

Agentic Routing: The Harness-Native Data Flywheel

TokenRhythm Technologies

<https://github.com/opensquilla/opensquilla>

Abstract

Large language model agents are increasingly executed not by a single model call, but by an execution harness that manages observation, context, control, action, state, and verification. At the same time, frontier and open models are becoming structurally specialized: a model that is strong at code editing, long-context recovery, tool use, mathematical reasoning, or low-latency response may not dominate on the other axes. This makes model selection inside an agent a core systems problem rather than a per-query serving trick. Existing routing methods mostly optimize single-turn cost-quality trade-offs and therefore miss the execution state, intermediate failures, and feedback loops that make agents different from chat completion. We propose HARNESS-NATIVE agentic routing, a step-level routing paradigm that selects either a single best-fit model for cost-effective execution or multiple complementary models for ensemble-style accuracy improvement, conditioned on the full harness state. The key insight is that every routing decision naturally produces a structured data record—consisting of the query, harness state, model choice or model set, execution trace, outcome, and cost—whose labels are supplied by the environment rather than by the router itself. These records form a *harness-native data flywheel*: execution traces train better routers and harness-native models, which improve cost-quality trade-offs and generate more traces under the same budget. We instantiate this idea in OPENSQUILLA with a four-layer router, an open LightGBM ranker, and trained router models for cost-saving single-model routing and high-accuracy multi-model routing. The report studies singleton and multi-model routing on agentic benchmarks including DRACO and PinchBench, and argues that agentic routing is not merely cost control, but a data engine for agent-native training.

1. Introduction

The recent evolution of AI agents can be summarized by a simple decomposition: $Agent = Base\ Models + Harness$ [11]. Base models [2, 28] provide latent capabilities such as language understanding, reasoning, code generation, tool-use priors, and multimodal perception, while the harness [7, 19, 29] provides the execution substrate that turns these capabilities into observable, controllable, and verifiable behavior. The two sides are complementary rather than hierarchical: a model without a harness remains a passive generator, and a harness without capable base models has no intelligence to orchestrate. Progress in agents is therefore an alternating process rather than a one-sided model-scaling story. Stronger base models make new behaviors possible; harness engineering converts those behaviors into reliable workflows; those workflows expose new bottlenecks such as context drift, brittle tool calls, poor recovery, or unverifiable

intermediate states; and these bottlenecks in turn define what the next generation of models should internalize. In this view, the transition from prompt engineering to harness engineering and agent-native training is a co-evolution between base models and the execution systems that make them useful.

At the same time, models are becoming increasingly specialized rather than interchangeable. Contemporary model families differ not only in average benchmark scores but also in their operational profiles: coding-oriented models may be more competitive on repository-scale editing, long-context models may be stronger at evidence recovery, reasoning models may be better for mathematics but slower and more expensive, lightweight models may be ideal for routine control decisions, and tool-use models may tolerate different action formats. These differences create a Pareto frontier across quality, latency, cost, context length, tool reliability, and safety instead of a single model that dominates every axis. The specialization is structural because many objectives compete with one another: fast and slow thinking favor different inference regimes, concise tool calls and deep reasoning pull output distributions in different directions, and a model’s apparent ability often changes with the harness in which it is embedded.

When model specialization is combined with the large adaptation space of the harness, routing is no longer the problem of simply choosing which model should answer a query. It becomes a joint optimization problem over a specialized model space and a specialized harness adaptation space. A routing decision can determine not only the model, but also how observations are formatted, how context is compressed, whether the control loop should use a cheap fast path or a stronger deliberative path, which action schema is exposed, how state is carried across steps, and what verification or recovery policy is applied after a failure. Model routing should be treated as a meta-decision across harness responsibilities rather than as a thin serving layer above fixed prompts.

We call this paradigm HARNESS-NATIVE agentic routing: step-level model selection during agent execution conditioned on the full harness state. At each decision point, the router can operate in two regimes. In the cost-effective regime, it selects a single best-fit model from the current model pool, and potentially a harness configuration, to achieve the best cost-quality trade-off for the current dialogue turn or execution step. In the high-accuracy regime, it selects a set of complementary models and aggregates their outputs through ensemble, verification, voting, debate, or fallback policies to improve task accuracy beyond any single cheap model. This formulation differs from query-level model routing because agent execution is multi-step, stateful, and recoverable: a cheap decision at one step may force costly correction later, while a strong model or a multi-model ensemble may be unnecessary for routine control but essential after verification detects drift. The router must therefore reason over execution trajectories, not only prompts, and it must treat cost, latency, and quality as coupled Pareto dimensions. Crucially, this definition also makes routing a data-producing operation. Every step-level decision can be logged as a structured record containing the query, the harness state, the model choice or model set, the execution trace, the outcome, and the realized cost. Unlike preference data or static benchmark labels, these records are produced by the agent’s own execution environment: the model choice is an action, while the outcome and cost are supplied by verification and runtime accounting as ground-truth labels. Routing decisions are therefore not only control decisions, but also the starting point of a harness-native data flywheel.

The resulting data flywheel closes the loop between routing, harness execution, and model training. Better routing increases the fraction of steps handled by cheaper capable models, which lowers cost under a fixed budget. Lower cost allows the harness to execute more tasks and collect more traces. More traces improve router models and create supervision for harness-native

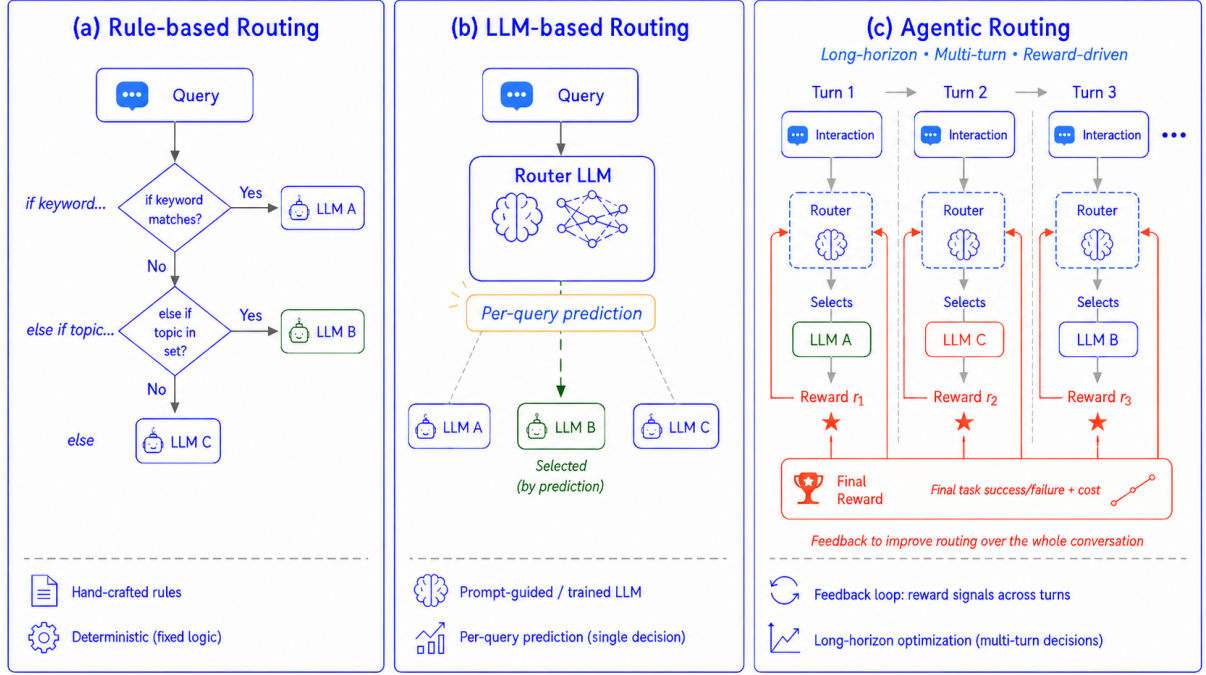


Figure 1 | **Architectural comparison of different model routing mechanisms.** (a) **Rule-based Routing:** A static approach that relies on hard-coded heuristic rules and keyword-based intent classification to assign user queries to specific models. (b) **LLM-based Routing:** A semantic-aware routing mechanism where a LLM dynamically serves as the router to make the optimal model selection. (c) **Agentic Routing (ours):** It evaluates the response history and goal alignment across multiple turns, leveraging an agentic reward signal to continuously optimize the router.

specialist models. These specialist models are then injected back into the model pool rather than replacing general foundation models, making the pool more diverse and increasing the value of routing. The positive feedback is not an assumption that the router is already good; it comes from environment-labeled traces, exploration coverage, and the label variance provided by a strong base-model pool.

