Clearing MCP Navigation Fog with Economics-Aware Hierarchical Tool Routing

Jieling Yu*, Zhenheng Tang*, Ruiting Zhou†, Baochun Li‡, and Bo Li*
* Hong Kong University of Science and Technology,
† Southeast University, ‡University of Toronto,
jyucm@connect.ust.hk, zhtang.ml@ust.hk, ruitingzhou@seu.edu.cn, bli@ece.toronto.edu, bli@cse.ust.hk

***Abstract*—The Model Context Protocol (MCP) standardizes how large language models (LLMs) connect to external tools and services, vastly enhancing the capability of LLM agents. However, as MCP ecosystems scale to tens or hundreds of servers and thousands of fine-grained tools, tool use tends to become unreliable and expensive: many tools appear semantically plausible but fail at execution time, naive schema injection causes severe prompt bloat, and similarity-only routing ignores real-world heterogeneity in monetary cost, latency, and reliability. We refer to this regime as *navigation fog*.**

In this paper, we propose Economics-aware Hierarchical Retrieval and Routing (EHRR), a lightweight MCP co-pilot routing mechanism designed for practical MCP deployment at scale. EHRR prompts the LLM to produce a structured *functional* subtask description only when external tool support is needed, and then performs coarse-to-fine routing. Specifically, at the server layer, EHRR ranks servers using a utility that trades off semantic alignment against expected operational overhead, and applies a low-communication posted-price reverse auction to filter out economically infeasible servers. At the tool layer, EHRR selects tools with a utility that jointly accounts for relevance, execution success probability, latency, and per-call price, and executes the chosen tool via MCP.

Experiments on 95 real-world MCP tasks over 81 servers and 810 tools, using two distinct LLM backends, show that EHRR consistently improves end-to-end task success while substantially reducing wasted routing/execution, token overhead, and tool spending. Compared with retrieval-only routing, EHRR more than doubles success and reduces token and monetary cost by up to 33.0% and 51.9%, respectively, while achieving orders-of-magnitude savings relative to prompt-saturation baselines.

***Index Terms*—MCP, LLM, retrieval, economic, hierarchical**

I. Introduction

The recent progress of large language models (LLMs) has accelerated the development of autonomous agents that can plan, reason, and act by invoking external tools. The Model Context Protocol (MCP) [1] further lowers the integration barrier by offering a standardized interface that connects LLMs to heterogeneous servers, APIs, and data repositories. In principle, MCP enables a scalable "agentic web" where an assistant can access a rapidly expanding universe of capabilities. In practice, however, MCP-scale deployments introduce a new bottleneck: as the number of available MCP servers and tools grows into the thousands, an agent must repeatedly decide *what capability is needed*, *which server is worth contacting*, and *which tool is actually executable* under real cost and reliability constraints. We call the resulting performance degradation *navigation fog*.

A naive response is to expose the full tool catalog to the LLM at each step. This approach quickly becomes infeasible. Tool schemas and descriptions consume context budget and are billed as input tokens; moreover, long-context prompting often degrades tool-choice reliability as irrelevant functions accumulate. Major providers explicitly recommend keeping the active tool set small (on the order of tens) to preserve accuracy [2]. In multi-step agentic workflows, early retrieval errors further propagate across the trajectory, producing redundant calls, inflated token usage, and unstable execution traces.

To mitigate prompt bloat, recent work has shifted toward retrieval-based tool routing. RAG-MCP [3] retrieves relevant MCP schemas from an external index to avoid injecting the full tool space, while hierarchical agent frameworks such as AnyTool [4] and MCP-Zero [5] adopt coarse-to-fine routing to reduce search complexity. Beyond similarity-only retrieval, execution-aware methods (e.g., GRETEL [6]) attempt to close the *semantic–functional gap* by validating candidates through trial execution and reranking. These directions substantially improve scalability and robustness, but they leave a critical practical dimension under-modeled: real MCP ecosystems are *economically and operationally heterogeneous*. Tool invocations may be metered and priced, servers differ in latency and reliability, and some tools are systematically fragile due to parameter constraints and runtime dependencies. Benchmarks and empirical studies repeatedly show that semantic plausible tools can fail at execution time because of argument mismatches, missing prerequisites, or environment constraints [7]–[9]. In this setting, selecting a tool is not only a semantic-matching problem; it is also an economics- and uncertainty-aware decision.

This paper proposes that *scalable MCP tool use requires routing objectives that explicitly integrate economics and execution uncertainty*. Similarity-only ranking cannot answer questions that arise routinely in deployment: Should the agent pay for a high-cost grounding tool or attempt a cheaper alternative first? Which server should be preferred when two candidates are semantically comparable but differ in latency

The research was supported in part by an NSFC grant 62432008, RGC RIF grant R6021-20, an RGC TRS grant T43-513/23N-2, RGC CRF grants C7004-22G, C1029-22G and C6015-23G, NSFC/RGC grant CRS_HKUST601/24 and RGC GRF grants 16207922, 16207423 and 16203824, and also an NSFC grant 62232004, Collaborative Innovation Center of Novel Software Technology and Industrialization.

or failure rate? How can we enforce cost feasibility without requiring every server to reveal detailed pricing or cost structure?

We address these challenges with **Economics-aware Hierarchical Retrieval and Routing (EHRR)** (Fig. 1), a decoupled MCP co-pilot router that separates *functional abstraction* from *server/tool retrieval*. When the LLM determines that external tool support is needed, it first generates a structured subtask description that specifies the required functionality. EHRR then performs hierarchical routing: (i) at the *server layer*, it ranks servers by a utility that combines semantic relevance with expected operational overhead (latency, reliability, and post-accept failure risk), and applies a posted-price reverse auction to filter out economically infeasible servers with minimal communication; (ii) at the *tool layer*, it evaluates tools within accepted servers using a utility that jointly accounts for relevance, execution success probability, latency, and per-call price, returning only the top-n candidates to the LLM. Finally, the LLM invokes the selected tool through EXECUTE-TOOL, and EHRR updates lightweight server/tool statistics online to improve future routing decisions.

EHRR is designed to be practical and deployable. It requires no fine-tuning, operates with simple embedding-based similarity plus exponentially smoothed runtime statistics, and reduces prompt overhead by returning a compact candidate set. The posted-price mechanism provides an explicit feasibility filter at the server layer and admits a dominant-strategy accept/reject policy with ex-post individual rationality (and buyer-side budget safety under posted prices) under the reverse-auction formulation, while keeping routing complexity polynomial in the number of servers and tools.

Our original contributions are as follows. *First*, we formalize *navigation fog* in MCP ecosystems as a deployment-driven failure mode caused by the combination of tool-space explosion, semantic–functional mismatch, and heterogeneous execution economics. *Second*, we propose **EHRR**, an economics-aware hierarchical router that (a) uses LLM-generated functional subtasks to anchor retrieval, (b) performs utility-driven server and tool retrieval, and (c) enforces cost feasibility via a low-communication posted-price reverse auction. *Third*, we analyze the server-layer mechanism and establish key theoretical properties (dominant-strategy accept/reject and ex-post individual rationality) along with polynomial-time routing complexity. *Finally,* we evaluate EHRR on 95 real-world MCP tasks over 81 servers and 810 tools using two representative LLM backends, showing consistent improvements in end-to-end success alongside substantial reductions in invalid calls, token usage, and tool cost.

II. RELATED WORK

Tool-using LLM agents and large-scale tool orchestration. Tool-augmented language models extend a base LLM with external functions, enabling agents to execute actions and access non-parametric knowledge during inference. Toolformer [10] introduced self-supervised learning for inserting API calls into text, while ReAct [11] interleaves reasoning and acting to improve multi-step decision making. As tool spaces grow, ToolLLM [12] studies large-scale API usage over real-world tool collections, and ToolGen [13] unifies retrieval and calling through generation. Hierarchical agent designs such as AnyTool [4] further organize decision making via coarse-to-fine routing to mitigate combinatorial tool search. These lines of work primarily study *agent planning and tool invocation behavior*; our focus is the complementary *routing problem* in MCP ecosystems—selecting executable tools under heterogeneous latency, reliability, and pricing constraints.

Tool retrieval and routing at scale (including MCP). A key practical bottleneck in tool-using agents is prompt and schema overload: injecting large tool catalogs increases token cost and can degrade selection accuracy. Retrieval-based routing addresses this by indexing tool schemas and returning only a small candidate set to the LLM. RAG-MCP [3] retrieves MCP schemas via semantic search to reduce prompt bloat, and TeaRAG [14] proposes token-efficient agentic retrieval mechanisms for long-horizon workflows. Within the MCP setting, ScaleMCP [15] emphasizes dynamic tool discovery and auto-synchronizing tool storage (with MCP servers as the source of truth), while Tool-to-Agent Retrieval [16] embeds tools and their parent agents/servers in a shared space to avoid context dilution in multi-agent routing. Recent MCP ecosystem measurements further indicate substantial marketplace noise and heterogeneity, with many low-value or fragile servers, reinforcing the need for robust routing beyond similarity-only matching [17]. Benchmarks such as MCPToolBench++ [18] highlight the practical consequences of MCP scale: long tool schemas, variable execution success across servers, and limited context windows that constrain naive tool exposure. EHRR is compatible with retrieval-based pipelines, but explicitly models the *operational and economic heterogeneity* that similarity-only routing and indexing approaches typically abstract away.

