ENVIRONMENT-AWARE LARGE LANGUAGE MODEL ROUTING FOR AGENTIC TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model routing aims to balance task performance and cost by selecting a suitable model for each query. In this work, we find that the preferred model for an agentic task can change with its execution environment, even when the query remains fixed. Changes in available tools, data scale, and execution feedback affect task difficulty and candidate models' performance and execution costs, revealing an important source of routing information that query-only routers overlook. We introduce EA-ROUTE, an environment-aware routing benchmark of 16k query–environment pairs from seven datasets spanning question answering, coding, and tool use. By varying tools, data scale, interface organization, and execution feedback while holding the query, answer, and evaluation fixed, EA-ROUTE shows that environmental changes can alter which model is preferred when accounting for task performance and execution cost. We propose a method for making routers environment-aware. Its key challenge is that environments are large and heterogeneous, so they cannot be passed to a router directly and must first be condensed into a compact, informative representation. To this end, we introduce an offline environment-understanding agent that explores environments and compiles routing-relevant properties into reusable extractors. These extractors generate compact descriptions without an LLM call at routing time and can be rerun as environment contents change. Optimized with downstream routing feedback, the resulting representation improves macro-averaged model-success prediction AUC by 8.9 points over query-only inputs, with consistent gains across routing paradigms and held-out environments. Our benchmark and agent-compiled representations demonstrate why execution environments matter for model routing and how to incorporate them effectively.

1 INTRODUCTION

Large language models (LLMs) differ in both capability and inference cost, making model routing important for balancing system performance and deployment cost (Chen et al., 2023; Ding et al., 2024; Yue et al., 2024). Existing methods typically predict model performance primarily from the user query, assigning simple requests to cheaper models and invoking stronger models only when necessary (Hu et al., 2024; Feng et al., 2025; Li et al., 2026). This paradigm makes an implicit assumption: the user query contains most of the information needed to estimate task difficulty and model suitability. This assumption is often reasonable for single-turn question answering, where the model only needs to produce an answer from the given query. Recent work has enriched routing inputs with retrieved passages or accumulated execution experience, broadening the information available for model selection beyond the query itself (Zhang et al., 2025a; Zhou et al., 2026).

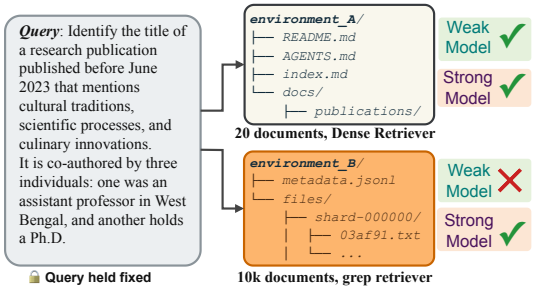

Figure 1: The preferred model for the same query can change with its environment when accounting for task performance and execution cost.

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

For agentic tasks, however, the same query can favor different models depending on the execution environment. An LLM agent completes a task by retrieving information, invoking tools, executing actions, and adapting to feedback within an *environment* — the deployment-side state beyond the user query that the model can observe or interact with during execution (Liu et al., 2024b). Because the required work depends on available tools, data scale, interface organization, or feedback quality, solving the same task can vary widely in difficulty. For example, as shown in Fig 1, the same retrieval request may require only a direct lookup in an environment containing a few relevant documents, but multiple rounds of retrieval and evidence filtering in another environment containing 10k documents, as models can degrade when relevant evidence is embedded in longer or more distracting contexts (Bai et al., 2024). Besides, environmental changes do not affect all models in the same way. Evidence position can affect performance in long contexts (Liu et al., 2024a), while stateful tool interactions introduce different failure modes (Lu et al., 2025). These capabilities make candidate models differently sensitive to environmental burden. Environmental variation can therefore change the optimal LLM selection.

We introduce Environment-Aware Route Benchmark (EA-ROUTE), the first routing benchmark to systematically study how changes in the execution environment affect model selection. Unlike existing benchmarks that evaluate each query in a single native environment (Jimenez et al., 2024; Chen et al., 2026), EA-Route pairs the same query with diverse environment variants. Its central design combines a Minimal Environment that preserves task solvability with Environment Knobs that vary scale, information access, interactive execution, and layout. The benchmark spans seven datasets covering document QA, coding, data analysis, and tool use. Actual candidate-model executions show that environmental changes alter task performance and execution cost enough to change the preferred model for the same query. These results motivate incorporating environment information into agentic model routing.

Beyond the benchmark, we propose *Agent-Compiled Environment Representations*, a new paradigm for bringing environment information into routing. Obtaining informative environment representations at low cost is challenging because environments are large, heterogeneous, and dynamic. Providing the entire environment to the router is costly and can obscure relevant properties; manually designed extractors require environment-specific engineering; and repeated LLM exploration adds recurring overhead. Our approach addresses these limitations through three key components. *Offline Compilation* uses an environment-understanding agent to actively explore the environment, identify observable routing-relevant properties, and compile procedures for extracting them into reusable executable code. *Online Profiling* executes this code to scan the current environment and produce an up-to-date environment description at runtime without LLM calls. *Routing-Guided Harness Optimization* refines the environment-profiling harness based on downstream routing performance, making the extracted information more useful for model selection.

Our experiments show that augmenting a representative query-only router (Feng et al., 2025) with environment descriptions improves success-prediction Macro-AUC by 8.91 points and yields a better quality–cost tradeoff. The benefit extends to other routing paradigms, such as preference-based and LLM-agent routers (Ong et al., 2025; Zhou et al., 2026). Ablations further show that the agent-compiled extractors outperform generic hand-crafted rules, and that routing-guided optimization adds further gains, supporting both components of our method. Finally, the gains persist in environments and datasets unseen during training.

2 ENVIRONMENT-AWARE ROUTE BENCHMARK

2.1 MINIMAL ENVIRONMENT AND ENVIRONMENT KNOBS

In agentic tasks, model performance and cost can vary with the execution environment, potentially changing routing decisions for the same query. Yet existing datasets typically provide only one environment per task, making environmental effects difficult to isolate. To study these effects while preserving task solvability, we construct a *minimal environment* with the required evidence and interfaces, then apply controlled *environment knobs* to generate variants.

A *minimal environment* consists of the evidence and operational interfaces necessary to complete a task. It provides a basis for controlled variation of the environment while retaining the task's essential requirements. The query, answer, and evaluator are unchanged, so the task is not simpli-

Category	Knob	Controlled variation	Example scenario
Environment Scale	Retrieval Corpus Size	Number of documents exposed to retrieval.	Answering a policy question from one team's handbook vs. the company-wide wiki.
	Data Record Volume	Number of records contained in a table, JSON, or CSV files.	Summarizing last month's sales vs. five years of transaction logs.
	Repository Size	Number of files in a coding repository.	Fixing a bug in a small script vs. a large monorepo.
	Tool Pool Size	Number of candidate tools.	An assistant connected only to email vs. to email, calendar, CRM, and ticketing.
Information Access	Retrieval Mode	Embedding-based dense retrieval or command-line `grep` search.	Searching an indexed knowledge base vs. keyword-searching a shared drive.
	SQL Query Access	SQL interface for structured-data retrieval.	Querying the sales database directly vs. working from exported spreadsheets.
Interactive Execution	Execution and Feedback	Whether the agent can execute code or tools, observe results, and revise subsequent actions.	Running tests to check a fix vs. a read-only setting with no execution.
Environment Layout	Substrate-Specific Layout	Organization of directories, filenames, indices, metadata, and declarations.	A well-organized project folder vs. files scattered under ad-hoc names.

Table 1: Controlled environment knobs in EA-ROUTE. The last column gives everyday examples in which a similar request meets different environments.

fied. For document QA, for example, it includes the complete documents containing the annotated evidence and the interfaces needed to search and read them, while removing unrelated documents. Through *environment knobs*, we construct a controlled variation framework spanning task modalities and intervention types. Our design captures four categories of agent–environment interaction: how much content the agent must process (Environment Scale), which interfaces it can use to access information (Information Access), whether it can execute actions and receive feedback (Interactive Execution), and how the environment is organized and presented (Environment Layout). Table 1 describes each knob under this taxonomy. Each knob changes an observable property of the execution environment while preserving the underlying task.

2.2 EA-ROUTE CONSTRUCTION

EA-ROUTE instantiates the framework across seven datasets spanning four task types. It includes document QA from BrowseComp-Plus (Chen et al., 2026) and OfficeQA-Pro (Opsahl-Ong et al., 2026), data analysis from CoDA-Bench (Zhang et al., 2026b), code tasks from SWE-bench Lite (Jimenez et al., 2024), and tool-use tasks from CRMArena-Pro (Huang et al., 2025), EnterpriseBench (Vishwakarma et al., 2025), and DeepPlanning-Shopping (Zhang et al., 2026a).

For each source task, we reconstruct a minimal environment from its annotations. We then select compatible knobs from Table 1, sample their settings, and assign a layout appropriate to the environment type. Dataset-specific knob assignments and layout families are detailed in Appendix A. These settings generate variants by adding distractor content, changing information-access or execution capabilities, or reorganizing the same content and tools. Each variant preserves the required evidence and operational support. Minimal environments undergo the answer-guided completeness check in Appendix A.1; a candidate's failure alone is not grounds for excluding an instance. Overall, EA-ROUTE comprises 16,266 query–environment pairs derived from 2,158 queries drawn from the seven source datasets.

2.3 How Environments Affect Routing Decisions

Models differ in their sensitivity to environmental changes. For the same query, an environmental change can shift candidate models' performance and cost by different amounts. On BrowseComp-Plus, growing the retrieval corpus from fewer than 160 to 46k–100k documents lowers Claude Opus 4.8's success rate by about 23 points but Qwen3-235B's by about 60 points, widening the gap between them from about 12 to about 50 points (Figure 2). Costs diverge as well: Qwen3-235B's cost levels off at larger corpus sizes, whereas Claude Opus 4.8's continues to rise (Appendix A.8). As a result, how much a stronger model's advantage is worth, and what it costs, depends on the environment rather than on the query alone.

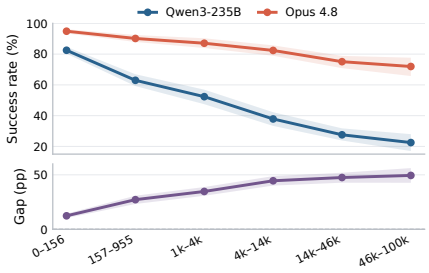

Figure 2: BrowseComp-Plus accuracy (top) and the Opus–Qwen gap (bottom) across corpus-size strata.

The preferred model for the same query can change across environments. We select the model with the higher task score and break ties by lower execution cost. For environment pairs that differ on a given knob, the *flipping rate* is the fraction for which the preferred model changes. Across dataset–knob combinations, this rate ranges from 14.5% to 55.7% (Appendix Table 7). Appendices A.6 and A.7 detail the calculation and repeated-run stability analysis, respectively.

3 Agent-Compiled Environment Representations

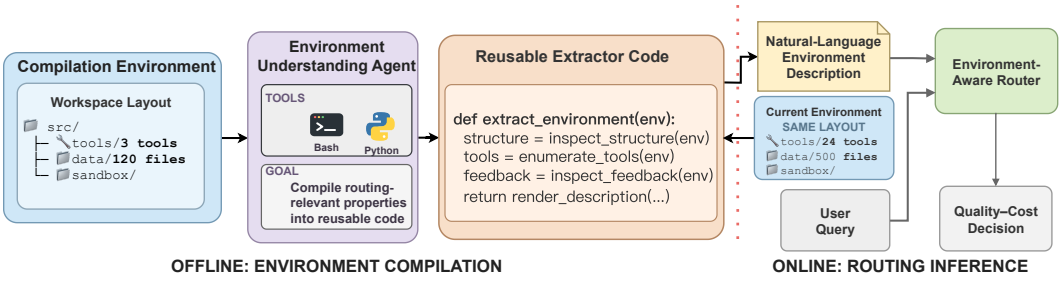

Figure 3: Agent-compiled environment representations. Offline, an environment-understanding agent explores an environment in a folder and generates reusable Python code. Online, the code describes the current environment for quality–cost routing with the user query.

