

ADASPEC: Adaptive Routing by Ranking for Speculative Agents

Kangqi Ni¹, Wenyue Hua², Jinbiao Wei³, Tianlong Chen¹

¹University of North Carolina at Chapel Hill,

²Microsoft Research, AI Frontiers, ³Yale University

Abstract

Long-horizon LLM agents execute sequences of reasoning, tool-use, and environment-interaction steps, making end-to-end latency a central deployment bottleneck. Speculative agents reduce this latency by leveraging a faster draft agent to propose the action trajectory while a stronger target agent verifies each action to ensure lossless action trajectories. Recent systems adapt how many future actions a preselected draft agent proposes before verification, but leave the drafter itself unchanged across tasks and episodes. Our experiments show that the best drafter varies across tasks and trajectory prefixes, revealing substantial latency headroom from adaptive drafter selection. This motivates the question: which draft agent should speculate for a given task and target agent to achieve low end-to-end latency? We propose ADASPEC, an adaptive routing framework for speculative agents with an objective distinct from standard model routing: selecting the drafter that minimizes realized speculative-execution latency for a target agent, rather than selecting the model that produces the final answer. ADASPEC profiles a pool of candidate draft-agent configurations for each target agent and trains a lightweight, latency-supervised router to rank them by measured speculative latency. The router supports both query-level routing and episode-level rerouting before each draft-and-check segment, while the target agent remains responsible for verifying actions. ADASPEC improves fixed-drafter baselines across multiple datasets and target-agent settings.

1 Introduction

Long-horizon LLM agents rarely complete a task with a single model call. Instead, they execute chains of dependent reasoning, tool-use, and environment-interaction steps. Several lines of work motivate this execution style: ReAct

(Yao et al., 2022) popularized interleaving reasoning with external actions; MRKL systems (Karpas et al., 2022) and Toolformer (Schick et al., 2023) motivated modular tool use; AutoGen (Wu et al., 2023) scaled multi-agent coordination; and ToolLLM/ToolBench-style work (Qin et al., 2023; Guo et al., 2024) expanded tool-use training and evaluation. Benchmarks such as AgentBench (Liu et al., 2025), OpenAGI (Ge et al., 2023), and TravelPlanner (Xie et al., 2024) make the latency cost concrete: agents must wait through repeated planner calls, tool invocations, database queries, and environment feedback. System optimizations reduce the cost of individual LLM calls through batching, memory management, scheduling, and heterogeneous serving (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024; Ni et al., 2026), but they do not remove the dependency between consecutive agent steps.

A complementary way to address this dependency is to overlap proposal and verification rather than wait for each decision to finish before considering future actions. Speculative execution follows this principle: a cheaper draft process proposes future work, while a stronger target process checks what can be safely committed. This idea is partly inspired by blockwise and lookahead decoding (Stern et al., 2018; Fu et al., 2024). For agents, the same draft-and-check principle moves from tokens to actions: a draft agent proposes future actions, and a target agent checks and commits the resulting trajectory. Recent agent systems study action-level speculation, parallel proposal and checking, dynamic speculation horizons, and speculative search-agent execution (Huang et al., 2025). These methods show that speculation can reduce waiting time in agent workflows. However, when they adapt at runtime, they typically adapt how far a preselected draft agent runs ahead rather than which draft agent proposes the next actions.

The fixed-drafter assumption leaves a question:

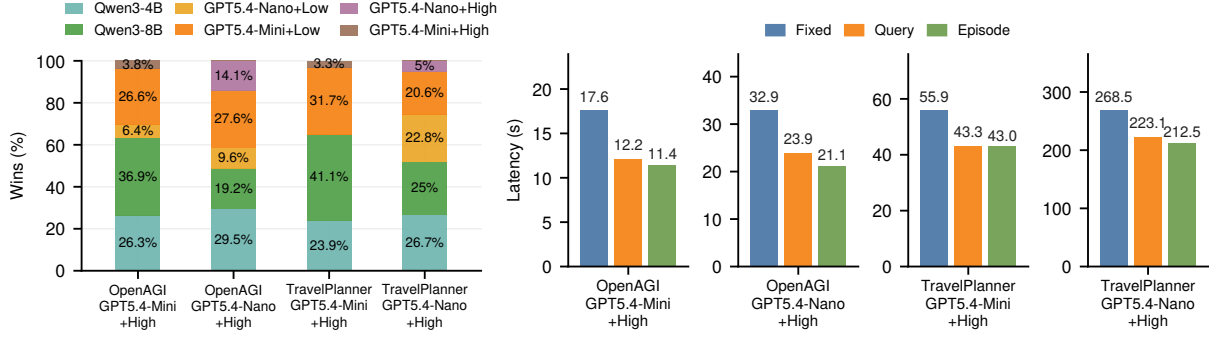


Figure 1: Oracle analysis of drafter selection. Left: query-level oracle winners are distributed across multiple draft agents; the largest winner share is only 41.1%, indicating that no single fixed drafter dominates. Right: oracle routing reduces latency relative to the best fixed- k drafter by 16.9%–30.9% at the query level and by 20.9%–35.9% at the episode level.

which draft agent should speculate for a given task and target agent? The best choice depends on a context-specific tradeoff: faster drafters may incur extra checks and corrections, while stronger drafters may match the target more often but cost more to run. The best option can also shift across tasks and trajectory prefixes. This makes the objective distinct from standard model routing. Standard routing typically optimizes which model can correctly answer a query or step, often under cost or latency constraints. Here, correctness is still governed by the fixed target agent, so the router optimizes realized end-to-end speculative-execution latency by choosing the proposal source. We therefore formulate drafter selection as a measured-latency ranking problem over candidate drafters, avoiding direct latency prediction while providing richer supervision than only classifying the fastest-drafter label.

We propose ADASPEC, an adaptive routing framework for selecting draft agents inside speculative execution. For each target agent, ADASPEC profiles a pool of candidate draft-agent configurations and trains a lightweight, latency-supervised listwise router to rank them by measured speculative latency. The router can select a draft agent once at the query level, or reroute before each speculative episode, defined as one draft-and-check segment of up to k proposed actions, based on the accepted action prefix. The target agent continues to verify committed actions, while the router optimizes the operational choice that determines task-level latency. Our contributions are as follows:

- *Problem formulation for adaptive draft-agent selection.* We formulate draft-agent selection as a routing problem inside speculative execution, re-

laxing the fixed-drafter assumption and exposing the headroom from adaptive drafter choice.