Execution-aware selection and the semantic–functional gap. A recurring failure mode in tool routing is the semantic–functional gap: tools that appear semantic relevant may fail at runtime due to parameter constraints, missing prerequisites, or brittle execution environments. Benchmarks and analyses show that retrieval models are often not "tool-savvy" and that tool-use errors are systematic across large tool spaces [7]–[9]. Execution-aware methods attempt to close this gap by incorporating feedback from trial execution and reranking (e.g., GRETEL [6]) or by rewriting invalid invocations into executable calls (Re-Invoke [19]). For MCP routing specifically, NetMCP [20] proposes network-aware routing that combines semantic relevance with QoS signals to reduce completion time and failures under varying network conditions. EHRR is complementary to these approaches: it uses lightweight online statistics to model success probability and latency at both server and tool levels, while additionally incorporating explicit pricing signals as first-class routing objectives.

Cost- and economics-aware tool use. Several recent studies incorporate cost into tool-augmented decision making, typically optimizing a performance–cost trade-off given known tool costs. CATP-LLM [21], for example, trains LLMs to produce cost-aware tool plans under execution-time costs. However, MCP deployments introduce an additional layer of

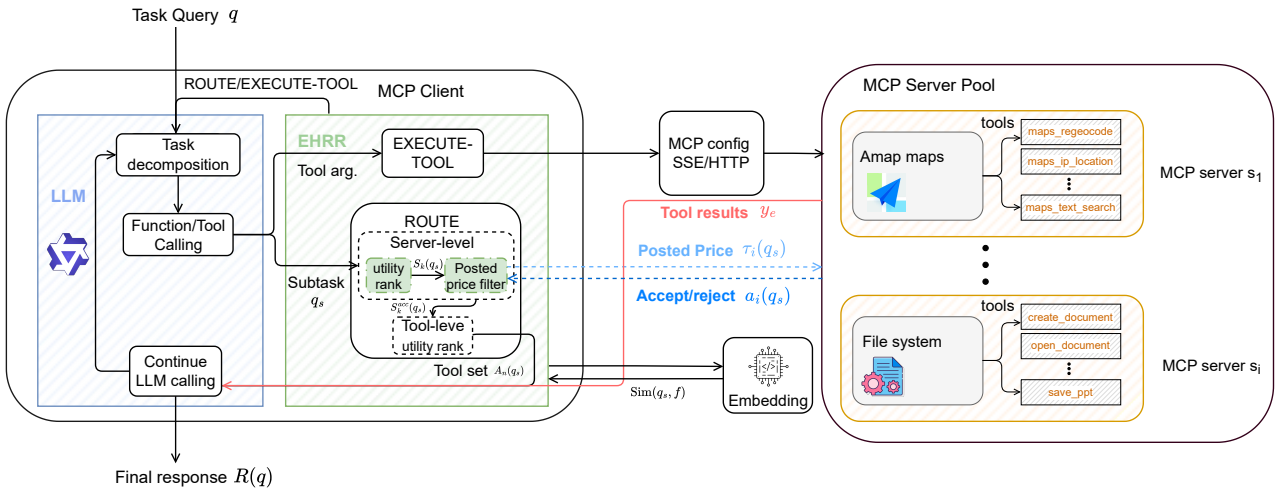

Fig. 1. Overall architecture of the EHRR framework under an MCP ecosystem. The LLM delegates tool discovery to ROUTE using a structured subtask description, and invokes the selected tool via EXECUTE-TOOL.

economic and operational heterogeneity: costs are provider-specific, may be partially private or coarse-grained at the server level, and are coupled with reliability and connection overhead. To address this setting, EHRR employs a posted-price reverse auction at the server layer to enforce economic feasibility with minimal communication and establishes a dominant-strategy accept/reject policy with ex-post individual rationality under the reverse-auction formulation [22]. This mechanism-based feasibility filter is largely orthogonal to prior retrieval- and execution-aware methods, which typically assume direct access to tool costs and do not model server-layer incentives.

III. Design of the EHRR MCP Co-pilot Router

Overall architecture and interfaces. We consider an MCP ecosystem consisting of a set of servers $\mathcal{S}$, where each server $s_i \in \mathcal{S}$ exposes a finite set of tools $\mathcal{T}_i$. Given a task query q, an LLM M operates in a multi-step reasoning–acting loop (Alg. 1) with access to two interfaces:

- ROUTE: given a structured functional subtask description q_s, return a small ranked candidate set of tools $\mathcal{A}_n(q_s)$ together with associated metadata (e.g., parameter schema, index/prices).
- EXECUTE-TOOL: execute a selected tool call through MCP and return the observed result.

A key design choice is *decoupling* (i) *functional abstraction* handled by the LLM (producing q_s) from (ii) *server/tool retrieval* handled by the router. This separation keeps the LLM prompt compact while allowing the router to optimize over operational and economic signals using lightweight online statistics.

System model and notation. For each server s_i, we assume a textual functional profile f_i (server-level description) and, for each tool $t_{i,j} \in \mathcal{T}_i$, a tool description $f_{i,j}$. A tool invocation produces (possibly stochastic) operational outcomes:

- *Latency.* Let $\ell_{i,j} \geq 0$ denote the end-to-end tool execution latency (excluding the LLM's own generation time), and let $l_{i,j} \approx \mathbb{E}[\ell_{i,j}]$ be its running estimate.
- *Execution success.* Let $Y_{i,j} \in \{0, 1\}$ indicate whether the tool call returns a *valid, usable* result. We maintain a running success estimate $r_{i,j} \approx \mathbb{P}(Y_{i,j} = 1)$.
- *Monetary price.* Let $p_{i,j} \geq 0$ be the per-invocation price charged for tool $t_{i,j}$.

In addition, the client incurs a routing overhead L^{route} (treated as constant) and a server-dependent connection establishment latency L_i^{conn}. Because the ROUTE module must score servers before choosing a specific tool, we also maintain a server-level average tool-call latency $\hat{L}_i^{\mathrm{call}}$ and a coarse server-level success estimate r_i aggregated over calls routed to server s_i.

Task decomposition and semantic similarity. When the LLM determines that external capabilities are required, it produces a structured subtask description q_s specifying the required functionality (e.g., *domain*, *operation type*, *target*). This q_s is the semantic anchor for routing.

We embed q_s and each description text f into a shared space using an embedding model, yielding vectors $\mathbf{e}(q_s), \mathbf{e}(f) \in \mathbb{R}^d$. We first compute cosine similarity

$$\cos(q_s, f) = \begin{cases} \frac{\langle \mathbf{e}(q_s), \mathbf{e}(f) \rangle}{\|\mathbf{e}(q_s)\| \, \|\mathbf{e}(f)\|}, & \|\mathbf{e}(q_s)\| \, \|\mathbf{e}(f)\| > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Since cosine similarity lies in $[-1, 1]$, we use a clipped compatibility score

$$\mathrm{Sim}(q_s, f) \triangleq \max\{0, \cos(q_s, f)\} \in [0, 1], \tag{1}$$

which avoids negative "relevance" while remaining consistent with utility maximization.

Server-layer scoring via expected time-to-success. At the server layer, the router must rank servers using only coarse information. We model two sources of execution uncertainty:

(i) tool-call failures captured by a server-level success probability r_i, and (ii) *post-accept server failure* (e.g., crash, timeout, dropped connection after agreeing to serve) captured by $p_i^{\text{fail}} \in [0, 1)$.

To be risk-aware without maintaining full posteriors, we track a running variance estimate σ_i^2 over the Bernoulli outcomes routed through server s_i, and form a conservative success estimate $\tilde{r}_i \triangleq \max\left(\epsilon,\ r_i - \sqrt{\sigma_i^2}\right)$, where $\epsilon > 0$ prevents division by zero.

Let $G_i \triangleq L^{\text{route}} + L_i^{\text{conn}}$ denote the fixed communication overhead before tool execution. If we approximate each end-to-end *attempt* through server s_i as costing $G_i + \hat{L}_i^{\text{call}}$ and succeeding with probability $(1-p_i^{\text{fail}})\tilde{r}_i$, then under geometric retries the expected time-to-success is

$$C_i^s(q_s) \triangleq \frac{G_i + \hat{L}_i^{\text{call}}}{\max\left(\epsilon,\ (1-p_i^{\text{fail}})\tilde{r}_i\right)}. \tag{2}$$

This C_i^s converts reliability and post-accept failure risk into a single latency-like quantity that can be traded off against semantic relevance.

We then define the server utility for subtask q_s as

$$U_i(q_s) = \text{Sim}(q_s, f_i) - \alpha_s\, C_i^s(q_s), \tag{3}$$

where $\alpha_s > 0$ controls the similarity–overhead trade-off. The router selects the top-k servers $\mathcal{S}_k(q_s) \subseteq \mathcal{S}$ by $U_i(q_s)$.

