout throughput by 5.9%. Similarly, with the 32B policy, ObsSpec reduces agent latency by 14.2% and synchronous batch latency by 7.8% and increases asynchronous rollout throughput by 11.5%. The mean rewards are similar across pipelines, as expected because the rollout collection is lossless. We hypothesize that the larger gains with the 32B policy arise because slower generation would leave more policy work available to overlap with tool execution.

3.4 Larger World Models Do Not Reduce Agent Latency

Larger world models reduce rollbacks but take longer to generate predictions. Figure 4 demonstrates this trade-off across four world models ranging from 0.6B to 8B parameters. As we scale the world model, rollbacks fall from 30.5% to 22.5%. At the same time, mean prediction latency increases from 0.105 to 0.350 seconds, and the cancel rate rises from 3.1% to 11.4% because tool execution is more likely to finish before the prediction. These effects largely cancel, with all four world models producing nearly identical agent latencies. We therefore prefer the 0.6B world model because it provides a similar reduction in agent latency for substantially less compute.

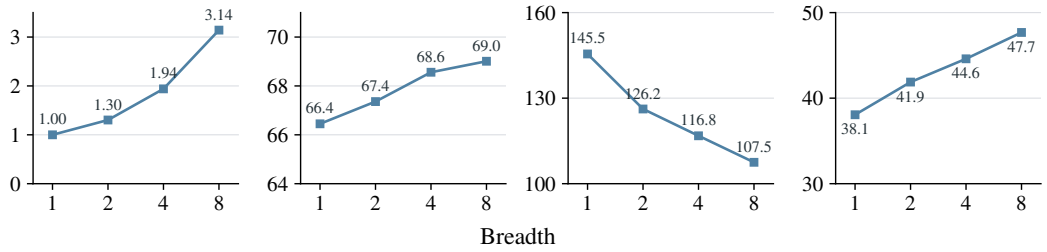
3.5 Less Is More Under Shared Policy Resources

Speculation depth. Increasing depth allows ObsSpec to speculate further ahead by repeatedly alternating predicted observations with conditional policy actions. However, in Figure 5, agent latency changes little across $D \in \{1, 2, 4, 8\}$. The pipeline also realizes substantially less depth than



(a) Realized depth. (b) Discarded policy tokens. (c) Policy decode rate (tok/s). (d) Agent latency (s).

Figure 5: Depth sweep with the 8B policy and 0.6B world model. Greater depth increases realized speculation and discarded policy computation without reducing agent latency.



(a) Realized breadth. (b) Match (%). (c) Policy decode rate (tok/s). (d) Agent latency (s).

Figure 6: Breadth sweep with the 8B policy and 0.6B world model. Greater breadth increases realized speculation and match rate, while reducing the policy decode rate and increasing agent latency.

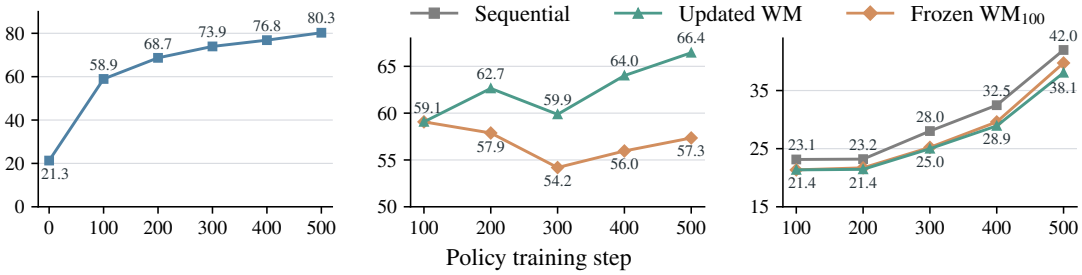
configured: increasing D from 1 to 8 raises the mean realized depth from 0.99 to 3.21, since speculation stops whenever tool execution catches up or an incorrect prediction invalidates the speculative work. Deeper speculation also makes rollbacks more expensive because all work downstream of a mismatched observation is discarded. Increasing D from 1 to 8 raises discarded policy tokens from 198 to 472 per rollout. As a result, policy decode rate falls from 145.5 to 139.5 tok/s. Greater depth consequently adds discarded computation and contention without reducing agent latency, making $D = 1$ our preferred configuration.