- *Listwise latency routing for speculative agents.* We introduce a latency-supervised listwise router that ranks candidate draft-agent configurations by measured speculative latency. The objective matches the online decision and avoids relying on calibrated latency regression or winner-only classification.
- *Evaluation on long-horizon agent benchmarks.* We evaluate ADASPEC on OpenAGI and TravelPlanner, showing that oracle drafter choices vary across contexts and that ADASPEC improves fixed-drafter baselines across multiple target-agent settings.

2 Related Work

Speculative Execution Speculative execution reduces latency by running potential future work before it is confirmed, with rollback on incorrect predictions. In LLM inference, speculative decoding uses a cheaper draft model to propose tokens that a stronger target verifies, preserving the target output while reducing sequential decoding latency (Leviathan et al., 2023; Chen et al., 2023a). Tree-based speculative inference, multi-head decoding, and feature-level proposals expand the number or structure of candidates checked per target step (Miao et al., 2024; Cai et al., 2024; Li et al., 2024). Agent-level work moves the same draft-and-check idea from tokens to actions and tool calls. Speculative Actions (Ye et al., 2026) formalizes lossless action-level speculation, Interactive Speculative Planning (Hua et al., 2025) parallelizes

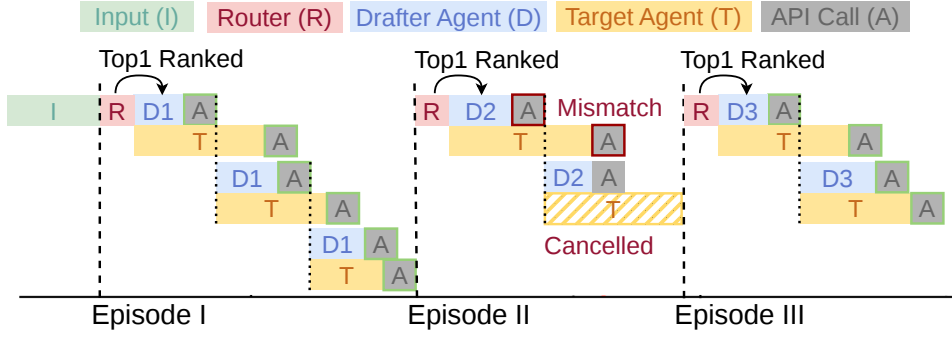


Figure 2: Overview of ADASPEC. A speculative episode is one draft-and-verify process of up to k proposed actions. For each episode, the router scores the candidate draft-agent pool for the current context and target agent, selects the top-ranked drafter, and runs speculation. Matched draft actions are committed, while a mismatched draft action is replaced by the target action before the next episode starts.

proposal and checking, and Dynamic Speculative Agent Planning (Guan et al., 2026) adapts how far a fixed approximation agent runs ahead.

ADASPEC complements these speculative-agent frameworks by making the drafter itself adaptive, selecting the draft-agent configuration for a given target and trajectory prefix rather than assuming a fixed draft agent.

Model Routing Model-routing systems choose among LLMs to trade off quality, latency, and cost. FrugalGPT (Chen et al., 2023b), AutoMix (Aggarwal et al., 2025), and HybridLLM (Ding et al., 2024) route or cascade queries across cheaper and stronger models, often using query difficulty or confidence to decide when to escalate. RouteLLM (Ong et al., 2025) learns routing decisions from preference data. RouterBench (Hu et al., 2024) and RouterArena (Lu et al., 2026) evaluate routers under multi-model quality-cost tradeoffs. Recent systems extend routing beyond a single query to multi-round or agentic orchestration, including Router-R1 (Zhang et al., 2026b), xRouter (Qian et al., 2025), ToolOrchestra (Su et al., 2025), EvoRoute (Zhang et al., 2026a), and GraphPlanner (Feng et al., 2026). These systems route the model, tool, or workflow that produces, aggregates, or executes the task output, so routing affects final-answer quality as well as cost and latency.

Speculative-agent routing instead chooses only the proposal source inside a fixed target-agent loop. Thus, ADASPEC does not route the answer-producing model; it ranks candidate drafters by realized speculative latency for each task, trajectory context, and target agent.

3 Motivation

Speculative agents accelerate long-horizon execution by allowing a cheaper drafter to run ahead while a stronger target agent checks the proposed actions. Although this preserves the target agent’s action trajectory, the optimal drafter is context-dependent: acceptance rates, draft latency, and correction cost can shift across tasks and contexts.

Figure 1 quantifies this context dependence through oracle drafter choices. In the left subfigure, query-level oracle winners are spread across several draft agents rather than concentrated on one fixed drafter: the largest winner share is only 36.9% for OpenAGI GPT-5.4-Mini+High, 29.5% for OpenAGI GPT-5.4-Nano+High, 41.1% for TravelPlanner GPT-5.4-Mini+High, and 26.7% for TravelPlanner GPT-5.4-Nano+High. The right subfigure shows why this variation matters for latency. Relative to the best fixed- k drafter, query-level oracle routing reduces end-to-end latency by 30.9% and 27.4% on the two OpenAGI targets, and by 22.6% and 16.9% on the two TravelPlanner targets. Episode-level routing also brings notable benefits in some settings, reaching reductions of 35.2% and 35.9% on OpenAGI and 23.1% and 20.9% on TravelPlanner. These results suggest that the best drafter can vary not only across tasks but also across trajectory prefixes. We therefore propose adaptive drafter routing to improve speculative-agent acceleration: For each target agent and context, select the drafter that minimizes realized end-to-end latency.

4 Methodology

We introduce ADASPEC as a routing layer inside speculative-agent execution. The method has three components: (1) a profiling stage that measures

candidate draft-agent configurations for each target agent, (2) a lightweight router trained to rank those candidates by end-to-end latency, and (3) an execution loop that invokes the selected drafter while leaving target checking unchanged.

4.1 Problem Setup

We consider a task x solved by a target agent T . An agent configuration specifies the model, reasoning effort, and reasoning style. At step t , the observable state is the task plus accepted action prefix, $s_t = (x, a_{1:t-1})$. Without speculation, T makes a blocking call for the next action a_t^T . With speculation, a draft agent d proposes one draft-and-check episode of up to k future actions, $\hat{a}_{t:t+k-1}^d = d(s_t)$, which T checks in order.

Let m be the longest accepted draft prefix: the largest value such that $\hat{a}_{t+i}^d$ is equivalent to a_{t+i}^T for every $0 \leq i < m$. The system commits these m actions; on the first rejection, it cancels remaining calls, waits to commit the target action at that position, and starts the next episode from the updated prefix. Thus, every committed action remains target-verified. A poor routing decision can waste time through slow drafts or extra rejections, but cannot bypass target verification.