Posted-price feasibility filter. In real MCP deployments, servers and tools differ substantially in per-call monetary cost. Exchanging full pricing information for all servers/tools is communication-heavy and often unnecessary. We therefore apply a low-communication feasibility filter at the server layer using a posted-price signal.

For each $i \in \mathcal{S}_k(q_s)$, the client computes a posted price $\tau_i(q_s)$ that increases with semantic alignment and (weakly) with the expected operational overhead:

$$\tau_i(q_s) = p_{\text{bs}} \cdot \text{Sim}(q_s, f_i) + p_{\text{ofs}} \cdot \log\left(1 + \frac{C_i^s(q_s)}{L_0}\right), \tag{4}$$

where $p_{\text{bs}}, p_{\text{ofs}} > 0$ are pricing coefficients and L_0 is a fixed reference time constant (we use $L_0 = 1$ second) so that the logarithm is dimensionless. The $\log(\cdot)$ term imposes diminishing returns, preventing overly aggressive price inflation for high-overhead servers.

Let c_i denote a scalar per-call ask cost for server s_i (e.g., an advertised typical call price, a server-provided quote, or a client-side estimate from historical prices). The server responds with a single bit

$$a_i(q_s) = \mathbb{1}\{c_i \le \tau_i(q_s)\}. \tag{5}$$

Only servers with $a_i(q_s) = 1$ are retained for tool-layer evaluation. This step yields $O(k)$ one-bit responses per subtask, keeping communication minimal.

Tool-layer utility and candidate construction. For each accepted server s_i and tool $t_{i,j}$, the router evaluates a tool-level cost that jointly accounts for latency, reliability, server failure risk, and monetary price. Let $l_{i,j}$ and $r_{i,j}$ be the tool-level latency and success estimates, and reuse the server-level post-accept failure estimate p_i^{fail}. With $G_i = L^{\text{route}} + L_i^{\text{conn}}$ as above, we define the execution-aware costs of tool $t_{i,j}$ as

$$C_{i,j}^t(q_s) = \frac{G_i + l_{i,j}}{\max\left(\epsilon,\ (1-p_i^{\text{fail}})\, r_{i,j}\right)} + \kappa\, p_{i,j}, \tag{6}$$

where κ converts dollars to latency-equivalent units (seconds/USD). The first term is an expected time-to-success proxy under retries, while the second term incorporates explicit monetary cost. In practice, agents often *reroute* after failures rather than repeatedly retrying the same paid tool, so we scale the time term by success probability while keeping the monetary term as one-invocation cost. In deployments where billing charges per attempt and retries are common, an optional variant is to replace $\kappa p_{i,j}$ by $\kappa p_{i,j} / \max\left(\epsilon,\ (1-p_i^{\text{fail}})\, r_{i,j}\right)$.

The tool utility is then

$$U_{i,j}(q_s) = \text{Sim}(q_s, f_{i,j}) - \alpha_t\, C_{i,j}^t(q_s), \tag{7}$$

with $\alpha_t > 0$ controlling the trade-off at tool granularity.

Finally, ROUTE returns the top-n tools by $U_{i,j}(q_s)$ among all tools hosted on accepted servers. In implementation, we additionally enforce the posted-price feasibility constraint $p_{i,j} \le \tau_i(q_s)$ when constructing candidates to avoid recommending tools that violate the client's per-call willingness-to-pay.

TABLE I
TOY EXAMPLE OF TWO MCP SERVERS.

Server	$\text{Sim}(q_s, f_i)$	C_i^s (s)	U_i	τ_i (USD)	c_i (USD)	Feasible?
s_1	0.75	1.78	0.572	0.0249	0.010	Yes
s_2	0.85	3.33	0.517	0.0351	0.050	No

A. Toy Example: When Economics and Reliability Override Similarity

We illustrate how EHRR can change routing outcomes relative to similarity-only retrieval. Consider a subtask q_s with two servers s_1, s_2. Assume routing and connection latencies yield $G_1 = 0.3$s and $G_2 = 0.6$s. Let the estimated call latencies be $\hat{L}_1^{\text{call}} = 0.9$s and $\hat{L}_2^{\text{call}} = 0.6$s. Suppose the semantic similarities are $\text{Sim}(q_s, f_1) = 0.75$ and $\text{Sim}(q_s, f_2) = 0.85$, so a similarity-only router would prefer s_2.

EHRR instead uses the conservative success estimate $\tilde{r}_i$ and the expected time-to-success proxy C_i^s (Eq. (2)). Let $(\tilde{r}_1, p_1^{\text{fail}}) \approx (0.709, 0.05)$ and $(\tilde{r}_2, p_2^{\text{fail}}) \approx (0.45, 0.20)$. With $\epsilon = 10^{-3}$, we obtain $C_1^s \approx 1.78$s and $C_2^s \approx 3.33$s. Using $\alpha_s = 0.1$ (Eq. (3)), the resulting server utilities are $U_1 \approx 0.572$ and $U_2 \approx 0.517$, so EHRR prefers s_1 despite its lower semantic similarity.

TABLE II
TOY EXAMPLE OF THREE TOOLS WITHIN s_1 (WITH $\tau_1 \approx 0.0249$).

Tool	Sim	$l_{1,j}$ (s)	$r_{1,j}$	$p_{1,j}$ (USD)	$p_{1,j} \le \tau_1$	$C_{1,j}^t$	$U_{1,j}$
$t_{1,1}$	0.90	1.0	0.70	0.030	No	–	–
$t_{1,2}$	0.75	0.5	0.90	0.002	Yes	0.94	0.516
$t_{1,3}$	0.85	0.8	0.80	0.020	Yes	1.47	0.483

Next, EHRR applies the posted-price feasibility filter (Eq. (4)–Eq. (5)). With $p_{\text{bs}} = 0.0025$, $p_{\text{ofs}} = 0.0225$, and $L_0 = 1$s, the posted prices are $\tau_1 \approx 0.0249$ and $\tau_2 \approx 0.0351$ USD per call. If the servers' ask costs are $c_1 = 0.010$ and $c_2 = 0.050$ USD, then s_1 accepts while s_2 is rejected,

pruning an economically infeasible option using only one-bit responses.

Finally, within the accepted server s_1, EHRR ranks tools using Eq. (7), subject to the feasibility constraint $p_{1,j} \le \tau_1$. Let $\alpha_t = 0.25$ and $\kappa = 1$. Consider three tools: an expensive high-similarity tool that violates feasibility, and two feasible alternatives. Even among feasible tools, EHRR can prefer a lower-similarity tool when it offers higher execution reliability and lower expected time-to-success.

This example highlights two practical consequences of EHRR's formulation: (i) server selection can favor a slightly less aligned server when reliability and post-accept failure risk dominate expected completion time; (ii) the posted-price constraint enforces economic feasibility, while the tool utility further prioritizes high value-to-cost candidates among feasible tools.

Online statistics and updates. EHRR maintains lightweight online statistics using exponential moving averages. After observing a routed tool call on server s_i with outcome $y_i \in \{0,1\}$, observed call latency l_i, and an indicator $\text{fail}_i \in \{0,1\}$ for a post-accept server failure event, we update

$$\begin{cases} r_i \leftarrow (1-\lambda_r) r_i + \lambda_r \, y_i, \\ \sigma_i^2 \leftarrow (1-\lambda_\sigma)\sigma_i^2 + \lambda_\sigma \, (y_i - r_i)^2, \\ p_i^{\text{fail}} \leftarrow (1-\lambda_p) p_i^{\text{fail}} + \lambda_p \, \text{fail}_i, \\ \hat{L}_i^{\text{call}} \leftarrow (1-\lambda_l) \hat{L}_i^{\text{call}} + \lambda_l \, l_i, \end{cases} \tag{8}$$

with smoothing coefficients $\lambda_r, \lambda_\sigma, \lambda_p, \lambda_l \in (0,1)$. Tool-level statistics $(r_{i,j}, l_{i,j})$ are updated analogously when a specific tool $t_{i,j}$ is executed.

Optimization view. For a fixed subtask q_s, define the feasible tool universe after hierarchical filtering as $\mathcal{F}(q_s) \triangleq \big\{(i,j) : i \in \mathcal{S}_k(q_s),\ a_i(q_s) = 1,\ j \in \mathcal{T}_i,\ p_{i,j} \le \tau_i(q_s)\big\}$.

The ROUTE module can be viewed as solving the cardinality-limited utility maximization

$$\max_{\{x_{i,j}\}} \quad \sum_{(i,j)\in\mathcal{F}(q_s)} x_{i,j} \, U_{i,j}(q_s) \tag{9a}$$

$$\text{s.t.} \quad \sum_{(i,j)\in\mathcal{F}(q_s)} x_{i,j} \le n, \qquad x_{i,j} \in \{0,1\}. \tag{9b}$$

Because the objective is additive and the only constraint is a cardinality bound, an optimal solution is obtained by selecting the top-n tools ranked by $U_{i,j}(q_s)$ (ties broken arbitrarily), which is exactly what EHRR implements. All key notations are listed in Tab. III.

