ODAR: INSTANCE–COMPUTE ALIGNMENT FOR ADAPTIVE TEST-TIME REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling is commonly framed as allocating more reasoning to more difficult inputs, yet static difficulty need not track the realized benefit of additional computation. We present ODAR, an adaptive reasoning system whose deployed router learns from coarse complexity and Fast-Agent failure signals, while all-path counterfactual outcomes are reserved for evaluation. In an all-path audit, this inexpensive routing proxy predicts positive compute benefit with 0.77 AUROC, compared with 0.58 for input length, 0.68 for predictive uncertainty, and 0.694 for a PRM score. Holding the Simple/Medium/Hard proportions and average calls fixed, within-benchmark permutation lowers audit performance by 4.29 points, isolating *instance–compute alignment*; frozen-router transfer retains 74.4–87.8% of source-pair gains. Across 22 public benchmarks plus one custom proof stress test, ODAR-API exceeds Self-Consistency on all 23 displayed task-specific metrics while averaging 2.55 model calls. Token-level FEP selection is evaluated in the open-weight configuration, where next-token likelihoods are observable. Together, these results establish realized compute benefit as an informative evaluation target for adaptive inference and show that inexpensive supervision can identify instances that benefit from additional reasoning.

1 INTRODUCTION

Large language models (LLMs) increasingly improve reasoning by spending additional computation at inference time through Chain-of-Thought deliberation, Self-Consistency, Best-of-N, verifier-guided search, and related test-time scaling strategies (Wei et al., 2022; Wang et al., 2022; Aminian et al., 2025; Lightman et al., 2023; Lifshitz et al., 2025). A central systems question is therefore no longer simply *whether* to spend more compute, but *which inputs should receive it*. Recent routing and orchestration methods address this problem using predicted difficulty, confidence, uncertainty, or learned dispatch policies (McGill and Perona, 2017; Pan et al., 2025; Su et al., 2025; Guo et al., 2025; Liu et al., 2025; Chen et al., 2025; Wang et al., 2025). The resulting question is which measurable property best determines where additional inference compute should be allocated.

We study **realized marginal compute benefit** as the evaluation target for adaptive routing. Static difficulty describes an input, whereas compute benefit measures what a more expensive inference action actually changes. The two can diverge: an apparently difficult query may gain little from escalation, while a routine-looking query may benefit from verification or deeper search. During evaluation, we execute every available path for the same query and ask whether a deployable routing signal assigns extra compute to instances with positive realized value.

In this work, we study this question through **ODAR**, an *open-domain adaptive reasoner* that treats inference as resource allocation. For each query, a lightweight **Difficulty Estimator (DE)** assigns one of three compute regimes—Simple, Medium, or Hard—implemented by complementary **Fast** and **Slow Agents**. Selection is backend-aware: the closed-API configuration uses likelihood-free verification, while **Open-ODAR** uses the FEP score when both local generators expose token likelihoods. This separates the routing principle from any provider-specific likelihood interface.

The broad benchmark suite establishes the performance scope, while the mechanism experiments test why adaptive allocation helps. We compare routing signals against counterfactual compute benefit, permute query-to-path assignments at a fixed allocation histogram, transfer a frozen router across unseen Fast/Slow pairs, and factor routing, heterogeneity, and selection in a matched-budget design.

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.

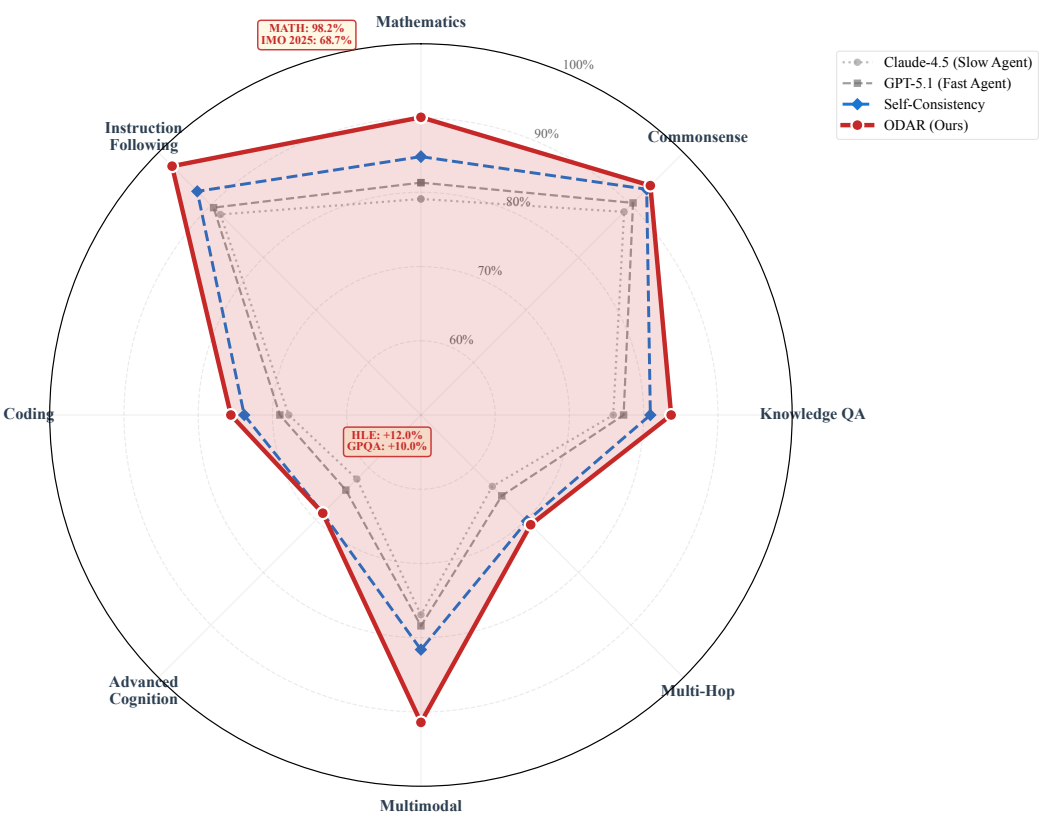

Figure 1: **Performance across 22 public benchmarks plus one custom proof stress test.** Values are task-specific percentage scores; the custom proof suite uses a verifier-based Proof Score rather than standard accuracy.