Incorporating environment information into model routing is challenging because environments are large, heterogeneous, and dynamic. An environment may contain thousands of documents, tools, or code files; different environments have different structures and interfaces; and their contents and available tools can change over time. Passing the entire environment to the router is costly and can obscure the properties that matter for model selection; hand-crafted environment extraction requires environment-specific engineering and maintenance; and repeatedly asking an LLM to explore the environment introduces recurring runtime overhead. To address these challenges, we introduce agent-compiled environment representations as shown in Figure 3. Our key idea is to let an environment-understanding agent autonomously identify important environment factors and compile their extraction procedures into a reusable program, enabling up-to-date environment representations to be generated efficiently at runtime.

Offline compilation of a reusable extractor. The goal of the offline stage is to obtain a reusable program rather than a one-time environment description. By inspecting representative seed environments, the environment-understanding agent selects observable factors that may affect model performance or execution cost. The agent then determines how to get each factor, such as which directories or interfaces to access, how to compute the corresponding measurements, and how to organize the results. After exploration, it compiles these procedures into an extractor and submits

the Python code. The resulting extractor is *executable Python code* that can read the current environment and output a natural-language description of it. Because the code stores inspection procedures rather than case-specific values, the same extractor can be reused as the environment changes.

Efficient environment profiling for online routing. At runtime, the system profiles the current environment by executing the reusable extractor rather than asking an LLM to analyze the environment again. The extractor measures the selected factors and organizes their **observed values** into a natural-language description as the environment representation. The router uses the environment representation and user query to select a model, which then performs inference for the task. Because the profiling procedure has been compiled into code, the same extractor can be rerun whenever the environment changes, efficiently refreshing the representation without repeated LLM calls.

Routing-guided harness optimization. We use downstream routing feedback to optimize the environment-profiling harness, from factor selection and measurement to description generation. The agent's initial decisions about what and how to inspect may not yield representations that best support model selection. Following Meta-Harness (Lee et al., 2026), we use an LLM proposer to iteratively revise the prompt that governs the agent's environment exploration and extractor construction. Each revision produces a candidate profiling harness, which is executed to generate environment descriptions. These descriptions are used to train a router on the training partition and evaluate it on the development set. We run ten rounds of revision and select the harness with the largest area under the development-set cost–performance curve. Algorithm 1 in Appendix D.4 summarizes the search procedure, selection criterion, and validation protocol.

4 Experiments

4.1 Experimental Setup

Benchmark and evaluation protocol. We evaluate on the seven source datasets in EA-Route using Qwen3-235B (Yang et al., 2025) and Claude Opus 4.8 (Anthropic, 2026) (medium reasoning effort), both accessed through the Amazon Bedrock API. We use Claude Opus 5 for both the proposer and the environment-understanding agent. Both candidates use the same Hermes Agent scaffold (Nous Research, 2026). This shared execution agent is an intentional control to reduce differences due to scaffolding. Routing occurs once before execution, and the selected model completes the entire rollout. Outcomes are determined by each source dataset's official evaluator. We split data at the query level, keeping all environment variants of a query in the same partition, and repeat experiments over three splits and their corresponding initialization seeds.

Router and input conditions. We extend IPR (Feng et al., 2025) with an execution-cost prediction head, jointly predicting candidate-model success and cost using a shared, fine-tuned Qwen3-Embedding-0.6B encoder (Zhang et al., 2025b). We compare query-only routing, which uses the query and candidate-model identity, with environment-aware routing, which additionally receives an optimized agent-compiled environment description. Both conditions use the same backbone, candidate models, data splits, and training and evaluation procedures; only the input condition differs. Architecture and training details appear in Appendix B.2.

Routing and evaluation. The router selects a model using predicted success probability and execution cost. Sweeping the value assigned to a successful completion yields the quality–cost operating points (Appendix B.3). For success prediction, each query–environment–model example has a binary label indicating whether the execution succeeds or fails. We compute ROC-AUC from these labels and predicted success probabilities within each dataset–model cell, then average equally over cells containing both classes to obtain Macro-AUC. For cost prediction, we compute the coefficient of determination R^2 between predicted and observed $\log_{10}$ costs within each dataset–model cell, then average equally over eligible cells. Higher R^2 indicates a better fit in log-cost space. Both prediction metrics are reported as means and standard deviations over the three runs.

4.2 Can Environment Information Improve Model Routing?

Environment information improves routing across the budget range. Figure 4 compares IPR routing with query-only inputs and with environment descriptions. On the macro-average across

Metric	Query only	+ Environment
Macro-AUC ↑	61.31 ± 2.35	$\mathbf{70.22} \pm 2.05$
Cost prediction R^2 ↑	0.029 ± 0.012	$\mathbf{0.410} \pm 0.013$

Table 2: Success and cost prediction for IPR with query-only and environment-aware inputs. Values are means and standard deviations over three runs; Macro-AUC uses a 0–100 scale.

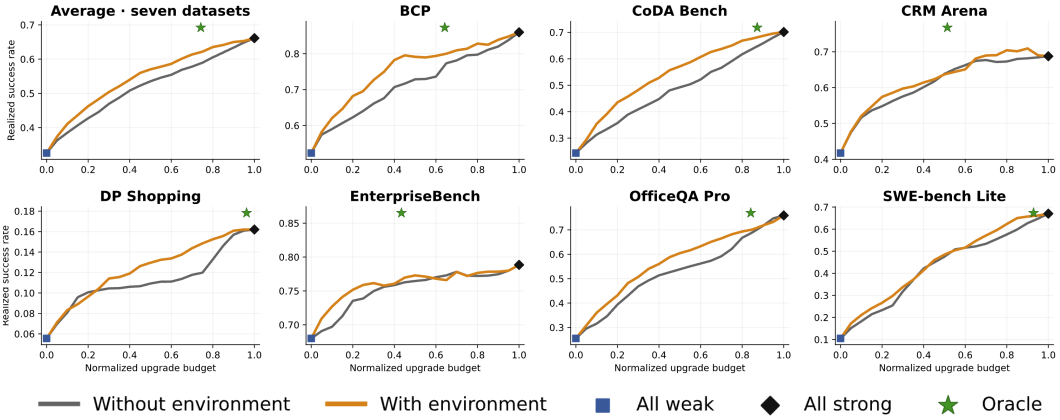

Figure 4: Realized success rate versus normalized upgrade budget for IPR on seven datasets and their macro-average. Curves compare routing without and with environment information. Squares and diamonds mark the all-weak and all-strong policies, respectively.

seven datasets, environment-aware routing matches or exceeds query-only success throughout the budget range. Appendix B.5 quantifies success gains at fixed budgets and the average and maximum success gains across the budget range. The benefit therefore extends beyond a single operating point: environment information enables more effective allocation of model calls across budgets, from low-cost operation to higher-success regimes.

Environment descriptions substantially improve success and cost prediction. As shown in Table 2, incorporating environment descriptions leads to an 8.91-point improvement in success Macro-AUC and a 0.381 increase in cost-prediction R^2. These improvements strengthen both quantities underlying the routing decision: the expected success gain from upgrading the model and the additional execution cost. Environment descriptions supply information missing from the query alone, helping the router judge more accurately when a more expensive model is worth using.

4.3 Does Environment Information Help Across Routing Paradigms?

Does the value of environment information depend on IPR's design, or can it benefit fundamentally different routing mechanisms? We examine two additional paradigms: a preference router adapted from RouteLLM (Ong et al., 2025), which directly predicts the relative execution outcomes of the two candidates, and Agent-as-a-Router (Zhou et al., 2026), which uses an LLM to select a candidate based on retrieved execution experience. We supply both methods with the same optimized environment descriptions, keeping their respective training or update procedures unchanged.

Paradigm	Metric ↑	Query only	+ Environment
Preference routing	OvR Macro-AUC	73.13 ± 2.79	$\mathbf{77.28} \pm 1.55$
Agent-as-a-Router	Balanced action accuracy	73.47	**77.89**

Table 3: Environment information across routing paradigms. Because LLM-based routing is costly, Agent-as-a-Router is evaluated once per input condition; no standard deviation is reported.

The benefits of environment information extend across routing paradigms. Table 3 shows that environment descriptions raise preference-routing OvR Macro-AUC from 73.13 to 77.28, a gain of 4.15 points, and Agent-as-a-Router balanced action accuracy from 73.47 to 77.89, a gain of 4.42 points. Together with IPR, these results span three distinct decision mechanisms: predicting individual model performance, comparing candidate preferences, and selecting models from execution experience. All three benefit from information about the environment that query-only routing omits. These findings position environment awareness as a core dimension of agent routing, with value beyond an architectural enhancement to a single method. Future work should examine not only how to build stronger routers, but also whether those routers receive the environmental information needed to make informed decisions. Environment-aware conditions should therefore be included in both method design and benchmark comparisons. Implementation and metric definitions appear in Appendices B.6 and B.7.

4.4 How Does Environment Representation Affect Routing?

We compare four representation settings and a privileged reference: **Query only** provides the query and candidate-model identity; **Hand-crafted** uses a fixed, generic extraction program generated from the benchmark's knob definitions and shared across environments; **Agent-compiled** uses the environment-understanding agent to explore the environment and its layout, then author a reusable extractor; and **Optimized** further uses routing feedback to guide the proposer in revising exploration strategies, prompting the agent to try different measurements and descriptions, and selecting extraction schemes based on development-set performance. Optimized corresponds to + Environment in the main experiments. **Privileged knob metadata** directly serializes the ground-truth environment-generation settings as the environment input, bypassing extraction. All environment-aware conditions use the same router architecture and training procedure. Appendix D.2 details the fixed-rule baseline and privileged input.

Representation	Success AUC ↑	Cost R^2 ↑
Query only	61.31 ± 2.35	0.029 ± 0.012
Hand-crafted	65.67 ± 3.29	0.318 ± 0.036
Agent-compiled	68.53 ± 0.44	0.342 ± 0.028
Optimized	$\mathbf{70.22} \pm 2.05$	$\mathbf{0.410} \pm 0.013$
Privileged knob metadata	71.87 ± 1.34	0.440 ± 0.041

Table 4: IPR representation ablation.

Layout-aware compilation and routing feedback progressively strengthen environment representations. Table 4 shows that hand-crafted descriptions raise success Macro-AUC from 61.31 to 65.67 and cost-prediction R^2 from 0.029 to 0.318. Agent compilation further improves these metrics to 68.53 and 0.342. The portability of hand-crafted rules comes at the expense of specificity: in the example in Appendix D.3, generic rules count auxiliary files such as indices alongside documents, whereas the agent-authored extractor distinguishes the document corpus from these supporting files, more directly characterizing the resources the model must search. However, understanding environment structure does not establish which attributes matter most for routing: an initial extractor may still omit useful factors or devote space to less predictive information. Routing-guided optimization evaluates alternative extraction schemes by their downstream performance. After optimization, Macro-AUC and cost R^2 reach 70.22 and 0.410, respectively, yielding cumulative gains of 4.55 AUC points and 0.092 in cost R^2 over hand-crafted descriptions. These gains support our design of combining agent-authored extraction with routing feedback to produce more effective environment representations. Optimized closely approaches the privileged-information reference. Thus, automatically extracted descriptions recover much of the predictive benefit available from explicit knowledge of the environment-generation settings, without requiring those settings as router inputs.