Speculation breadth. Increasing breadth allows OBSSPEC to sample multiple candidate observations and generate a conditional action for each distinct candidate. However, the additional samples are frequently duplicates. At $K = 8$, deduplication leaves only 3.14 distinct candidates per attempt on average, and the share of attempts with a match rises by only 2.6 percentage points over $K = 1$. This marginal gain introduces substantial contention, with agent latency increasing by 25.3% at $K = 8$, relative to $K = 1$. Over the same range, the policy decode rate falls by 26.1%, and discarded policy tokens increase by roughly 5x. Thus, $K = 1$ is preferred.

3.6 WORLD MODELS SHOULD TRACK THE POLICY

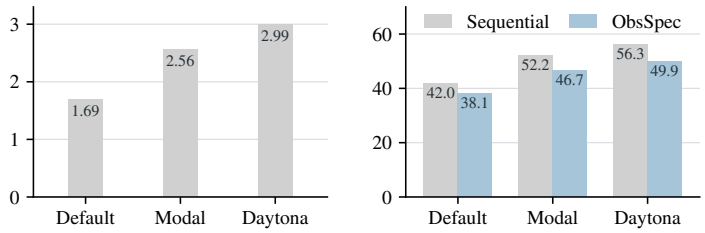
As the policy improves (Figure 7a), it changes which tool calls it makes and therefore which tool outputs the world model needs to predict. Co-training the world model with the policy allows the world model to adapt to these changes. To evaluate the benefit of co-training, we compare a world model trained through the same step as the policy with a world model frozen after policy step 100. In Figure 7b, the gap in prediction accuracy between the frozen and updated world models widens with the number of policy training steps. The gap grows from 4.8 percentage points at step 200 to 9.1 points at step 500.

This materially affects the agent latency of OBSSPEC, as seen in Figure 7c. While speculating with either world model consistently outperforms the sequential baseline, the difference in agent latency between the two world models widens during training. Concretely, the latency advantage of the



(a) Reward (%). (b) Match (%). (c) Agent latency (s).

Figure 7: Policy improvement and the effects of co-training the world model on match rate and agent latency.



(a) Tool latency (s). (b) Agent latency (s).

Figure 8: Sensitivity to sandbox infrastructure with the 8B policy and 0.6B world model.

updated world model over the frozen world model grows from 0.3 seconds at policy step 200 to 1.7 seconds at step 500. This widening gap suggests that keeping the world model up to date increases the match rate and decreases agent latency.

3.7 ObsSpec Is Robust to Sandbox Latency

Figure 8 tests whether ObsSpec remains effective across sandbox providers with different execution latencies. On the 8B policy, tool latency increases from 1.69 seconds with our default sandbox service to 2.56 seconds on Modal and 2.99 seconds on Daytona. ObsSpec consistently reduces agent latency, by 9.3% with our default sandbox service, 10.6% on Modal, and 11.3% on Daytona. The gains increase only modestly because speculation at $D = 1$ can hide at most one conditional policy generation. With our default sandbox service, the next action finishes generating before the tool returns in 64.6% of speculation attempts, leaving little additional policy work to overlap during longer tool executions.

3.8 Not All Tasks Benefit from ObsSpec

We also evaluate an additional mathematical reasoning workload: tool-integrated reasoning (TIR) (Gou et al., 2024; Yang et al., 2024) on competition-level mathematical problems from DAPO-Math-17k (Yu et al., 2026a). On this task, the policy interleaves natural-language reasoning with code execution for arithmetic, case enumeration, and intermediate verification before writing its final answer to `/home/user/answer.txt`. We follow the same experimental configuration as in our main experiments. Agent latency is 55.82 seconds with the sequential pipeline and 56.42 seconds with ObsSpec using a 0.6B world model.

We identify three reasons why ObsSpec is ineffective on TIR. First, the workload provides fewer opportunities to speculate than the terminal-use tasks, averaging only 3.44 tool calls per rollout compared with 11.73. Second, tool execution accounts for only 21.2% of TIR agent latency, versus 47.1% on the terminal-use tasks, leaving a much smaller fraction of the critical path available for overlap. Third, its observations are harder to predict. With a 0.6B world model, the match rate falls from 66.4% on the terminal-use tasks to 55.0% on TIR. Scaling the world model to 8B raises the

match rate to 59.5% but triples prediction latency, leaving agent latency effectively unchanged at 55.79 seconds.