Our decision variable is the drafter. For each target T , let $\mathcal{D}_T = \{d_1, \dots, d_M\}$ be the executable pool of candidate draft-agent configurations. For routing context s , let $\ell(s, d_i, T)$ be the measured end-to-end speculative latency when d_i drafts for T , including draft generation, target checks, rejections and corrections, cancellations, and the realized wall-clock schedule. The oracle route is

$$d^*(s, T) = \arg \min_{d_i \in \mathcal{D}_T} \ell(s, d_i, T).$$

We learn to approximate d^* from profiled executions, either per task or each speculative episode.

4.2 Adaptive Drafter Routing

Routing granularities. ADASPEC supports two routing granularities. Query routing is called once before the planner starts and uses only the raw task description. Episode routing is called immediately before each speculative episode. Its input contains the task description plus the accepted action prefix rendered in the same trajectory format used by the planner. Episode contexts also include the current step index, the trajectory length or step budget when available, and the fixed episode length k .

Candidate pools. The candidate pool contains draft-agent configurations that can be executed in the current system. Each configuration is specified by the model, the reasoning effort, and the reasoning style.

Configuration-aware scoring. The router is a compact transformer encoder with a scalar scoring head. For each candidate d_i , it receives a single text sequence

$$[\text{DRAFT}=d_i] [\text{TARGET}=T] q(s),$$

where $q(s)$ is either the task text or the task-plus-prefix context. This conditioning lets the same task or prefix be scored under different candidate-target pairs. The same encoder and scoring head are used for all candidates; only the draft marker, target marker, and context text change. The scorer outputs $z_i = f_\theta(s, d_i, T)$, with higher scores indicating a better latency rank. These scores are not calibrated seconds; only their ordering within the current candidate pool is used. At inference, ADASPEC selects

$$\hat{d}(s, T) = \arg \max_{d_i \in \mathcal{D}_T} f_\theta(s, d_i, T).$$

The same scoring interface is used for query and episode routing; only the context text and metadata differ. At runtime, ADASPEC scores the candidate texts as a batch and executes the chosen drafter. Figure 2 summarizes this execution flow.

4.3 Learning from Profiled Latency

Latency labels from profiled runs. ADASPEC trains on fixed- k profiling runs where every candidate drafter is evaluated with the same target agent and benchmark task. We group profiling rows by task id, target configuration, and routing context, so each training item contains a comparable candidate pool. A candidate is compared only with other candidates run for the same decision point. Groups with missing runs or fewer than three comparable candidates are discarded, and task-level splits keep the candidates for a given decision point together. This grouping makes the latency labels comparable: candidates are ranked only against other candidates evaluated on the same task, target, and routing context. Each training item supervises the operational decision made at runtime: choosing the lowest-latency candidate within the current candidate pool.

For query routing, the label for candidate d_i is its measured task-level latency $L_{\text{task}}(d_i, T)$. For

episode routing, we need a latency target for a suffix beginning at step t . We use the recorded per-step times to allocate the task latency to the remaining suffix:

$$\ell_{\text{suffix}}(t, d_i, T) = L_{\text{task}}(d_i, T) \frac{\sum_{j=t}^n \tau_j^{d_i, T}}{\sum_{j=1}^n \tau_j^{d_i, T}},$$

where n is the trajectory length and $\tau_j^{d_i, T}$ is the recorded per-step time on candidate d_i 's profiled trajectory: the drafter's step time on accepted steps and the target's step time on rejected steps. This approximation preserves the measured end-to-end ranking signal while giving episode routing a prefix-dependent target.

We convert the comparable latency values in each candidate pool into normalized relevance scores,

$$r_i = \begin{cases} \frac{\ell_{\max} - \ell_i}{\ell_{\max} - \ell_{\min}}, & \ell_{\max} > \ell_{\min}, \\ 1, & \text{otherwise,} \end{cases}$$

where $\ell_{\min}$ and $\ell_{\max}$ are the fastest and slowest candidate latencies in the same pool. Higher relevance therefore means lower measured speculative latency. Equal-latency groups provide no ranking signal and are omitted during training; when retained for analysis, all candidates receive equal relevance.

Listwise latency objective. ADASPEC optimizes the candidate ordering directly. This is important because the absolute latency scale is sensitive, hard to predict, and can shift across machines, API backends, and load conditions, while the relative ranking within a candidate pool is more reliable. Let candidates be sorted by decreasing relevance and partitioned into tie groups $G_1, \dots, G_J$. Define $R_j = \cup_{h=j}^J G_h$ as the remaining candidates when group G_j is selected. For a valid candidate set C , the tie-aware ListMLE (Xia et al., 2008) loss is

$$\mathcal{L}_{\text{ListMLE}} = \frac{1}{|C|} \sum_{j=1}^J \left(\log \sum_{c \in R_j} \exp z_c - \log \sum_{c \in G_j} \exp z_c \right).$$

The loss is averaged over valid candidate groups. Relevance values determine the ordering and tie groups, but they are not used as extra weights inside the ListMLE terms. This trains the router on the full measured ordering rather than only a fastest-drafter label or calibrated wall-clock target, matching the runtime decision where only the best candidate in the current pool must be selected.

5 Experiments

5.1 Datasets

OpenAGI This dataset contains natural-language requests over image, text, and multimodal inputs, including restoration, captioning, classification, visual QA, translation, summarization, sentiment analysis, and text-to-image generation. Each task requires the agent to choose and order API calls for the requested input-output transformation.

TravelPlanner This dataset contains user travel requests for complete itineraries with constraints such as cities, dates or duration, party size, budget, and preferences. The agent queries database-backed tools over flights, restaurants, attractions, accommodations, and distances to produce a feasible plan.

5.2 Experimental Setup

Configuration Each agent configuration is specified by a model name, reasoning effort, and reasoning style. Model names are the official OpenAI API and Hugging Face identifiers (gpt-5.4, gpt-5.4-mini, gpt-5.4-nano, Qwen3-4B, Qwen3-8B); the effort setting maps to the `reasoning_effort` request parameter for GPT models, and NO denotes non-reasoning mode for Qwen3. We write model-effort pairs as MODEL+EFFORT, e.g., GPT-5.4-Nano+High. The target-agent configurations are GPT-5.4-Nano+High, GPT-5.4-Mini+High, and GPT-5.4+Medium, each evaluated with ReAct and multi-agent-debate (MAD) reasoning styles. The draft-agent configurations are Qwen3-4B+No, Qwen3-8B+No, GPT-5.4-Nano+Low, and GPT-5.4-Mini+Low; all draft agents use the direct-answering reasoning style.