4.5 Can Environment-Aware Routing Generalize to Unseen Environments?

We consider four shifts beyond the training distribution. **Unseen knob dimensions** vary an environmental attribute held fixed during training, such as training with a fixed tool count and testing

with different counts. **Scale extrapolation** tests values beyond the training maximum for an attribute already varied during training, such as larger document collections. **Unseen layouts** change resource organization, for example from flat to nested document directories. **Unseen datasets** use leave-one-dataset-out evaluation, training on six source datasets and testing on the seventh in each of seven folds. For unseen layouts and datasets, the environment-understanding agent recompiles the extractor for the target structure, without using target execution outcomes to update the router or optimize the harness. Full protocols appear in Appendix C.3.

Test condition	Query only	+ Environment
Knob dimension	56.56 ± 1.67	$\mathbf{61.45} \pm 3.61$
Scale extrapolation	56.86 ± 2.34	$\mathbf{59.11} \pm 2.90$
Layout	57.48 ± 1.44	$\mathbf{62.48} \pm 2.38$
Dataset	57.86 ± 2.91	$\mathbf{63.29} \pm 1.64$

Table 5: OOD success Macro-AUC. Each condition uses its own test set.

Environment information remains valuable in previously unseen execution settings. Table 5 shows Macro-AUC gains of 4.89, 2.25, 5.00, and 5.43 points for unseen knob dimensions, scale extrapolation, unseen layouts, and unseen datasets, respectively. The gains cover both environmental changes within familiar task sources and task sources absent from training. Environment information thus remains useful for judging model performance in unfamiliar settings. Real deployments cannot be expected to match the resource scales, tool configurations, and task sources covered by training data; these results support treating environment awareness as a basic design consideration for agent routing, with value beyond improving performance on a fixed test distribution.

4.6 Analysis

Frozen routers transfer to native execution environments. To test whether the gains depend on the environmental variation introduced by our benchmark construction, we transfer routers trained on EA-Route to the source datasets' original environments. The fixed extractors run on the native environments to supply their descriptions. For example, BrowseComp-Plus uses its corpus of approximately 100k documents and an original dense retriever. Across the datasets, environment descriptions improve Macro-AUC from 70.27 to 75.62, a gain of 5.35 points. Complete settings and eligibility rules appear in Appendix C.4. These native-environment gains address the concern of circular reasoning: the evidence for environment-aware prediction extends to original benchmark environments, rather than relying solely on the variants we construct.

The gains from environment information vary across datasets. The per-dataset AUC results in Appendix C.2 show that environment descriptions improve success prediction across all seven datasets, with the largest gains for both candidates on CoDA-Bench and BrowseComp-Plus, and a further substantial gain for Opus on DeepPlanning-Shopping. Improvements are smaller on CRMArena-Pro and EnterpriseBench, consistent with the closer routing curves in Figure 4. These differences suggest that environment information is particularly valuable when resource discovery and processing account for a substantial part of the workload. In document retrieval and data analysis, corpus size, file organization, and retrieval interfaces change the amount of information an agent must search and process, even for the same query. In tool-use environments such as CRMArena-Pro and EnterpriseBench, descriptive tool names and schemas provide more direct access to the required operations, while business rules and action sequencing remain central challenges. The value of environment information therefore depends on how strongly environmental variation changes the demands placed on the candidate models.

Environment information improves prediction across fusion architectures. We compare query–environment fusion operators while fixing the agent-compiled descriptions, encoder, data splits, and optimizer settings. Every evaluated operator improves both success AUC and cost-prediction R^2 over query-only routing. The predictive value of environment information can therefore be exploited by different architectures, rather than being confined to one fusion operation. Dif-

ferences among the operators also identify further room to improve how query and environment representations interact. Architecture definitions and full results appear in Appendix C.5.

Offline compilation keeps online overhead negligible. Because extractors are compiled offline, generating a description at routing time requires no language-model call. Compiled extractors take 0.056 seconds on average, compared with 11.0 seconds for on-demand LLM description generation. The one-time authoring cost is also modest: authoring a BrowseComp-Plus extractor is estimated at \$0.36–\$0.63 in API charges, comparable to a single Opus task execution (\$0.35 on average). This cost is incurred once per layout rather than once per query, since environments sharing a layout reuse the extractor and unchanged environments can reuse cached descriptions. If one extractor serves all 830 BrowseComp-Plus queries, the amortized authoring cost falls to 0.12%–0.22% of the Opus execution cost per query. Measurement scope, pricing assumptions, and the reuse scenario are detailed in Appendix C.6.

5 Related Work

Model routing. As LLMs move from single-turn question answering to retrieval, coding, and tool use, selecting suitable models for agents becomes increasingly important for balancing task performance and deployment cost. FrugalGPT learns model cascades to reduce expensive model calls while maintaining response quality (Chen et al., 2023); RouteLLM learns from preference data to decide when a stronger model is needed (Ong et al., 2025); and IPR predicts candidate-model quality to support user-controlled quality–cost trade-offs (Feng et al., 2025). RouterBench provides a common evaluation framework for such systems (Hu et al., 2024). Recent work enriches routing inputs beyond the query: RAGRouter incorporates retrieved content (Zhang et al., 2025a), while Agent-as-a-Router draws on accumulated execution experience (Zhou et al., 2026). However, the difficulty of agentic execution depends not only on the query but also on environmental conditions such as corpus size, tool configuration, and feedback capabilities. To our knowledge, EA-Route is the first routing benchmark centered on comparing the same query across multiple environments to systematically study how execution environments affect model selection for agentic tasks. We establish the value of environment information across routing paradigms, advancing model selection informed jointly by the query and its execution environment.

Agent benchmarks and environment generation. AgentBench evaluates agents across interactive domains (Liu et al., 2024b), SWE-bench tests issue resolution in real code repositories (Jimenez et al., 2024), and τ-bench evaluates multi-turn interactions involving tools, users, and domain policies (Yao et al., 2024). These benchmarks extend evaluation from textual responses to task execution governed by environment state and action outcomes. To broaden training and evaluation coverage, ToolEmu emulates tool execution with language models for safety assessment (Ruan et al., 2024), AgentGym integrates diverse interactive environments (Xi et al., 2024), and procedural generation methods synthesize executable tool environments and tasks at scale (Sullivan et al., 2025; Song et al., 2026; Tu et al., 2026). Yet scenario diversity alone does not provide the comparisons needed for environment-aware routing: if both the environment and the query change, a model's failure cannot readily be attributed to a harder environment rather than a harder task. Adapting these generation strategies to our objective therefore requires preserving query-level comparability across environments. EA-Route fixes queries, task requirements, and evaluation criteria while varying resource scale, capabilities, and layout, and records candidate-model performance and cost to support model-selection analysis for the same task under different execution conditions.

6 Conclusion

Model routing has so far been treated as a function of the query. We show that for agentic tasks this view is incomplete: changing the execution environment alters task difficulty and execution cost, and can change whether a stronger model is worth using for the same query. With EA-Route and agent-compiled environment representations, we provide both the evidence for this effect and a practical way to exploit it, with gains across routers and unseen environments. As agents are deployed in ever larger and more dynamic environments, the execution environment should be treated as a first-class input to model routing.

AI USE STATEMENT

Generative AI tools assisted with language editing. They also supported implementation and debugging. The authors retain responsibility for the research decisions and final manuscript, including the accuracy of references, experimental claims, and AI-assisted code and artifacts.

REPRODUCIBILITY STATEMENT

Implementation details and evaluation protocols are provided in the paper and appendix. We will publicly release our dataset to support reproducibility and future research.

ETHICS STATEMENT

This work builds on existing benchmarks. Reuse of source assets remains subject to their licenses and access restrictions. Deployment of environment-aware routing maintains appropriate safeguards for agent execution. Environment-aware routing may help small companies and research teams with limited budgets deploy capable agents more affordably by allocating model calls more effectively.

REFERENCES

Agentic AI Foundation. AGENTS.md: An open format for guiding coding agents. AGENTS.md specification, 2026.

Anthropic. Claude opus 4.8 system card. Technical report, Anthropic, 2026. URL https://www.anthropic.com/system-cards.

Anthropic and Agent Skills contributors. Agent skills: An open standard for extending agent capabilities. Agent Skills specification, 2026.

Apache Arrow. Apache arrow: Dataset partitioning. Apache Arrow documentation, 2026.

Apache Maven Project. Introduction to the standard directory layout. Apache Maven documentation, 2026.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33538–33547, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1702. URL https://aclanthology.org/2025.emnlp-main.1702/.

Thomas Breuel and WebDataset contributors. WebDataset. WebDataset repository, 2019.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Sahel Sharifymoghaddam, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Hosna Oyarhoseini, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-plus: A fair and disentangled evaluation benchmark for deep search agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22349–22370, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1023. URL https://aclanthology.org/2026.acl-long.1023/.

Commit0 Project. Commit0: Library generation from scratch. https://github.com/commit-0/commit0, 2026. Repository layout documentation.

dbt Labs. About dbt projects. dbt documentation, 2026.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

EvoCodeBench Project. EvoCodeBench repository. https://github.com/seketeam/evocodebench, 2026. Source code and dependency data layout.

Exercism. Practice exercise structure. https://github.com/exercism/docs/blob/main/building/tracks/practice-exercises/README.md, 2026. Practice exercise directory documentation.

Aosong Feng, Balasubramaniam Srinivasan, Yun Zhou, Zhichao Xu, Kang Zhou, Sheng Guan, Yueyan Chen, Xian Wu, Ninad Kulkarni, Yi Zhang, et al. Ipr: Intelligent prompt routing with user-controlled quality-cost trade-offs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 2484–2498, 2025.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. Crmarena-pro: Holistic assessment of llm agents across diverse business scenarios and interactions. *arXiv preprint arXiv:2505.18878*, 2025.

Hugging Face. Create an image dataset. https://huggingface.co/docs/datasets/image_dataset, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Kaggle. Official Kaggle CLI. Kaggle CLI repository, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, 2021. doi: 10.18653/v1/2021.emnlp-demo.21. URL https://aclanthology.org/2021.emnlp-demo.21/.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, et al. Llmrouterbench: A massive benchmark and unified framework for llm routing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37733–37754, 2026.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

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, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*, 2024.

Model Context Protocol. Model context protocol reference servers. MCP reference servers, 2026.

Multi-SWE-bench Project. Multi-SWE-bench repository. https://github.com/multi-swe-bench/multi-swe-bench, 2026. Evaluation working directory layout.

Subhrangshu Nandi, Arghya Datta, Rohith Nama, Udita Patel, Nikhil Vichare, Indranil Bhattacharya, Shivam Asija, Arushi Gupta, Giuseppe Carenini, Jing Xu, et al. Sop-bench: Complex industrial sops for evaluating llm agents. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 9604–9615, 2026.

Nous Research. Hermes agent. https://github.com/NousResearch/hermes-agent, 2026.

NVIDIA. NVIDIA NeMo agent toolkit. NVIDIA documentation, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. OfficeQA Pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026. URL https://arxiv.org/abs/2603.08655.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

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

Qwen Team. Qwen-agent. https://github.com/QwenLM/Qwen-Agent, 2026.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. In *International Conference on Learning Representations*, volume 2024, pp. 27031–27098, 2024.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026.

Michael Sullivan, Mareike Hartmann, and Alexander Koller. Procedural environment generation for tool-use agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18555–18573, 2025.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*, 2021.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, et al. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. *arXiv preprint arXiv:2602.06820*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Harsh Vishwakarma, Ankush Agarwal, Ojas Patil, Chaitanya Devaguptapu, and Mahesh Chandran. Can llms help you at work? a sandbox for evaluating llm agents in enterprise environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9167–9201, 2025.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, et al. AgentGym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thought representations for cost-efficient reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 21691–21728, 2024.

Jiarui Zhang, Xiangyu Liu, Yong Hu, Chaoyue Niu, Fan Wu, and Guihai Chen. Ragrouter: Learning to route queries to multiple retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 38:15709–15737, 2025a.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7377–7407, 2026a.