4 Related Work

Speculative execution for agents. Prior work accelerates tool-using agents by speculating either future actions or future observations. Action speculation predicts and executes future tool calls while the policy generates the current action (Sui et al., 2026; Nichols et al., 2025; Bai et al., 2026; Huang et al., 2025; Zhong et al., 2026; Liu et al., 2026b). These methods require special care for tools that mutate the environment's state. Observation speculation instead predicts the output of the current tool call and begins the next policy generation before execution finishes. Speculative Actions (Ye et al., 2025) increases speculative breadth by predicting multiple candidate outputs from a pending tool call and pre-executing the next call implied by each. SpecHop increases speculative depth across multiple retrieval steps and verifies them asynchronously (Saberi et al., 2026). Speculate with Memory stores memories of prior transitions to improve speculation accuracy (Li et al., 2026). More recently, AOSpec combines action and observation speculation and, like ObsSpec, requires the predicted observation to exactly match the actual observation (Chen et al., 2026a). ObsSpec studies observation speculation for RL under three requirements: lossless rollout collection, faster rollout collection under shared policy resources, and accurate predictions as the policy learns. We retain speculative generations only when predicted and actual observations match exactly, limit speculation to manage contention among concurrent rollouts, and co-train the world model with the policy. Our experiments measure how these choices affect rollout latency and throughput.

Efficient RL rollouts. An extensive body of work accelerates RL rollouts. Asynchronous systems overlap rollout collection with policy updates (Fu et al., 2026; Piché et al., 2025), and sample-efficient algorithms reduce the number of rollouts required for learning (Chen et al., 2025; Gao et al., 2025a; Shrivastava et al., 2026). Other methods accelerate policy inference through low-precision arithmetic (Xiao et al., 2023; Qiu et al., 2026), speculative decoding (Leviathan et al., 2023; Chen et al., 2026b; Zhang et al., 2026; He et al., 2025; Shao et al., 2026; Chang et al., 2026; Liu et al., 2025; Pham et al., 2026), or multi-token prediction (Gloeckle et al., 2024; Wang et al., 2026). These methods reduce the time or compute required for RL and can be combined with ObsSpec, which overlaps policy generation with environment execution.

World models for agents. Prior work uses world models to support planning and agent training. WMA (Chae et al., 2025) and WebDreamer (Gu et al., 2024) predict the results of candidate web actions to improve action selection. WebEvolver uses a co-evolving world model to generate training data for the policy and to simulate future observations during inference (Fang et al., 2025), while CoMAP uses predicted feedback to refine actions (Liu et al., 2026a). GTM (Ren et al., 2025), Qwen-AgentWorld (Zuo et al., 2026), WebWorld (Xiao et al., 2026), WebSynthesis (Gao et al., 2025b), and World Model RL (Yang et al., 2026) use world models to simulate the environment for agent training rather than call real tools. ECHO (Shrivastava et al., 2026), PaW (Lu et al., 2026), and RWML (Yu et al., 2026b) train the policy itself to predict observations as an auxiliary objective. ObsSpec uses a separate lightweight world model to reduce agent latency.

5 Conclusion

We presented ObsSpec, an observation speculation pipeline for agentic RL that overlaps policy generation with environment execution while keeping rollout collection lossless. On terminal-use tasks with 8B and 32B policies, a small co-trained world model reduces both agent and batch latency and increases asynchronous rollout throughput. Our experiments show that increasing speculation depth or breadth increases contention for shared policy resources and can even increase agent latency. As the policy learns, co-training keeps the world model's predictions accurate and reduces agent latency compared with a frozen world model. Together, these results establish observation speculation as a new direction for improving the efficiency of agentic RL. A promising future direction is to speculate selectively based on a tool call's output predictability, expected execution time, and the current utilization of policy resources.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods and clean and reformat datasets. We have not used generative AI tools to design or provide feedback on research methodology or experiments or interpret results. The use of generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, assist with translation, and support qualitative and thematic data analysis is not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, creating or editing software code, editing a research paper to improve readability, and identifying relevant literature. We have reviewed all AI-assisted work. AI-generated code, figures, and writing were manually checked against experimental outputs by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We take great care to ensure our work is reproducible. Section 2 and Algorithm 1 describe our proposed method. Section 3.1 and Table 1 detail the experimental configurations and hyperparameters. Appendix C reports the complete results with standard errors. Code and artifacts will be released upon acceptance.