We instantiate this idea in OPENSQUILLA for both singleton routing and multi-model ensemble routing. The open router follows a four-layer design: a token-complexity filter sends trivial requests to cheap models, a task-type classifier separates coding, reasoning, chat, and tool-use cases, a context-aware refiner incorporates harness state, and a LightGBM ranker selects the cheapest capable model from the remaining candidates. We further train router models for two deployment regimes: a cost-saving single-model mode that reduces cost at similar quality, and a high-accuracy multi-model mode that selects complementary models from a richer pool and combines their outputs to improve end-to-end quality beyond single-model routing. The open-source boundary is deliberate: the LightGBM router can be released, while high-accuracy router models and the accumulated trace data can be served through the OPENSQUILLA API.

Our evaluation is organized around the two operating modes of agentic routing. For singleton routing, we measure whether the router can preserve task quality while reducing cost on PinchBench and DRACO. For multi-model routing, we measure whether selecting several complementary models and aggregating their outputs improves accuracy or cost-quality frontier points on DRACO and PinchBench. The evaluation is also designed to ablate routing granularity

and harness-state features to test whether step-level harness-native information is necessary. Initial results show that the open LightGBM router can preserve aggregate task quality while cutting realized cost by roughly 90% on PinchBench and about 43% on DRACO relative to a fixed strong-model baseline, and that multi-model routing can push the DRACO quality frontier upward while often lowering cost. The full evaluation studies whether these gains persist in task-level agent execution and whether multi-model routing improves the quality frontier.

2. Related Work

Model routing. Model routing studies how to improve the cost-quality trade-off of LLM serving by selecting among models with different prices and capabilities. Prior systems use cascades [3], difficulty or meta-model estimation [8, 24], preference data [21], and model-specific performance or benchmark-derived outcomes [17, 27] to decide whether a request can be handled by a cheaper model or should be escalated to a stronger one. Recent benchmarks standardize this setting with multi-model outcome matrices [12, 16], while deployment-oriented work studies budget constraints, user-controlled cost-quality tolerance, and lightweight or training-free routing [6, 10, 23]. A related line of work ensembles multiple LLM outputs through ranking, fusion, or multi-agent aggregation to improve quality [13, 33]. These works establish that many inputs do not require the strongest model, but their routing problem is usually defined over a static prompt or a single input-output pair. The router observes the user request, and sometimes a predicted quality score, but not the execution state of an agent. It therefore does not directly model tool failures, context pressure, verifier feedback, recovery attempts, or the accumulated trajectory that determines whether a cheap model remains sufficient at the current step.

Agent harnesses and agent-native data. Recent agent systems show that model behavior is strongly shaped by the surrounding harness, including tool schemas, observation interfaces, memory, context management, verification loops, and recovery policies [11, 26, 32, 34, 37]. Work on tool use and agent tuning further suggests that agent data should encode actions, feedback, and task execution rather than only static instruction-response pairs [5, 25, 40]. Some recent routing methods have begun to move toward agentic settings, such as routing in multi-agent collaboration, tool calling, explainable agent workflows, or preference-aligned domain-action selection [1, 20, 30, 39]. Our work connects these threads by treating routing not only as a serving-time optimization, but also as a harness-native data interface. At each agent step, the routing decision reflects the current task, harness state, available skills or tools, verification signals, recovery history, and the cost and risk of alternative model choices. By recording which model or model set was sufficient under a given trajectory, and when escalation, verification, or recovery was needed, agentic routing produces structured supervision for future agent-native training.

3. What is Agentic Routing

3.1. Problem Definition and Objective

Agentic routing is step-level model or model-set selection during agent execution conditioned on the harness state. Let q be a user task, $\mathcal{M}$ be a pool of candidate models, and h_t be the harness state at decision step t . The state h_t includes the current observation, compressed and raw context, control-loop status, available actions, artifact state, tool history, recovery status, and

verification signals.

We do not define routing as a single cost-minimization problem. Prior model-routing and cascading systems have shown that heterogeneous model pools can improve the cost-quality trade-off of LLM serving, but they usually operate at the level of static prompts or queries [3, 12, 18, 21]. Recent agentic routing benchmarks and routers move this decision closer to execution trajectories [35, 43]. Our formulation takes the next step: a router may preserve quality at lower cost, compose cheaper specialist models to dominate a strong singleton baseline, or spend more budget at high-risk states to improve task success. We therefore define a single quality-cost frontier for harness-native agent execution.

We model routing as an operator g that, conditioned on the current harness state h_t , selects a set of models $S_t = g(\mathcal{M} \mid h_t)$ from the pool $\mathcal{M}$ for the next execution step and, when it selects more than one, a policy for combining them. The router optimizes the Pareto front between task loss and cost:

$$\min_g \left(\mathbb{E}_{q \sim \mathcal{D}, \tau \sim g} [\ell(\tau)], \mathbb{E}_{q \sim \mathcal{D}, \tau \sim g} [C(\tau)] \right), \quad S_t = g(\mathcal{M} \mid h_t), \quad 1 \leq |S_t| = k \leq K, \quad (1)$$

where the objective pair is minimized in the Pareto sense, $\ell(\tau) = 1 - R_{\text{task}}(\tau)$ is the task loss of the resulting trajectory, $C(\tau) = \sum_{t=1}^T \sum_{m \in S_t} c(m, h_t)$ is its realized cost, and $k = |S_t|$ is the number of models fielded at step t , capped by K . Setting $k = 1$ recovers singleton routing (Section 3.2), where g fields exactly one model; $k > 1$ gives multi-model routing (Section 3.3), where g fields several models together with a policy for combining them. Equivalently, sweeping the scalarized objective $\mathbb{E}_{\tau \sim g} [\ell(\tau) + \lambda C(\tau)]$ over $\lambda \geq 0$ traces the same frontier, and latency enters as an optional third axis when a deployment requires it.

Reading Equation 1, a router g is better when it lowers task loss, lowers cost, or both. Crucially, a multi-model action ($k > 1$) does not necessarily cost more than a singleton action, because S_t may combine several cheap or specialist models whose total cost is below that of a single expensive frontier model. Different operating points on the frontier answer different questions: a low-cost point asks how cheaply the harness can preserve task quality, while a high-accuracy point spends more budget at difficult, high-risk, or weakly recoverable states. We instantiate this frontier along the model and model-set dimension of the routing action: the action fields a model set S_t , together with an aggregation policy when $k > 1$. The wider harness adaptation that agentic routing can in principle control—context formatting, compression, control-loop path, action schema, and verification or recovery policy—is held fixed in this work, so the router optimizes model capability under a given harness; jointly adapting the harness together with the model choice is a natural extension we leave to future work.

The loss is task-level rather than purely step-level. Agent execution is recoverable: a locally weak step can be corrected later, while a locally cheap step can create downstream recovery cost. This view is consistent with agent work that treats reasoning, acting, tool use, memory, and feedback as interleaved execution rather than isolated text generation [25, 26, 31, 34, 36]. The supervision signal therefore should not assume that every step has an independent correctness label; the natural reward is terminal or verification-derived, and g must learn from trajectories rather than isolated prompts. The trajectory fields

$$(q, h_t, S_t, z_t, c_t, y), \quad (2)$$

define the schema of harness-native arena records: execution records annotated with outcome, cost, and verification signals. In this report, the open LightGBM router is the cold-start policy that starts this record stream, while later router-model iterations use the accumulated arena records to improve future routing decisions.

3.2. Singleton Routing Model

We first instantiate the frontier in the singleton setting. Setting $k = 1$ in Equation 1 restricts the routing operation to a single model,

$$S_t = g(\mathcal{M} \mid h_t) = \{m_t\}. \quad (3)$$

The router still optimizes the same task-level frontier, but its online action is now to field exactly one model from a heterogeneous model pool for the next harness step.

We call this view *capability matching*. The harness issues a state-conditioned capability demand; the model pool supplies heterogeneous capability profiles with different prices, latencies, context limits, tool-use reliability, and failure modes; and the singleton router selects the cheapest risk-adjusted model whose capability is sufficient for the current state. The model pool is thus an arena of specialists, and the router pairs each state’s demand with the model best suited to it under uncertainty. The router is not merely classifying the prompt. It is allocating model capability inside the execution economy of the harness, where a superficially cheap model can become expensive if its failure causes retries, tool repair, context reconstruction, or downstream verification recovery.

Let the singleton routing state be

$$x_t = (q, h_t), \quad (4)$$

where q is the original user task and h_t is the current harness state. The harness state includes not only the visible prompt, but also context pressure, artifact state, tool history, recovery status, recent routing decisions, and verification signals. The required capability is therefore not a fixed property of the user query. It is a trajectory-dependent demand induced by the current execution state.