Yuxin Zhang, Ju Fan, Meihao Fan, Shaolei Zhang, and Xiaoyong Du. Coda-bench: Can code agents handle data-intensive tasks? *arXiv preprint arXiv:2606.15300*, 2026b.

Pengfei Zhou, Zhiwei Tang, Yixing Ma, Jiasheng Tang, Yizeng Han, Zhenglin Wan, Fanqing Meng, Wei Wang, Bohan Zhuang, Wangbo Zhao, et al. Agent-as-a-router: Agentic model routing for coding tasks. *arXiv preprint arXiv:2606.22902*, 2026.

A EA-ROUTE CONSTRUCTION DETAILS

A.1 MINIMAL ENVIRONMENT RECONSTRUCTION

For each source query, we reconstruct a minimal environment from the source annotations while preserving the query, target answer, and evaluation procedure. The retained resources depend on the task substrate:

BrowseComp-Plus. We retain the complete annotated gold documents and the interfaces needed to search and read them. The minimal corpus contains the evidence supporting the answer, rather than unrelated corpus documents.

OfficeQA-Pro. We retain the annotated evidence documents, including the source bulletins needed to answer the query, together with their access interfaces.

CoDA-Bench. We retain the source data containing the values needed to obtain the reference result, together with the files and loading metadata required to access those data. These are task inputs from which the result must be obtained, rather than a replacement of the task by its reference answer.

SWE-bench Lite. We retain the files requiring modification according to the gold patch and configure the supporting components needed to execute the task. The retained environment therefore includes the necessary execution support in addition to the patch-touched files; the repository-size knob adds other repository files.

CRMArena-Pro. We retain the tools used in the reference trajectory and the dependent tables and records needed for those operations.

EnterpriseBench. We retain the reference-trajectory tools and the supporting case files and data needed to carry out the annotated workflow.

DeepPlanning-Shopping. We retain the tools used in the reference trajectory and the associated data entries required to carry out the reference plan.

Checking resource completeness. After constructing the minimal environment, we let Opus-5 attempt the task. A successful execution provides evidence that the retained resources support a solution. A failed attempt does not cause the instance to be discarded: instead, Opus-5 is given the reference answer to check whether the environment lacks information needed to support that answer. Any identified omissions are supplemented from the source resources. This is an answer-guided resource-completeness check, not a requirement to keep modifying an environment until Opus succeeds. The reference answer supplied for this check is not added to the task-facing environment. Inclusion is based on the availability of the required resources, not on either candidate's ability to solve the instance. Both candidates subsequently receive the same reconstructed environment for the routing evaluation.

Preservation under environment variation. Scale interventions add distractor resources without removing the retained evidence or required operational support. Added files and functions do not collide with existing names, so they do not overwrite or shadow the resources retained for the task. Layout changes reorganize resources while preserving their contents and accessibility. Information-access and execution knobs change the available assistance or feedback while retaining the evidence and operations required to produce an answer under the unchanged evaluator. Thus, added distractors can be ignored in a valid solution, while capability changes preserve a route to completing the underlying task.

Table 6 gives the number of queries, environment records, and layout families for each source dataset. The seven datasets contain 2,158 queries and 16,266 query–environment records in total. We use an approximately 70%/10%/20% query-level train/dev/test split; all environment variants of a query remain in the same partition. The splits contain 1,511 training, 215 development, and

432 test queries each. Development data support model and harness selection, while test data are reserved for final evaluation.

Dataset	Queries	Environment records	Layout families
BrowseComp-Plus	830	4,967	10
CoDA-Bench	251	1,504	8
CRMArena-Pro	200	1,999	12
DeepPlanning-Shopping	120	1,080	13
EnterpriseBench	273	1,911	13
OfficeQA-Pro	245	2,450	10
SWE-bench Lite	239	2,355	5
Total	2,158	16,266	71

Table 6: Dataset composition. Environment records count query–environment pairs, including repeated records.

A.2 Knob Ranges and Sampling

For numerical knobs, we sample realized values from dataset-specific strata spanning the minimal and maximum sampled environments. The sampled ranges extend up to 100,132 additional documents for BrowseComp-Plus, 540 source folders for CoDA-Bench, 400 data entries for DeepPlanning-Shopping, 50 bulletins for OfficeQA-Pro, and 2,544 additional repository files for SWE-bench Lite. Tool-pool knobs add up to 270 distractors for CRMArena-Pro, 1,000 for EnterpriseBench, and 983 for DeepPlanning-Shopping.

Categorical knobs draw from their admissible states, including retrieval mode, SQL-tool availability, and sandbox execution capability. For datasets with multiple knobs, configurations are randomly assigned across variants of the same query. For the per-knob flipping analysis, we compare only pairs with identical non-target knob values; these background values may differ across cases (Appendix A.6). The assignments and random seeds are fixed before candidate rollouts.

Dataset	**Task type**	**Applied knob**	**Flipping rate**
BrowseComp-Plus	Document QA	Retrieval Corpus Size	55.7%
		Retrieval Mode	26.9%
OfficeQA-Pro	Document QA	Retrieval Corpus Size	36.9%
		Retrieval Mode	25.1%
CoDA-Bench	Data analysis	Data Record Volume	43.6%
SWE-bench Lite	Code	Repository Size	14.8%
		Execution and Feedback	14.5%
CRMArena-Pro	Tool use	Tool Pool Size	24.0%
		SQL Query Access	28.0%
EnterpriseBench	Tool use	Tool Pool Size	30.8%
		Execution and Feedback	24.8%
DeepPlanning-Shopping	Tool use	Data Record Volume	38.6%
		Tool Pool Size	36.4%

Table 7: Model-preference flipping rates for the non-layout knobs applied to each dataset in EA-ROUTE. For each query, we calculate the share of environment pairs differing on the indicated knob for which the quality-first model preference changes, then average across queries; score ties are broken by lower execution cost. Layout knobs are analyzed separately in Appendix A.9, where we report success-rate differences rather than flipping rates.

A.3 LAYOUT FAMILIES

After sampling the knob state, we randomly assign a layout family appropriate to the substrate. Layouts change directory structure, file naming, indices, or tool declarations while preserving the task content and capabilities, query, target answer, and evaluator. The cited systems provide the organizational conventions; Table 8 specifies the paths and indices implemented in EA-ROUTE.

Format family	Defining organization and source
Document corpora (BrowseComp-Plus, OfficeQA-Pro)	
`beir_corpus`	BEIR-style opaque document IDs (Thakur et al., 2021), stored in a flat `corpus/` without an index.
`hf_folder_readme`	Semantic filenames with `README.md` and a manifest (Lhoest et al., 2021).
`hf_imagefolder`	ImageFolder-style partitions (Hugging Face, 2026) with an `index.md`.
`agents_md_repo`	Two-level corpus with root `TASK.md` and `AGENTS.md` guidance (Agentic AI Foundation, 2026).
`webdataset_shards`	Numbered WebDataset shards (Breuel & WebDataset contributors, 2019) with a `metadata.jsonl` index.
`maven_resources`	Corpus under `src/main/resources/` (Apache Maven Project, 2026).
`agents_md_nested`	Nested documents with `README.md`, index, and `AGENTS.md` (Agentic AI Foundation, 2026).
`memgpt_memory`	Topic directories with manifest and persistent memory file, organized as a MemGPT-style memory hierarchy (Packer et al., 2023).
`agent_skill`	Documents alongside a `SKILL.md` and references (Anthropic & Agent Skills contributors, 2026).
`gaia_attachments`	Opaque attachments indexed by metadata (Mialon et al., 2024).
Tool catalogues (EnterpriseBench, DeepPlanning-Shopping, CRMArena-Pro)	
`single_toolspec`	One schema file and one implementation module (Nandi et al., 2026).
`one_module_registry`	All tools in a single Python module (Huang et al., 2025).
`api_grouped`	Paired schema and implementation files grouped by service (Patil et al., 2025).
`per_tool_module`	One Python module per tool and a shared registry (Qwen Team, 2026).
`category_tree`	Two-level category/tool directory tree (Qin et al., 2024).
`server_dirs`	Self-contained server directories with manifests and READMEs (Model Context Protocol, 2026).
`decorator_modules`	Thematic modules with decorator-registered functions (Lu et al., 2025).
`spec_no_impl`	NESTFUL-style tool signatures (Basu et al., 2025) without callable implementations in this layout.
`yaml_manifest`	YAML declarations resolved through a registry (NVIDIA, 2025).
`parallel_renderings`	The same API in standard, function-calling, and OpenAPI forms (Trivedi et al., 2024).
`domain_bundle`	Domain directories bundle policies, tools, and tasks (Barres et al., 2025).
`skills_dir`	Tool catalogue with `SKILL.md` files and root `AGENTS.md` (NVIDIA, 2025; Anthropic & Agent Skills contributors, 2026; Agentic AI Foundation, 2026).
`opaque_ids`	Category tree with opaque file identifiers (Xie et al., 2024).
Code repositories (SWE-bench Lite)	
`swebench_testbed`	Single checkout under `testbed/` (Jimenez et al., 2024).
`commit0_repos_dir`	Target and sibling repositories under `repos/`, with root configuration (Commit0 Project, 2026).
`evocodebench_source`	Repositories under `Source_Code/`, alongside `Dependency_Data/` (EvoCodeBench Project, 2026).
`mswebench_pr_workdir`	Per-pull-request working directory `pr-<n>/` (Multi-SWE-bench Project, 2026).
`exercism_practice`	Exercise directory under `python/exercises/practice/` with hidden metadata (Exercism, 2026).
Data analysis (CoDA-Bench)	
`kaggle_native`	Dataset folders with descriptive names and metadata (Kaggle, 2026).

Continued from previous page

Format family	Defining organization and source
`kaggle_opaque`	Same folders with opaque names and no metadata sidecar (this work).
`hf_folder_readme`	Flat data folder with README and manifest (Lhoest et al., 2021).
`hf_imagefolder`	Data files in ImageFolder-style subdirectories (Hugging Face, 2026) with an index.
`webdataset_shards`	Numbered WebDataset shards (Breuel & WebDataset contributors, 2019) with a metadata index.
`hive_partitioned`	Data files in `key=value` partition directories (Apache Arrow, 2026).
`dbt_project`	Static data under `seeds/` with a root `dbt_project.yml` (dbt Labs, 2026).
`beir_corpus`	BEIR-style opaque document IDs (Thakur et al., 2021), stored in a flat data directory without a listing.

Table 8: Layout families in EA-ROUTE. Each changes the environment organization while preserving the task and evaluator.

A.4 ROLLOUT PROTOCOL

We execute each of the 16,266 query–environment pairs once with Qwen3-235B-A22B-2507 and once with Claude Opus 4.8, yielding 32,532 candidate-model rollouts. Within a dataset, both candidates receive the same workspace, tool interface, interaction budget, and termination conditions. Both candidates use the Hermes Agent scaffold with a maximum of 40 execution turns on every dataset. Each task is scored by its source dataset's official evaluator. For every rollout, including failures, we reconstruct execution cost from input, output, cache-read, and cache-write token counts using published API prices.

A.5 CANDIDATE SUCCESS RATES AND EXECUTION COSTS

Table 9 summarizes the observed binary success rate and mean dollar cost per rollout for each candidate on each dataset, using all benchmark records. Success rates are fractions of recorded successful executions; mean costs include both successes and failures. We reconstruct costs from the recorded token usage and model rates. The cost averages include estimated token usage where needed. Across all 16,266 records, the mean costs are $0.074 for Qwen3-235B and $0.40 for Opus 4.8.