Together these tests measure *where compute is spent* directly rather than inferring the mechanism from aggregate accuracy.

Contributions. Our main contributions are:

- **A counterfactual evaluation target for adaptive compute.** We distinguish static difficulty from realized compute benefit and evaluate routing signals against all-path counterfactual outcomes. The deployed DE remains cheaply supervised—coarse complexity plus Fast-Agent failure—yet aligns with this target substantially better than length, uncertainty, PRM scores, and LLM difficulty judgments.

- **Direct evidence for instance–compute alignment.** A matched permutation control preserves the Simple/Medium/Hard proportions, model mixture, and average calls but destroys which query receives which path; the audit score drops by 4.29 points under within-benchmark permutation. A full $2 \times 2 \times 2$ decomposition separates routing, heterogeneous model complementarity, and candidate selection.

- **Generalization across tasks, budgets, and model pairs.** ODAR-API exceeds Self-Consistency on all 23 displayed task-specific metrics across 22 public benchmarks plus one custom proof stress test. A fully open implementation reduces compute by 82% relative to open-weight Self-Consistency, and a frozen router retains 74.4–87.8% of its source-pair gain on unseen model pairs without retraining.

2 Related Work

Recent work on test-time compute has moved from fixed budgets toward adaptive allocation through routing, early exit, conditional sampling, and instance-adaptive scaling (Pan et al., 2025; Su et al., 2025; Gupta et al., 2025; Shi et al., 2025; Park et al., 2025). Route-to-Reason (RTR) Pan et al. (2025) and CP-Router Su et al. (2025) dispatch inputs using predicted difficulty or uncertainty, including calibrated or conformal-style signals; HierRouter Gupta et al. (2025) learns hierarchical coordination among specialized experts, while InferenceDynamics Shi et al. (2025) emphasizes capability profiling and dispatch policies. A parallel line of work studies multi-agent orchestration: MoMA Guo et al. (2025) and RCR-Router Liu et al. (2025) route requests among role- or domain-specialized agents using capability-aware or structured control. Dual-process reasoning and conditional computation motivate separating inexpensive proposal generation from more expensive verification Segneri et al. (2020); Chen et al. (2025), while Active Inference and free-energy objectives have also been explored for planning and action selection, including PRACT Liu et al. (2024). ODAR connects these directions through depth-adaptive reasoning: a lightweight estimator selects among one-shot inference, Fast–Slow verification, and multi-candidate exploration, with candidate selection matched to the information exposed by each backend. Its main distinction is the evaluation target. Rather than assuming that static difficulty, uncertainty, or a learned score implies that more computation will help, ODAR evaluates routing signals against counterfactual compute benefit—the realized utility change from assigning the same instance to a more expensive inference action. A matched-allocation permutation control preserves the global Simple/Medium/Hard mixture and average calls while breaking query–path correspondence, isolating the value of instance-level allocation. In the open-weight configuration, the FEP selector combines normalized likelihood with a variance penalty on likelihood fluctuations and is evaluated as a separate candidate-selection mechanism.

3 Method

3.1 Routing Formulation and Difficulty Estimation

As shown in Figure 2, ODAR consists of a Difficulty Estimator $d(x) \in [0, 1]$, an Adaptive Router that assigns compute budgets, a Fast/Slow dual-agent stack, and a backend-compatible selector. The selector is likelihood-free for the proprietary API stack and FEP-based for Open-ODAR. Given an input query x, ODAR predicts d and assigns one of three paths: Simple ($d < 0.3$, $c = 1$), Medium ($0.3 \le d < 0.7$, $c = 2$), or Hard ($d \ge 0.7$, $c = 6$). The upper orchestration layer provides lightweight domain and strategy dispatch, while the three execution paths perform one-shot inference, Fast–Slow verification, or multi-candidate exploration.

ODAR uses a lightweight estimator $d_\phi(x)$ trained from input features. Its supervision combines signals available from the inexpensive training pipeline, while marginal-utility and all-path counterfactual outcomes are reserved for the evaluation audit. We construct an ***EFE-inspired*** routing target d_i^* from a coarse complexity signal and an empirical fast-path failure signal. For a training query x_i, we define

$$d_i^* = \underbrace{\alpha \cdot \frac{T_{\text{comp}}(x_i)}{\max_j T_{\text{comp}}(x_j)}}_{\text{Complexity proxy}} + \underbrace{(1-\alpha) \cdot (1 - \text{Acc}_{\text{fast}}(x_i))}_{\text{Fast-path failure signal}}, \tag{1}$$

where $T_{\text{comp}}(x_i)$ denotes a dataset-specific complexity proxy and $\text{Acc}_{\text{fast}}(x_i)$ is the Fast Agent's correctness indicator. The first term captures coarse problem complexity, while the second records whether the inexpensive Fast path fails; d_i^* is the empirical supervision signal for compute allocation.

We set $\alpha = 0.5$ and parameterize the estimator as

$$d(x) = \sigma(W \cdot \phi(x) + b), \tag{2}$$

where $\phi(x) \in \mathbb{R}^{24}$ contains structural and semantic features; full definitions are provided in Appendix I. The estimator uses explicit structural cues alongside semantic features, and Appendix T.3 analyzes their cross-domain generalization.