REFERENCES

Huajun Bai, Weiwei Lv, Huichuan Zheng, Youyou Lu, and Jiwu Shu. Spork: Self-speculative forking to accelerate agentic llm inference. *arXiv preprint arXiv:2607.03333*, 2026.

Sheng Cao, Zhao Chang, Chang Li, Hannan Li, Liyao Fu, and Ji Tang. The auton agentic ai framework. *arXiv preprint arXiv:2602.23720*, 2026.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025.

Chi-Chih Chang, Siqi Zhu, Zhichen Zeng, Haibin Lin, Jiaxuan You, Mohamed S Abdelfattah, Ziheng Jiang, and Xuehai Qian. Srt: Accelerating reinforcement learning via speculative rollout with tree-structured cache. *arXiv preprint arXiv:2601.09083*, 2026.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Hao Mark Chen, Jinnan Guo, Wayne Luk, and Hongxiang Fan. Aospec: Action and observation co-speculation for low-latency agent serving. *arXiv preprint arXiv:2608.00881*, 2026a.

Qiaoling Chen, Zijun Liu, Peng Sun, Shenggui Li, Guoteng Wang, Ziming Liu, Yonggang Wen, Siyuan Feng, and Tianwei Zhang. Respec: Towards optimizing speculative decoding in reinforcement learning systems. *Proceedings of Machine Learning and Systems*, 8:367–379, 2026b.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. Webevolver: Enhancing web agent self-improvement with co-evolving world model. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8970–8986, 2025.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *Advances in Neural Information Processing Systems*, 38:36256–36282, 2026.

Kanishk Gandhi, Shivam Garg, Noah D Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.

Chang Gao, Chujie Zheng, Xiong-Hui Chen, Kai Dang, Shixuan Liu, Bowen Yu, An Yang, Shuai Bai, Jingren Zhou, and Junyang Lin. Soft adaptive policy optimization. *arXiv preprint arXiv:2511.20347*, 2025a.

Yifei Gao, Junhong Ye, Jiaqi Wang, and Jitao Sang. Websynthesis: World-model-guided mcts for efficient webui-trajectory synthesis. *arXiv preprint arXiv:2507.04370*, 2025b.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, volume 2024, pp. 48362–48395, 2024.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl. *arXiv preprint arXiv:2508.18588*, 2025.

Zixiao Huang, Wen Zeng, Tianyu Fu, Tengxuan Liu, Yizhou Sun, Ke Hong, Xinhao Yang, Chengchun Liu, Yan Li, Quanlu Zhang, et al. Reducing latency of llm search agent via speculation-based algorithm-system co-design. *arXiv preprint arXiv:2511.20048*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

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.

Yu Li, Qinyuan Ye, Prafulla Kumar Choubey, Jiaxin Zhang, and Chien-Sheng Wu. Speculate with memory: Lossless acceleration for llm agents. *arXiv preprint arXiv:2607.12236*, 2026.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Xu Han, Peng Li, Anxiang Zeng, and Jinsong Su. Spec-rl: Accelerating on-policy reinforcement learning with speculative rollouts. *arXiv preprint arXiv:2509.23232*, 2025.

Youwei Liu, Jian Wang, Hanlin Wang, and Wenjie Li. Comap: Co-evolving world models and agent policies for llm agents. *arXiv preprint arXiv:2606.02372*, 2026a.

Zeyu Liu, Souvik Kundu, and Peter A Beerel. Speculative macro commit for faster tool-using agents. *arXiv preprint arXiv:2609.03236*, 2026b.

Ning Lu, Baijiong Lin, Shengcai Liu, Jiahao Wu, Haoze Lv, Yanbin Wei, Lingting Zhu, Shengju Qian, Xin Wang, Ying-Cong Chen, et al. Policy and world modeling co-training for language agents. *arXiv preprint arXiv:2606.02388*, 2026.

Daniel Nichols, Prajwal Singhania, Charles Jekel, Abhinav Bhatele, and Harshitha Menon. Optimizing agentic language model inference via speculative tool calls. *arXiv preprint arXiv:2512.15834*, 2025.

Nhat Minh Pham, Duy Tung Doan, Thi Duyen Ngo, Vinh Van Nguyen, and Khac-Hoai Nam Bui. Specroll: Fast-slow verifier-feedback adaptation for speculative reinforcement learning rollouts. *arXiv preprint arXiv:2608.04962*, 2026.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation. *arXiv preprint arXiv:2509.19128*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Zhaopeng Qiu, Shuang Yu, Jingqi Zhang, Shuai Zhang, Xue Huang, Jingyi Yang, and Junjie Lai. Fp8-rl: A practical and stable low-precision stack for llm reinforcement learning. *arXiv preprint arXiv:2601.18150*, 2026.