	Qwen3-235B		Opus 4.8	
Dataset	Success (%)	Cost (USD)	Success (%)	Cost (USD)
BrowseComp-Plus	61.12	0.0818	88.26	0.3500
CoDA-Bench	23.01	0.0291	73.74	0.2285
CRMArena-Pro	43.22	0.0356	65.28	0.3177
DeepPlanning-Shopping	37.04	0.2303	61.85	1.1770
EnterpriseBench	68.03	0.0042	83.05	0.1468
OfficeQA-Pro	29.02	0.0730	75.92	0.4420
SWE-bench Lite	11.76	0.1037	66.41	0.4850

Table 9: Observed candidate success rates and mean execution costs across all environment records of each dataset. Costs include unsuccessful executions.

A.6 MODEL PREFERENCE AND FLIPPING RATE

We determine the preferred model for each query–environment pair using a *quality-first* rule. We select the candidate with the higher task score and, when scores are equal, the candidate with the lower realized execution cost. Task scores are the original evaluation outputs for the task, and costs are computed from the token usage and corresponding prices recorded for each rollout. The preference therefore reflects both candidates' observed outcomes and costs in that environment. A flip denotes a change in the model selected by this rule, rather than necessarily a reversal in success

rankings. For example, if Qwen is cheaper and both models succeed in one environment, the rule selects Qwen; if only Opus succeeds in another environment for the same query, it selects Opus. Such a transition captures an environment-dependent need to use the stronger model, even if Opus remains at least as successful in both environments. The reported flip rate includes these transitions and should not be interpreted as the frequency of strict capability-ranking reversals.

Within-pair control. For each dataset and target knob, we compare environments for the same query that differ in the target knob while holding all other knob values fixed within the pair. Writing an environment configuration as $(a, \mathbf{b})$, where a is the target knob and $\mathbf{b}$ contains the remaining knob values, an eligible comparison has the form $(q, a, \mathbf{b})$ versus $(q, a', \mathbf{b})$, with $a \neq a'$. Pairs that also differ in another knob are not used to compute the target knob's flipping rate.

Variation across cases and aggregation. The non-target configuration $\mathbf{b}$ is fixed within each comparison, but need not be identical across cases. For example, when testing knob A, knob B has the same value at both endpoints of a given pair; another case may use a different value of B, which is again held fixed within that case's pair. Thus, we vary the target knob under multiple sampled background configurations rather than fixing one common background configuration for the entire dataset. For each query, we calculate the fraction of its eligible pairs in which the preferred model changes. We then average these query-level fractions to obtain the flipping rate reported in Table 7. Each query receives equal weight, rather than contributing in proportion to its number of environment pairs. The resulting rate summarizes preference changes under target-knob variation across the sampled queries and background configurations; it does not assume that the target knob has the same effect in every background configuration.

A.7 Execution Randomness and Preference Stability

Repeated executions can produce different trajectories, task scores, and costs even when the query and environment are unchanged. To assess whether this variability could account for the observed preference changes, we conduct a repeated-run experiment. Repeating the full benchmark would incur substantial execution cost, so we randomly sample 3,000 query–environment pairs from the benchmark for this check.

For each selected pair, we run each candidate model three times while holding the query, environment, model settings, and execution protocol fixed. We apply the same quality-first rule to the two candidates' outcomes in each round. A query–environment pair is counted as unstable if its preferred model differs across any of the three rounds; we report the fraction of selected pairs meeting this condition.

Only approximately 3% of the selected query–environment pairs exhibit a preference change across the three rounds. Although this statistic and the cross-environment flipping rate use different aggregation units, the observed instability is substantially smaller than the 14.5%–55.7% flipping rates across environment configurations. This check suggests that model preferences are relatively stable under repeated execution on the evaluated sample and that execution randomness alone is unlikely to account for the much larger cross-environment variation.

A.8 Execution Cost Across Environment Sizes

Figure 5 shows that corpus size is associated with changes in execution cost as well as task performance. Both candidates generally become more expensive as the corpus grows, but their cost curves differ. Qwen3-235B's cost levels off at larger corpus sizes, whereas Claude Opus 4.8's cost continues to rise. Thus, the cost of selecting a candidate is not fixed by model identity alone: it also varies with the environment in which the task is executed. Together with the performance curves, this example motivates conditioning both performance and cost predictions on the query and its environment. The plotted costs describe realized execution expenditure; the curves alone do not establish which changes in execution behavior account for the increase.

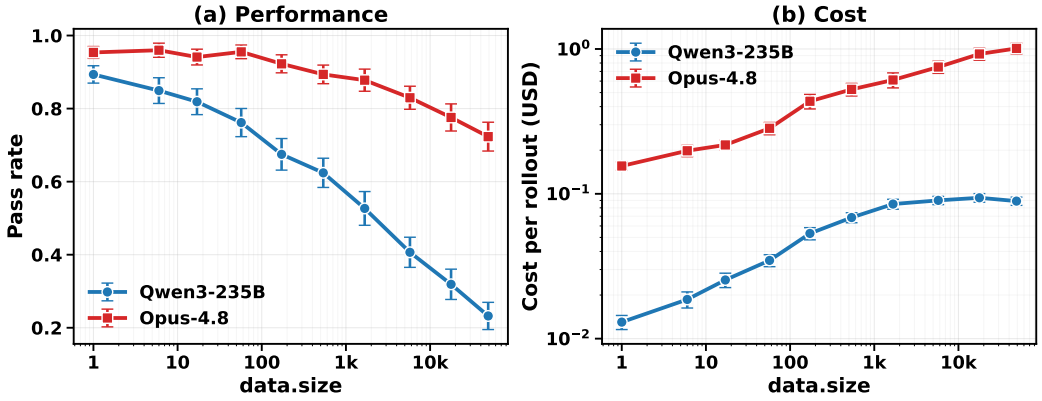

Figure 5: Task performance (left) and execution cost per rollout in USD (right) across corpus sizes on BrowseComp-Plus. The horizontal axes and the cost axis use logarithmic scales.

A.9 Effects of Environment Layout

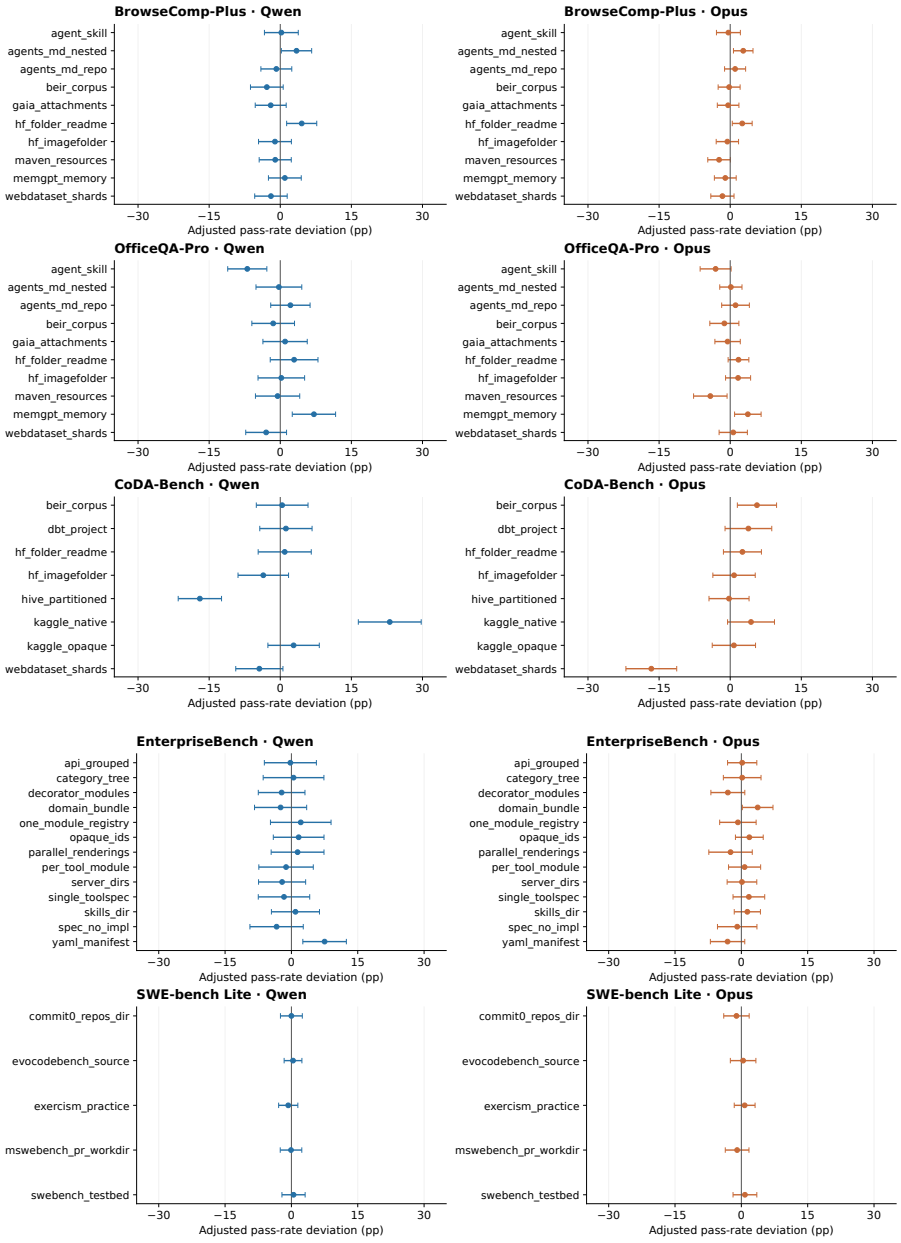

Figure 6: Adjusted success-rate deviations across layout families (percentage points). Points are centered on the record-weighted layout mean for each dataset and model; whiskers show 95% query-clustered confidence intervals. The analysis controls for query and knob states. CRMArena-Pro and DeepPlanning-Shopping are omitted because layout does not vary within query.

Layout is associated with model success after controlling for query and sampled knob states. Thus, layout can matter differently to the two candidates. These results are correlational.

B Routing Method Details

B.1 Evaluation Rationale and the Query-Only Baseline

Why pair the same query with multiple environments? Pairing each query with multiple environments holds query content fixed while varying execution conditions, allowing us to study information that is absent from the query alone. The environment interventions are defined from the task substrates and source annotations; they are not selected to require changes in success labels or model preferences. The empirical questions are how much these conditions affect execution, whether observable environment properties predict the resulting outcomes on held-out queries, and whether those predictions improve routing.

What does the query-only comparison measure? The query-only IPR router receives the query and candidate-model identity, so it assigns the same prediction to a given query–model pair across environment variants. This information restriction measures the value of observing the environment when execution conditions vary. The baseline can still learn differences in difficulty across queries and differences in performance between candidate models. Both input conditions use the same query-level splits, with all variants of a query confined to one partition. The query-only comparison establishes the value of environment information, while the Hand-crafted and privileged knob comparisons assess the effectiveness of the proposed representation.

Repeated queries do not force chance-level AUC. Success AUC is computed across query–environment examples within each dataset–model cell, not separately within each query. A positive–negative pair consisting of two variants of the same query receives tied query-only predictions and contributes 0.5 to the AUC comparison. Comparisons between different queries also contribute, however, and their predictions can be ranked differently. Consequently, repeated queries do not force the aggregate AUC to 0.5. In Table 12, query-only AUC ranges from 0.504 to 0.749: the 0.504 value occurs for SWE-bench Lite with Qwen, whereas the CRMArena-Pro and EnterpriseBench cells range from 0.690 to 0.749. These results show that query information retains substantial predictive value in several datasets despite the shared multi-environment design. Providing an environment description also does not guarantee improvement: its contents must predict held-out outcomes. Since improved success prediction need not improve model selection, we separately evaluate realized routing success and execution cost.