For each target agent T , we compare policies for choosing the draft agent under the same speculative episode length $k = 4$. The fixed-drafter rows report the four non-target draft agents above and the same-target direct fallback. For OpenAGI and TravelPlanner, we use the same 60%/40% task split for router data construction and reported comparisons. Experiments were run on a server with NVIDIA RTX A6000 GPUs and AMD EPYC 7763 CPUs.

Baselines We compare ADASPEC against three baselines: No Speculation, Fixed k , and Dynamic Speculative Agent Planning (DSP) (Guan et al.,

Target Agent	Method	Drafter Agent	ReAct			MAD		
			Lat. (s)	Steps/Ep.	Accept. (%)	Lat. (s)	Steps/Ep.	Accept. (%)
GPT-5.4-Nano+High	No Speculation	–	43.0	–	–	131.3	–	–
	Fixed k	Qwen3-4B+No	34.5	0.7	35.3	82.2	0.5	27.9
		Qwen3-8B+No	34.8	1.2	51.1	78.5	0.6	32.8
		GPT-5.4-Nano+Low	39.5	1.6	62.5	77.0	1.1	45.8
		GPT-5.4-Mini+Low	32.9	1.4	61.1	76.3	0.8	35.7
		GPT-5.4-Nano+High	36.4	2.0	69.5	77.8	1.1	50.2
	DSP	GPT-5.4-Mini+Low	33.8	1.5	62.5	77.4	0.8	36.3
	ADASPEC (Query)	Adaptive	28.7	1.4	57.2	72.8	0.9	39.2
GPT-5.4-Mini+High	ADASPEC (Episode)	Adaptive	26.5	1.3	56.7	72.2	0.8	36.7
	No Speculation	–	26.9	–	–	118.0	–	–
	Fixed k	Qwen3-4B+No	19.1	1.0	44.7	49.7	1.1	45.7
		Qwen3-8B+No	17.6	1.5	57.0	47.4	1.5	57.1
		GPT-5.4-Nano+Low	20.6	1.6	59.6	48.2	1.6	60.1
		GPT-5.4-Mini+Low	18.2	2.4	84.5	47.1	2.4	84.9
		GPT-5.4-Mini+High	26.4	2.5	85.4	50.1	2.6	86.3
	DSP	Qwen3-8B+No	16.9	1.5	55.8	46.1	1.5	56.3
	ADASPEC (Query)	Adaptive	15.5	1.7	62.1	42.5	2.4	85.4
GPT-5.4+Medium	ADASPEC (Episode)	Adaptive	15.9	1.8	63.6	43.4	2.6	88.4
	No Speculation	–	35.2	–	–	125.0	–	–
	Fixed k	Qwen3-4B+No	28.0	1.0	42.3	58.5	0.9	39.6
		Qwen3-8B+No	26.7	1.5	58.5	56.3	1.4	54.0
		GPT-5.4-Nano+Low	36.6	1.7	63.0	62.5	1.7	61.8
		GPT-5.4-Mini+Low	31.0	2.1	75.7	55.9	2.2	77.7
		GPT-5.4+Medium	53.3	2.2	79.9	60.1	2.5	84.5
	DSP	Qwen3-8B+No	26.9	1.5	58.9	55.3	1.4	53.5
	ADASPEC (Query)	Adaptive	23.6	2.2	80.7	53.5	2.5	85.1
	ADASPEC (Episode)	Adaptive	21.0	2.3	81.4	50.8	2.6	85.8

Table 1: Main comparison of No Speculation, Fixed k , DSP, and ADASPEC on the **OpenAGI** benchmark. The best routed variant reduces latency by 5.4%–21.3% over the strongest fixed drafter and by 6.7%–21.9% over DSP. Green shading marks the lowest and second-lowest latency settings for each target agent and reasoning style; darker green indicates the lowest latency.

2026). No Speculation runs the target agent sequentially and serves as the target-reference behavior. Fixed k uses one static drafter for all tasks and episodes. DSP adapts the speculative episode length while keeping the drafter fixed. To isolate horizon adaptation from drafter selection, we instantiate DSP only with the strongest Fixed k drafter from the preceding sweep for the same dataset, target agent, and reasoning style. By contrast, ADASPEC adapts drafter selection: ADASPEC (Query) selects one drafter per task, whereas ADASPEC (Episode) can reselect the drafter before each speculative episode.

Metrics For the main-result tables, we report **Lat. (s)**, the end-to-end wall-clock latency in seconds; **Steps/Ep.**, the mean number of accepted draft steps per speculative episode; and **Accept. (%)**, the percentage of proposed draft actions accepted by the target checker. We do not define episode-based metrics for no-speculation rows, and latency breakdowns separately attribute time to drafter, target,

and router components.

5.3 Main Results

OpenAGI Table 1 presents the main OpenAGI comparison across target agents and reasoning styles. The strongest Fixed- k drafter provides a strong starting point, reducing latency by 23.5%–60.1% relative to No Speculation, which confirms that draft-and-check execution substantially mitigates sequential agent latency. ADASPEC consistently improves on this speculative baseline: across the six OpenAGI settings, the best routed variant reduces latency by 5.4%–21.3% over the strongest fixed drafter and by 6.7%–21.9% over DSP. The best routing granularity depends on the target setting. Episode routing is most effective for GPT-5.4-Nano+High, reaching 26.5s on ReAct and 72.2s on MAD, and for GPT-5.4+Medium, reaching 21.0s on ReAct and 50.8s on MAD; in these cases, rerouting after accepted prefixes yields clear gains. For GPT-5.4-Mini+High, query routing is

Target Agent	Method	Drafter Agent	ReAct			MAD		
			Lat. (s)	Steps/Ep.	Accept. (%)	Lat. (s)	Steps/Ep.	Accept. (%)
GPT-5.4-Nano+High	No Speculation	–	475.6	–	–	334.2	–	–
	Fixed k	Qwen3-4B+No	268.5	0.5	28.5	247.7	0.8	38.2
		Qwen3-8B+No	279.6	0.6	30.7	227.0	1.0	45.0
		GPT-5.4-Nano+Low	268.5	0.8	39.7	254.5	0.8	39.7
		GPT-5.4-Mini+Low	272.4	0.4	22.9	244.3	0.7	33.3
		GPT-5.4-Nano+High	337.7	0.8	39.8	403.4	1.1	47.1
	DSP	Qwen3-8B+No	273.7	0.6	31.2	222.0	1.0	44.1
	ADASPEC (Query)	Adaptive	254.7	0.5	28.1	219.0	1.1	49.5
	ADASPEC (Episode)	Adaptive	235.1	0.4	28.5	213.0	1.1	47.0
GPT-5.4-Mini+High	No Speculation	–	112.4	–	–	194.6	–	–
	Fixed k	Qwen3-4B+No	70.9	0.5	29.7	152.5	0.9	39.4
		Qwen3-8B+No	55.9	0.9	43.8	122.3	1.4	53.9
		GPT-5.4-Nano+Low	119.4	0.6	31.7	154.5	0.5	28.3
		GPT-5.4-Mini+Low	64.8	1.1	47.7	129.1	1.6	56.8
		GPT-5.4-Mini+High	89.4	1.0	48.2	131.9	2.0	67.8
	DSP	Qwen3-8B+No	57.4	0.9	43.0	125.2	1.3	53.1
	ADASPEC (Query)	Adaptive	52.1	0.9	42.6	119.6	1.4	54.8
	ADASPEC (Episode)	Adaptive	53.9	0.9	43.6	114.4	1.2	53.4