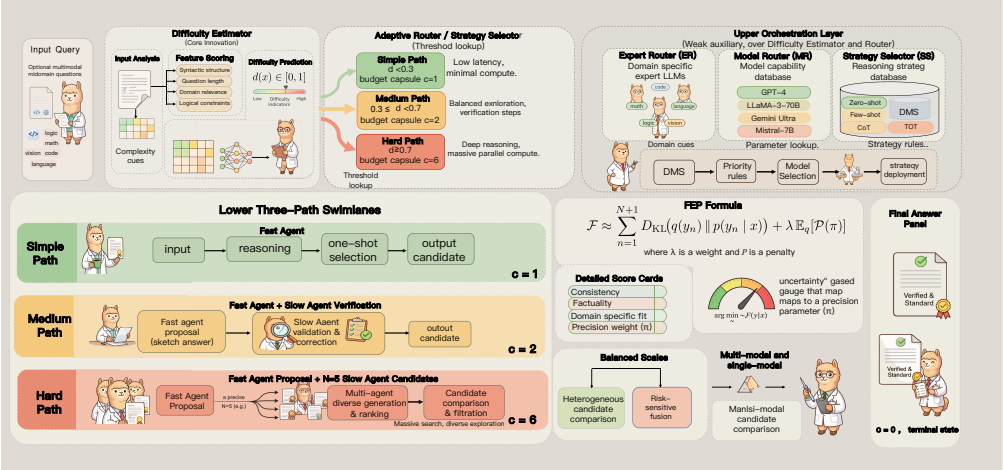

Figure 2: **ODAR architecture.** The Difficulty Estimator assigns each query to a **Simple** ($c = 1$), **Medium** ($c = 2$), or **Hard** ($c = 6$) path. Candidate selection uses likelihood-free verification in the closed-API configuration and FEP scoring in the open-weight configuration where token likelihoods are available.

Given the predicted score d, ODAR selects the reasoning path using thresholds $(\tau_1, \tau_2) = (0.3, 0.7)$:

$$\mathrm{Path}(x) = \begin{cases} \text{Simple } (\pi_{\mathrm{S}}), & d < \tau_1, \\ \text{Medium } (\pi_{\mathrm{M}}), & \tau_1 \le d < \tau_2, \\ \text{Hard } (\pi_{\mathrm{H}}), & d \ge \tau_2. \end{cases} \tag{3}$$

Thus, $d(x)$ is an inexpensive routing proxy learned independently of the all-path utility audit. Section 4.2 then evaluates its alignment with the realized marginal value of additional compute.

3.2 Adaptive Execution with Fast/Slow Agents

ODAR uses two complementary agents with distinct cost–capability profiles: a Fast Agent (A_θ; GPT-5.1, T=0.2, 512 tokens) for rapid hypothesis generation and a Slow Agent (A_γ; Claude-4.5 Sonnet, T=0.3, 1024 tokens) for verification and deeper exploration. The Fast Agent runs at 1.8s average latency, while a Slow pass averages 6.4s. Homogeneous-agent ablations reduce accuracy by 1.8% when both agents use GPT-5.1 and increase cost by 47% when both use Claude-4.5 (Section 4.3), supporting the separation of inexpensive proposal generation from slower refinement.

ODAR allocates test-time compute through three routing paths determined by the predicted difficulty score d. When $d < 0.3$, the model follows a simple path and executes a single Fast-Agent call:

$$\hat{y} = A_\theta(x). \tag{4}$$

This path minimizes latency when additional verification is not warranted. When $0.3 \le d < 0.7$, the Fast Agent produces y_θ and the Slow Agent verifies or refines it to obtain y_γ. A selector $\mathcal{S}$ returns the final response from $\{y_\theta, y_\gamma\}$, requiring two model calls.

When $d \ge 0.7$, ODAR escalates to multi-candidate exploration: the Fast Agent generates y_θ and the Slow Agent produces $n = 5$ additional candidates. Let

$$\mathcal{C} = \{y_\theta\} \cup \{y_\gamma^{(i)}\}_{i=1}^{n}. \tag{5}$$

ODAR then selects

$$\hat{y} = \mathcal{S}(\mathcal{C}, x). \tag{6}$$

The Hard path uses six total calls. In the closed-API configuration, $\mathcal{S}$ is the likelihood-free verification selector used by the evaluation harness; in Open-ODAR, $\mathcal{S}$ is the FEP selector in Section 3.3.

Under the empirical routing distribution, the expected inference cost is

$$\mathbb{E}[\text{calls}] = 0.41(1) + 0.35(2) + 0.24(6) = 2.55, \tag{7}$$

which corresponds to a 1.96× reduction in model invocations relative to Self-Consistency with $n = 5$.

3.3 LIKELIHOOD-BASED FEP SELECTOR FOR OPEN-ODAR

The FEP selector operates on sampled-token likelihood sequences and is therefore instantiated in **Open-ODAR**, where both local generators expose next-token logits. The closed-API stack uses the likelihood-free verification selector defined in Appendix O.2. For likelihood-capable models, ODAR applies a risk-sensitive free-energy-inspired score that favors confident candidates while penalizing unstable token-level likelihood profiles; character normalization and model-specific standardization align scores across heterogeneous generators.

For a candidate y_i generated by model m, we compute

$$\begin{aligned}\mathcal{F}_{\mathrm{char}}(y_i \mid x; m) = -\frac{1}{|y_i|_{\mathrm{char}}} \cdot \\ \sum_{t=1}^{|y_i|_{\mathrm{tok}}} \log p_m(y_{i,t} \mid y_{i,<t}, x) \\ + \lambda \cdot \\ \mathrm{Var}_t\Big[\log p_m(y_{i,t} \mid y_{i,<t}, x)\Big],\end{aligned} \tag{8}$$

where the first term is a per-character energy density and the second penalizes unstable likelihood profiles.

We then standardize this score using held-out calibration statistics (μ_m, σ_m):

$$\tilde{\mathcal{F}}(y_i \mid x; m) = \frac{\mathcal{F}_{\mathrm{char}}(y_i \mid x; m) - \mu_m}{\sigma_m}. \tag{9}$$

This maps candidates from different generators onto a common scale while preserving within-model ordering.