Operationally, singleton routing is Equation 1 restricted to $k = 1$ and applied greedily at the current step. Fixing an operating point on the frontier through the trade-off weight λ of Equation 1, the router fields the single model whose selection is expected to give the best task-loss/cost trade-off for the remainder of the trajectory:

$$m_t = \arg \min_{m \in \mathcal{M}_t} \mathbb{E}_{\tau \sim g} [\ell(\tau) + \lambda C(\tau) \mid S_t = \{m\}, h_t]. \quad (5)$$

This is exactly the scalarized objective of Equation 1 evaluated for a single-model action, with no additional symbols. It is expressive because ℓ and C are trajectory-level: a superficially cheap model that later triggers retries, tool repair, context reconstruction, or verification recovery raises the downstream cost $C(\tau)$, while a model that under-serves the current state raises the task loss $\ell(\tau)$. The capability-matching intuition—matching a state-conditioned capability demand against the heterogeneous capability supply of the pool, and pricing residual risk for irreversible or weakly recoverable actions—is therefore a reading of how $\ell(\tau)$ and $C(\tau)$ respond to the choice of m , not a separate optimization problem. The main point is the same: a model is not cheap if its failure creates expensive recovery, and a model is not strong if its capability does not match the state-specific demand.

The required capability profile is multi-dimensional. Routine control, short tool-call formatting, low-risk context bookkeeping, or cacheable response generation may require only a lightweight model. Ambiguous planning, difficult code edits, long-context debugging, recovery after verification failure, safety-sensitive decisions, or final artifact generation may require a stronger model. The same user task can therefore require different models at different steps. Context pressure can make long-context reliability more important than raw reasoning score;

a recent tool error can make action-format robustness more important than benchmark performance; a failed verifier can raise the required capability for the next recovery step. This is the difference between harness-native singleton routing and ordinary query-level routing: the model choice is conditioned not only on what the user asked, but on where the harness currently is in the execution trajectory.

The open singleton router in OPENSQUILLA implements this matching view as a lightweight cold-start mechanism. It performs four passes: *order admission*, which sends clearly trivial and low-risk states to cheap fast paths; *demand construction*, which builds a coarse capability profile from the task and harness state; *risk pricing*, which adjusts that demand using context pressure, tool failures, verifier results, recovery status, action irreversibility, and downstream correction risk; and *capability matching*, which binds the priced demand to a concrete model or tier through the deployment registry. This decomposition is deliberately stronger than a model-ranking pipeline. It separates the question of what the next harness step requires from the question of which currently deployed model should satisfy it.

In the released open implementation, the first three passes are realized by cheap deterministic filters and lightweight classifiers, while the final matching pass is implemented by a LightGBM ranker over the remaining model choices [15]. LightGBM is therefore useful but not fundamental. It is the cold-start matching engine, not the market itself. The market consists of the harness state, the model pool, the deployment registry, the verifier, and runtime accounting. The harder learning problem is to infer, from harness-native trajectories, the state-conditioned capability demand, the risk of under-routing, and the expected recovery cost that Equation 5 prices only implicitly through $\ell(\tau)$ and $C(\tau)$.

This positioning is important for the open-source boundary. The LightGBM router is the first practical point on the singleton frontier: sample-efficient enough for cold start, fast enough for step-level routing, inspectable enough for open release, and simple enough to improve from logged execution. It should not be read as the final router model. It is the open seed that starts the arena-record stream.

The router model should instead be viewed as an *iterated matching policy*. Each batch of arena records updates the same latent objects that define singleton routing: demand construction, risk pricing, and capability matching. The state and selected model are logged as the action; final outcome, verifier events, recovery cost, realized cost, and latency are supplied by the environment. These records allow later router-model iterations to learn when the open policy under-routes a state, when it overpays for unnecessary capability, and when a model’s deployment profile has shifted because the model pool or harness has changed.

In the singleton setting, router-model iterations are evaluated in cost-saving mode: each iteration must field one model for the current step while preserving task quality and reducing unnecessary calls to expensive models. The learning target is outcome improvement rather than imitation of the previous router, matching the logged-feedback and off-policy learning view in contextual bandit evaluation [9]. A bad LightGBM decision is therefore useful feedback rather than poisoned supervision, because the environment supplies the label through verification, recovery, realized cost, and latency.

In this sense, singleton routing is not model selection in the narrow serving-layer sense. It is outcome-aware allocation of model capability under harness-state uncertainty. The LightGBM router is the open seed of this capability-matching flywheel, while the router model under iteration is the next-stage matching policy progressively learned from the records produced by that seed.

3.3. Multi-Model Ensemble Routing

Multi-model ensemble routing is the $k > 1$ instantiation of Equation 1: the router g fields a model *set* rather than a single model. Given the harness state h_t and the candidate pool $\mathcal{M}_t \subseteq \mathcal{M}$ that remains after the context-aware refiner prunes $\mathcal{M}$ under h_t , the routing operation returns a proposer set together with a combination policy,

$$g(\mathcal{M}_t \mid h_t) = (P_t, a_t), \quad P_t \subseteq \mathcal{M}_t, \quad a_t \in \mathcal{G}, \quad (6)$$

where the proposers in P_t generate candidate outputs and the aggregation policy a_t fuses the candidates into a single executable result; $\mathcal{G}$ contains voting, verifier-based selection, judge models, and fallback policies. When the aggregator is itself a model call, the selected set of Equation 1 is

$$S_t = P_t \cup \{m_t^{\text{agg}}\}, \quad k = |S_t|, \quad (7)$$

where m_t^{agg} is the aggregator model chosen as part of a_t ; for rule-based, voting, or verifier aggregators, $S_t = P_t$. The cost $C(\tau)$ of Equation 1 now accounts for every proposer call and the aggregation overhead, while latency is tracked separately as an overhead metric.

As in the singleton case, the same frontier is instantiated per step. Write $\ell(u \mid h_t) = \mathbb{E}_{\tau \sim g}[\ell(\tau) \mid u_t = u, h_t]$ and $C(u \mid h_t) = \mathbb{E}_{\tau \sim g}[C(\tau) \mid u_t = u, h_t]$ for the expected task loss and cost of a routing action u at state h_t . With the multi-model action $u_t = (P_t, a_t)$, the router minimizes the scalarized objective of Equation 1, augmented with a complementarity regularizer:

$$(P_t, a_t) = \arg \min_{P \subseteq \mathcal{M}_t, a \in \mathcal{G}} \left[\ell((P, a) \mid h_t) + \lambda C((P, a) \mid h_t) - \alpha V(P \mid h_t) \right]. \quad (8)$$

The first two terms are the scalarized frontier objective of Equation 1 at $k > 1$, applied greedily at the current step; the third term $-\alpha V(P \mid h_t)$ is a complementarity regularizer with weight α , defined below. The singleton rule of Equation 5 is the special case $u_t = (\{m\}, \emptyset)$, where V vanishes.

The cost C aggregates the proposer calls and the aggregation overhead, so a larger set is penalized exactly to the extent that it spends more. The complementarity term $V(P \mid h_t)$ rewards proposer sets whose members fail in *decorrelated* ways: it is estimated from the arena corpus through historical error correlation, state-conditioned disagreement, and verifier feedback rather than surface output difference, so it captures useful complementarity rather than mere diversity, and it is submodular and therefore admits near-optimal greedy selection. In principle the task loss ℓ already prefers such sets, since proposers that reduce correlated failures lower the expected loss while redundant models sharing training distributions and failure modes raise cost without lowering loss. But because ℓ is trajectory-level, off-policy, and noisy while the choice of P is combinatorial, the explicit term $-\alpha V$ acts as a regularizer that makes the subset search easier to optimize and generalize, steering exploration toward complementary sets instead of collapsing onto the logging policy’s observed favorites. It is an inductive bias on the surrogate, not a change to the frontier of Equation 1: α is kept small and annealed toward zero as the loss estimates sharpen, so the true cost–quality frontier is recovered at convergence and the router is never rewarded for diversity that does not reduce loss. This complementarity term is what separates multi-model routing from top- k selection. The size cap $|P| \leq K$ is inherited from Equation 1, and a latency budget enters, when a deployment requires it, as the optional third axis of Equation 1.

The two regimes are two readings of Equation 8 at different values of the cost weight λ , compared against the strong-singleton reference action $u_t^{\text{ref}} = (\{m_t^{\text{ref}}\}, \emptyset)$, where m_t^{ref} is a strong single-model reference for the current state. When λ is light, the minimizer favors the accuracy-seeking ensemble: it spends additional calls at states where a single-model error is costly and

hard to recover from—final answer generation, complex code modification, irreversible tool actions, safety-sensitive decisions, long-context synthesis, or low router confidence—driving the loss below the reference, $\ell(u_t | h_t) < \ell(u_t^{\text{ref}} | h_t)$, at the price of higher cost up to a per-step budget $C(u_t | h_t) \leq B_t$. A typical division of labor lets one model produce the primary proposal, another identify errors from a different perspective, and a third provide a cheap sanity check. When λ is heavy, the same minimizer favors the cost-saving ensemble: it composes medium- or low-cost models to replace the expensive reference, keeping the loss within a tolerance ϵ of the reference, $\ell(u_t | h_t) \leq \ell(u_t^{\text{ref}} | h_t) + \epsilon$, while spending strictly less, $C(u_t | h_t) < C(u_t^{\text{ref}} | h_t)$; when the loss is also strictly below the reference, the same composition is simultaneously accuracy-improving. Both conditions follow from Equation 8 rather than adding new objectives. The economics of multi-model routing is thus a search for a better cost–quality operating point among a strong singleton, a cheap singleton, and a composition of cheaper models.