IV. Algorithms and Theoretical Properties

EHRR is implemented as a lightweight hierarchical router that is invoked on demand by the LLM during multi-step problem solving. The LLM performs task decomposition and decides *when* to retrieve or execute tools, while EHRR performs *which server/tool* retrieval using the utility functions defined in Sec. III.

A. Algorithms

LLM-orchestrated execution loop. Alg. 1 makes two roles explicit: the LLM determines *whether* routing/execution is needed and constructs the functional subtask q_s, while EHRR deterministically returns a compact candidate set based on runtime statistics and economics-aware utility. This reduces prompt bloat and prevents the LLM from repeatedly reasoning over the full tool catalog.

TABLE III
NOTATIONS AND DESCRIPTIONS.

Symbol	Description
q, q_s	task query/subtask description
$s_i \in \mathcal{S}$	MCP server i
$\mathcal{T}_i$	tool set hosted by server s_i
$t_{i,j}$	tool j on server s_i
$f_i, f_{i,j}$	server/tool textual description
$\text{Sim}(q_s, f)$	clipped cosine similarity in $[0,1]$ between q_s and description f
L^{route}	routing overhead (client-side constant)
$L_i^{\text{conn}}, \hat{L}_i^{\text{call}}$	connection, and avg. tool call latency on server s_i
$l_{i,j}, p_{i,j}$	expected execution latency and per-call monetary price of tool $t_{i,j}$
r_i, σ_i^2	server-level success estimate and variance
$r_{i,j}$	tool-level success probability estimate
p_i^{fail}	post-accept server failure probability estimate
C_i^s	server-level expected time-to-success proxy
$C_{i,j}^t$	tool-level EGL cost
U_i,$U_{i,j}$	server/tool utility
τ_i	posted price for server s_i
a_i	accept/reject bit returned by server s_i
$\mathcal{S}_k(q_s)$	top-k servers by server utility for subtask q_s
$\mathcal{A}_n(q_s)$	returned top-n tool candidates for subtask q_s
Variables	**Description**
$x_{i,j}$	whether select tool $t_{i,j}$ or not

LLM–router interaction. Alg. 1 makes the division of labor explicit. When the current dialogue context is insufficient to proceed without external capabilities, the LLM emits a ROUTE decision and produces a structured *functional* subtask description q_s^e. EHRR then returns a compact ranked set of candidates $\mathcal{A}_n(q_s^e)$ with lightweight metadata, which is appended to the context. Crucially, only this small candidate set is exposed to the LLM, avoiding repeated prompt-saturation with the full tool catalog. When the LLM commits to an action, it emits EXECUTE with a selected tool and arguments; the observed outcome and runtime signals (e.g., success/failure, latency, and post-accept server failure) are fed back to update EHRR's exponential moving-average statistics (Eq. (8)), thereby adapting future routing decisions online. The loop terminates once the LLM judges the accumulated context $\mathcal{C}_e$ sufficient and outputs STOP.

Economics-aware hierarchical routing (ROUTE). Alg. 2 performs a coarse-to-fine decision: (i) rank servers using a conservative expected time-to-success proxy, (ii) apply a posted-price accept/reject filter with $O(k)$ one-bit responses, and (iii) rank tools within accepted servers using the fine-grained utility that incorporates price, latency, and execution success probability.

Coarse-to-fine routing with feasibility screening. Alg. 2 implements a three-stage pipeline. First, it computes server

utilities (Eq. (3)) that trade off semantic compatibility against a conservative expected-time-to-success proxy (Eq. (2)), and retains only the top-k servers to control routing complexity. Second, it applies a posted-price feasibility filter: for each shortlisted server, the client posts a willingness-to-pay threshold $\tau_i(q_s)$ (Eq. (4)), and the server returns a single-bit accept/reject response (Eq. (5)). This step enforces economic feasibility with minimal communication and without requiring servers to disclose detailed cost structures. Third, within accepted servers, EHRR scores tools using the tool utility (Eq. (7)) and returns the top-n candidates together with metadata required for execution.

Algorithm 1 EHRR MCP Co-pilot Router (LLM Orchestration Loop)

Input: Task query q, server set $\mathcal{S}$, tool sets $\{\mathcal{T}_i\}_{i\in\mathcal{S}}$, router statistics; LLM M equipped with ROUTE and EXECUTE-TOOL

Output: Final natural-language response $R(q)$

Initialize dialogue context $\mathcal{C}_0 \leftarrow \{q\}$, step counter $e \leftarrow 0$
while true do
  M reads $\mathcal{C}_e$ and outputs a decision $d_e \in \{\text{ROUTE}, \text{EXECUTE}, \text{STOP}\}$
  if $d_e = \text{ROUTE}$ **then**
    M produces a structured subtask description q_s^e
    M calls $\text{ROUTE}(q_s^e)$, obtaining ranked candidates $\mathcal{A}_n(q_s^e)$ and auxiliary metadata (e.g., prices, scores)
    Update context $\mathcal{C}_{e+1} \leftarrow \mathcal{C}_e \cup \{\mathcal{A}_n(q_s^e)\}$
  else if $d_e = \text{EXECUTE}$ **then**
    Extract tool identifier (i_e, j_e) and arguments d_e.arg
    Call $\text{EXECUTE-TOOL}(t_{i_e,j_e}, d_e.\text{arg})$ to obtain result y_e and observed statistics
    Update router statistics using the observed outcome
    Update context $\mathcal{C}_{e+1} \leftarrow \mathcal{C}_e \cup \{y_e\}$
  else
    $R(q) \leftarrow$ final answer generated by M based on $\mathcal{C}_e$
    break
  end if
  $e \leftarrow e + 1$
end while
return $R(q)$

Implementation notes for deployability. (i) *Embedding cache.* EHRR precomputes and persists embeddings for server profiles $\mathbf{e}(f_i)$ and tool descriptions $\mathbf{e}(f_{i,j})$. At runtime, ROUTE only embeds the subtask q_s and performs similarity lookup against cached vectors, keeping routing latency low and avoiding repeated schema serialization. (ii) *Explicit failure surfacing.* EHRR returns routing outcomes and execution failures directly to the LLM. An empty candidate set indicates insufficient coverage (prompting subtask refinement), while execution errors (e.g., invalid arguments, timeouts, or server crashes) are surfaced verbatim to enable corrective retries or rerouting. This design keeps the router lightweight and deterministic, while leveraging the LLM's adaptive reasoning for recovery.

B. Properties of the Expected Time-to-Success Proxy

We next justify the geometric-attempts structure that underlies the server-layer proxy C_i^s (Eq. (2)) and the time-to-success term in the tool-layer cost (Eq. (6)).

Algorithm 2 ROUTE (Economics-aware Hierarchical Retrieval and Routing)

Input: Subtask q_s, server set $\mathcal{S}$ with tool sets $\{\mathcal{T}_i\}$, router statistics

Output: Top-n candidate tools $\mathcal{A}_n(q_s)$ (ranked)

Server-layer scoring:
for each server $s_i \in \mathcal{S}$ **do**
  Compute server cost $C_i^s(q_s)$ by Eq. (2)
  Compute server utility $U_i(q_s)$ by Eq. (3)
end for
Select top-k servers by utility: $\mathcal{S}_k(q_s)$
Posted-price feasibility filter:
$\mathcal{S}_k^{\text{acc}}(q_s) \leftarrow \emptyset$
for each $i \in \mathcal{S}_k(q_s)$ **do**
  Compute posted price $\tau_i(q_s)$ by Eq. (4)
  Obtain acceptance bit $a_i(q_s)$ by Eq. (5)
  if $a_i(q_s) = 1$ **then**
    Add s_i to $\mathcal{S}_k^{\text{acc}}(q_s)$
  end if
end for
Tool-layer scoring:
$\mathcal{A}(q_s) \leftarrow \emptyset$
for each accepted server $i \in \mathcal{S}_k^{\text{acc}}(q_s)$ **do**
  for each tool $j \in \mathcal{T}_i$ **do**
    if $p_{i,j} \le \tau_i(q_s)$ **then**
      Compute tool utility $U_{i,j}(q_s)$ by Eq. (7)
      Add $(i, j, U_{i,j}(q_s))$ to $\mathcal{A}(q_s)$
    end if
  end for
end for
Sort $\mathcal{A}(q_s)$ by utility and return top-n elements as $\mathcal{A}_n(q_s)$
return $\mathcal{A}_n(q_s)$ with associated metadata

Lemma 1 (Geometric Attempts: Expected Time-to-Success)**.** *Consider repeated independent execution attempts. Each attempt succeeds with probability $\beta \in (0, 1]$ and incurs deterministic time cost $L \ge 0$. Let N be the number of attempts until the first success. Then $\mathbb{E}[N] = 1/\beta$ and the expected time-to-success is $\mathbb{E}[LN] = L/\beta$.*

Proof. N is geometrically distributed with parameter β, so $\mathbb{E}[N] = 1/\beta$. Linearity of expectation gives $\mathbb{E}[LN] = L\,\mathbb{E}[N] = L/\beta$. □

Corollary 1 (Derivation of Server-Layer Proxy)**.** *Under the approximation that an attempt through server s_i costs $L = G_i + \hat{L}_i^{\text{call}}$ and succeeds with probability $\beta = (1 - p_i^{\text{fail}})\tilde{r}_i$, Lemma 1 yields $\mathbb{E}[\textit{time-to-success}] \approx \frac{G_i + \hat{L}_i^{\text{call}}}{(1-p_i^{\text{fail}})\tilde{r}_i}$, matching Eq. (2) up to the ϵ-stabilization.*

In agent traces, the LLM often reroutes after failures rather than blindly retrying the same tool; the geometric form is still a useful penalty shaping that inflates "effective overhead" as success probability decreases.