Evidence beyond constructed environment variants. The native-environment evaluation in Appendix C.4 directly tests whether the prediction benefit extends beyond our constructed variants. Held-out queries are executed in the source datasets' original environments, without applying our environment interventions at evaluation time. These queries are disjoint from training and development, and router weights, extractors, and decision thresholds transfer without adaptation. Over the five datasets with eligible cells, environment descriptions improve Macro-AUC from 70.27 to 75.62. Thus, the observed benefit is not confined to evaluation on artificially varied environments. Gains also persist under unseen knob dimensions and held-out datasets (Table 5; Appendix C.3), supporting transfer beyond the environmental variation and task sources observed during router training. These evaluations complement the controlled benchmark by testing the usefulness of environment information outside its training conditions.

Deployment scope. This setting is relevant to routers shared across execution environments or reused as environments evolve. The controlled benchmark identifies and measures the value of environment information, while the native and out-of-distribution evaluations test its transfer. The magnitude of deployment gains will depend on the environments encountered and the information already available to the router.

B.2 Intelligent Prompt Routing

Input construction. Following IPR (Feng et al., 2025), we convert every benchmark record into two training examples, one for each candidate model. Each example contains the query, the environment representation under the evaluated condition, the candidate-model identity, and a binary label indicating whether that model successfully completes the task. The query-only condition omits the

environment representation. The hand-crafted, agent-compiled, and optimized agent-compiled conditions use the same router architecture and differ only in the text supplied through the environment channel. The compilation prompt and harness-search algorithm appear in Appendices D.5 and D.

Encoder and fusion. We use Qwen3-Embedding-0.6B (Zhang et al., 2025b) as a shared, end-to-end fine-tuned text encoder. The query and environment are prefixed with "Query:" and "Environment:", encoded separately with the same parameters, and L2-normalized. Each input has an independent maximum sequence length of 320 tokens. The encoder produces a 1,024-dimensional embedding. For environment-aware conditions, we use the NLI-style feature combination of concatenation, absolute difference, and elementwise product (Reimers & Gurevych, 2019):

$$[\mathbf{e}_q; \mathbf{e}_e; |\mathbf{e}_q - \mathbf{e}_e|; \mathbf{e}_q \odot \mathbf{e}_e],$$

where $\mathbf{e}_q$ and $\mathbf{e}_e$ denote the query and environment embeddings.

The candidate-model identity is represented by a trainable 32-dimensional embedding. Its values are randomly initialized and optimized jointly with the remaining router parameters. Concatenating this embedding with the NLI features produces a 4,128-dimensional fused representation. The query-only condition instead concatenates the 1,024-dimensional query embedding with the same 32-dimensional model embedding.

Success head. The success head is a three-layer MLP with dimensions $d_{\text{in}} \to 256 \to 256 \to 1$. The two hidden layers use ReLU activations and dropout with probability 0.1; the final scalar is transformed by a sigmoid into the predicted success probability $\hat{p}_m$. We train this head with binary cross-entropy, using a positive-class weight equal to the ratio of negative to positive examples in the training partition. One shared router predicts both candidates: model identity is an input rather than assigning a separate network to each model.

Cost head. The cost-aware IPR variant adds a second MLP with the same input and hidden-layer architecture. It shares the text encoder, fused representation, and model-identity embedding with the success head, but has independent MLP parameters. Its target is the base-10 logarithm of the observed execution cost of the corresponding candidate model. We standardize this target using the mean and standard deviation computed over available training costs and optimize it with mean squared error. Records without recoverable cost information remain in the success loss and are masked only from the cost loss. The joint objective is

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \lambda \mathcal{L}_{\text{cost}},$$

with a cost-loss weight of 1.0. At inference time, we undo the training-set standardization and exponentiate the predicted log-cost to obtain dollars for the routing rule.

The cost-aware decision rule and operating-point sweep are specified in Appendix B.3.

Optimization and evaluation. The encoder and two heads are jointly optimized with AdamW (Loshchilov & Hutter, 2019) and weight decay 0.01. We use an encoder learning rate of 10^{-5}, a head learning rate of 10^{-4}, batch size 16, five epochs, gradient clipping at norm 1.0, and a cosine learning-rate schedule with a 6% warm-up and a minimum learning-rate ratio of 0.1. The encoder runs in bfloat16 with gradient checkpointing, while fusion and head computation use float32. Results are aggregated over three independently generated query-level train/dev/test splits and their corresponding initialization seeds. All environment variants of a query remain in the same partition. We fit the router on train and select the checkpoint with the highest pooled dev ROC-AUC over candidate-model examples. The dev split also supports harness selection; the test split is evaluated only after the router and harness are frozen.

Evaluation metrics and curve aggregation are specified in Appendix B.4.

B.3 Cost-Aware Routing Policy

For each query–environment pair, the router evaluates both candidate-model identities to obtain success probabilities $\hat{p}_m$ and execution costs $\hat{c}_m$ in dollars. Given a value $V \geq 0$, in dollars per successful completion, it selects

$$\hat{m} = \arg \max_{m \in \{\text{Qwen}, \text{Opus}\}} \left[V \hat{p}_m - \hat{c}_m \right],$$

with ties assigned to Qwen. We sweep V over zero and 240 logarithmically spaced values from 10^{-3} to 10^{3}. Each operating point is evaluated using the selected model's realized outcome and cost. The rule uses dollar-scale cost predictions obtained by reversing the training-target standardization and exponentiating the predicted log-cost.

B.4 Evaluation Metrics and Curve Aggregation

Execution cost. We reconstruct complete-rollout costs from API-reported input, output, cache-read, and cache-write token counts using the corresponding prices[1].

Prediction metrics. For each query–environment–model example, the binary success label is 1 for success and 0 otherwise. Within each dataset–model cell, ROC-AUC uses this label and the predicted success probability as its ranking score. Macro-AUC is the unweighted mean over cells containing both classes. Cost-prediction R^2 is computed within each cell using the unstandardized $\log_{10}$-dollar targets and predictions, then averaged equally over cells with at least three recoverable costs and nonzero target variance. Both metrics are summarized by their mean and standard deviation across three runs.

Quality–cost curves. For dataset d, let $C_d(V)$ be the mean realized dollar cost per query–environment instance under the policy in Appendix B.3, and let $S_d(V)$ be the fraction of selected executions that succeed, using the same binary outcomes as the AUC labels. We express cost as a normalized upgrade budget,

$$b_d(V) = \frac{C_d(V) - C_d^{\mathrm{Qwen}}}{C_d^{\mathrm{Opus}} - C_d^{\mathrm{Qwen}}},$$

where C_d^{Qwen} and C_d^{Opus} are the mean costs of always selecting the respective candidate on the same evaluation instances. Thus, budgets of zero and one correspond to the costs of always-Qwen and always-Opus. Each dataset curve plots $S_d(V)$ against $b_d(V)$. The macro-average gives each of the seven datasets equal weight at a common normalized budget, rather than averaging their dollar costs; the same budget can correspond to different values of V across datasets.

B.5 Realized IPR Routing Gains

Success at fixed budgets. To quantify the routing gains beyond success-prediction ROC-AUC, we summarize the realized quality–cost curves at normalized upgrade budgets of 0.25, 0.50, and 0.75. Using the budget definition in Appendix B.4, a budget b corresponds to mean execution cost $C_d^{\mathrm{Qwen}} + b(C_d^{\mathrm{Opus}} - C_d^{\mathrm{Qwen}})$ within dataset d. Thus, each comparison uses the same mean execution cost for both input conditions within that dataset. Let $S_d^{\mathrm{env}}(b)$ and $S_d^{\mathrm{query}}(b)$ denote their success rates. We report $100[S_d^{\mathrm{env}}(b) - S_d^{\mathrm{query}}(b)]$ in percentage points, rather than relative percentage improvements.

Average success gain across budgets. We summarize the full curves by the average success-rate improvement over the common normalized budget interval $[0, 1]$, expressed in percentage points:

$$\overline{\Delta S}_d = 100 \int_0^1 [S_d^{\mathrm{env}}(b) - S_d^{\mathrm{query}}(b)] \, \mathrm{d}b.$$

We compute this integral by trapezoidal integration of the evaluated curves on the shared grid $b = 0, 0.05, \ldots, 1$, equivalently using piecewise-linear interpolation between adjacent grid points. The three fixed-budget values lie on this grid. Signed differences are retained, including operating points where environment-aware routing performs worse. Because the interval has unit width, this integral gives the average absolute success-rate gain under uniform weighting of normalized budget. It summarizes realized routing outcomes, rather than success-prediction ROC-AUC or relative percentage improvement. Macro results average datasets equally.

[1] https://aws.amazon.com/bedrock/pricing/

Maximum success gain. We additionally report the largest success-rate difference over the 21 evaluated budget points, together with the budget at which it occurs. For the macro result, we first average the seven dataset curves at each budget and then take the maximum difference; we do not average the individual dataset maxima. These maxima describe the evaluated curves and are not operating points selected on development data.

	Success gain (percentage points)		
Dataset	$b = 0.25$	$b = 0.50$	$b = 0.75$
BrowseComp-Plus	+5.54	+6.26	+1.85
CoDA-Bench	+6.82	+7.93	+5.91
CRMArena-Pro	+2.30	-0.15	+1.90
DeepPlanning-Shopping	+0.14	+2.03	+2.86
EnterpriseBench	+2.00	+0.82	-0.04
OfficeQA-Pro	+5.15	+6.61	+5.98
SWE-bench Lite	+4.32	+0.92	+4.02
Macro-average	+3.75	+3.49	+3.21

Table 10: Realized IPR routing gains from environment descriptions relative to Query only, in percentage points (pp). Budgets are normalized upgrade budgets. Macro-averages give each dataset equal weight. Entries are descriptive point estimates from the evaluated curves.

Magnitude and variation. Environment descriptions improve macro-average success by 3.75, 3.49, and 3.21 percentage points at budgets 0.25, 0.50, and 0.75, respectively. Averaged uniformly across the budget interval, the macro success-rate gain is 2.74 percentage points; its maximum is 3.75 points at budget 0.25. The largest individual-dataset gain is 8.55 points on CoDA-Bench at budget 0.60. The largest budget-averaged gains occur on CoDA-Bench, OfficeQA-Pro, and BrowseComp-Plus. Gains vary by operating point: CRMArena-Pro decreases by 0.15 points at budget 0.50, and EnterpriseBench decreases by 0.04 points at budget 0.75, although both have positive budget-averaged gains. These summaries quantify the practical magnitude and heterogeneity of the routing benefit without equating prediction AUC improvements with routing gains. These quantities are descriptive point estimates computed from the aggregate curves.

B.6 Preference Label Routing

Labels and examples. Following the preference-based routing formulation of RouteLLM (Ong et al., 2025), we train an adapted three-class classifier with one example per query–environment pair, rather than one example per candidate model as in Appendix B.2. The label is Qwen-win when only Qwen succeeds, Opus-win when only Opus succeeds, and tie when both succeed or both fail. These labels use the candidates' binary task outcomes; execution cost does not enter the label. Candidate-model identity is not supplied as an input because the label already compares the two candidates.

Architecture. Preference routing uses the Qwen3-Embedding-0.6B encoder, query and environment prefixes, separate encoding, L2 normalization, and NLI-style fusion described in Appendix B.2. Each text input has a 320-token limit and yields a 1,024-dimensional embedding. The environment-aware fusion produces a 4,096-dimensional vector, which is passed through an MLP with dimensions $4{,}096 \to 256 \to 256 \to 3$. Its two hidden layers use ReLU activations and dropout 0.1; the output is a three-class softmax distribution. Query-only omits the environment and uses the 1,024-dimensional query embedding as the head input. The hand-crafted, agent-compiled, and optimized conditions share this architecture and differ only in the supplied environment description.