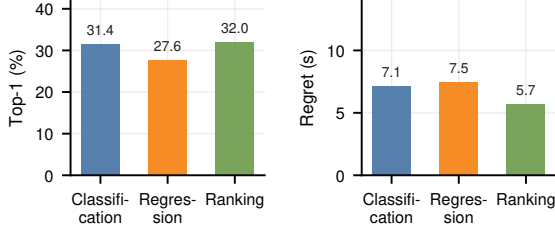
Table 2: Main comparison of No Speculation, Fixed k , DSP, and ADASPEC on the **TravelPlanner** benchmark. ADASPEC achieves the lowest latency in each reported setting, improving over the strongest fixed drafter by 6.2%–12.4% and over DSP by 4.1%–14.1%. Green shading marks the lowest and second-lowest latency settings for each target agent and reasoning style; darker green indicates the lowest latency.

slightly better at 15.5s/42.5s, indicating that the initial task context is often sufficient to identify a good drafter. The acceptance results further show that latency is not optimized by simply choosing the most target-like drafter: on GPT-5.4-Mini+High ReAct, ADASPEC (Query) achieves the lowest latency with 62.1% acceptance, while fixed drafters with 84.5% and 85.4% acceptance are slower. Effective drafter selection must therefore balance drafter latency against acceptance rate, which ADASPEC captures through its end-to-end latency-ranking objective.

TravelPlanner Table 2 reports the corresponding results on TravelPlanner, where trajectories are longer and constraints evolve across the itinerary. Fixed- k speculation again delivers large improvements over No Speculation, with the best fixed drafter reducing latency by 32.1%–50.3% across the four reported settings. ADASPEC achieves the lowest latency in each setting and further reduces latency by 6.2%–12.4% over the strongest fixed drafter and by 4.1%–14.1% over DSP. For GPT-5.4-Nano+High, episode routing is consistently preferred, improving query routing from 254.7s to 235.1s on ReAct and from 219.0s to 213.0s on MAD; this suggests that accepted itinerary prefixes reveal useful information about the remaining tool and constraint structure. For GPT-5.4-Mini+High, query routing is best on ReAct at 52.1s, while

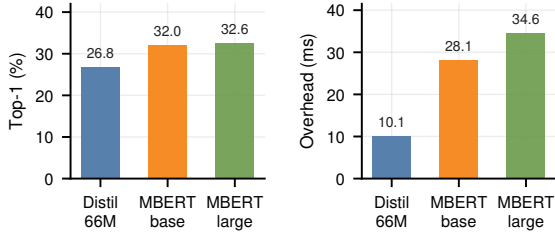
episode routing is best on MAD at 114.4s, showing that the advantage of prefix-level rerouting is setting-dependent rather than uniform. The acceptance columns support the same tradeoff interpretation as OpenAGI. On GPT-5.4-Nano+High ReAct, the best routed run has only 28.5% acceptance but lower latency than higher-acceptance fixed drafters; conversely, the low-acceptance Qwen3-4B+No drafter on GPT-5.4-Mini+High MAD remains much slower than routing. Thus, the best drafter is neither the most accurate nor the cheapest in isolation, but the one that minimizes realized latency under the draft-and-check execution dynamics.

Router Objectives Figure 3 compares three router training targets on the same OpenAGI routing contexts. Top-1 is the fraction of contexts where the router selects the lowest-latency candidate in the profiled pool; regret is the extra end-to-end latency relative to that candidate. Ranking has the best Top-1, 32.0% versus 31.4% for classification and 27.6% for regression, but the clearer gain is in regret: 5.7s versus 7.1s and 7.5s. Thus, classification and regression incur $1.25\times$ and $1.31\times$ as much regret as ranking. Ranking therefore helps not only by selecting the fastest candidate slightly more often, but also by making mistakes less costly on average. This matches the supervision: classi-



(a) Top-1: fastest drafter selected (%) (b) Regret: extra latency over fastest (s)

Figure 3: Router objective ablation on OpenAGI. Panels compare supervision targets by exact selection of the lowest-latency candidate and by the latency cost of selecting a non-fastest candidate.



(a) Top-1: fastest drafter selected (%) (b) Overhead: router time (ms)

Figure 4: Router backbone ablation on OpenAGI. Panels compare encoder choices by fastest-candidate selection accuracy and local router inference cost.

fication keeps only the fastest-candidate label and treats all other choices alike, regression requires calibrated absolute latencies, and ranking preserves the measured ordering of near-best and poor alternatives within each candidate pool.

5.4 Performance Analysis

Overhead Analysis Table 3 reports router overhead for the OpenAGI ReAct setting with GPT-5.4-Nano+High as the target. Even with episode-level rerouting, ModernBERT-base routing accounts for less than 0.3% of end-to-end latency. Thus, ADASPEC adds very little overhead, even when it reroutes before each speculative episode.

Long-Horizon Behavior Table 4 reports ReAct-style speculation acceptance rates by step-position bucket for TravelPlanner. We use non-overlapping action-position buckets and report mean per-task acceptance in each bucket. For GPT-5.4-Nano+High, episode routing improves later-step acceptance from 23.8% to 27.5% at Steps 3–4 and from 12.5% to 25.0% after Step 5. For GPT-5.4-Mini+High, the gap is larger at Steps 3–4, where episode routing reaches 41.2% versus 20.0% for

Method	E2E (s)	Calls/Task	Router (s)
ADASPEC (Query)	28.7	1.0	0.03 (0.10%)
ADASPEC (Episode)	26.5	2.4	0.07 (0.26%)

Table 3: Router overhead for OpenAGI ReAct-style target agents with GPT-5.4-Nano+High. Parentheses in the router column report router time as a percentage of E2E latency.