We implement multi-model routing as a two-stage procedure. Stage one is task-profile-driven subset selection: the routing model first emits a task profile ϕ_t computed from (q, h_t) , summarizing signals such as task type, difficulty, required capabilities, risk level, verification availability, cost and latency tolerance, and routing uncertainty. This profile is a compact representation of the state-conditioned quantities in the objective above rather than a new environment state. Given ϕ_t and the candidate pool $\mathcal{M}_t$, the selector chooses a proposer subset P_t whose capabilities match the profile and whose members provide complementary evidence. Additional proposers are scored by their fit to the task profile, their own cost and latency profiles, expected marginal value, verification compatibility, uncertainty reduction, and complementarity with the models already selected. A valuable proposer is not merely strong in isolation; it supplies a capability or perspective required by the task profile, such as long-context evidence recovery, code synthesis, adversarial critique, or low-cost error detection. Selection is therefore profile-conditioned subset construction rather than top- k ranking, and the preference is regime-dependent: reliability gain at critical states for the accuracy-seeking ensemble, quality retention at lower total cost for the cost-saving ensemble.

Stage two is state-aware aggregation. Each proposer $m \in P_t$ independently generates a candidate r_m , and the aggregator produces the final result

$$\hat{r}_t = a_t(\{(m, r_m)\}_{m \in P_t}, h_t). \quad (9)$$

The policy a_t is also conditioned on the task profile and the strongest verification signal observable to the harness: executable signals such as unit tests, static analysis, or patch application in software tasks; schema, permission, and safety constraints in tool-use tasks; rubric-based judging, consistency, factuality, and citation checks in open-ended research tasks. a_t is not a post-processing module but part of the routing action: the router jointly decides which models propose and how their candidates are fused. In the cost-saving regime the aggregator must itself be cost-aware, since an expensive aggregator over cheap proposers can eliminate the proposer-side savings; a_t is then a lightweight judge, a rule-based or verifier-based selector, or a low-cost model aggregator. The accuracy-seeking regime may use a stronger aggregator when task risk, verification difficulty, and the resulting quality improvement justify its cost.

Each ensemble step is also a data-producing operation: the per-proposer candidates r_m provide outcomes from several models on the same (q, h_t) decision point, contributing to the counterfactual coverage that Section 3.5 builds through controlled exploration. The two-stage design preserves the singleton cost–quality trade-off: the system degenerates to singleton routing in routine states, selectively introduces complementary proposers where a single-model failure is expensive, and replaces a strong singleton with a composition of cheaper models in cost-sensitive states—expanding the quality frontier while reducing the cost of reaching a given

quality level.

3.4. Router-Model Iterations: Implementation and Evolution

The router is deployed not as a single model but as a sequence of policies $g^{(0)}, g^{(1)}, g^{(2)}, \dots$ that share the objective of Equation 1 but differ in representation and training data. The zeroth policy $g^{(0)}$ is the open LightGBM ranker of Section 3.2: hand-engineered features feed a shallow cost-quality ranker wrapped by the four cheap passes—order admission, demand construction, risk pricing, and capability matching [15]. It is deliberately not the final router; it is the cheapest policy that can run at every step and begin the arena-record stream of Section 3.5. Every later policy $g_\theta^{(r)}$ is a learned model trained on that stream, and this subsection describes how such a policy is built, trained, and rolled forward.

Architecture. A router-model iteration factorizes into three components. A *state encoder* maps the harness state h_t —task and instruction, compressed and raw context, artifact and tool history, verifier outputs, recovery status, and recent route history—to a state vector, replacing the hand-engineered features of $g^{(0)}$. A *supply encoder* maps each candidate model $m \in \mathcal{M}_t$ to a profile embedding built from its model card and observed outcome history: price, latency, context length, tool-use reliability, and task-family success. Because candidates are described by profiles rather than by fixed output classes, a newly released model can be scored from its profile without retraining a label space. A *prediction head* then estimates, for a candidate action $u = (S, a)$ at state h_t , the expected task loss $\hat{\ell}(u \mid h_t)$ and expected cost $\hat{C}(u \mid h_t)$, and the router acts by exactly the rule of Equations 5 and 8, selecting $\arg \min_u [\hat{\ell}(u \mid h_t) + \lambda \hat{C}(u \mid h_t)]$.

Training. Each iteration is trained to minimize the frontier objective of Equation 1 on the accumulated arena corpus $\mathcal{D}^{(r)}$ of Section 3.5:

$$\theta^{(r+1)} = \arg \min_{\theta} \widehat{\mathbb{E}}_{\mathcal{D}^{(r)}} [\ell(\tau) + \lambda C(\tau)]. \quad (10)$$

The expectation is an *off-policy* estimate: because every record was produced by an earlier policy $g^{(r)}$ rather than by uniform sampling over actions, the estimator reweights logged outcomes with an inverse-propensity or doubly-robust correction, so the update targets outcome improvement rather than imitation of the logging policy [9]. No separate reward, recovery, or risk terms are needed: recovery cost and under-routing enter through the trajectory-level $C(\tau)$ and $\ell(\tau)$ exactly as in Sections 3.2 and 3.3, and λ selects the operating point on the frontier.

Coverage. The estimator is only as good as the corpus’s coverage. A policy that always exploits reveals the outcome of the action it took but not of the actions it skipped. Router-model iterations therefore depend on the controlled exploration of Section 3.5—uncertainty-triggered escalation, and occasional alternative-model or oracle replay on a sampled subset of states—to obtain the counterfactuals that let $g^{(r+1)}$ learn where $g^{(r)}$ under-routed a state or overpaid for capability.

Evolution. The policies form a ladder, each generation promoted only if it moves the frontier of Equation 1 against the previous one. The released seed $g^{(0)}$ is the LightGBM ranker over hand-engineered features. The first learned generation $g^{(1)}$ replaces those features with a trained state encoder and prediction head, fitted offline on the first corpus and still gated behind the

cheap admission filters of $g^{(0)}$. The next generation $g^{(2)}$ adds supply-encoder profiles for new-model generalization and exploration-filled counterfactuals, and can be distilled into a small fast router to stay within the step-level latency budget. Subsequent generations $g^{(r)}$ are continual updates as the corpus grows and the model pool changes. A candidate generation is shipped only when it lowers realized cost at equal task reward or raises task reward under a fixed budget; otherwise the deployment rolls back to the previous policy. Agreement with $g^{(r-1)}$ is never the target—movement of the quality–cost frontier is.

Deployment. Because the router runs at every step, its own cost and latency are part of the trade-off. The deployment stays two-tier: the cheap admission filters and the $g^{(0)}$ -style gate resolve clearly trivial or clearly hard states, and the heavier learned router is invoked only on the uncertain, high-value, or weakly recoverable states where a better matching decision is worth its overhead. This keeps the router on the useful side of its own cost–quality trade-off.

This staged path also fixes the open-source boundary. The $g^{(0)}$ seed can be released because it is inspectable, locally trainable, and useful for cold start; later generations depend on accumulated arena records, changing model-supply profiles, and continuous frontier evaluation, so they are an evolving OPENSQUILLA policy path rather than a finalized artifact. The claim is not that some $g^{(r)}$ has converged, but that harness-native routing produces exactly the arena-record stream required to keep moving the frontier.

3.5. Routing Data as a Model-Arena Corpus

The data side of agentic routing is not an ordinary query log. A query log records what was asked and what a model answered. A harness-native routing record instead records a selected action and its later scoring by the execution environment. This distinction is central to the data flywheel: the router’s decision is an action, while the outcome, verification result, recovery path, realized cost, and latency are labels supplied by the harness and runtime. The data is therefore a model-arena corpus rather than a transcript to be imitated.