Training. We minimize weighted three-class cross-entropy. If n_k is the number of training examples in class k, its weight is $w_k = (1/n_k)/(\frac{1}{3}\sum_j 1/n_j)$, so the three weights have mean one. We use AdamW with weight decay 0.01, encoder and head learning rates of 10^{-5} and 10^{-3}, batch size 16, five epochs, gradient clipping at norm 1.0, and the same cosine schedule as IPR: 6% warm-up and a minimum learning-rate ratio of 0.1. We use three independently generated query-level train/dev/test splits and corresponding initialization seeds, with all environment variants of a query

kept in the same partition. The router is fitted on train, and the checkpoint with the highest dev macro one-vs-rest AUC is selected. The dev split also supports harness selection; test is evaluated only after both the router and harness are frozen.

Prediction and evaluation. Within each dataset and split, we compute one-vs-rest ROC-AUC separately for each class: examples of that class are positive, the other two classes are negative, and the class's predicted probability is the ranking score. A class AUC is undefined when either positive or negative examples are absent. We average the defined class AUCs within each dataset, then take the unweighted mean of the dataset-level scores. The reported mean and standard deviation are over the three splits. For a single discrete decision, the default policy selects Opus only when Opus-win has the highest predicted probability and otherwise selects Qwen. For cost–accuracy curves, we instead sweep a threshold on the predicted Opus-win probability, choosing Opus above the threshold and Qwen otherwise; realized outcomes and costs of the selected candidates determine each operating point.

B.7 AGENT-AS-ROUTER

Router and inputs. Agent-as-Router (Zhou et al., 2026) uses Claude Sonnet 5 as the routing model and is evaluated using pre-recorded candidate-model outcomes. The routing agent receives the dataset name, the current query, cards describing Qwen3-235B and Opus 4.8, and retrieved execution experience. In the environment-aware condition, both the current prompt and the retrieval key additionally include the generated environment description. Queries and environment descriptions are capped at 800 and 1,000 characters, respectively. Sonnet produces at most 256 tokens.

Online memory. Each routing condition maintains an independent online cosine-kNN memory containing only outcomes of models that it previously selected. A memory entry stores the visible context, selected model, task score, execution cost, and reward

$$r = s - 0.1c,$$

where s is the task score and c is execution cost in dollars. Counterfactual outcomes of the unselected candidate are never inserted into memory or shown to the router.

The retrieval encoder uses lowercase word unigrams and adjacent word bigrams. Features are hashed into 8,192 dimensions, weighted by $1 + \log(\text{count})$, and L2-normalized. Retrieval uses cosine similarity with $k = 10$, and memory retains the 20,000 most recent entries. The query-only condition excludes environment text from both retrieval keys and displayed historical snippets; the environment-aware condition includes it in both.

Prompt and response format. The system prompt instructs Sonnet to select exactly one candidate while maximizing task score and accounting for USD cost under the stated reward. Retrieved examples report similarity, selected model, score, cost, reward, and the visible historical context. Sonnet must return {"model": "<model_name>", "reasoning": "<12 words maximum>"}. Unparsable responses default to Qwen. On API errors, the router selects the model with the highest reward in retrieved memory, defaulting to Qwen if memory is empty.

Exploration and feedback. The online policy tries each candidate at least once within every dataset, then applies ϵ-greedy exploration with $\epsilon = 0.05$. Exploration uses a fixed random seed. Exploration remains active during both warm-up and test. After the routing decision, only the selected candidate's pre-recorded score and cost are revealed, and memory is updated before routing the next query.

Stream construction. Because LLM-based routing is costly, we evaluate each input condition once using a shared random seed. The evaluation initializes memory with 200 query-disjoint training groups sampled proportionally across datasets and then evaluates the complete test stream. One environment is selected deterministically for each query group, and the same selected environment is used by all representation conditions. Warm-up decisions are excluded from reported test metrics. Each condition maintains and updates its own memory.

Evaluation metric. We evaluate Agent-as-a-Router using balanced action accuracy, which measures how reliably the router selects the candidate with the higher realized reward. For test instance i, we define the reward of candidate m as

$$r_{i,m} = s_{i,m} - 0.1c_{i,m},$$

where $s_{i,m}$ is its task score and $c_{i,m}$ is its execution cost in USD. The target routing decision is

$$m_i^\star = \arg \max_{m \in \{\text{Qwen},\text{Opus}\}} r_{i,m},$$

with ties assigned to Qwen.

For each dataset d, we separately compute the fraction of correct decisions among instances whose target is Qwen and among those whose target is Opus, then average the two:

$$\text{BA}_d = \frac{1}{2}\left(\frac{\sum_{i \in I_{d,\text{Qwen}}} \mathbf{1}[\hat{m}_i = \text{Qwen}]}{|I_{d,\text{Qwen}}|} + \frac{\sum_{i \in I_{d,\text{Opus}}} \mathbf{1}[\hat{m}_i = \text{Opus}]}{|I_{d,\text{Opus}}|}\right),$$

where $I_{d,m} = \{i \in d : m_i^\star = m\}$ and $\hat{m}_i$ is the router's selected candidate. This gives equal weight to decisions favoring each candidate, regardless of their frequency.

The scores in Tables 3 and 11 are 100 times the unweighted mean of BA_d across datasets containing both target decisions. We compute these scores on the test stream, excluding warm-up instances. Both candidates' recorded outcomes are used to construct evaluation targets; the routing agent receives feedback only for the candidate it selects.

C Additional Routing Analyses

C.1 Representation Ablations Across Routing Paradigms

Representation	Preference Router OvR Macro-AUC ↑	Agent-as-a-Router Balanced action accuracy ↑
Query only	73.13 ± 2.79	73.47
Hand-crafted	75.47 ± 2.35	75.95
Agent-compiled	76.54 ± 1.97	77.61
Optimized	$\mathbf{77.28} \pm 1.55$	**77.89**

Table 11: Representation ablations for additional routing paradigms. Preference scores are means and standard deviations over three splits; action accuracy is averaged over eligible datasets from one run per condition, owing to the cost of LLM-based routing. Scores use a 0–100 scale and are comparable only within each paradigm.

Table 11 separates the contributions of hand-crafted descriptions, agent compilation, and harness optimization within each additional routing paradigm. Optimized descriptions correspond to the + Environment condition in Section 4.3. Training and evaluation follow Appendices B.6 and B.7.

C.2 Per-Dataset Prediction Quality

Table 12 reports success-prediction AUC for each dataset and candidate model. Environment descriptions improve AUC in all 14 dataset–model cells. Averaged over the seven datasets, the increases are 8.10 AUC points for Qwen and 9.71 points for Opus. CoDA-Bench has the largest gains for both models (22.6 and 18.4 points, respectively), followed by BrowseComp-Plus (16.6 and 17.5 points). DeepPlanning-Shopping also shows a substantial gain for Opus (15.3 points), compared with 3.6 points for Qwen. Gains are smaller on CRMArena-Pro (4.0 points for Qwen and 2.5 for Opus), EnterpriseBench (2.3 and 1.3 points), and OfficeQA-Pro (3.3 and 2.7 points). SWE-bench Lite improves by 4.3 points for Qwen and 10.3 for Opus. These results show that the value of environment information varies across datasets and candidate models, complementing the quality–cost curves in Figure 4.

Table 12: Success-prediction AUC by dataset and candidate model, comparing query-only and environment-enhanced routing. AUC values use a 0–1 scale.

	Query-only		Environment-enhanced	
Dataset	Qwen3-235B	Opus 4.8	Qwen3-235B	Opus 4.8
BrowseComp-Plus	0.559	0.541	0.725	0.716
CoDA-Bench	0.553	0.543	0.779	0.727
CRMArena-Pro	0.749	0.749	0.789	0.774
DeepPlanning-Shopping	0.610	0.561	0.646	0.714
EnterpriseBench	0.690	0.747	0.713	0.760
OfficeQA-Pro	0.593	0.569	0.626	0.596
SWE-bench Lite	0.504	0.616	0.547	0.719

C.3 Out-of-Distribution Evaluation Protocols

We evaluate environment-aware IPR under four distribution shifts. Within each setting, Query only and + Environment use the same target test set, candidate models, and training and evaluation procedures. Each setting has its own test set, so Table 5 compares input conditions within a setting rather than the relative difficulty of different shifts.

Unseen knob dimensions. We evaluate every environment knob in turn, holding that attribute fixed during training and varying it at test time, then aggregate across the tested knobs. For example, training environments may share a fixed tool count while test environments vary tool counts. The held-out element is the dimension of variation, rather than a particular value of a dimension already varied in training. This setting tests prediction under a type of environmental change absent from the training examples.

Scale extrapolation. For each numerical environment attribute, we order the sampled scale values and use the lower 70% for training; the upper 30% are reserved for extrapolation evaluation. Test values therefore exceed the corresponding maximum observed during training. Training and test environments vary the same attribute, but test environments extend beyond the observed scale, for example by using larger document collections. This setting evaluates extrapolation rather than interpolation between observed values.

Unseen layouts. For each dataset, we randomly select two complete layout families to hold out from router training and evaluate on those families. Layout changes alter how resources are organized or presented, such as document directory structures or tool declaration formats, while preserving task content and semantics. The environment-understanding agent recompiles an extractor from observable target-environment contents, without access to gold answers or candidate-model outcomes. The resulting extractor is fixed for evaluation, and the router receives no outcome-based adaptation on the target layout.

Unseen datasets. We perform seven leave-one-dataset-out folds, training the router on six source datasets and evaluating it on the seventh. The environment-understanding agent compiles an extractor from the held-out dataset's observable environment contents. Target gold answers, routing labels, and candidate-model outcomes are not used for extractor compilation, router training, or harness optimization. This setting evaluates a transferred router with an extractor adapted to the target environment structure.

Metrics and aggregation. All settings use success Macro-AUC as defined in Appendix B.4: ROC-AUC is computed within each dataset–candidate-model cell containing both success and failure labels, then averaged equally across eligible cells. Leave-one-dataset-out evaluation aggregates eligible cells from the held-out datasets across all seven folds. Table 5 reports means and standard deviations over three random seeds.

C.4 NATIVE-ENVIRONMENT EVALUATION

We evaluate held-out queries in the original execution environments supplied by the source datasets to test whether the benefits of environment information extend beyond the controlled variants constructed for EA-ROUTE. For example, BrowseComp-Plus retains its corpus of approximately 100k documents and its original Qwen3-Embedding dense retriever, instead of our modified corpus sizes or retrieval configurations. Evaluation queries are disjoint from training and development queries. Router weights, environment extractors, and decision thresholds are transferred without adaptation, and candidate-model outcomes from executions in these native environments are used for evaluation.

CRMArena-Pro and DeepPlanning-Shopping have single-class dataset–model cells in this evaluation, for which ROC-AUC is undefined. These cells are excluded from the aggregate. Over the remaining five datasets, Macro-AUC is 70.27 for Query only and 75.62 for + Environment, a gain of 5.35 points. Metric computation follows Appendix B.4.

C.5 QUERY–ENVIRONMENT FUSION

Fusion operator	Macro-AUC ↑	Cost-prediction R^2 ↑
Query only (main-table reference)	61.31	0.029
NLI features (Reimers & Gurevych, 2019)	68.53	0.342
Self-attention (Vaswani et al., 2017)	68.12	0.382
Concatenation	68.43	0.247
Addition + gated interaction	67.74	0.305
Addition	67.52	0.407
Element-wise product	67.23	0.351

Table 13: Sensitivity to query–environment fusion. All variants also receive a model-identity embedding. AUC is on a 0–100 scale. The query-only row is the main-table reference; the other six rows are the fusion sweep.

All six evaluated fusion operators use agent-compiled environment descriptions, the same Qwen3-Embedding-0.6B encoder, model-identity embedding, query-level splits, cost target, and optimizer settings. Every operator improves both reported metrics over the query-only reference. Checkpoint selection is rerun for each operator.

Operator definitions. Let q and e be the separately encoded, normalized query and environment embeddings, and let m be the model-identity embedding. Concatenation uses $[q; e; m]$, element-wise product uses $[q \odot e; m]$, and NLI features use $[q; e; |q - e|; q \odot e; m]$ (Reimers & Gurevych, 2019). Each resulting vector is passed to separate success and cost MLP heads. The query-only reference instead uses $[q; m]$.