Target Agent	Method	Step 1–2	Step 3–4	≥ Step 5
GPT-5.4-Nano+High	ADASPEC (Query)	33.8	23.8	12.5
	ADASPEC (Episode)	36.2	27.5	25.0
GPT-5.4-Mini+High	ADASPEC (Query)	67.5	20.0	25.0
	ADASPEC (Episode)	71.2	41.2	29.2

Table 4: TravelPlanner routed-run acceptance by step-position bucket on ReAct-style target agents. Entries report mean per-task acceptance in percent.

query routing, and remains higher after Step 5, at 29.2% versus 25.0%. These trends suggest that prefix-aware rerouting helps the drafter stay aligned with the evolving itinerary context over longer trajectories.

Router Backbone Figure 4 shows the tradeoff between routing accuracy and cost. DistilBERT has the lowest overhead at 10.1ms per decision, but reaches only 26.8% Top-1. ModernBERT-base raises Top-1 to 32.0%, a 5.2 percentage-point gain, with 28.1ms overhead. ModernBERT-large does not provide a favorable incremental tradeoff in the plotted metrics. It increases Top-1 by only 0.6 percentage points over ModernBERT-base, from 32.0% to 32.6%, while overhead rises from 28.1ms to 34.6ms. This is a $1.23\times$ overhead increase for a small accuracy gain. We therefore use ModernBERT-base: it captures most of the accuracy improvement over DistilBERT while avoiding the additional router overhead of the larger encoder.

6 Conclusions

Across OpenAGI and TravelPlanner, fixed-drafter speculation leaves consistent latency gains unrealized: oracle winners are distributed across draft agents, and ADASPEC improves over the strongest fixed-drafter and DSP baselines in every reported setting. The results show that the best drafter is not simply the most target-like or the cheapest model, but the one that minimizes realized draft-and-check latency after draft cost, acceptance, target verification, and correction time interact. Query-level routing is sufficient in some settings, while episode-level rerouting helps when accepted prefixes reveal useful trajectory context. Listwise latency

ranking further reduces regret relative to classification or regression, with negligible routing overhead. These findings show that drafter choice is an effective control knob for speculative agents: its gains come from context-dependent drafter selection that avoids high-regret drafters.

Limitations

ADASPEC is designed as a software-layer router for choosing draft agents under a fixed speculative execution protocol. Therefore, our evaluation focuses on latency, acceptance behavior, and target-verified task execution rather than hardware-based routing, where drafter choices also depend on the accelerator type, device placement, energy profile, or hardware cost of each candidate. This scope isolates the effect of drafter selection from infrastructure-level variation. Future directions include extending the router to make hardware-aware drafter choices.

Acknowledgments

This work is partially supported by Amazon Research Award and Cisco Faculty Award.

References

- Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqi, and Mausam. 2025. [Automix: Automatically mixing language models](#). *Preprint*, arXiv:2310.12963.
- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 117–134.
- Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023a. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.
- Lingjiao Chen, Matei Zaharia, and James Zou. 2023b. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*.
- Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pages 41348–41366.
- Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. 2026. Graphplanner: Graph memory-augmented agentic routing for multi-agent llms. *arXiv preprint arXiv:2604.23626*.
- Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*.
- Yingqiang Ge, Wenye Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. 2023. [Openagi: When llm meets domain experts](#). *Preprint*, arXiv:2304.04370.
- Yilin Guan, Qingfeng Lan, Fei Sun, Dujian Ding, Devang Acharya, Chi Wang, William Wang, and Wenye Hua. 2026. Dynamic speculative agent planning. In *International Conference on Learning Representations*, volume 2026, pages 7695–7717.
- Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156.
- Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*.
- Wenye Hua, Mengting Wan, Jagannath Vadrevu, Ryan Nadel, Yongfeng Zhang, and Chi Wang. 2025. Interactive speculative planning: Enhance agent efficiency through co-design of system and user interface. In *International Conference on Learning Representations*, volume 2025, pages 14256–14283.
- Zixiao Huang, Wen Zeng, Tianyu Fu, Tengxuan Liu, Yizhou Sun, Ke Hong, Xinhao Yang, Chengchun Liu, Yan Li, Quanlu Zhang, Guohao Dai, Zhenhua Zhu, and Yu Wang. 2025. [Reducing latency of llm search agent via speculation-based algorithm-system co-design](#). *Preprint*, arXiv:2511.20048.
- Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhl-gay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholz. 2022. [Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning](#). *Preprint*, arXiv:2205.00445.

- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pages 19274–19286. PMLR.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2025. [Agentbench: Evaluating llms as agents](#). *Preprint*, arXiv:2308.03688.
- Yifan Louie Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. 2026. Routerarena: An open platform for comprehensive comparison of llm routers. In *International Conference on Learning Representations*, volume 2026, pages 44385–44406.
- Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunshen Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. [Specinfer: Accelerating large language model serving with tree-based speculative inference and verification](#). In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949. ACM.
- Kangqi Ni, Wenyue Hua, Xiaoxiang Shi, Jiang Guo, Shiyu Chang, and Tianlong Chen. 2026. Chimera: Latency-and performance-aware multi-agent serving for heterogeneous llms. *arXiv preprint arXiv:2603.22206*.
- Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. 2025. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pages 34433–34448.
- Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025. [xrouter: Training cost-aware llms orchestration system via reinforcement learning](#). *Preprint*, arXiv:2510.08439.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. [Toolllm: Facilitating large language models to master 16000+ real-world apis](#). *Preprint*, arXiv:2307.16789.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.
- Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31.
- Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhurina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. 2025. [Toolorchestra: Elevating intelligence via efficient model and tool orchestration](#). *Preprint*, arXiv:2511.21689.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryan W White, Doug Burger, and Chi Wang. 2023. [Autogen: Enabling next-gen llm applications via multi-agent conversation](#). *Preprint*, arXiv:2308.08155.
- Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199.
- Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. 2026. Speculative actions: A lossless framework for faster ai agents. In *The Fourteenth International Conference on Learning Representations*.
- Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pages 521–538.

Guibin Zhang, Haiyang Yu, Kaiming Yang, Bingli Wu, Fei Huang, Yongbin Li, and Shuicheng Yan. 2026a. Evoroute: Experience-driven self-routing llm agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 38213–38225.

Haozhen Zhang, Tao Feng, and Jiaxuan You. 2026b. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:141233–141265.

A Additional Benchmark Details

A.1 OpenAGI

OpenAGI evaluates tool-use planning over text, vision, and multimodal instructions. Each task is a natural-language request that must be solved by choosing a sequence of benchmark-valid tools. The planner observes the task description and an accepted action prefix, then emits exactly one next action. The benchmark action vocabulary covers text processing, visual analysis, image restoration, image generation, and termination.