Zhenzhen Ren, Xinpeng Zhang, Zhenxing Qian, Yan Gao, Yu Shi, Shuxin Zheng, and Jiyan He. Gtm: Simulating the world of tools for ai agents. *arXiv preprint arXiv:2512.04535*, 2025.

Mehrdad Saberi, Keivan Rezaei, and Soheil Feizi. Spechop: Continuous speculation for accelerating multi-hop retrieval agents. *arXiv preprint arXiv:2605.21965*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpay Ariyak, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, Ce Zhang, et al. Beat the long tail: Distribution-aware speculative decoding for rl training. *Proceedings of Machine Learning and Systems*, 8:1626–1638, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Vaishnavi Shrivastava, Piero Kauffmann, Ahmed Awadallah, and Dimitris Papailiopoulos. Echo: Terminal agents learn world models for free. *arXiv preprint arXiv:2605.24517*, 2026.

Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, Kaiqiang Xu, Kai Chen, and Yuqing Yang. Parallelizing tool execution and llm generation for low-latency agent serving. *arXiv preprint arXiv:2603.18897*, 2026.

Ke Wang, Aohan Zeng, Zhengxiao Du, Yuxuan Hu, Bohan Zhang, Xinyi Wang, Jie Tang, and Jing Zhang. Mtp-rl: Acceleration of reinforcement learning rollouts with policy-aligned multi-token prediction. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37530–37542, 2026.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Zikai Xiao, Jianhong Tu, Chuhang Zou, Yuxin Zuo, Zhi Li, Peng Wang, Bowen Yu, Fei Huang, Junyang Lin, and Zuozhu Liu. Webworld: A large-scale world model for web agent training. *arXiv preprint arXiv:2602.14721*, 2026.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Xiyuan Yang, Sheikh Sarwar, Jingru Cheng, Zhan Shi, Duanshun Li, Huiyuan Chen, Haiyang Zhang, Chenlei Guo, Jingrui He, and Zhenyu Liao. Scaling automatic research agents via world models. *arXiv preprint arXiv:2608.12564*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster agentic systems. *arXiv preprint arXiv:2510.04371*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026a.

Xiao Yu, Baolin Peng, Ruize Xu, Yelong Shen, Pengcheng He, Suman Nath, Nikhil Singh, Jiangfeng Gao, and Zhou Yu. Reinforcement world model learning for llm-based agents. *arXiv preprint arXiv:2602.05842*, 2026b.

Yizhou Zhang, Ning Lv, Teng Wang, and Jisheng Dang. Fastgrpo: Accelerating policy optimization via concurrency-aware speculative decoding and online draft learning. In *International Conference on Learning Representations*, volume 2026, pp. 59620–59636, 2026.

Shuzhang Zhong, Baotong Lu, Qi Chen, Chuanjie Liu, Fan Yang, and Meng Li. Dualspec: Accelerating deep research agents via dual-process action speculation. *arXiv preprint arXiv:2603.07416*, 2026.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, et al. Qwen-agentworld: Language world models for general agents. *arXiv preprint arXiv:2606.24597*, 2026.

A ObsSpec Algorithm

Algorithm 1 ObsSpec rollout collection for $D = K = 1$.

Inputs instruction x, policy π_θ, world model q_ϕ, environment $\mathcal{E}$, initial state s_1

$t \leftarrow 1$, $h_1 \leftarrow (x)$, and sample $a_t \sim \pi_\theta(\cdot \mid h_t)$
while a_t is not `done` and the rollout budget remains **do**
  Start environment execution $(s_{t+1}, o_t) \leftarrow \mathcal{E}(s_t, a_t)$ asynchronously
  Start prediction $\widehat{o}_t \sim q_\phi(\cdot \mid h_t, a_t)$ asynchronously
  Wait until the environment execution or prediction finishes
  if the prediction finishes first **then**
    Start sampling $a_{t+1} \sim \pi_\theta(\cdot \mid h_t, a_t, \widehat{o}_t)$ asynchronously
  else
    Cancel the prediction
  end if
  Await the environment execution to obtain o_t
  $h_{t+1} \leftarrow (h_t, a_t, o_t)$
  if the prediction finished first and $\widehat{o}_t = o_t$ **then**
    Await the speculative generation of a_{t+1}
  else
    Cancel speculative generation if it is active
    Sample $a_{t+1} \sim \pi_\theta(\cdot \mid h_{t+1})$
  end if
  $t \leftarrow t + 1$
end while
return (h_t, a_t)