Self-attention (Vaswani et al., 2017) treats $[q, e]$ as a sequence of two vectors, applies eight-head self-attention with dropout 0.1, flattens both output vectors, and concatenates m before the prediction heads. Addition uses separate query and environment MLP towers and sums their scalar outputs: $f_q([q; m]) + f_e([e; m]) + b$ for the success logit, with separate towers for cost prediction.

Addition + gated interaction uses the additive towers plus a gated interaction residual, $\alpha f_{\text{int}}([q; e; q \odot e; m])$, where the learned scalar α is initialized to zero; the cost head has its own towers and gate.

C.6 ENVIRONMENT DESCRIPTION LATENCY AND AMORTIZED AUTHORING COST

Description latency. On 3,266 records for which both methods succeed (BrowseComp-Plus, EnterpriseBench, and SWE-bench Lite), compiled extractors take 0.056 seconds on average (median 0.042), compared with 11.0 seconds (median 10.3) for on-demand LLM generation. Timings cover description generation only.

Authoring cost. Estimated from API token usage and prices, authoring one BrowseComp-Plus extractor costs about \$0.36–\$0.63, including retries; the range reflects uncertainty in long-context pricing. The estimate covers extractor authoring.

Amortization. The average Opus execution cost on BrowseComp-Plus is $0.35 per task. If one extractor serves all 830 BrowseComp-Plus queries, the amortized authoring cost is 0.12%–0.22% of this per query. Extractors are rerun when an environment changes, while unchanged environments can reuse cached descriptions.

D Environment Understanding Agent Details

This section describes the compiler interface, representation baselines, example environment descriptions, and routing-guided harness search. The Optimized condition uses the selected harness from the ten-round search described below.

D.1 Compiler Interface and Information Available During Authoring

The compiler contains 71 Python extractors, one for each dataset–layout family. An offline environment-understanding agent inspects representative environments using Bash and Python, then submits a module defining `describe_environment(env_root) -> str`. Seed workspaces are read-only. Each extractor measures the current environment at runtime and can be reused on environments of the same family with different contents and scales.

D.2 Fixed Rules and Privileged Knob Input

Hand-crafted. The Hand-crafted baseline uses a single generic extraction program shared across the evaluated environments, rather than environment-specific exploration and compilation. We use Opus 5 to generate this program from the benchmark's knob definitions, then keep the extraction rules fixed across environments. The design is informed by the environmental dimensions varied in the benchmark, while requiring one program to operate across heterogeneous environments without a supplied parser for each substrate or layout. Knowing that retrieval mode or tool availability matters does not itself specify how an unfamiliar environment exposes those properties: files, indices, and tool declarations can encode them differently. This baseline therefore tests a reusable generic rule set under that information constraint; the agent-compiled approach instead explores the environment to identify substrate-specific resources and interfaces and generate the corresponding extraction code. The program measures the current workspace at runtime; fixed rules do not mean fixed output values. Its generic measurements include file counts, directory depth, and file-size statistics. Knowledge of the knob definitions during code generation does not give the runtime extractor direct access to the ground-truth knob values, nor guarantee that it identifies every substrate-specific property. The BrowseComp-Plus example below illustrates this distinction.

Privileged knob metadata. This reference bypasses environment extraction and supplies the ground-truth knob configuration used to generate each environment as a compact text description. The query and candidate-model identity remain the other router inputs. For example, a 12-document environment with literal search can be represented as:

Privileged knob metadata (illustrative input)

```
data.size=12, tool=grep
```

The environment input contains knob settings rather than an LLM-generated explanation. This condition provides a privileged-information reference for assessing how closely automatically extracted descriptions approach the predictive performance obtained with direct access to the generation settings.

D.3 Environment Description Examples

BrowseComp-Plus.

Hand-crafted

```
15 files (63.8KB) across 25 directories, max depth 3, up to 12
    entries per directory. Two formats: .md (60%) and .txt (40%).
    Files are small: median 5.0KB, p90 9.7KB, largest 15.8KB.
```

Agent-compiled

```
The corpus contains 12 files (6 .md, 6 .txt) totaling about 62.8 KB (
    roughly 16K tokens). File sizes range from 185 B to 15.8 KB (
    median 5.3 KB, mean 5.2 KB).

No task-specific tools are exposed (n_visible_tools = 0): there is no
     semantic retrieval, embedding search, or query interface. The
    agent can only use literal grep, guessing search terms and
    reading candidate files manually without relevance ranking.
```

Optimized Agent

```
docs=12,64294 B; types=.md:6,.txt:6;
layout=sharded directory tree; shards_with_files=12;
doc_path_depth=3; index_entries=12;
retrieval=grep literal match; semantic retrieval=none.
```

The three descriptions show what each stage adds. The hand-crafted rules apply the same file-system statistics to every environment: they count all 15 files, including the index and other auxiliary files, and cannot identify which files form the searchable corpus or which retrieval interface the model will use. The environment-understanding agent instead inspects the workspace itself, including file contents, the index, and the exposed interfaces. Its extractor therefore separates the 12 corpus documents from supporting files and reports that only literal `grep` search is available, with no semantic retrieval or task-specific tools. These properties bear directly on how much search a model must perform, consistent with the gain from 65.67 to 68.53 Macro-AUC in Table 4. Routing-guided harness search replaces this style with a denser `field=value` format. The optimized description drops explanations and subjective ratings, and uses the saved space for additional structural measurements, including the directory layout, the number of populated shards, document path depth, and index size. Because the router encodes descriptions within a fixed input length, this format lets more measured properties reach the router, while the relation between these properties and model success or cost is left to be learned from data. Macro-AUC rises further to 70.22.

D.4 Routing-Guided Harness Search

We run ten rounds of prompt and harness revision, starting from the initial agent-compiled configuration. At each round, the proposer receives the complete history of previous revisions, their validation feedback, and training cases with routing results. It uses this history to propose a revised exploration and extraction prompt. The environment-understanding agent then compiles extractors under the revised harness, and the resulting descriptions are used to train and evaluate the router with fixed data partitions and training settings.

Validation and selection. Candidate extractors must execute successfully, measure the current workspace rather than hard-code seed-environment values, and return deterministic descriptions without solving the task. Router fitting uses the training partition; checkpoint selection and harness comparison use development data. The final test partition is withheld until the harness and router are frozen. The harness-selection objective is the area under the development-set cost–performance curve, using the routing sweep and normalized-cost aggregation in Appendix B.4. Success ROC-AUC and cost-prediction R^2 remain diagnostic metrics and are not combined into a separate selection score. After all ten rounds, we select the configuration with the best development curve area. Algorithm 1 summarizes the procedure.

Algorithm 1 Routing-guided environment-harness optimization (ten rounds).

Require: Initial harness H_0; proposer $\mathcal{P}$; fixed train/dev/test partitions
Ensure: Selected harness $H^\star$
Evaluate H_0 with train/dev to initialize its curve area and history $\mathcal{L}$
$\mathcal{H} \leftarrow \{H_0\}$
for $t = 1, \ldots, 10$ **do**
    $H_t \leftarrow \mathcal{P}(\mathcal{L}, \text{training cases and routing results})$
    Compile extractors with H_t and validate their execution and descriptions
    if validation fails **then**
        Append the revision and validation failures to $\mathcal{L}$
        continue
    end if
    Generate environment descriptions with the validated extractors
    Train router on train; select its checkpoint on dev
    $a_t \leftarrow$ area under the dev cost–performance curve
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{H_t\}$
    Append revision, validation feedback, and routing results to $\mathcal{L}$
end for
$H^\star \leftarrow$ harness in $\mathcal{H}$ with the largest dev curve area
Freeze $H^\star$ and the router before evaluating on test
return $H^\star$

D.5 Original Compiler Prompt

The following paraphrases the initial compiler prompt used for the Agent-compiled condition. The Optimized condition uses the selected prompt described in Appendix D.6.

> **Why this work exists.** We study **model routing**. The same task placed in different environments is not equally difficult: a small, inexpensive model succeeds in some environments, while only a large, expensive model succeeds in others. We need a router that chooses a model using only an environment description, **without seeing the outcome in advance**.
>
> **Your output is the source of that environment description.** Therefore, you explore not "what content is in this environment" but **how much difficulty this environment adds to the task** and which factors make the task harder or easier.
>
> You do not need to solve the task in the environment. You judge how difficult completing it would be, not what its answer is.
>
> **How to work: explore first, then write code. Step 1—Explore.** You may run Bash commands and Python code to inspect the environment. You decide what to inspect, but should keep asking where the difficulty comes from. You may see the actual command output and continue exploring until you understand the relevant properties of the environment.
>
> **Step 2—Write code.** After understanding the environment, write `extractor_rule.py`:
>
> ```
> def describe_environment(env_root) -> str
> ```
>
> `describe_environment` must read the **current contents** of `env_root` and return a passage of **natural language** describing how difficult the environment is and where the difficulty lies.
>
> Include the properties discovered during exploration that affect difficulty. You have inspected this environment yourself; you judge which properties affect whether a model can complete the task. The description must be grounded in actual measurements, without speculation.
>
> **The sole hard requirement: probe the environment, do not solve the task.** This applies both to exploration and to the code you write. You may list directories, count, read files, and measure sizes and lengths. You **may not** try to solve the task. Do not run the environment's retrieval or tool scripts to obtain an answer, search documents for the answer to a question, or try to identify which content contains the correct answer. Your task is to assess difficulty, not complete the task.

Two constraints. `describe_environment()` must **measure the environment itself at runtime**. Do not hard-code numbers observed during exploration. It must work on other environments of the same family with different content and scale. Two calls on the same environment must return the same result.

Output format. After exploration, return the **complete contents** of `extractor_rule.py`:

```
# extractor_rule.py
...full module...
```

The module must be self-contained, define `describe_environment(env_root)`, run as-is, and be submitted through `submit_extractor_rule`.

D.6 Optimized Compiler Prompt

You are an offline environment-understanding agent. Your task is to write reusable environment-extraction rules for a model router. The router combines the user query with the environment description to learn which candidate model to choose; you provide **observable environment facts** and do not predict difficulty or recommend a model for the router.

Step 1: Explore the environment. Use Bash and Python to inspect the provided seed environments. Choose useful measurements according to the environment type: numbers of documents or tools, file-size distributions, directory and index structure, retrieval interfaces, execution capabilities, and feedback visibility. Do not copy a single fixed template; document collections, tool catalogs, and code repositories may require different fields.

Step 2: Write the extractor. Submit the complete `extractor_rule.py` defining:

```
def describe_environment(env_root) -> str:
    ...
```

The function must read the current `env_root` on **every run** and measure the current environment, rather than hard-code the numbers of files, their sizes, or tool states from the seed environments. Repeated calls on the same environment must return the same output. The code must work for environments of the same family with different contents and scales.

Description length and content. Keep only measurable facts that help distinguish environments, and be as brief as possible. Prefer compact "field=value/status" expressions; remove introductions, task background, repeated layout names, and sentences explaining why a field matters. There is no target length to fill, and invariant or redundant information should not be added merely for detail.

Do not output subjective ratings such as "simple," "difficult," or "moderate difficulty." Do not write consequence claims such as "therefore it is harder to search" or model advice such as "a small model can complete this." Report "files=4312, total size=138 MB" rather than "the corpus is large." The router learns from observable facts; the extractor does not reach a conclusion for it.

Probing boundaries and output. You may list directories, count, read files, and measure sizes and lengths. Do not try to solve the task: do not run the environment's retrieval or tool scripts to obtain an answer, search for the correct answer to a query, or locate the document containing the answer.

After exploration, return the complete, self-contained code for `extractor_rule.py` and submit it through `submit_extractor_rule`. The environment description returned by the extractor should end with a measurement, without a concluding judgment.