For each decision point t , we store a typed arena record

$$\mathcal{R}_t = (q, h_t, \mathcal{M}_t, \hat{\kappa}_t, u_t, s_t, \text{trace}_{t:T}, v_{t:T}, y, C_{t:T}, D_{t:T}, \omega_t), \quad (11)$$

where q is the original task, h_t is the harness state, $\mathcal{M}_t$ is the candidate model or tier slate available at the decision point, $\hat{\kappa}_t$ is the router’s estimated capability demand, and

$$u_t = (S_t, a_t) \quad (12)$$

is the routing action. The term s_t stores the router-side scores, confidence, ranking margins, and exploration flags. The subsequent execution trace is $\text{trace}_{t:T}$, the verification and runtime events are $v_{t:T}$, the final or intermediate outcome is y , the realized cost and duration are $C_{t:T}$ and $D_{t:T}$, and ω_t records provenance such as production exploitation, uncertainty-triggered exploration, offline replay, or oracle-style evaluation.

This schema deliberately separates prediction, action, and outcome. The fields $(h_t, \mathcal{M}_t, \hat{\kappa}_t, s_t)$ describe what the router believed before acting, the field u_t records what it actually fielded, and the fields $(\text{trace}_{t:T}, v_{t:T}, y, C_{t:T}, D_{t:T})$ record how the environment scored that action. Because the action and its outcome are stored separately, a weak model choice is never the imitation target—the outcome is; we develop the consequences of this separation for self-poisoning in Section 5.3.

The record is also the interface between the router and the rest of the harness. Routing-decision signals record the selected model or model set, optional harness configuration, candidate ranking, score margin, confidence, and aggregation policy. Tool-call signals record action schema, tool success, execution error, retry count, and recovery path. Context-management signals record context length, compression trigger, retained artifacts, dropped evidence, and post-compression verification. Verification and runtime signals record test results, judge outcomes, constraint violations, task-level success, realized cost, and latency overhead. These signals make the data harness-native: they describe not only what the model saw, but also how the execution system shaped, constrained, and evaluated the model’s behavior.

These typed records accumulate into the arena corpus that drives the harness-native data flywheel of Section 5. There we develop how the accumulated corpus is used to train routers, harness-native specialists, and harness policies (Section 5.2); how controlled exploration—distinguished in the record by the provenance field ω_t —maintains the coverage needed for off-policy learning without turning production into uncontrolled experimentation, and why the loop does not self-poison (Section 5.3) [9]; and why the accumulated corpus is hard to crawl, buy, or reconstruct (Section 5.4). Here we fix only the data object: a typed, environment-labeled record of what was fielded and how the harness scored it.

4. Experiments

4.1. Experimental Setup

The experiments are organized around the two operating regimes of agentic routing. The first regime is singleton routing, where the router selects one model for the current turn or execution step and is evaluated by cost-quality trade-off. The second regime is multi-model ensemble routing, where the router selects several complementary models and an aggregator produces the final answer; this regime is evaluated by accuracy improvement under controlled additional cost. This separation is important because the two regimes answer different deployment questions: whether routing can make existing quality cheaper, and whether routing can make difficult tasks more accurate.

We evaluate on two main benchmarks. DRACO is a cross-domain deep-research benchmark that evaluates factual accuracy, completeness, objectivity, presentation quality, and citation quality across complex research tasks [42]. PinchBench (version 1.2.1) evaluates OpenClaw-style agents on real-world tasks and reports success rate, execution time, cost per task, and value score, making it suitable for cost-quality routing analysis. We also use public agent benchmarks such as SWE-bench and τ -bench when evaluating software-engineering and tool-agent-user interaction settings [14, 38].

We report separate metrics for the two regimes. For singleton routing, the main metrics are task success, cost ratio, quality retention, latency, value score, and Pareto coverage. For multi-model routing, the main metrics are task success or Pass@1, accuracy lift over the best singleton router, ensemble cost multiplier, aggregation success rate, and oracle gap to the best observed model set. Across both regimes, we report router accuracy when an oracle or best observed decision is available, and we report quality-cost curves rather than a single operating point.

Table 1 | Singleton-routing results on PinchBench.

Method	Agent	Score	Cost (\$)	Tokens (K)
Opus 4.8	OpenClaw	93.35	0.2224	187.7
Opus 4.8	OPENSQUILLA	94.33	0.1649	97.7
Agentic routing (ours)	OPENSQUILLA	93.14	0.0204	51.3

Notes: Score is the mean across tasks and costs are rounded to cents; total tokens are input plus output tokens in thousands.

Table 2 | Singleton-routing results on DRACO.

Method	Agent	Score	Cost (\$)	Tokens (K)
Opus 4.8	OpenClaw	52.13	1.1420	54.2
Opus 4.8	OPENSQUILLA	52.36	0.6559	103.5
Agentic routing (ours)	OPENSQUILLA	52.33	0.3729	108.6

Notes: Score is the mean across tasks and costs are rounded to cents; total tokens are input plus output tokens in thousands; the routing threshold is set to 0.95.

4.2. Singleton Routing: Preliminary Cost-Quality Frontier Evidence

This section evaluates the $K = 1$ operating region of Equation 1. The current singleton evidence is aggregate rather than task-level: each row in Tables 1 and 2 is a full benchmark run, not an individual routing decision or per-task confidence interval. We therefore use these results as preliminary cost-quality frontier evidence. The question is whether singleton capability matching can preserve aggregate task score while reducing realized billed cost against a fixed strong-model baseline. In these runs the router (Agentic routing) selects among Opus 4.8, GLM 5.2, and DS4 Flash on PinchBench, and among Opus 4.8, GLM 5.2, and DS4 Pro on DRACO; the strong-model baselines are Opus 4.8 run in the OPENSQUILLA and OpenClaw harnesses.

The comparison is intentionally simple. OPENSQUILLA uses singleton routing over the available model pool, while the OpenClaw baseline uses Claude Opus 4.8 as a fixed strong-model policy. This comparison follows the harness-centric evaluation view that the same base model can behave differently under different agent adapters, tool interfaces, and benchmark protocols [34, 41]. The harness, benchmark split, and scoring protocol are held fixed within each aggregate setting. We report the benchmark score, total tokens, and realized billed cost.

4.2.1. PinchBench

On PinchBench, the OPENSQUILLA runs expose two singleton operating points relative to the fixed OpenClaw baseline, which scores 93.35 at \$0.2224 per task. The routed configuration (Agentic routing) nearly matches this baseline while cutting cost by an order of magnitude: it scores 93.14, a gap of only 0.21 points and a quality retention of 99.77%, at \$0.0204 per task—a cost ratio of 9.17% and roughly 10.9× lower cost. The Opus4.8-backed OPENSQUILLA point instead shifts toward quality: it scores 94.33, about 0.98 points above the OpenClaw baseline, while still reducing per-task cost to \$0.1649, a cost ratio of 74.1%. Both points should be read as aggregate operating-point comparisons rather than as isolated causal attributions to the router.

4.2.2. DRACO

The DRACO aggregate shows a different savings mechanism. DRACO is particularly useful here because it scores complex deep-research outputs along factual accuracy, completeness, objectivity, presentation quality, and citation quality, rather than only exact-match task completion [42]. Agentic routing retains 99.94% of the strong-model baseline score, moving from 52.36 to 52.33, while reducing per-task cost from \$0.6559 to \$0.3729. This corresponds to a cost ratio of 56.85% and a cost reduction of 43.15%. Unlike PinchBench, the routed run uses slightly more input tokens than the fixed baseline. The cost reduction therefore cannot be explained only by prompt shortening or lower token volume. It indicates that singleton routing also changes the price composition of model calls: cheaper sufficient models can be fielded for parts of the execution while preserving aggregate quality.

4.2.3. Summary

Taken together, these aggregate frontier points support the feasibility of the singleton capability-matching view in Section 3.2. The router does not globally weaken the agent. Instead, it attempts to field the cheapest sufficient model for each execution state. In PinchBench, this produces both token-side and model-price-side savings. In DRACO, the score is preserved and cost decreases even when input-token volume does not decrease, which is consistent with cost savings coming from state-conditioned model allocation rather than only from context reduction.

These results should be interpreted carefully. Tables 1 and 2 do not yet isolate the causal contribution of step-level granularity, individual harness-state features, risk-pricing terms, or router-model iterations. They also do not provide task-level confidence intervals or category-level breakdowns. We therefore treat these tables as preliminary aggregate frontier evidence, not as a complete ablation study. The next evaluation step is to decompose these aggregate gains into task-level outcomes, route-level decisions, and arena records, so that future versions can test whether the observed savings come from routing granularity, harness-state signals, capability matching, or model-price composition.

4.3. Multi-Model Ensemble Routing: Accuracy and Cost-Quality Evaluation

The multi-model-routing experiment tests whether selecting several complementary models and aggregating their outputs improves the quality-cost frontier of agent execution. We compare fixed singleton baselines with evaluated multi-model routing groups that vary proposer composition, sampling, and aggregation or prefiltering. The primary quantities are benchmark-native quality, average monetary cost per task, token or tool usage, and p50/p95 latency. Because the runs span different benchmarks, harnesses, and provider settings, the tables below should be read as descriptive operating points rather than as a single controlled-cost ablation or a statistical significance study. In each table, **Ours** denotes the highest-quality multi-model configuration among the evaluated routing groups for that specific setting.