Given a candidate set $\mathcal{C}$, ODAR converts aligned scores into Boltzmann-style weights. For the medium path, with $\mathcal{C} = \{y_\theta, y_\gamma\}$, we compute

$$p_i = \frac{\exp\Big(-\tilde{\mathcal{F}}(y_i \mid x; m(y_i))/\tau\Big)}{\sum_{y_j \in \mathcal{C}} \exp\Big(-\tilde{\mathcal{F}}(y_j \mid x; m(y_j))/\tau\Big)}, \tag{10}$$

with $\tau = 1.0$ unless otherwise specified.

For the hard path, with $\mathcal{C} = \{y_\theta\} \cup \{y_{\gamma,i}\}_{i=1}^{N}$, ODAR selects

$$\hat{y} = \arg\min_{y \in \mathcal{C}} \tilde{\mathcal{F}}(y \mid x; m(y)) = \arg\max_{y \in \mathcal{C}} p(y), \tag{11}$$

where $m(y)$ denotes the generator model of candidate y.

For later calibration analysis, we define confidence as

$$c_{\mathrm{FEP}} = \max_{y \in \mathcal{C}} p(y), \tag{12}$$

and margin as

$$\Delta_{\mathcal{F}} = \tilde{\mathcal{F}}_{(2)} - \tilde{\mathcal{F}}_{(1)}, \tag{13}$$

where $\tilde{\mathcal{F}}_{(1)}$ and $\tilde{\mathcal{F}}_{(2)}$ are the smallest and second-smallest aligned energies in $\mathcal{C}$. The full routing procedure is summarized in Algorithm 2: easy queries receive one Fast call, intermediate cases trigger Slow verification, and hard cases invoke multi-candidate exploration. Appendix O.2 specifies the selector used by each backend and the likelihood-capable setting in which token-level FEP is instantiated.

4 EXPERIMENTS

We evaluate ODAR on 23 benchmarks spanning 8 task categories, covering matched-budget performance, counterfactual compute benefit, instance–compute alignment, and cross-model transfer.

Table 1: **Compute-matched performance across 22 public benchmarks and one custom proof stress test.** Values are task-specific percentage scores; the proof suite uses a verifier-based Proof Score. The final row is an unweighted descriptive macro score; paired bootstrap intervals are in Appendix Y.

Category	Dataset	GPT-5.1	Claude-4.5	Len-Adapt.	Self-Consist.	ODAR-API (Ours)	Gain vs. GPT-5.1
Mathematics	MATH Hendrycks et al. (2021)	91.8	89.2	94.0	94.5	**98.2**	+6.4
	GSM8K Cobbe et al. (2021)	95.9	94.5	97.0	97.2	**99.1**	+3.2
	Custom Proof Suite (Proof Score)	48.5	45.2	55.6	55.4	**68.7**	+20.2
	MathVista Lu et al. (2023)	89.1	87.4	91.7	92.3	**96.5**	+7.4
Commonsense	ARC-Challenge Clark et al. (2018)	92.5	90.8	94.6	94.2	**98.5**	+6.0
	OpenBookQA Mihaylov et al. (2018)	91.2	89.5	93.5	93.8	**97.8**	+6.6
	BoolQ Clark et al. (2019)	93.4	92.1	95.1	95.2	**98.1**	+4.7
	StrategyQA Geva et al. (2021)	84.5	82.3	87.6	88.9	**93.4**	+8.9
Knowledge	MMLU-Pro Wang et al. (2024)	86.2	85.1	89.0	89.5	**94.2**	+8.0
	GPQA Rein et al. (2024)	68.5	66.8	72.0	72.4	**78.5**	+10.0
Multi-Hop	HotpotQA Yang et al. (2018)	78.5	76.4	82.4	82.1	**89.5**	+11.0
	MuSiQue Trivedi et al. (2022)	52.3	50.8	56.8	58.4	**65.2**	+12.9
Multimodal	ScienceQA Saikh et al. (2022)	90.5	89.2	92.9	93.1	**97.4**	+6.9
	AOK-VQA Schwenk et al. (2022)	86.2	84.8	89.1	89.5	**94.6**	+8.4
	MMMU-Pro Yue et al. (2025)	58.5	56.9	62.1	62.4	**68.7**	+10.2
Advanced	BBH Suzgun et al. (2023)	88.0	86.5	90.6	91.2	**95.4**	+7.4
	BBEH Kazemi et al. (2025)	48.5	46.2	52.1	52.8	**58.9**	+10.4
	TruthfulQA Lin et al. (2022)	78.2	76.5	81.8	82.4	**88.6**	+10.4
	HLE Phan et al. (2025)	42.8	40.5	47.0	48.5	**54.8**	+12.0
Coding	SWE-bench Jimenez et al. (2023)	62.5	60.8	66.6	68.4	**74.2**	+11.7
	LIVEBENCH White et al. (2024)	75.6	74.8	78.0	79.3	**82.4**	+6.8
Instruction	IFEval Zhou et al. (2023)	89.5	88.2	92.5	92.6	**98.1**	+8.6
Abstract	ARC-AGI-2 Chollet et al. (2025)	30.2	28.5	34.2	36.4	**41.5**	+11.3
Secondary Macro Score (Unweighted)		74.9	73.2	78.1	78.7	**84.0**	**+9.1**

Table 2: **Open-ODAR with local likelihood-capable backends.** Llama 4 + DeepSeek V3.2 expose token logits for the FEP selector.