C. Monotonicity and Sensitivity Properties

Proposition 1 (Monotonicity of Execution-Aware Costs)**.** *Fix $G_i \geq 0$, $l_{i,j} \geq 0$, $p_{i,j} \geq 0$, and $\kappa > 0$. Ignoring the ϵ-clipping for clarity, the tool-layer cost $C^t_{i,j} = \frac{G_i + l_{i,j}}{(1-p_i^{\text{fail}})\, r_{i,j}} + \kappa p_{i,j}$ is strictly increasing in $l_{i,j}$, $p_{i,j}$, and p_i^{fail}, and strictly decreasing in $r_{i,j}$ over their natural domains. Analogous monotonicity holds for the server-layer proxy C^s_i in Eq.* (2)*.*

Proof. For the displayed $C^t_{i,j}$, the partial derivatives satisfy $\partial C^t_{i,j}/\partial l_{i,j} = 1/((1-p_i^{\text{fail}})r_{i,j}) > 0$, $\partial C^t_{i,j}/\partial p_{i,j} = \kappa > 0$, $\partial C^t_{i,j}/\partial r_{i,j} = -(G_i + l_{i,j})/((1-p_i^{\text{fail}})r_{i,j}^2) < 0$, and $\partial C^t_{i,j}/\partial p_i^{\text{fail}} = (G_i + l_{i,j})/((1-p_i^{\text{fail}})^2 r_{i,j}) > 0$. The server-layer case is identical in form. □

Proposition 2 (Diminishing Returns and Robustness of the Posted Price)**.** *Fix* $\text{Sim} \in [0,1]$ *and consider $\tau_i(q_s)$ as a function of $C^s_i \geq 0$: $\tau_i = p_{bs} \cdot \text{Sim}(q_s, f_i) + p_{ofs} \cdot \log\left(1 + \frac{C^s_i}{L_0}\right)$.*

Then τ_i is increasing and concave in C^s_i, with $\frac{\partial \tau_i}{\partial C^s_i} = \frac{p_{ofs}}{L_0 + C^s_i} \leq \frac{p_{ofs}}{L_0}$. Consequently, any estimation error ΔC in C^s_i perturbs τ_i by at most $(p_{ofs}/L_0)\,|\Delta C|$.

Proof. Differentiate directly. Concavity follows since $\frac{\partial^2 \tau_i}{\partial (C^s_i)^2} = -\frac{p_{\text{ofs}}}{(L_0 + C^s_i)^2} < 0$. The perturbation bound follows from the derivative upper bound. □

D. Server-Layer Incentive Properties

We analyze the posted-price filter in Eq. (5) as a posted-price procurement mechanism applied to top-k servers. For a fixed subtask q_s and a fixed posted price $\tau_i(q_s)$, let $c_i(q_s)$ denote server s_i's private per-call cost of providing service (in the minimal information setting, the server's strategy reduces to a one-bit accept/reject response).

Theorem 1 (Dominant-Strategy Acceptance and Ex-Post IR)**.** *Fix a subtask q_s and a posted price $\tau_i(q_s)$ offered to each server s_i. Server s_i has private per-call cost $c_i(q_s) \geq 0$. Under the mechanism: (i) server s_i reports a single bit $a_i \in \{0,1\}$ (accept/reject); (ii) only accepting servers may be selected for execution by an arbitrary rule that does* not *condition on $c_i(q_s)$ (which is never reported); and (iii) if selected, server s_i is paid exactly $\tau_i(q_s)$ and incurs cost $c_i(q_s)$. Then reporting* accept *iff $c_i(q_s) \leq \tau_i(q_s)$ is a dominant strategy. Moreover, under this strategy the mechanism is* ex-post individually rational*: whenever a server is selected, its realized utility is non-negative.*

Proof. If server s_i rejects, it is never selected and its utility is 0. If it accepts and is selected, its realized utility is $\tau_i(q_s) - c_i(q_s)$; if it accepts and is not selected, its utility is 0. Thus, when $c_i(q_s) \leq \tau_i(q_s)$, accepting weakly dominates rejecting (utility is never negative and is strictly positive if selected). When $c_i(q_s) > \tau_i(q_s)$, accepting can yield negative realized utility upon selection, while rejecting guarantees 0, so rejecting weakly dominates accepting. Therefore, accept iff $c_i(q_s) \leq \tau_i(q_s)$ is dominant. Ex-post IR follows immediately: under the dominant strategy, whenever selected, $\tau_i(q_s) - c_i(q_s) \geq 0$. □

Corollary 2 (Buyer Budget Safety)**.** *Under the posted-price rule, if server s_i is executed then the payment is exactly $\tau_i(q_s)$. In particular, the buyer never pays more than the posted price to any executed server.*

Proposition 3 (Budget-Capped Posted Prices Preserve DSIC/IR)**.** *Let $\tau'_i(q_s) = \min\{\tau_i(q_s), B\}$ for some buyer budget cap $B > 0$. Then Theorem 1 continues to hold with $\tau'_i(q_s)$.*

Proof. $\tau'_i(q_s)$ is still a posted price. The same accept/reject dominance argument applies. □

E. Optimality of Tool-Layer Top-n Selection

Proposition 4 (Top-n Optimality Under Additive Utility)**.** *For a fixed subtask q_s, fix the feasible tool set $\mathcal{F}(q_s)$ defined in Sec. III. The solution to the cardinality-limited maximization in Eq.* (9) *is obtained by selecting the top-n tools with the largest utilities $U_{i,j}(q_s)$.*

Proof. Eq. (9) is a linear objective with a single cardinality constraint. Any feasible solution can be improved (or left unchanged) by replacing a selected tool of lower utility with an unselected tool of higher utility, without violating feasibility. Repeating this exchange argument yields the set of the n largest utilities as an optimal solution. □

F. Robustness to Utility Estimation Error

Proposition 5 (Near-Optimality Under Bounded Utility Estimation Error)**.** *Fix a subtask q_s and feasible set $\mathcal{F}(q_s)$. Let $U_{i,j}(q_s)$ be the* true *utilities and let $\widehat{U}_{i,j}(q_s)$ be the estimated utilities used by the router. Assume a uniform bound $|\widehat{U}_{i,j}(q_s) - U_{i,j}(q_s)| \leq \delta$ for all $(i,j) \in \mathcal{F}(q_s)$. Let $\widehat{\mathcal{A}}_n$ denote the top-n set returned by maximizing $\widehat{U}_{i,j}$, and let $\mathcal{A}^\star_n$ denote an optimal size-n set under the true utilities. Then $\sum_{(i,j)\in\mathcal{A}^\star_n} U_{i,j}(q_s) - \sum_{(i,j)\in\widehat{\mathcal{A}}_n} U_{i,j}(q_s) \leq 2n\delta$.*

Proof. For any size-n set $\mathcal{A}$, the uniform bound implies $\left|\sum_{(i,j)\in\mathcal{A}} \widehat{U}_{i,j} - \sum_{(i,j)\in\mathcal{A}} U_{i,j}\right| \leq n\delta$. Since $\widehat{\mathcal{A}}_n$ maximizes the estimated sum, $\sum_{\widehat{\mathcal{A}}_n} \widehat{U} \geq \sum_{\mathcal{A}^\star_n} \widehat{U}$. Therefore, $\sum_{\widehat{\mathcal{A}}_n} U \geq \sum_{\widehat{\mathcal{A}}_n} \widehat{U} - n\delta \geq \sum_{\mathcal{A}^\star_n} \widehat{U} - n\delta \geq \sum_{\mathcal{A}^\star_n} U - 2n\delta$. □

G. Computational Complexity

Proposition 6 (Computational Complexity)**.** *For each subtask q_s, Alg. 2 runs in $O(|\mathcal{S}| \log k + m \log m)$ time, where $m = \sum_{i \in \mathcal{S}^{\text{acc}}_k(q_s)} |\mathcal{T}_i|$ is the number of tools evaluated in the tool layer. In expectation, $m \approx k\bar{T}$, where $\bar{T}$ is the average number of tools per accepted server.*

Proof. Server scoring computes $U_i(q_s)$ for all servers and performs a top-k selection, costing $O(|\mathcal{S}| \log k)$ using a heap or partial sort. The posted-price filter scans the k servers, costing $O(k)$. The tool-layer evaluates m tools and sorts them to extract the top-n candidates, costing $O(m \log m)$. Summing yields the stated bound. □