DRACO is the primary benchmark for multi-model routing because deep research tasks often benefit from complementary model strengths: one model may retrieve better evidence, another may synthesize more coherent analysis, and another may produce more reliable citations. The aggregator can score candidate reports using factuality checks, citation coverage, completeness criteria, and consistency across model outputs. PinchBench provides short-task cost-quality evidence, and the Hermes mixture-of-agents configuration serves as an additional multi-model baseline in the DRACO comparison.

Table 3 | DRACO results comparing singleton baselines with our recommended multi-model ensemble routing configuration.

Method	Quality	Efficiency			
	Score	Cost (\$)	Tokens (K)	p50 (s)	p95 (s)
Fable 5	59.80	1.2122	93.7	187.7	343.5
Opus 4.8	52.36	0.6559	103.5	165.3	270.6
GPT-5.5	50.22	0.4505	81.9	278.1	457.4
GLM 5.2	48.28	0.1214	116.8	207.0	351.5
DeepSeek V4 Pro	50.32	0.1320	83.4	213.1	346.0
K2.7-code	45.48	0.0676	86.3	186.2	491.7
Qwen 3.7	49.34	0.0432	99.5	240.9	368.9
Gemini 3 Flash	40.79	0.0117	9.5	18.7	168.4
Ours	60.82	0.3766	579.7	535.5	3097.0

Notes: Costs are averaged per task; token counts are shown in thousands and latencies in seconds. Values are rounded for display. The Fable 5 baseline refused six DRACO tasks, so its reported values are computed over the 94 executed tasks.

Unless otherwise noted, costs are averages per task, token counts are shown in thousands, and latencies are shown in seconds. Boldface highlights the primary comparison value in each table, usually the selected configuration’s quality and, when applicable, its cost advantage. We use the benchmark’s native score scale in each table; for example, PinchBench scores are normalized to $[0, 1]$, and DRACO and Hermes report their native quality scores.

4.3.1. DRACO

For DRACO with the default DuckDuckGo web-search provider, Table 3 reports all singleton baselines by model name and the best-quality multi-model configuration among our evaluated routing groups. Our configuration uses DeepSeek V4, GLM 5.2, Gemini 3 Flash, and Qwen 3.7 as proposers, GLM 5.2 as the aggregator, and additional stochastic samples for the cheaper Gemini and Qwen proposers.

Among the singleton baselines, fable5 is the strongest quality point, with an average score of 59.80 and an average cost of \$1.2122 over the 94 tasks it executed. Our multi-model ensemble reaches 60.82 on the full 100-task DRACO run while reducing average cost to \$0.3766, improving quality by 1.02 points and reducing cost by 68.9% relative to fable5. This result illustrates the cost-efficient substitution regime predicted by the frontier formulation: complementary cheaper proposers plus an aggregator can outperform the strongest evaluated singleton baseline on both quality and monetary cost. The trade-off is latency, since the ensemble makes more model calls; the point should therefore be read as a quality–cost improvement rather than a latency-optimized deployment.

Table 4 directly compares our selected DRACO ensemble with the Hermes MoA configuration. Our configuration improves the native score from 59.55 to 60.82 while reducing actual average cost from \$0.4460 to \$0.3766 per task, a quality–cost advantage over the mixture-of-agents baseline.

We also evaluate the same DRACO setting with Brave as the web-search provider, replacing the default DuckDuckGo provider. Table 5 reports the singleton baselines and the highest-quality multi-model configuration in this provider setting.

Table 4 | DRACO comparison of routing and baseline configurations.

Method	Score	Cost (\$)
Hermes MoA	59.55	0.4460
Sakana/fugu-ultra	46.70	0.6344
Ours	60.82	0.3766

Notes: Costs use actual average cost per task for comparability. Hermes MoA denotes the evaluated Hermes mixture-of-agents configuration, with 100/100 coverage, 100 healthy tasks, no short outputs, and no judge errors. The sakana/fugu-ultra row uses DuckDuckGo search and 91 successful rows. Token usage is omitted because it is not reported for Hermes MoA.

Table 5 | DRACO results with Brave web search comparing singleton baselines with our recommended multi-model ensemble routing configuration.

Method	Quality	Efficiency			
	Score	Cost (\$)	Tokens (K)	p50 (s)	p95 (s)
Opus 4.8	59.11	1.6177	257.7	233.5	418.2
GPT-5.5	53.28	0.8407	189.4	213.5	587.6
Ours	64.09	0.1218	500.1	656.8	2096.0

Notes: Costs are averaged per task; token counts are computed as Avg Visible plus Avg Reason and shown in thousands; latencies are in seconds. All rows use Brave as the web-search provider.

With Brave search, the best multi-model configuration improves DRACO quality by 4.98 points over Opus 4.8 and by 10.81 points over GPT-5.5, while reducing average cost by 92.5% and 85.5%, respectively. This is a stronger cost-efficient substitution point than the default-provider DRACO run: the ensemble benefits from complementary candidate generation while the aggregate monetary cost remains far below both singleton baselines. The trade-off remains latency, with higher p50 and p95 durations due to additional model calls.

4.3.2. PinchBench

For PinchBench, Table 6 reports the same compact metrics from the task-level evaluation workbook. We use the task rows rather than the workbook’s overall summary row, and report G16 as the highest-quality multi-model configuration among the evaluated routing groups.

On PinchBench, the best multi-model configuration essentially matches the Opus 4.8 quality point: the gap is only 0.0003 score units, while average cost falls from \$0.1649 to \$0.1349, an 18.2% reduction. Relative to GPT-5.5, the same configuration improves quality by 0.0058 score units but uses higher cost and latency. This suggests that PinchBench is already close to saturation for the strongest singleton baseline, and that the main benefit of the evaluated multi-model configuration is matching that quality at lower monetary cost rather than creating a large quality lift.

4.4. Cost-Score Frontier Summary

Figure 2 aggregates the singleton (Section 4.2) and multi-model ensemble (Section 4.3) results into a single cost-score view for each benchmark, excluding the OpenClaw harness baseline and any baseline cheaper than our singleton router. Each point is a method, labeled by name; gray

Table 6 | PinchBench results comparing singleton baselines with our recommended multi-model ensemble routing configuration.

Method	Quality	Efficiency			
	Score	Cost (\$)	Tokens (K)	p50 (s)	p95 (s)
Opus 4.8	0.9433	0.1649	97.7	23.1	150.1
GPT-5.5	0.9373	0.0963	56.7	29.0	76.1
Ours	0.9431	0.1349	272.4	96.0	223.7

Notes: Costs are averaged per task; token counts are input plus output tokens in thousands, and latencies are in seconds. PinchBench scores are normalized to [0, 1].

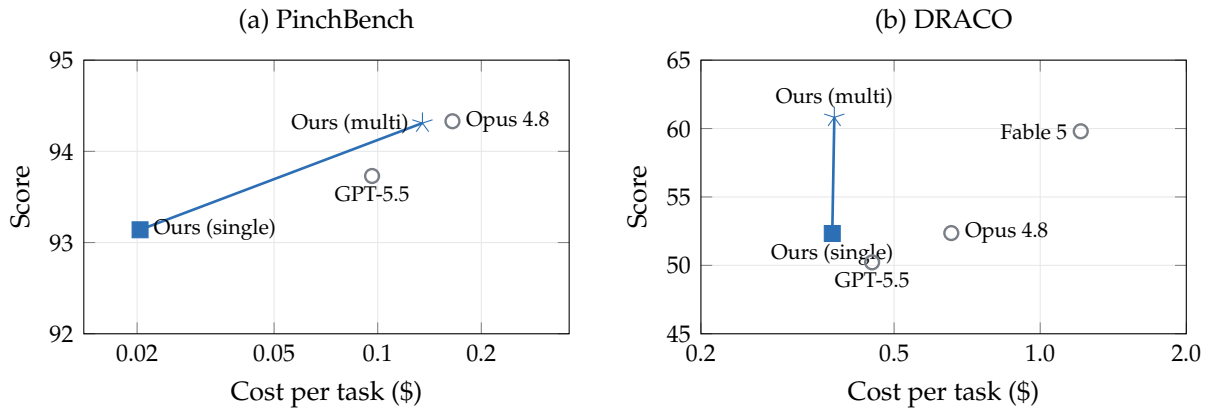


Figure 2 | Aggregate cost–score view on PinchBench and DRACO, combining the singleton and multi-model ensemble results. Gray circles are fixed singleton baselines; the blue square and star, joined by a line, are our singleton router and multi-model ensemble. Cost is shown on a log scale.

circles are fixed singleton baselines, and the blue square and star—joined by a line—are our singleton router and multi-model ensemble. On DRACO our two points dominate the strongest baselines: the multi-model ensemble exceeds both Opus 4.8 and Fable 5 in score at far lower cost, and the singleton router matches the Opus 4.8 quality point at a fraction of its cost. On PinchBench, where the strong baseline is already near saturation, the singleton router is the cheapest operating point and the ensemble nearly reaches the Opus 4.8 quality point at lower cost.