B TRAINING AND SERVING CONFIGURATION

Parameter	Value
Data	
Batch size	16 prompts
Group size	16 rollouts per prompt
Prompt budget	2,048 tokens
Response budget	16,384 tokens
Per-turn generation budget	2,048 tokens
Max turns	32 turns
Policy optimization	
Objective	GRPO
Optimizer	AdamW
Learning rate	1×10^{-6}
Learning-rate schedule	Constant with no warmup
Adam (β_1, β_2)	$(0.9, 0.999)$
Weight decay	0.01
Gradient clipping	1.0
Loss aggregation	token-mean
PPO epochs	1
GRPO clip range	$[0.20, 0.28]$
KL coefficient	0.0
Entropy coefficient	0.0
Training steps	500
World model optimization	
Objective	Masked cross-entropy
Optimizer	AdamW
Learning rate	1×10^{-5}
Learning-rate schedule	Constant with no warmup
Adam (β_1, β_2)	$(0.9, 0.999)$
Weight decay	0.01
Gradient clipping	1.0
Loss aggregation	token-mean
Training steps	500
Policy serving	
Serving engine	vLLM
TP size	2
Temperature	1.0
Top-p	1.0
World model serving	
Serving engine	vLLM
TP size	1
Temperature	0.0 (or 1.0 for additional candidates when $K > 1$)
Top-p	1.0

Table 1: Training and serving configuration for the policy and world model.

C Full Results

Policy	Pipeline	Reward (%) ↑	Agent (s) ↓	Batch (s) ↓	Rollouts/h ↑
Qwen3-8B	Sequential	78.64 ± 0.64	41.98 ± 0.39	156.26 ± 2.94	$13{,}762 \pm 156$
	ObsSpec	79.76 ± 0.63	38.06 ± 0.38	149.42 ± 3.06	$14{,}571 \pm 175$
Qwen3-32B	Sequential	84.62 ± 0.56	42.59 ± 0.46	185.63 ± 2.40	$14{,}066 \pm 155$
	ObsSpec	83.91 ± 0.57	36.53 ± 0.44	171.16 ± 2.81	$15{,}689 \pm 199$

Table 2: Full results underlying Figure 1.

Command	Count	Tool (s)	WM (s)	Match (%)	Rollback (%)	Cancel (%)
`cat`	19,544	1.597 ± 0.006	0.113 ± 0.001	79.95 ± 0.29	16.75 ± 0.27	3.31 ± 0.13
`ls`	10,681	1.614 ± 0.008	0.121 ± 0.001	39.43 ± 0.47	59.54 ± 0.47	1.03 ± 0.10
`mkdir`	2,943	1.607 ± 0.016	0.036 ± 0.000	96.67 ± 0.33	3.33 ± 0.33	0.00 ± 0.00
`echo`	1,169	1.612 ± 0.026	0.038 ± 0.000	98.37 ± 0.37	1.63 ± 0.37	0.00 ± 0.00
`sha256sum`	946	1.625 ± 0.028	0.178 ± 0.002	35.20 ± 1.55	50.00 ± 1.63	14.80 ± 1.15
`sqlite3`	668	1.631 ± 0.033	0.102 ± 0.004	78.89 ± 1.58	15.57 ± 1.40	5.54 ± 0.89
`make`	664	1.766 ± 0.038	0.170 ± 0.016	42.02 ± 1.92	47.74 ± 1.94	10.24 ± 1.18
`date`	412	1.541 ± 0.044	0.063 ± 0.001	0.24 ± 0.24	99.76 ± 0.24	0.00 ± 0.00
`find`	357	1.501 ± 0.046	0.081 ± 0.004	38.94 ± 2.58	30.81 ± 2.44	30.25 ± 2.43
`python3`	338	8.011 ± 0.737	0.200 ± 0.010	48.52 ± 2.72	45.56 ± 2.71	5.92 ± 1.28
Overall	47,762	1.685 ± 0.008	0.105 ± 0.001	66.45 ± 0.22	30.50 ± 0.21	3.05 ± 0.08

Table 3: Full results underlying Figure 3.