Category	Dataset	(A) Llama 4 (Fast Agent)	(B) DeepSeek (CoT)	(C) DeepSeek (SC $k = 5$)	(D) Open-ODAR (Llama+DeepSeek)	Gain (vs. SC)
Math	MATH Hendrycks et al. (2021)	76.5	87.8	90.2	**92.4**	+2.2%
	GSM8K Cobbe et al. (2021)	88.4	96.3	97.1	**97.8**	+0.7%
Coding	LiveBench White et al. (2024)	58.2	71.4	73.5	**76.1**	+2.6%
	SWE-bench Jimenez et al. (2023)	45.5	62.5	65.8	**68.2**	+2.4%
Reasoning	GPQA Rein et al. (2024)	35.8	68.4	70.5	**72.8**	+2.3%
	HotpotQA Yang et al. (2018)	74.2	81.5	83.8	**86.4**	+2.6%
Knowledge	MMLU-Pro Wang et al. (2024)	68.5	85.0	86.2	**89.1**	+2.9%
Instruction	IFEval Zhou et al. (2023)	86.5	88.2	88.9	**92.5**	+3.6%
Average		66.7	80.1	82.0	**84.4**	**+2.4%**
Cost (Normalized)		$1.0\times$	$5.0\times$	$25.0\times$	$\sim 4.5\times$	**-82%**

4.1 Experimental Setup and Audit Protocol

ODAR uses GPT-5.1 as the Fast Agent and Claude-4.5-Sonnet as the Slow Agent with task-invariant hyperparameters (Appendix M). Results average three random seeds; datasets above 5,000 instances use stratified $K{=}5$ fold estimates. Baselines share prompting, answer extraction, and budget accounting. We report model calls, normalized compute, and API spend separately: ODAR-API averages 2.55 calls/query versus 5.00 for Self-Consistency, with mean reported API spend of $0.42 versus $0.75 per query.

For the counterfactual allocation audit, each query is executed offline through every action $a \in \{S, M, H\}$. For action output $y_a(x)$, task reward $R(y_a(x))$, and measured cost C_a, define

$$U_a(x) = R(y_a(x)) - \lambda_c C_a, \qquad \Delta U_a(x) = U_a(x) - U_S(x), \tag{14}$$

Table 3: **Budget-matched adaptive-compute comparison.** All methods use the unified evaluation harness; Avg. Calls is model invocations per query.

Method	Type	MATH	GSM8K	GPQA	Avg. Calls
GPT-5.1 Base	Single-pass	91.8	95.9	68.5	1.00
Self-Consistency (N=5)	Fixed Sampling	94.5	97.2	72.4	5.00
Best-of-N (N=5)	Fixed Sampling + PRM Ranking	95.3	97.6	73.8	5.00
Park-style IAS (Park et al., 2025)	Calibrated PRM + IAS (reimpl.)	96.8	98.4	75.1	3.45
ODAR-API (Ours)	**Adaptive routing + verification**	**98.2**	**99.1**	**78.5**	**2.55**

and the counterfactual oracle action as

$$a^*(x) = \arg \max_{a \in \{S,M,H\}} U_a(x). \tag{15}$$

Compute-Benefit AUROC tests whether a routing signal predicts $\max_{a \in \{M,H\}} \Delta U_a(x) > 0$. Counterfactual outcomes define the offline audit and oracle upper bound; deployed routing uses Eq. 1. Because benchmark metrics mix Accuracy, EM/F1, Pass@1, Similarity, and Proof Score, 84.0 is an unweighted descriptive macro score; primary comparisons remain task-specific or use their designated audit pools.

4.2 Counterfactual compute benefit and instance–compute alignment

We evaluate routing signals against counterfactual outcomes obtained by executing all available paths on the same queries. ODAR reaches a Compute-Benefit AUROC of 0.77 and oracle-action accuracy of 0.742, compared with 0.58/0.487 for length, 0.68/0.613 for predictive uncertainty, 0.694/0.648 for PRM score, and 0.716/0.681 for an LLM difficulty judge. Thus, static difficulty and the realized value of additional compute are related but non-equivalent targets.

A matched permutation control preserves the 41/35/24% Simple/Medium/Hard proportions and 2.55 average calls while permuting query-to-path assignments. Within-benchmark permutation reduces the audit score from 85.13 to 80.84 (-4.29 points), directly isolating *instance–compute alignment*. A frozen router transferred to unseen Fast/Slow pairs retains 74.4–87.8% of its source-pair gain, with Benefit AUROC remaining 0.704–0.746. Table 4 summarizes these mechanism tests; full tables are reported in Appendix X.

The $2 \times 2 \times 2$ factorial decomposition, multi-budget Pareto frontier, and oracle-gap analysis are given in Appendix X.

4.3 Results, Generalization, and Component Analysis

Table 1 summarizes 22 public benchmarks plus the custom proof stress test. ODAR-API exceeds Self-Consistency on all 23 displayed task-specific metrics. As a secondary descriptive summary, the unweighted arithmetic mean is 84.0 for ODAR-API versus 78.7 for Self-Consistency (+5.3). The per-row gain column is measured against GPT-5.1; it ranges from +3.2 on GSM8K to +20.2 Proof-Score points on the custom proof suite. Because the suite mixes Accuracy, EM/F1, Pass@1, Similarity, and a custom Proof Score, we call the aggregate a *macro score*, not average accuracy. Gains versus GPT-5.1 are largest on sustained multi-step reasoning, including the custom proof suite (+20.2 Proof-Score points), HLE (+12.0), GPQA (+10.0), BBEH (+10.4), MuSiQue (+12.9), and SWE-bench (+11.7). The counterfactual analysis in Section 4.2 links these gains to instance-dependent value of compute rather than to the global budget alone.

Table 4: **Mechanism summary.** The controls isolate value-of-compute prediction, instance–compute alignment, and cross-model transfer without changing the deployed ODAR pipeline.

Mechanism	ODAR result	Matched comparison	Finding
Does static difficulty equal compute benefit?	Benefit AUROC 0.770; oracle-action acc. 0.742	Length 0.580; uncertainty 0.680; PRM 0.694; LLM judge 0.716	ODAR is more aligned with marginal value of extra reasoning than conventional difficulty proxies.
Instance–compute alignment	Audit score 85.13 at 2.55 calls; path mix 41/35/24%	Within-benchmark permutation: 80.84 at the same 2.55 calls and path mix	Correct query–path alignment contributes 4.29 accuracy points.
Cross-model transfer	Frozen transfer retains 74.4–87.8% of source-pair gain	No router retraining or threshold refit on unseen model pairs	The routing signal transfers substantially across model families.