5. The Harness-Native Data Flywheel

5.1. Closed Loop

The harness-native data flywheel describes how agent execution can generate the data needed to improve future agents. The loop is:

Base Models → Harness Execution → Routing Data → Harness-Native Models → Model Pool.

Base models provide broad capabilities. The harness executes tasks and records routing decisions, traces, outcomes, and costs. The resulting data trains better routers and specialist models. These specialist models are added back into the model pool, where routing decides when they

should replace or complement general foundation models. The important point is that the flywheel does not eliminate foundation models; it makes the model pool more heterogeneous and therefore makes routing more valuable.

The flywheel is budget-amplifying. Better routing increases the fraction of steps handled by cheap capable models. Lower average cost allows more tasks to be executed under the same budget. More tasks produce more trace data. More trace data improves routers and specialists, which further lowers cost or improves quality. The expected growth is not unbounded exponential acceleration. A realistic curve is S-shaped: early improvements can compound quickly once the cold-start router is good enough, but the system eventually approaches a steady state as easy savings are exhausted and remaining tasks require strong models.

More precisely, the loop is an iteration on the frontier of Equation 1. At iteration r , the routing policy $g^{(r)}$ executes inside the harness and appends its arena records to the corpus, $\mathcal{D}^{(r+1)} = \mathcal{D}^{(r)} \cup \mathcal{R}^{(r)}$; these records then update the router, the post-trained models, and the harness-native specialists. Progress is measured not by agreement with the previous policy but by *frontier movement*: lower realized cost at comparable task reward, or higher task reward under a controlled budget.

5.2. How to use the Accumulated Corpus

The value of the flywheel is not any single record but the accumulated corpus, which becomes a reusable training substrate for the whole system. As it grows, the same records are put to several uses. They train the router iterations of Section 3.4, improving demand construction, risk pricing, and capability matching. They expose repeated execution states—code repair, long-context recovery, tool-call formatting, or verification-aware editing—where a cheaper specialist could replace or complement a strong general model, yielding distillation and specialist-training sets. They supervise harness policies beyond model choice, such as when to compress context, when to verify, when to retry, and when to escalate. And they tell the operator what to build next: whether the next improvement should be a routing change, a distilled specialist, a verifier update, or a base-model post-training update.

The corpus also weakly supervises the models being routed. The routed action is weak evidence, not a gold label: the fact that the old policy selected a model does not make it optimal, and a skipped model is not necessarily wrong. Supervision comes from the environment’s verdict—whether the output passed verification, whether it was later repaired, whether the task succeeded, and at what cost. A signal compiler Γ maps each record into several post-training views,

$$\Gamma(\mathcal{R}_t) \rightarrow \{\mathcal{D}_{\text{SFT}}, \mathcal{D}_{\text{pref}}, \mathcal{D}_{\text{reward}}, \mathcal{D}_{\text{distill}}, \mathcal{D}_{\text{curr}}, \mathcal{D}_{\text{safety}}\}, \quad (13)$$

so that a cheap-model failure that a strong model fixes becomes an SFT or distillation target, and a verified answer beating a failed one becomes a preference pair. The reward must be decomposed, because cost used directly as a language-model preference teaches brevity rather than genuine efficiency: quality and safety train the model, while cost and latency train the router,

$$R_{\text{model}}(x, y) = Q(x, y) - \lambda_S S(x, y) - \lambda_E E(x, y), \quad (14)$$

$$R_{\text{route}}(x, y, m) = Q(x, y) - \lambda_C C(m, y) - \lambda_D D(m, y) - \lambda_{\text{rec}} C_{\text{rec}}(x, y, m). \quad (15)$$

Across all of these uses the pattern is the same: routing first reveals a capability gap, and the accumulated corpus then indicates how to close it. This is what makes the corpus, rather than any single router, the durable asset of the flywheel.

Table 7 | Harness-native routing data differs from ordinary web or preference data because it is produced by execution and labeled by the environment.

Dimension	Ordinary interaction data	Harness-native arena corpus
Reward signal	Often implicit, noisy, or preference-based.	Task success, verification result, recovery path, and exact cost.
Counterfactual value	Usually weak because unchosen model outputs are unavailable.	Strengthened by shadow evaluation, replay, and alternative-model audits.
Crawlability	Partly visible from public outputs.	Hidden inside runtime traces and harness state.
Structure	Usually weakly structured text or interaction logs.	Typed multi-field record with execution trace and state features.
Open-source implication	Code release may reveal much of the pipeline.	Code can be open while the arena corpus and high-accuracy router models remain proprietary.

5.3. Why the Flywheel Does Not Self-Poison

A natural concern is that a router trained on data produced by earlier routers may simply imitate their mistakes. This concern applies to behavior cloning of the router’s own decisions, but it does not describe the harness-native data schema. In our schema, the model choice is an action, while success, failure, recovery cost, and latency are labels supplied by the environment. A bad router that selects a weak model for a difficult state generates a negative example: this state-action pair led to failure or high recovery cost. The data is therefore logged feedback for off-policy learning, not a transcript to be copied blindly.

The real failure modes are coverage, selection bias, label variance, credit assignment, and verifier noise. Coverage fails when the router never explores alternative models for a state; selection bias arises because logged records come from an old policy, so observed actions are not an unbiased sample of all routes; label variance fails when all early attempts either succeed or fail uniformly, leaving no signal; credit assignment fails when terminal success cannot be attributed to decisions inside a long trajectory; and verifier noise arises when the outcome label itself is unreliable. These are handled with matching mitigations: uncertainty-triggered exploration, replay, and oracle traces for coverage; logged route probabilities and propensity-aware or doubly-robust estimation for selection bias; a strong and diverse base-model pool for label variance; stored traces, intermediate verifier signals, and cost deltas for credit assignment; and multiple combined verification signals with audits of high-impact buckets for verifier noise.

5.4. Why the Data Is Hard to Replace

Harness-native routing data is difficult to crawl or purchase because it is produced inside execution. Public text can show a final answer, and benchmark leaderboards can show aggregate scores, but neither contains the sequence of harness states, candidate model slates, route probabilities, rejected alternatives, tool errors, verification events, context-compression decisions, recovery costs, and exploration provenance that led to the outcome. This makes the data valuable even when parts of the router implementation are open-source. Table 7 contrasts this harness-native arena corpus with ordinary web or preference data along the dimensions that make it hard to replace.

5.5. End State: A Three-Layer Model Pool

The flywheel suggests a three-layer model pool. The first layer contains general foundation models that remain necessary for long-tail tasks, ambiguous user intent, and high-stakes reasoning. The second layer contains harness-native specialist models trained from execution traces for common task families, tool schemas, verification policies, or context-management regimes. The third layer contains ultra-cheap fast paths such as small models, classifiers, retrieval shortcuts, or cache hits for trivial decisions. Agentic routing is the policy that composes these layers into one execution system.

This end state changes the role of the harness. The harness is not merely an adapter that wraps a fixed model; it becomes the environment that defines tasks, supplies feedback, and cultivates specialist models. Model providers can continue to produce stronger general capabilities, while harness operators produce the execution data that determines how those capabilities should be deployed and distilled into specialists. The more diverse the model pool becomes, the more valuable routing becomes; the more mature the harness becomes, the better the training signal for future models. The more fine-grained the supply grows, the more valuable capability matching becomes, and the more precisely the corpus can indicate what to train next.

Several assumptions bound these claims. This report specifies the flywheel’s mechanism and gives preliminary evidence; it does not claim that the loop has converged. Its value is strongest when model specialization persists, though even under convergence the execution traces retain independent worth for verification, recovery, and post-training. Routing must also save more than its own latency and cost overhead, which is why cheap filters and LightGBM ranking precede heavier router models, and why step-level rather than token-level routing is the practical target for current harnesses. Finally, because outcome labels can be noisy and an arena corpus contains tool traces, artifacts, and user state, it must store provenance and multiple verification signals and be treated as governed operational data rather than as ordinary public text.