H. Communication and Prompt Complexity

Proposition 7 (One-Bit Feasibility Signaling)**.** *For each subtask q_s, the posted-price feasibility stage in Alg. 2 requires exactly one accept/reject bit from each of the k shortlisted servers. Hence, the server-layer feasibility signaling cost is k bits per routing step (up to protocol overhead), i.e., $O(k)$.*

Proof. Each shortlisted server returns a single-bit response $a_i(q_s) \in \{0, 1\}$. □

Proposition 8 (Tool-Schema Exposure Bound)**.** *Assume each returned candidate tool description plus metadata consumes at most B tokens when appended to the LLM context. Over an episode with E routing steps, EHRR appends at most $E \cdot n \cdot B$ tokens of tool-candidate information, since it exposes at most n tools per routing step. In contrast, prompt-saturation baselines that inject all tools can incur $\Theta(E \cdot |\bigcup_i \mathcal{T}_i| \cdot B)$ candidate-description tokens.*

Proof. Immediate from the fact that ROUTE returns at most n candidates per step. □

I. Online Statistics: Convergence and Stability

Lemma 2 (EMA Convergence Under Stationarity)**.** *Let $(Z_t)_{t\geq 0}$ be i.i.d. with mean μ and variance σ^2. Consider the exponential moving average update $x_{t+1} = (1-\lambda)x_t + \lambda Z_t$ for some $\lambda \in (0, 1)$. Then $\mathbb{E}[x_t] = \mu + (1-\lambda)^t(x_0 - \mu)$, so the mean error decays geometrically. Moreover, $\mathrm{Var}(x_t)$ converges to $\lim_{t\to\infty} \mathrm{Var}(x_t) = \frac{\lambda}{2-\lambda}\sigma^2$.*

Proof. Taking expectations yields $\mathbb{E}[x_{t+1}] = (1-\lambda)\mathbb{E}[x_t] + \lambda\mu$, which solves to the stated form. For the variance, using independence of Z_t and x_t gives $\mathrm{Var}(x_{t+1}) = (1-\lambda)^2\mathrm{Var}(x_t) + \lambda^2\sigma^2$. Solving the fixed-point equation $v = (1-\lambda)^2 v + \lambda^2\sigma^2$ yields $v = \lambda\sigma^2/(2-\lambda)$. □

Corollary 3 (Effective Window Length)**.** *In the stationary regime, Lemma 2 implies an "effective sample size" $N_{\text{eff}} \approx \frac{2-\lambda}{\lambda}$ in the sense that $\mathrm{Var}(x_\infty) \approx \sigma^2/N_{\text{eff}}$.*

V. EXPERIMENTAL RESULTS

Commercial MCP-style tool ecosystems exhibit heterogeneous and rapidly evolving pricing. To obtain a reproducible abstraction grounded in public pricing evidence, we derive a two-tier per-invocation cost model from representative providers (e.g., Google Gemini and Azure/OpenAI) and serverless primitives (AWS Lambda) [23]–[25]. Empirically, tool prices are bimodal, reflecting a separation between lightweight operations and grounding-intensive services.

We model each tool's per-call price $p_{i,j}$ using two complexity tiers: (i) **Low-complexity tier** $\mathcal{L}$ (e.g., embedding retrieval and local code execution), with $p_{i,j} \sim \mathcal{U}(0, 2.5 \times 10^{-3})$ USD/call; (ii) **High-complexity tier** $\mathcal{H}$ (e.g., web search and geospatial grounding), with $p_{i,j} \sim \mathcal{N}(2.25 \times 10^{-2}, (5 \times 10^{-3})^2)$ USD/call, truncated to $p_{i,j} \geq 0$. Each tool is assigned to $\mathcal{L}$ or $\mathcal{H}$ based on its declared functionality in the MCP description. Unless otherwise stated, all compared methods share the same sampled prices to ensure controlled comparisons.

Finally, the posted-price coefficients in Eq. (4) are set to $p_{\text{bs}} = 0.0025$ and $p_{\text{ofs}} = 0.0225$, matching the characteristic price scales of $\mathcal{L}$ and $\mathcal{H}$. This choice ensures that the willingness-to-pay thresholds $\tau_i(q_s)$ operate on the same order of magnitude as the underlying tool prices.

TABLE IV
TASK DISTRIBUTION ACROSS DIFFERENT CATEGORIES.

Office	Lifestyle	Leisure	Finance	Travel	Shopping	Total
31	15	14	14	12	9	95

A. Experimental Settings

We conduct our experiments on an MCP ecosystem comprising 81 servers and 810 fine-grained tools. We evaluate EHRR on 95 real-world MCP-based tasks spanning six categories—*Finance*, *Leisure*, *Lifestyle*, *Office*, *Shopping*, and *Travel*— drawn from existing MCP-based benchmarks [26]. The category-wise task distribution is summarized in Table IV.

All experiments are orchestrated on a local Apple M1-based machine, which manages the interaction with MCP servers, LLMs, and routing logic. We consider two representative large language models, namely *Qwen-Plus* and *Claude-Sonnet-4*, both accessed via their respective service-provider APIs to reflect realistic deployment settings. For text embedding, we employ *Qwen3-Embedding-0.6B* [27], deployed on a remote L40 GPU server and accessed through a vLLM API interface via SSH port forwarding. This setup decouples embedding computation from the main control process while preserving low-latency embedding lookup during routing.

Unless otherwise specified, all methods share the same pricing, and control hyperparameters to ensure a fair comparison; the complete set of hyperparameters is summarized in Table V. We adopt small candidate pool sizes (k and n) to reduce prompt length and token cost while maintaining sufficient coverage. The utility weights α_s and α_t are preselected based on experiment with annotated ground truth. This configuration achieves a stable balance between coverage and ranking quality without extensive tuning.

TABLE V
KEY HYPERPARAMETERS.

Parameter	**Value**
p_{bs} (base price)	0.0025
p_{ofs} (offset price)	0.0225
Top-k servers	5
Top-n tools	3
α_s (server utility weight)	0.1
α_t (tool utility weight)	0.25
$\lambda_r, \lambda_\sigma, \lambda_l, \lambda_p$	0.15
κ	1
ϵ	1e-3

Benchmarks. We compare against the following benchmarks:

- **Baseline.** The task query and tool descriptions are directly fed to the LLM, which performs end-to-end reasoning and tool invocation without any explicit routing.
- **RAG-MCP.** [3] A retrieval-augmented benchmark that performs semantic retrieval over available MCP tools using the *original task query* without task decomposition; we further enhance it with hierarchical server–tool routing to improve efficiency.

B. Results

We evaluate EHRR from both effectiveness and efficiency perspectives, reporting task success rate, token consumption, total tool cost, invalid route calls, invalid tool calls, and final tool utility. For each metric, we present results under two representative LLMs, *Qwen-Plus* and *Claude-Sonnet-4*. To ensure consistent and objective assessment, task success rate, invalid tool calls, and invalid route calls are evaluated using *DeepSeek-v3* as an external judge, following an analysis of the entire question-answering traces.

TABLE VI
SR IMPROVEMENTS ACROSS TASK CATEGORIES.

Category	Qwen-Plus			Claude-Sonnet-4		
	SR	↑ vs RAG	↑ vs Base	SR	↑ vs RAG	↑ vs Base
Finance	0.667	+144.4%	+1000.0%	0.714	+100.0%	+900.0%
Leisure	0.286	+33.3%	+1000.0%	0.286	+300.0%	+300.0%
Lifestyle	0.533	+100.0%	+700.0%	0.733	+266.7%	+266.7%
Office	0.516	+128.6%	+700.0%	0.742	+228.6%	+130.0%
Shopping	0.333	+1000.0%	+1000.0%	0.444	+33.3%	+1000.0%
Travel	0.500	+50.0%	+500.0%	0.583	+16.7%	+133.3%
Overall	0.484	+112.0%	+1037.1%	0.621	+136.0%	+227.8%

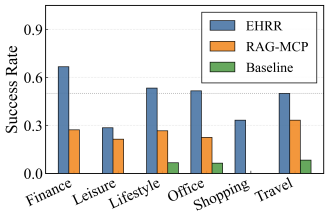

Fig. 2. Task success rate by category (Qwen-Plus).

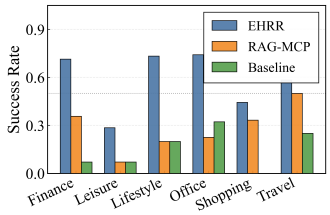

Fig. 3. Task success rate by category (Claude-Sonnet-4).

Task Success Rate. Task success rate (SR) is the fraction of tasks completed end-to-end; a task is counted as successful only if the final response satisfies all requirements and any required tool invocations return valid and usable results. We evaluate SR using DeepSeek-v3 as an external judge over the full execution traces.