WM	WM latency (s)	Agent latency (s)	Match	Rollback	Cancel
0.6B	0.105 ± 0.001	38.06 ± 0.38	$66.45 \pm 0.22\%$	$30.50 \pm 0.21\%$	$3.05 \pm 0.08\%$
1.7B	0.137 ± 0.001	37.84 ± 0.37	$68.50 \pm 0.21\%$	$28.71 \pm 0.21\%$	$2.80 \pm 0.08\%$
4B	0.239 ± 0.001	37.89 ± 0.36	$68.74 \pm 0.21\%$	$24.88 \pm 0.20\%$	$6.38 \pm 0.11\%$
8B	0.350 ± 0.002	37.95 ± 0.37	$66.17 \pm 0.22\%$	$22.47 \pm 0.19\%$	$11.36 \pm 0.15\%$

Table 4: Full results underlying Figure 4.

Depth	Agent latency (s)	Realized depth	Policy tok/s	Discarded policy tok.
1	38.06 ± 0.38	0.992 ± 0.001	145.5 ± 0.1	198.0 ± 3.4
2	38.27 ± 0.38	1.871 ± 0.006	143.0 ± 0.1	342.5 ± 5.8
4	38.02 ± 0.37	2.773 ± 0.017	140.5 ± 0.1	426.7 ± 7.5
8	38.47 ± 0.39	3.208 ± 0.029	139.5 ± 0.1	472.2 ± 9.8

Table 5: Full results underlying Figure 5.

K	Agent latency (s)	Match	Realized breadth	Policy tok/s	Discarded policy tok.
1	38.06 ± 0.38	$66.45 \pm 0.22\%$	1.0	145.5 ± 0.1	198.0 ± 3.4
2	41.88 ± 0.40	$67.36 \pm 0.22\%$	1.303 ± 0.002	126.2 ± 0.1	332.8 ± 5.8
4	44.60 ± 0.44	$68.56 \pm 0.21\%$	1.940 ± 0.006	116.8 ± 0.1	574.9 ± 10.3
8	47.68 ± 0.46	$69.01 \pm 0.21\%$	3.142 ± 0.013	107.5 ± 0.1	994.9 ± 19.0

Table 6: Full results underlying Figure 6.

Policy step	Reward (%)	Updated match (%)	Frozen match (%)	Sequential agent (s)	Updated agent (s)	Frozen agent (s)
100	58.95 ± 2.52	59.07 ± 0.23	59.07 ± 0.23	23.14 ± 0.42	21.35 ± 0.40	21.35 ± 0.40
200	68.68 ± 2.38	62.64 ± 0.23	57.88 ± 0.24	23.20 ± 0.27	21.44 ± 0.26	21.70 ± 0.30
300	73.95 ± 2.25	59.87 ± 0.22	54.19 ± 0.22	28.03 ± 0.33	24.99 ± 0.30	25.22 ± 0.28
400	76.84 ± 2.16	64.01 ± 0.21	55.96 ± 0.21	32.47 ± 0.31	28.92 ± 0.28	29.60 ± 0.29
500	80.26 ± 2.04	66.45 ± 0.22	57.35 ± 0.23	41.98 ± 0.39	38.06 ± 0.38	39.74 ± 0.39

Table 7: Full results underlying Figure 7.

Sandbox	Tool latency (s)	Sequential agent (s)	ObsSpec agent (s)	Reduction
Default	1.69 ± 0.01	41.98 ± 0.39	38.06 ± 0.38	9.3%
Modal	2.56 ± 0.01	52.17 ± 0.44	46.66 ± 0.51	10.6%
Daytona	2.99 ± 0.02	56.30 ± 0.59	49.94 ± 0.54	11.3%

Table 8: Full results underlying Figure 8. All measured quantities report means and standard errors.

Method	Reward (%)	Agent (s)	WM (s)	Match (%)	Rollback (%)	Cancel (%)
Sequential	78.25 ± 0.64	55.82 ± 0.83	–	–	–	–
ObsSpec, 0.6B	78.47 ± 0.64	56.42 ± 0.86	0.120 ± 0.003	54.96 ± 0.42	40.42 ± 0.41	4.62 ± 0.18
ObsSpec, 8B	78.39 ± 0.64	55.79 ± 0.82	0.367 ± 0.007	59.47 ± 0.41	31.72 ± 0.39	8.81 ± 0.24

Table 9: Full results for the TIR experiments in Section 3.8.