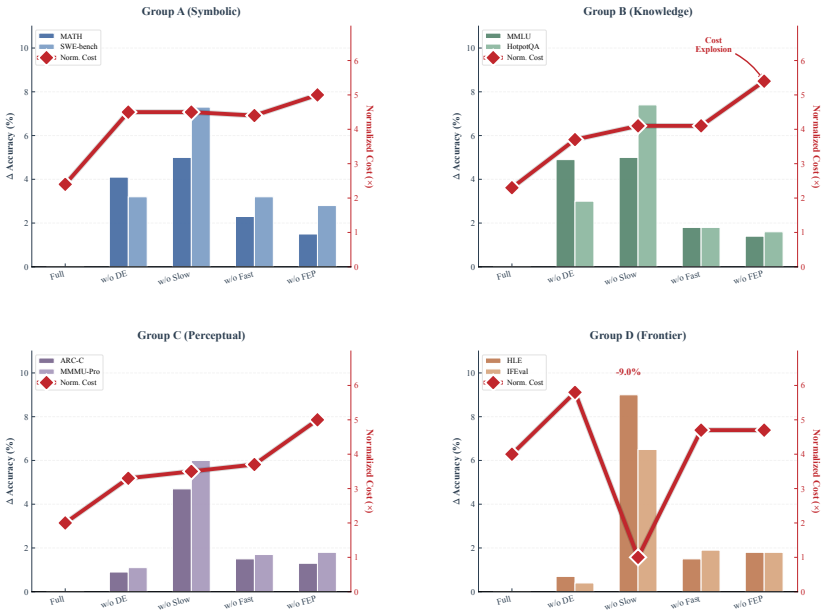

Figure 3: **Multi-dimensional ablation matrix.** Bars report accuracy drops after removing each component, while the line reports normalized inference cost. The Difficulty Estimator primarily improves call efficiency, whereas Slow-Agent reasoning and candidate selection contribute most on harder tasks.

We evaluate Open-ODAR with Llama 4 Scout as the Fast Agent and DeepSeek V3.2 as the Slow Agent. The reference implementation uses fixed local weights, full probability outputs, FP8 quantization, pre-computed calibration statistics (μ_m, σ_m), and a Dockerized runtime (Appendix P). Open-ODAR reaches 84.4% average accuracy versus 82.0% for DeepSeek Self-Consistency ($k = 5$), while reducing normalized cost from $25.0\times$ to approximately $4.5\times$ (82%); gains span mathematics, coding, multi-hop reasoning, knowledge, and instruction following. Under matched backbones, prompting, answer extraction, and compute accounting, ODAR-API reaches 98.2/99.1/78.5 on MATH/GSM8K/GPQA with 2.55 calls, compared with 3.45 for Park-style IAS (Table 3). On the counterfactual audit pool at $5\times$ normalized cost, ODAR scores 85.13 versus 83.19 for Park-style IAS, 83.02 for CertaIndex-style adaptive scaling, 82.66 for Learning How Hard to Think, and 82.47 for Difficulty-Adaptive Self-Consistency; Appendix X reports the full $2\times$–$10\times$ frontier.

The component analyses separate routing from candidate conversion. FEP reliability is measured on the log-probability-capable Open-ODAR stack with fixed heterogeneous candidate pools on

TRUTHFULQA, MATH, and GSM8K, while Appendix L reports selector ablations. Figure 3 shows that removing adaptive routing raises average calls from 2.55 to 6.00, whereas Slow-Agent verification and candidate selection contribute most on hard tasks. At identical path proportions and call counts, matched permutation reduces the audit score by 4.29 points (Table 35); the full factorial decomposition in Appendix X further separates adaptive routing, heterogeneous specialization, and calibrated selection.

System-level measurements connect these component effects to deployment behavior. ODAR reaches 88.5% accuracy at $1.4\times$ normalized cost, corresponding to a 4-fold reduction relative to Self-Consistency, and 41% of queries complete within 3s through the Simple Path while the Hard Path forms the long-latency tail. Under adversarial routing perturbations, complexity injection raises compute by 154% with a -0.2% accuracy change, whereas simplicity masking removes useful compute and reduces accuracy by 6.4%. Manual error analysis assigns 34% of sampled failures to routing, verification, or candidate selection and 66% to cases in which all available paths fail, separating allocation errors from base-model capability errors.

The matched factorial analysis places these effects in a common hierarchy: adaptive routing contributes the largest main effect (+2.61 Audit Score), followed by heterogeneous Fast/Slow specialization (+1.49) and calibrated selection (+0.72), with positive interactions among all three components. Relative to the Fast-Agent reference, ODAR closes 63.72% of the counterfactual oracle allocation gap. This decomposition reinforces the central separation between deciding where compute should be spent and converting that allocated compute into a final answer.

5 DISCUSSION

Our results support adaptive test-time reasoning as an *instance–compute alignment* problem rather than uniform scaling. The central question is not simply whether more reasoning improves performance, but whether additional computation is assigned to queries for which it has positive realized value. ODAR's lightweight Difficulty Estimator predicts this benefit more effectively than input length, uncertainty, PRM scores, and LLM-based difficulty judgments. The matched permutation experiment further isolates query–path correspondence: preserving the Simple/Medium/Hard proportions and average calls while disrupting instance-level assignments reduces the audit score by 4.29 points, showing that the global compute mixture alone does not explain the gain.

The experiments also separate the roles of routing, heterogeneous Fast/Slow reasoning, and candidate selection. Routing determines where additional computation is spent, while specialization and selection determine how effectively that budget is converted into a final answer. Factorial decomposition identifies adaptive routing as the largest individual contributor, with complementary gains from heterogeneous specialization and calibrated selection. Frozen-router transfer and multi-budget evaluation further show that the allocation signal generalizes across model pairs and compute regimes, supporting selective escalation as a practical way to improve the accuracy–compute trade-off.