6. Conclusions and Future Work

This report argues that agentic routing is a first-class mechanism for agent systems, because both sides of the agent equation—models and harnesses—are becoming specialized. The core idea is to route at execution steps using the full harness state and to treat each routing decision as a data-producing operation. This reframes cost control as a data problem: routing decisions create environment-labeled records, those records train better routers and harness-native models, and the models return to the pool to improve future execution. Our preliminary results support the feasibility of this direction. In the singleton regime, the open cold-start router preserves aggregate task quality while cutting realized cost by roughly 90% on PinchBench and about 43% on DRACO relative to a fixed strong-model baseline. In the multi-model regime, composing complementary proposers moves the DRACO quality frontier upward while often lowering monetary cost, illustrating the cost-efficient substitution and high-accuracy escalation regimes predicted by the frontier formulation. These are aggregate operating-point comparisons rather than controlled causal ablations: isolating the contribution of step-level granularity, harness-state features, and trained router-model iterations is left to further evaluation.

Several future directions are important. First, edge-side routing introduces a latency paradox: local routing can protect privacy, but weak local hardware may make routing slower than a strong cloud model. Second, model switching interacts with KV-cache reuse; switching models can destroy cache locality and erase routing gains unless cache compatibility is considered.

Third, rapid model releases require automatic model profiling and benchmark refresh so routers can adapt to new candidates. Fourth, cross-harness transfer remains open: a router trained in OPENSQUILLA may or may not transfer to other agent harnesses without losing state semantics. Finally, the granularity frontier may move from step-level to token-level routing, especially when combined with speculative decoding and multi-model generation.

The broader conclusion is that harnesses do more than execute models. They define the environment in which model capabilities are observed, evaluated, and transformed into future training data. Agentic routing is the control layer that exposes this data stream. If agent-native training is the next stage of agent development, then harness-native routing is one concrete mechanism for producing the data that makes it possible.

References

- [1] S. Agarwal, P. Namyar, A. Wolman, R. Ambavat, A. Gupta, and Q. Zhang. Switchcraft: AI model router for agentic tool calling. *arXiv preprint arXiv:2605.07112*, 2026. URL <https://arxiv.org/abs/2605.07112>.
- [2] Anthropic. System card: Claude opus 4.7. Anthropic Technical Report, April 2026. URL <https://www-cdn.anthropic.com/037f06850df7f7be871e206dad004c3db5fd50340/Claude%20opus%204.7%20System%20Card.pdf>. Accessed: 2026-06-24.
- [3] L. Chen, M. Zaharia, and J. Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.
- [4] L. Chen, M. Zaharia, and J. Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024. URL <https://openreview.net/forum?id=cSimKw5p6R>.
- [5] Z. Chen, K. Liu, Q. Wang, W. Zhang, J. Liu, D. Lin, K. Chen, and F. Zhao. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9354–9366, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.557. URL <https://aclanthology.org/2024.findings-acl.557/>.
- [6] Z. Chen, Z. Wei, Y. Bai, X. Xiong, and J. Wu. TagRouter: Learning route to LLMs through tags for open-domain text generation tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 21539–21564, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1110. URL <https://aclanthology.org/2025.findings-acl.1110/>.
- [7] O. Contributors. Opensquilla — token-efficient ai agent. <https://github.com/opensquilla/opensquilla>, 2026. GitHub repository.
- [8] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Ruhle, L. V. S. Lakshmanan, and A. H. Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=02f3mUtqnM>.
- [9] M. Dudik, J. Langford, and L. Li. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*, 2011.

- [10] A. Feng, B. Srinivasan, Y. Zhou, Z. Xu, K. Zhou, S. Guan, Y. Chen, X. Wu, N. Kulkarni, Y. Zhang, Z. Shen, D. Besspalov, S. S. Mishra, Y. Teng, D. Y.-B. Wang, H. Ding, and L. L. Cheong. 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*, pages 2484–2498, Suzhou, China, 2025. Association for Computational Linguistics. URL <https://aclanthology.org/2025.emnlp-industry.170/>.
- [11] J. Guo, Z. Hao, C. Wang, C. Fan, T. Luo, H. Li, Y. Gao, H. Mei, J. Peng, R. Xu, et al. From question answering to task completion: A survey on agent system and harness design. *arXiv preprint arXiv:2606.20683*, 2026.
- [12] Q. J. Hu, J. Bieker, X. Li, N. Jiang, B. Keigwin, G. Ranganath, K. Keutzer, and S. K. Upadhyay. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*, 2024. URL <https://arxiv.org/abs/2403.12031>.
- [13] D. Jiang, X. Ren, and B. Y. Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL <https://aclanthology.org/2023.acl-long.792/>.
- [14] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
- [15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, 2017.
- [16] H. Li, Y. Zhang, Z. Guo, C. Wang, S. Tang, Q. Zhang, Y. Chen, B. Qi, P. Ye, L. Bai, Z. Wang, and S. Hu. LLMRouterBench: A Massive Benchmark and Unified Framework for LLM Routing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 37733–37754, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1881. URL <https://aclanthology.org/2026.findings-acl.1881/>.
- [17] A. Mohammadshahi, A. R. Shaikh, and M. Yazdani. Routoo: Learning to route to large language models effectively. *arXiv preprint arXiv:2401.13979*, 2024. URL <https://arxiv.org/abs/2401.13979>.
- [18] Y. Moslem and J. D. Kelleher. Dynamic model routing and cascading for efficient llm inference: A survey. *arXiv preprint arXiv:2603.04445*, 2026.
- [19] Nous Research. Hermes-agent. <https://github.com/NousResearch/hermes-agent>, 2026. GitHub repository.
- [20] M. Okamoto, A. K. Erol, and M. Riedl. Explainable model routing for agentic workflows. *arXiv preprint arXiv:2604.03527*, 2026. URL <https://arxiv.org/abs/2604.03527>.
- [21] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

- [22] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=8sSqNtaMr>.
- [23] P. Panda, R. Magazine, C. Devaguptapu, S. Takemori, and V. Sharma. Adaptive LLM routing under budget constraints. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23934–23949, Suzhou, China, Nov. 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.1301. URL <https://aclanthology.org/2025.findings-emnlp.1301/>.
- [24] M. Šakota, M. Peyrard, and R. West. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining, WSDM '24*, pages 606–615, 2024. doi: 10.1145/3616855.3635825. URL <https://doi.org/10.1145/3616855.3635825>.
- [25] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=Yacmpz84TH>.
- [26] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.
- [27] T. Shnitzer, A. Ou, M. Silva, K. Soule, Y. Sun, J. Solomon, N. Thompson, and M. Yurochkin. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023. URL <https://arxiv.org/abs/2309.15789>.
- [28] A. Singh, A. Fry, A. Perelman, A. Tart, A. Ganesh, A. El-Kishky, A. McLaughlin, A. Low, A. Ostrow, A. Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.
- [29] P. Steinberger and OpenClaw Contributors. Openclaw — personal ai assistant. <https://github.com/openclaw/openclaw>, 2026. GitHub repository.
- [30] C. Tran, S. Paracha, A. Hafeez, and S. Chen. Arch-router: Aligning LLM routing with human preferences. *arXiv preprint arXiv:2506.16655*, 2025. URL <https://arxiv.org/abs/2506.16655>.
- [31] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- [32] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL <https://openreview.net/forum?id=ehfRiFOR3a>.
- [33] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Y. Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=h0ZfDIrj7T>.

- [34] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.
- [35] P. Yang, W. Chen, T. Yang, P. Feng, J. Xing, W. Guo, Y. Yao, Y. Han, H. Li, and X. Wang. Twinrouterbench: Fast static and live dynamic evaluation for realistic agentic llm routing. *arXiv preprint arXiv:2605.18859*, 2026.
- [36] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- [37] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.
- [38] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.
- [39] Y. Yue, G. Zhang, B. Liu, G. Wan, K. Wang, D. Cheng, and Y. Qi. MasRouter: Learning to route LLMs for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.757. URL <https://aclanthology.org/2025.acl-long.757/>.
- [40] A. Zeng, M. Liu, R. Lu, B. Wang, X. Liu, Y. Dong, and J. Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.181. URL <https://aclanthology.org/2024.findings-acl.181/>.
- [41] M. Zheng, K. Han, B. Li, H. Xu, Y. Tian, W. He, H. Zhou, J. Guo, H. Hu, L. Ma, C. Xu, G. Dai, L. Xia, Y. Wei, Y. Wang, and W. Yu. Claw-swe-bench: A benchmark for evaluating openclaw-style agent harnesses on coding tasks. *arXiv preprint arXiv:2606.12344*, 2026.
- [42] J. Zhong, H. Zhang, C. Southern, J. Yang, T. Wang, K. Jung, S. Zhang, D. Yarats, J. Ho, and J. Ma. Draco: A cross-domain benchmark for deep research accuracy, completeness, and objectivity. *arXiv preprint arXiv:2602.11685*, 2026.
- [43] P. Zhou, Z. Tang, Y. Ma, J. Tang, Y. Han, Z. Wan, F. Meng, W. Wang, B. Zhuang, W. Zhao, and Y. Yang. Agent-as-a-router: Agentic model routing for coding tasks. *arXiv preprint arXiv:2606.22902*, 2026.