A.2 TravelPlanner

TravelPlanner evaluates long-horizon information collection and itinerary synthesis. Each user query specifies a trip request with constraints such as origin, destination, dates or duration, party size, budget, transportation preferences, accommodation requirements, cuisine preferences, and attractions. The agent gathers evidence through database-backed tools, writes relevant retrieval results into a notebook, and then calls a planner tool to produce a complete itinerary.

B Experimental Details

B.1 Agent Roles

Each executable agent role is identified by a model, a reasoning-effort setting, and a controller style. The router uses the same role string during training and inference:

model|effort|style.

The target role is the configuration used for verification. Draft roles are candidate configurations that propose future actions under the same benchmark action format. The router selects a draft role; the target role still checks and commits actions.

B.2 OpenAGI Action Space

OpenAGI actions are emitted inside XML-style tool tags. The planner emits one tool name per step; the parser normalizes the model response to the benchmark tool name.

B.3 TravelPlanner Function API

TravelPlanner actions are function-style tool calls. Each step calls one function. Retrieval tools return database evidence; notebook writes make retrieved evidence visible to the final planner.

B.4 Runtime Routing Interface

At runtime, the router receives a compact description of the current routing state: the task text or task-plus-prefix text, the target role, the available draft roles, the routing granularity, and the fixed speculation window. For episode-level routing, the context also includes the current prefix position. The router scores the available draft roles and returns the selected drafter for the next speculative episode. The target controller still verifies the committed actions; the router only chooses which draft role proposes them.

C Router Data and Training Details

C.1 Data Construction

Router training examples are derived from profiled fixed- k speculative runs. Each example corresponds to a single routing decision and compares only draft roles evaluated under the same benchmark task, target role, controller style, routing granularity, and speculation window. This grouping makes the ranking target well defined: the router learns to prefer the candidate role with lower measured end-to-end latency under a matched decision context.

Query-level examples use the raw task description as the router context. Episode-level examples combine the task description with the accepted action prefix and prefix position. The stored example also records the candidate and target role strings, the routing granularity, the speculation window, and the training target used by the ranking objective. Splitting is performed at the task level, so examples derived from the same task do not cross the train/test boundary.

C.2 Router Input Format

The router is a transformer encoder with a scalar scoring head. A separate input string is constructed

Task Type	Example Request	Expected Tool Sequence
Text Transformation	Translate, summarize, classify sentiment, or answer a document question.	Call the requested text tool and terminate after the requested output is produced.
Image Understanding	Caption an image, classify an image, detect objects, or answer a visual question.	Call the relevant vision tool, then use a text tool only if the requested answer format requires it.
Image Restoration	Restore a noisy, blurry, grayscale, or low-resolution image.	Call the requested restoration tool, optionally call a follow-up vision tool, then terminate.
Multimodal Question Answering	Answer a question that depends on both image content and text output requirements.	Use a vision tool first, then QA, translation, or summarization as required.
Image Generation	Produce an image from a text description and optionally describe the generated image.	Call text-to-image generation, then call a vision tool only if inspection is requested.

Table 5: Representative OpenAGI task types.

Task Type	Example Request	Expected Tool Sequence
Single-City Itinerary	Plan a trip to one destination under a fixed budget.	Search transportation, lodging, restaurants, and attractions, then call the planner.
Multi-City Route	Visit several cities in a state or region.	Search cities, check travel distance, store route evidence, then gather city-specific options.
Transportation-Constrained Trip	Use or avoid a specified travel mode.	Use the allowed transportation tool before searching lodging and activities.
Accommodation-Constrained Trip	Require private rooms, family lodging, or party-size constraints.	Search accommodations, store admissible options, then search dining and attractions.
Preference-Specific Itinerary	Include cuisines, attractions, or activity preferences.	Search preference-specific options for each city and store the relevant results before planning.

Table 6: Representative TravelPlanner task types.

Field	Values Used in Experiments	Meaning
Model	GPT-5.4 target and draft roles; Qwen3 draft roles.	Base model used by the controller.
Effort	No, Low, Medium, High.	Reasoning-effort setting passed to the model call when supported.
Style	Direct, ReAct, multi-agent debate.	Prompting and controller style for next-action generation.
Target Role	Model, effort, and style of the verifier.	Produces the next action used for speculative checking.
Draft Role	Model, effort, and style of a candidate drafter.	Produces up to k proposed actions for the next speculative episode.

Table 7: Agent role fields used by the routing layer.

for each candidate draft role in the current set:

The [DRAFT=...] and [TARGET=...] markers make the candidate and target roles explicit in the encoded text. The same input construction is used for training and runtime inference.

C.3 Training Objective and Hyperparameters

Table 10 lists the router objective and training hyperparameters.

C.4 Prompt and Controller Templates

We provide additional prompt and controller templates.

D LLM Usage

LLMs were used only to edit and polish the writing of this manuscript, such as refining wording and improving clarity. They were not involved in the research ideas, methodology, experiments, or analysis.

Tool	Description
Sentiment Analysis	Analyze the sentiment of an input text.
Text Summarization	Produce a concise summary of a text passage.
Machine Translation	Translate text into the requested language.
Fill Mask	Fill a missing or masked phrase in text.
Question Answering	Answer a question from text or provided context.
Image Classification	Predict an image-level class label.
Object Detection	Detect objects in an image.
Colorization	Colorize a grayscale image when required.
Image Super-Resolution	Increase image resolution.
Image Denoising	Remove noise from an image.
Image Deblurring	Restore a blurred image.
Visual Question Answering	Answer a question grounded in an image.
Image Captioning	Produce a caption for an image.
Text-to-Image Generation	Generate an image from text.
TERMINATE	End the trajectory when the task is complete.

Table 8: OpenAGI tool vocabulary used by the planner.

Function	Arguments	Description
FlightSearch	Departure city, destination city, date.	Retrieve flight options for an allowed air-travel leg.
GoogleDistanceMatrix	Origin, destination, mode.	Estimate distance, travel time, and cost for ground transportation.
AccommodationSearch	City.	Retrieve accommodation options in a city.
RestaurantSearch	City.	Retrieve dining options in a city.
AttractionSearch	City.	Retrieve attractions in a city.
CitySearch	State.	Retrieve candidate cities in a state.
NotebookWrite	Short description.	Store the latest relevant retrieval result for later planning.
Planner	Original user query.	Generate a complete itinerary from the user query and notebook evidence.

Table 9: TravelPlanner function API used by the information-collection agent.

Candidate Scoring Input.

```
[DRAFT={draft_model|draft_effort|draft_style}]
[TARGET={target_model|target_effort|target_style}]
{serialized task or task-plus-prefix context}
```

Figure 5: Router input template.