6 CONCLUSION AND LIMITATIONS

We introduced ODAR, an adaptive reasoning framework that allocates test-time computation according to estimated instance-level reasoning demand. The deployed router learns from inexpensive complexity and Fast-Agent failure signals, while an all-path counterfactual audit evaluates whether this proxy identifies queries that actually benefit from additional reasoning. Across matched-allocation, factorial, budget-frontier, and cross-model transfer experiments, the results consistently show that instance–compute alignment contributes beyond simply increasing the global inference budget.

ODAR currently operates over three discrete reasoning paths, leaving finer-grained or continuous allocation as a natural extension for more precise control of reasoning depth and latency. Token-level FEP selection is evaluated in the open-weight configuration where next-token likelihoods are observable, while the proprietary API stack uses likelihood-free verification. Because the benchmark suite contains heterogeneous task metrics, aggregate macro scores are descriptive summaries, with primary conclusions based on task-specific results, matched-budget comparisons, and counterfactual allocation analyses. The custom proof stress test is likewise interpreted through its verifier-based Proof Score.

AI USE STATEMENT

Generative AI tools were used solely for language polishing. All content was reviewed and approved by the authors.

REFERENCES

Gholamali Aminian, Idan Shenfeld, Amir R Asadi, Ahmad Beirami, and Youssef Mroueh. Best-of-n through the smoothing lens: Kl divergence and regret analysis. *arXiv preprint arXiv:2507.05913*, 2025.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. *arXiv preprint arXiv:2502.13842*, 2025.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.

Xiyu Guo, Shan Wang, Chunfang Ji, Xuefeng Zhao, Wenhao Xi, Yaoyao Liu, Qinglan Li, Chao Deng, and Junlan Feng. Towards generalized routing: Model and agent orchestration for adaptive and efficient inference. *arXiv preprint arXiv:2509.07571*, 2025.

Nikunj Gupta, Bill Guo, Rajgopal Kannan, and Viktor K Prasanna. Hierrouter: Coordinated routing of specialized large language models via reinforcement learning. *arXiv preprint arXiv:2511.09873*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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?, 2024. *URL https://arxiv. org/abs/2310.06770*, 7, 2023.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Yuanzhu Peter Chen, et al. Big-bench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26473–26501, 2025.

Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 3214–3252, 2022.

Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjekye, Hao Tang, Geng Yuan, Wei Niu, et al. Rcr-router: Efficient role-aware context routing for multi-agent llm systems with structured memory. *arXiv preprint arXiv:2508.04903*, 2025.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Zuxin Liu, Liangwei Yang, Rithesh RN, Tian Lan, Ming Zhu, Juntao Tan, Shirley Kokane, et al. Pract: Optimizing principled reasoning and acting of llm agent. In *Proceedings of the 28th Conference on Computational Natural Language Learning*, pages 442–446, 2024.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, pages 2363–2372. PMLR, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Zhihong Pan, Kai Zhang, Yuze Zhao, and Yupeng Han. Route to reason: Adaptive routing for llm and reasoning strategy selection. *arXiv preprint arXiv:2505.19435*, 2025.

Young-Jin Park, Kristjan Greenewald, Kaveh Alim, Hao Wang, and Navid Azizan. Know what you don't know: Uncertainty calibration of process reward models. *arXiv preprint arXiv:2506.09338*, 2025.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Tanik Saikh, Tirthankar Ghosal, Amish Mittal, Asif Ekbal, and Pushpak Bhattacharyya. Scienceqa: A novel resource for question answering on scholarly articles. *International Journal on Digital Libraries*, 23(3):289–301, 2022.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-okvqa: A benchmark for visual question answering using world knowledge. In *European conference on computer vision*, pages 146–162. Springer, 2022.

Marco Segneri, Hongjie Bi, Simona Olmi, and Alessandro Torcini. Theta-nested gamma oscillations in next generation neural mass models. *Frontiers in computational neuroscience*, 14:47, 2020.

Haochen Shi, Tianshi Zheng, Weiqi Wang, Baixuan Xu, Chunyang Li, Chunkit Chan, Tao Fan, Yangqiu Song, and Qiang Yang. Inferencedynamics: Efficient routing across llms through structured capability and knowledge profiling. *arXiv preprint arXiv:2505.16303*, 2025.

Jiayuan Su, Fulin Lin, Zhaopeng Feng, Han Zheng, Teng Wang, Zhenyu Xiao, Xinlong Zhao, Zuozhu Liu, Lu Cheng, and Hongwei Wang. Cp-router: An uncertainty-aware router between llm and lrm. *arXiv preprint arXiv:2505.19970*, 2025.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Ji Wang, Kashing Chen, Xinyuan Song, Ke Zhang, Lynn Ai, Eric Yang, and Bill Shi. Symphony: A decentralized multi-agent framework for scalable collective intelligence. *arXiv preprint arXiv:2508.20019*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 4, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, et al. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15134–15186, 2025.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

APPENDIX CONTENTS

A. Code Availability 13
B. Notation and Nomenclature 13
C. Theoretical Motivation: Risk-Sensitive Control 13
D. Algorithm Specifications 15
E. Difficulty Estimator Training Details 15
F. Architectural Analogy 16
G. Extended Comparison with Concurrent Frameworks 16
H. Design-to-Implementation Mapping 17
I. Feature Extraction for Difficulty Estimation 17
J. Error Analysis and Taxonomy 21
K. FEP Fusion Case Study 22
L. Fusion Method Ablation 22
M. Hyperparameters and Configuration 23
N. Dual-Agent Prompt Templates 24
O. Implementation Details 27
P. Detailed Reproducibility Protocols 28
Q. Dataset Details 29
R. Additional Performance Analysis 30
S. Pareto Efficiency Analysis 31
T. Additional Ablation Studies 33
U. Comparison with Learned Verifiers 36
V. Adversarial Robustness Analysis 37
W. Case Studies 37
X. Counterfactual Allocation and Factorial Analysis 38
Y. Paired Statistical Analysis and Subsampling Stability 41

A CODE AVAILABILITY

The release package contains the ODAR implementation, Difficulty Estimator training scripts, FEP-based fusion module, evaluation suite for all 23 benchmarks, pre-trained Difficulty Estimator weights, and the Dockerized Open-ODAR environment.