Fig. 2, Fig. 3, and Table VI show that EHRR improves SR across all six categories under both LLM backends. Overall, EHRR achieves SR $= 0.484$ on Qwen-Plus and $= 0.621$ on Claude-Sonnet-4, corresponding to relative gains of $+112.0\%$ and $+136.0\%$ over RAG-MCP, respectively. The improvements are particularly show in tool-intensive categories where similarity-only routing often retrieves semantically plausible but non-executable candidates, and where prompt-saturation baselines frequently fail due to unstable tool retrieval. These gains are consistent with EHRR's design: subtasks reduce semantic ambiguity during retrieval, while execution- and economics-aware utilities suppress low-value exploration and promote candidates with higher expected payoff.

Invalid Route Calls and Invalid Tool Calls. Invalid tool calls produce empty, erroneous, or unhelpful outputs, reflecting execution-level waste. Invalid route calls select candidates that are never invoked or trigger repeated re-routing, reflecting retrieval imprecision. We report *combined invalid calls* (C-Inv) as their aggregate, since routing errors often propagate downstream.

As shown in Table VII and Fig. 4–Fig. 5, EHRR reduces C-Inv in nearly all categories under both LLMs. On Qwen-Plus, EHRR reduces total C-Inv by 54.7% relative to RAG-MCP and by 48.6% relative to the Baseline. For example, in *Finance*, EHRR incurs only 5 invalid calls versus 72 under RAG-MCP. On Claude-Sonnet-4, EHRR reduces total C-Inv by 58.6% relative to RAG-MCP and by 32.2% relative to the Baseline, with especially large reductions in *Lifestyle* and *Travel*. The only exception is *Office* under Claude-Sonnet-4, where EHRR shows a marginal $+1.1\%$ increase relative to the Baseline; nevertheless, it still reduces C-Inv by 51.9% relative to RAG-MCP in the same setting. Overall, these results indicate that EHRR improves both retrieval precision and execution quality by explicitly modeling success probability, latency, and economic feasibility during routing.

TABLE VII
COMBINED INVALID CALLS (C-INV) ACROSS TASK CATEGORIES.

Category	Qwen-Plus			Claude-Sonnet-4		
	EHRR C-Inv	↓ vs RAG	↓ vs Base	EHRR C-Inv	↓ vs RAG	↓ vs Base
Finance	5.0	93.0%	93.0%	54.0	42.6%	20.6%
Leisure	28.0	36.4%	58.8%	25.0	77.5%	51.0%
Lifestyle	41.0	51.8%	37.9%	16.0	79.5%	81.4%
Office	143.0	49.3%	20.1%	185.0	51.9%	+1.1%
Shopping	19.0	48.6%	64.8%	11.0	38.9%	69.4%
Travel	27.0	56.5%	63.5%	16.0	71.4%	44.8%
Overall	263.0	54.7%	48.6%	307.0	58.6%	32.2%

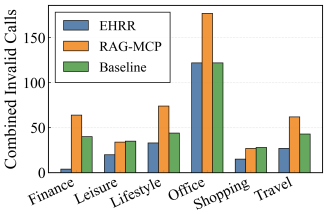

Fig. 4. Invalid route and tool calls by category (Qwen-Plus).

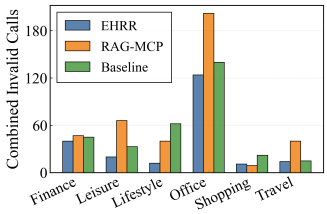

Fig. 5. Invalid route and tool calls by category (Claude-Sonnet-4).

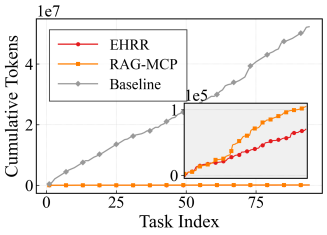

Fig. 6. Cumulative token consumption over tasks (Qwen-Plus).

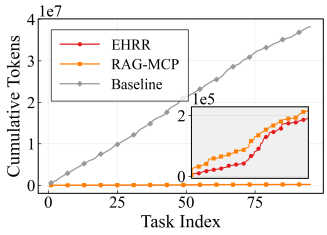

Fig. 7. Cumulative token consumption over tasks (Claude-Sonnet-4).

Token Usage. We measure token efficiency by the total number of *input* tokens consumed during routing and tool-execution prompting. Fig. 6–Fig. 7 and Table VIII show that the prompt-saturation Baseline incurs prohibitive token usage, because it repeatedly injects large tool catalogs and schemas into the LLM context. This leads to near-linear growth in prompt length over multi-step trajectories, resulting in 5.23×10^7 tokens (Qwen-Plus) and 3.82×10^7 tokens (Claude-Sonnet-4).

In contrast, RAG-MCP and EHRR restrict the context to a small retrieved set. Importantly, our RAG-MCP baseline uses the same hierarchical server–tool structure and the same

Top-k/Top-n settings as EHRR; thus, the comparison isolates the impact of economics- and execution-aware scoring. Under this controlled setting, EHRR further reduces token usage by 33.0% (Qwen-Plus) and 12.4% (Claude-Sonnet-4) relative to RAG-MCP, while achieving up to 99.9% reduction relative to the Baseline.

TABLE VIII
TOKEN USAGE AND TOOL COST COMPARISON.

Metric	Benchmarks			Reduction by EHRR	
	EHRR	RAG-MCP	Baseline	↓ vs RAG (%)	↓ vs Base (%)
Qwen-Plus					
Tokens	70,260	104,809	52,278,678	33.0%	99.9%
Cost (USD)	4.41	6.88	2500.31	35.9%	99.8%
Claude-Sonnet-4					
Tokens	191,310	218,410	38,196,230	12.4%	99.5%
Cost (USD)	7.48	15.57	1826.79	51.9%	99.6%

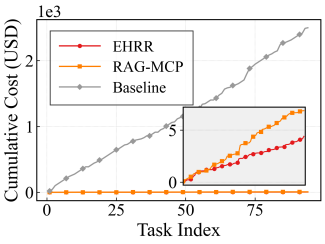

Fig. 8. Cumulative tool cost over tasks (Qwen-Plus).

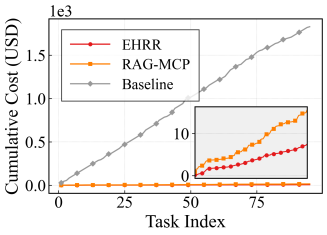

Fig. 9. Cumulative tool cost over tasks (Claude-Sonnet-4).

Total Tool Cost. We compute total tool cost as the sum of prices of all invoked tools during routing and execution, reflecting the economic efficiency of each method under the shared pricing model. Fig. 8–Fig. 9 and Table VIII show that EHRR achieves the lowest total spending under both LLM backends. While RAG-MCP already reduces cost by limiting the candidate set relative to prompt-saturation, EHRR further decreases total cost by 35.9% on Qwen-Plus and 51.9% on Claude-Sonnet-4. These savings are consistent with EHRR's feasibility screening and cost-aware tool utility, which jointly suppress repeated exposure to high-cost, low-payoff tools.

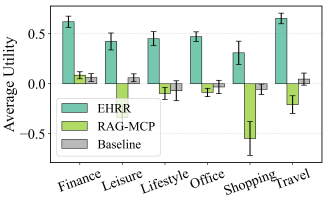

Fig. 10. Average utility per task category (Qwen-Plus).

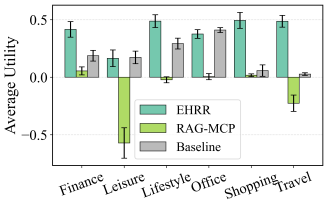

Fig. 11. Average utility per task category (Claude-Sonnet-4).

Tool Utility. Because different methods may invoke different numbers of tools, total accumulated utility can conflate decision quality with exploration scale. We therefore report the *average utility per selected tool*, which measures how effectively a method converts individual tool selections into net utility under Eq. (7).

Fig. 12–Fig. 13 show that EHRR consistently attains the highest average utility across categories for both backends. In several categories, RAG-MCP and the Baseline yield low or negative average utility, indicating that semantically plausible selections can still be net-negative once latency, failure risk, and monetary cost are accounted for. EHRR maintains positive average utility throughout, suggesting that its routing decisions better align with the intended value-to-cost objective rather than relying on similarity alone.

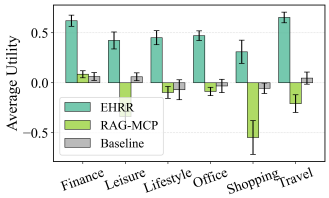

Fig. 12. Average utility per task category (Qwen-Plus).

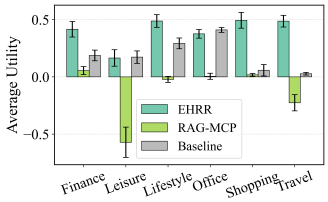

Fig. 13. Average utility per task category (Claude-Sonnet-4).

C. Ablation Study

To quantify the contribution of each component in EHRR, we conduct controlled ablations by removing one component at a time while keeping others fixed. To mitigate bias from LLM-based evaluation, we additionally report label-based routing metrics, including MRR and Recall@k, computed against annotated ground truth.