Setting	Value
Backbone	answerdotai/ModernBERT-base.
Primary Objective	Latency-derived listwise ranking over comparable candidate sets.
Fixed Speculation Window	$k = 4$ for fixed- k router data and routed execution.
Train/Test Split	60%/40% task-level split.
Learning Rate	2×10^{-5} .
Epoch Budget	6 epochs.
Weight Decay	0.01.
Query Router Batch Size	32.
Episode Router Batch Size	64.

Table 10: Router training settings.

OpenAGI Task Prompt.

Problem: {task_description}

Please solve this problem using the following tools step by step.

Available tools are: Sentiment Analysis, Text Summarization, Machine Translation, Fill Mask, Question Answering, Image Classification, Object Detection, Colorization, Image Super-Resolution, Image Denoising, Image Deblurring, Visual Question Answering, Image Captioning, Text-to-Image Generation, and TERMINATE.

For each step of the plan, specify the tool to use. If the task is complete, use <tool>TERMINATE</tool>. The final action must follow the exact XML tag format, for example <tool>Question Answering</tool>.

Figure 6: OpenAGI base planner prompt template.

Direct Next Action Prompt.

{base_task_prompt}

{accepted_action_trajectory}

Directly state the **next action** given the current action trajectory. Use <tool> and </tool> for formatting.

What should the action be in Step {step_number}?

Step {step_number}:

Figure 7: Direct draft-agent prompt template.

Target Thought Prompt.

{base_task_prompt}

{accepted_action_trajectory}

Carefully reason about the **next action** given the current action trajectory.

Generate only the thought.

Thought {step_number}:

Target Action Prompt.

{thought_prompt} {generated_thought}

Generate only the action based on the thought. Use <tool> and </tool> for formatting.

Action {step_number}:

Figure 8: Two-stage ReAct target prompt template.

Multi-Agent-Debate Target Prompt Pattern.

Each MAD step reuses the benchmark-specific next-action context {base_next_action_prompt} and returns one target action.

Agent A, Round 1.

{base_next_action_prompt}

Discuss with another agent about the **one next action step** given the current trajectory. Provide your thought and answer.

Agent B, Round 1.

{base_next_action_prompt}

{agent_A_proposal}

Decide whether the other agent's idea is useful, then provide your thought and answer.

Reflection Rounds.

Each agent sees its own previous proposal and the other agent's proposal, then summarizes, reflects, and updates its next-action decision.

Final Action.

The target model sees the updated proposals, summarizes and reflects, then emits the final next action in the benchmark-required action format.

Figure 9: Multi-agent-debate target controller template.

TravelPlanner ReAct Prompt.

Collect information for a travel plan using interleaved Thought, Action, and Observation steps. Gather valid information related to transportation, dining, attractions, and accommodation. Relevant information should be written in NotebookWrite, which is then provided as input to the Planner tool. Each action calls one function once.

Action Types. FlightSearch, GoogleDistanceMatrix, AccommodationSearch, RestaurantSearch, AttractionSearch, CitySearch, NotebookWrite, and Planner.

Query: {query}{scratchpad}

Figure 10: TravelPlanner information-collection prompt template.

TravelPlanner Final Planner Prompt.

Based on the provided information and the user query, produce a detailed day-by-day travel plan. The plan should include transportation, meals, attractions, and accommodations when applicable. All details should be derived from the provided notebook evidence and should follow the requested constraints.

Given information: {notebook_text}

Query: {query}

Travel Plan:

Figure 11: TravelPlanner final planner prompt template.

Prompt	Instantiated Context	Expected Output
OpenAGI Direct	Task: “Given a low-resolution image, produce a German caption.” Prefix: none. The prompt asks for Step 1.	<tool>Image Super-Resolution</tool>
OpenAGI Direct, Step 2	Task: same image-captioning request. Prefix: Step 1 is Image Super-Resolution. The prompt asks for Step 2.	<tool>Image Captioning</tool>
OpenAGI Target Action	Task: “Answer the document question in German.” Prefix: Step 1 is Question Answering. The target has generated a short thought that the answer should now be translated.	<tool>Machine Translation</tool>
TravelPlanner Search	Query: “Plan a three-day trip from Boston to Chicago for two people from 2022-03-10 to 2022-03-12 under a 3000 dollar budget.” Scratchpad: none.	FlightSearch[Boston, Chicago, 2022-03-10]
TravelPlanner Notebook Write	Query: same trip request. Latest observation contains flight candidates for Boston to Chicago on the departure date.	NotebookWrite[Flights from Boston to Chicago on 2022-03-10]
TravelPlanner Final Plan	Notebook contains selected transportation, lodging, restaurants, and attractions. Query is the original trip request.	Day-by-day itinerary with transportation, meals, attractions, and accommodation fields.

Table 11: Concrete prompt examples with synthetic task text.

Task Type	Example Task	Tool Sequence
Answer Then Translate	Answer an English question and provide the final response in another language.	Question Answering → Machine Translation → TERMINATE.
Enhance Then Caption	Produce a caption for a low-resolution image.	Image Super-Resolution → Image Captioning → TERMINATE.
Restore Then Inspect	Restore a degraded image and answer a visual question.	Restoration tool → Visual Question Answering → TERMINATE.
Fill Then Answer	Complete a masked phrase before answering the query.	Fill Mask → Question Answering → TERMINATE.
Detect Then Describe	Detect objects and describe their relation.	Object Detection → Image Captioning or Visual Question Answering → TERMINATE.
Generate Then Validate	Generate an image from text and describe or inspect the result.	Text-to-Image Generation → optional visual tool → TERMINATE.

Table 12: Representative OpenAGI task examples.

Task Type	Example Task	Information Collection Sequence
Flight-Based City Trip	Plan a short trip between distant cities.	FlightSearch → NotebookWrite → local searches → Planner.
Ground-Transport Trip	Plan a trip where self-driving or taxi travel is relevant.	GoogleDistanceMatrix → NotebookWrite → lodging and local searches → Planner.
Multi-City State Route	Visit several cities in one state.	CitySearch → distance checks → notebook writes → per-city searches → Planner.
Accommodation-Constrained Trip	Require private rooms, family-friendly lodging, or party-specific constraints.	AccommodationSearch → NotebookWrite → restaurant and attraction searches → Planner.
Cuisine-Preference Trip	Include meals matching cuisine preferences.	Route evidence → RestaurantSearch per city → NotebookWrite → Planner.
Budget-Sensitive Itinerary	Produce a feasible plan under a fixed total budget.	Transportation and lodging cost evidence → notebook writes → final planner synthesis.

Table 13: Representative TravelPlanner task examples.