Fig. 14. Ablation Study: Success Rate.

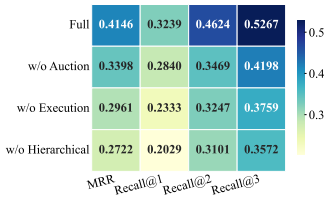

Fig. 15. Ablation Study: MRR & Recall@k.

Discussion. The results in Fig.14 and Fig.15 show consistent degradation across all metrics when any component is removed. Notably, the full model achieves the best routing quality across MRR and Recall@k (MRR: 0.41, Recall@3: 0.52), indicating that the complete design more effectively identifies and ranks relevant tools, which in turn translates into higher success rate (37.7%).

Removing the posted-price mechanism (**w/o Auction**) leads to a clear drop in both ranking quality (MRR: 0.34) and success rate (31.0%), accompanied by a significant decrease in average utility. This highlights the importance of feasibility-aware filtering in eliminating low-quality or high-cost MCP servers early. Excluding execution-aware signals (**w/o Execution**) further degrades performance (MRR: 0.30), confirming that semantic similarity alone is insufficient; incorporating execution success and latency is essential for identifying practically viable tools. The largest performance drop is observed when removing hierarchical routing (**w/o Hierarchical**), with SR decreasing to 24.1% and MRR to 0.2722. This highlights the importance of coarse-to-fine pruning in large-scale MCP environments, where directly ranking all tools introduces substantial noise and reduces both accuracy and efficiency.

VI. CONCLUDING REMARKS

This paper studies *navigation fog* in large-scale Model Context Protocol (MCP) ecosystems: as the number of available servers and tools grows, agents face a compounded failure mode where (i) semantic plausibility no longer implies

executability, (ii) prompt-saturation and uncontrolled exploration inflate tokens and tool spending, and (iii) heterogeneous latency, reliability, and pricing create non-trivial operational trade-offs that similarity-only routing cannot capture.

To address this regime, we propose **Economics-aware Hierarchical Retrieval and Routing (EHRR)**, a deployable co-pilot router that cleanly separates *functional abstraction* from *server/tool retrieval*. The LLM is responsible only for deciding when external support is needed and for generating a structured functional subtask description, while EHRR deterministically performs coarse-to-fine routing. At the server layer, EHRR ranks servers by a conservative expected-time-to-success proxy that unifies latency, reliability, and post-accept failure risk, and then applies a low-communication posted-price feasibility filter to screen out economically infeasible servers using only one-bit accept/reject responses. At the tool layer, it selects a compact top-n candidate set by maximizing an additive utility that jointly accounts for semantic alignment, execution success probability, latency, and per-call price, while maintaining lightweight online statistics via exponential moving averages.

We provide formal guarantees for key components: the posted-price server-layer filter has a dominant-strategy accept/reject policy and is ex-post individually rational (and buyer-side budget safe under posted prices), the tool-layer top-n rule is optimal for the induced cardinality-constrained additive objective, and the overall routing procedure runs in polynomial time per subtask. Empirically, experiments over 95 real-world MCP tasks spanning six categories (81 servers and 810 tools) show that EHRR improves end-to-end task success while substantially reducing invalid routing/tool calls, prompt token consumption, and total tool cost across two representative LLM backends. These results indicate that routing objectives must explicitly model both *execution uncertainty* and *economic feasibility* to scale MCP-enabled agents beyond similarity-only tool retrieval.

REFERENCES

[1] Anthropic, "Introducing the Model Context Protocol," https://www.anthropic.com/news/model-context-protocol, 2024, accessed: 2024-11-25.

[2] OpenAI, "Function Calling," https://platform.openai.com/docs/guides/function-calling, 2024.

[3] T. Gan and Q. Sun, "Rag-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation," *arXiv preprint arXiv:2505.03275*, 2025.

[4] Y. Du, F. Wei, and H. Zhang, "AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls," in *Proc. 41st International Conference on Machine Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=qFILbkTQWw

[5] X. Fei, X. Zheng, and H. Feng, "MCP-Zero: Proactive Toolchain Construction for LLM Agents from Scratch," *arXiv preprint arXiv:2506.01056*, 2025.

[6] Z. Wu, Y. Guo, C. Liang, and R. Li, "GRETEL: A Goal-driven Retrieval and Execution-based Trial Framework for LLM Tool Selection Enhancing," *arXiv preprint arXiv:2510.17843*, 2025.

[7] Z. Shi, Y. Wang, L. Yan, P. Ren, S. Wang, D. Yin, and Z. Ren, "Retrieval Models Aren't Tool-Savvy: Benchmarking Tool Retrieval for Large Language Models," in *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, 2025. [Online]. Available: https://aclanthology.org/2025.findings-acl.1258/

[8] Z. Guo, S. Cheng, H. Wang, S. Liang, Y. Qin, P. Li, Z. Liu, M. Sun, and Y. Liu, "StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models," in *ACL (Findings)*, 2024.

[9] S. Kokane, M. Zhu, T. M. Awalgaonkar, J. Zhang, A. Prabhakar, T. Q. Hoang, Z. Liu, R. R. N, L. Yang, W. Yao, J. Tan, Z. Liu, S. Heinecke, H. Wang, J. C. Niebles, C. Xiong, and S. Savarese, "ToolScan: A Benchmark For Characterizing Errors In Tool-Use LLMs," in *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. [Online]. Available: https://openreview.net/forum?id=09tnQgqKuZ

[10] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," *Advances in Neural Information Processing Systems*, vol. 36, pp. 68 539–68 551, 2023.

[11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *The 11th International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X

[12] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, dahai li, Z. Liu, and M. Sun, "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=dHng2O0Jjr

[13] R. Wang, X. Han, L. Ji, S. Wang, T. Baldwin, and H. Li, "ToolGen: Unified Tool Retrieval and Calling via Generation," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=XLMAMmowdY

[14] C. Zhang, Y. Wang, D. Xu, H. Zhang, Y. Lyu, Y. Chen, S. Liu, T. Xu, X. Zhao, Y. Gao *et al.*, "TeaRAG: A Token-Efficient Agentic Retrieval-Augmented Generation Framework," *arXiv preprint arXiv:2511.05385*, 2025.

[15] E. Lumer, A. Gulati, V. K. Subbiah, P. H. Basavaraju, and J. A. Burke, "ScaleMCP: Dynamic and Auto-Synchronizing Model Context Protocol Tools for LLM Agents," *arXiv preprint arXiv:2505.06416*, 2025.

[16] E. Lumer, F. Nizar, A. Gulati, P. H. Basavaraju, and V. K. Subbiah, "Tool-to-Agent Retrieval: Bridging Tools and Agents for Scalable LLM Multi-Agent Systems," *arXiv preprint arXiv:2511.01854*, 2025.

[17] H. Guo, Y. Hao, Y. Zhang, M. Xu, P. Lv, J. Chen, and X. Cheng, "A Measurement Study of Model Context Protocol," *arXiv preprint arXiv:2509.25292*, 2025.

[18] S. Fan, X. Ding, L. Zhang, and L. Mo, "MCPToolBench++: A Large Scale AI Agent Model Context Protocol MCP Tool Use Benchmark," *arXiv preprint arXiv:2508.07575*, 2025.

[19] Y. Chen, J. Yoon, D. Sachan, Q. Wang, V. Cohen-Addad, M. Bateni, C.-Y. Lee, and T. Pfister, "Re-Invoke: Tool Invocation Rewriting for Zero-Shot Tool Retrieval," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024, pp. 4705–4726.

[20] E. Li, H. Du, and K. Huang, "NetMCP: Network-Aware Model Context Protocol Platform for LLM Capability Extension," *arXiv preprint arXiv:2510.13467*, 2025.

[21] D. Wu, J. Wang, Y. Meng, Y. Zhang, L. Sun, and Z. Wang, "CATP-LLM: Empowering Large Language Models for Cost-Aware Tool Planning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2025/papers/Wu_CATP-LLM_Empowering_Large_Language_Models_for_Cost-Aware_Tool_Planning_ICCV_2025_paper.pdf

[22] X. Tang and H. Yu, "Fairness-Aware Reverse Auction-based Federated Learning," *IEEE Internet of Things Journal*, 2024.

[23] Google DeepMind, "Google Gemini API Pricing," https://ai.google.dev/gemini-api/docs/pricing, 2025, accessed: 2025-02-01.

[24] Microsoft Azure, "Azure OpenAI Service Pricing," https://azure.microsoft.com/pricing/details/cognitive-services/openai-service/, 2025, accessed: 2025-02-01.

[25] "AWS Lambda Pricing," https://aws.amazon.com/cn/lambda/pricing/, Amazon Web Services, 2025.

[26] G. Mo, W. Zhong, J. Chen, X. Chen, Y. Lu, H. Lin, B. He, X. Han, and L. Sun, "Livemcpbench: Can agents navigate an ocean of mcp tools?" *arXiv preprint arXiv:2508.01780*, 2025.

[27] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin *et al.*, "Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models," *arXiv preprint arXiv:2506.05176*, 2025.