B NOTATION AND NOMENCLATURE

To facilitate understanding of the theoretical derivations and algorithmic implementation, we summarize the key mathematical notations and system variables in Table 5.

Table 5 separates three roles that are easy to conflate in an adaptive system: $d(x)$ controls *allocation*, the path set Π defines the available compute actions, and $\mathcal{F}$ governs *selection* within likelihood-capable candidate pools. This separation is central to the empirical design. The counterfactual audit evaluates whether the allocation signal sends compute to useful instances, whereas the FEP ablations evaluate how well already-generated candidates are ranked. Keeping these objects distinct prevents selector quality from being mistaken for evidence about routing quality.

C THEORETICAL MOTIVATION: RISK-SENSITIVE CONTROL AND APPROXIMATION ASSUMPTIONS

The FEP selector uses risk-sensitive control as a mean–variance motivation and makes the approximation assumptions explicit below. Its role in ODAR is a practical candidate-scoring objective whose empirical contribution is tested independently in the selector ablations.

Table 5: Nomenclature and key variables used throughout the ODAR framework.

SYMBOL	DESCRIPTION
Variables & Distributions	
x	INPUT QUERY (TEXT OR MULTIMODAL)
$y, \hat{y}$	CANDIDATE ANSWER SEQUENCE / FINAL SELECTED ANSWER
s	LATENT COGNITIVE STATE (REASONING PATH)
$q(s\|y)$	VARIATIONAL BELIEF DISTRIBUTION OVER HIDDEN STATES
$\mathcal{F}(y\|x)$	VARIATIONAL FREE ENERGY (ACCURACY + COMPLEXITY)
$G(\pi\|x)$	EXPECTED FREE ENERGY (EFE) FOR POLICY SELECTION
System Components & Routing	
m_θ / A_θ	FAST AGENT (GPT-5.1)
m_γ / A_γ	SLOW AGENT (CLAUDE-4.5)
$\phi(x)$	FEATURE VECTOR EXTRACTED FROM QUERY x
$d(x)$	PREDICTED DIFFICULTY SCORE, $d \in [0, 1]$
Π	POLICY SET $\{\pi_S, \pi_M, \pi_H\}$ (SIMPLE, MEDIUM, HARD)
$\mathcal{C}$	SET OF CANDIDATE ANSWERS FOR SELECTION
Parameters & Weights	
τ_1, τ_2	ROUTING THRESHOLDS (0.3, 0.7)
n	NUMBER OF SAMPLES FOR BEST-OF-N STRATEGY
w_θ, w_γ	FEP-BASED FUSION WEIGHTS FOR AGENT AGGREGATION
λ	COMPLEXITY PENALTY WEIGHT IN FEP FORMULATION

C.1 RISK-SENSITIVE MEAN–VARIANCE MOTIVATION

Let $L(y) = -\log p(y \mid x)$ denote sequence surprisal. The entropic risk measure

$$\mathcal{J}_\gamma(x) = \frac{1}{\gamma} \log \mathbb{E}_{y \sim p(\cdot|x)} \left[\exp(\gamma L(y))\right] \tag{16}$$

admits the cumulant expansion

$$\mathcal{J}_\gamma(x) = \mathbb{E}[L(y)] + \frac{\gamma}{2} \operatorname{Var}[L(y)] + \mathcal{O}(\gamma^2). \tag{17}$$

This motivates a mean–variance penalty when γ is small enough for the second-order approximation to be informative.

C.2 VARIANCE DECOMPOSITION

For a latent reasoning state s, the law of total variance gives

$$\operatorname{Var}[L] = \mathbb{E}_s[\operatorname{Var}(L \mid s)] + \operatorname{Var}_s(\mathbb{E}[L \mid s]) \,. \tag{18}$$

The second term can be interpreted as variability associated with uncertainty over latent reasoning states, while the first captures within-state variation. Consequently,

$$\operatorname{Var}[L] \geq \operatorname{Var}_s(\mathbb{E}[L \mid s]) \,. \tag{19}$$

Thus total variance is an *upper bound* on this epistemic component. Under the additional **epistemic-dominance assumption**

$$\operatorname{Var}_s(\mathbb{E}[L \mid s]) \gg \mathbb{E}_s[\operatorname{Var}(L \mid s)] \,, \tag{20}$$

the total variance approximates the epistemic component from above.

C.3 TOKEN-LEVEL PRACTICAL ESTIMATOR

A single sampled response supplies token-level surprisal along one decoding trajectory rather than an ensemble estimate of $\operatorname{Var}_y[L(y)]$. We therefore use the following **temporal-surprisal approximation**:

$$\operatorname{Var}_{y \sim p(\cdot|x)}[L(y)] \approx \widehat{\operatorname{Var}}_t \left[-\log p(y_t \mid y_{<t}, x)\right] . \tag{21}$$

Equation 21 is the estimator assumption that connects the risk-sensitive motivation to the implementable token-level score. Formatting-only and special tokens are excluded before computing temporal variance.

Scope of the theoretical claim. Risk-sensitive control motivates the mean–variance form, variance decomposition motivates the uncertainty interpretation, and Eq. 21 supplies the practical token-level approximation. This establishes the intended theoretical scope of the FEP score: a risk-sensitive scoring construction evaluated by its candidate-selection behavior in Open-ODAR.

D Algorithm Specifications

This appendix specifies the adaptive routing procedure and the FEP selector used in the log-probability-capable Open-ODAR configuration. The proprietary API configuration uses the likelihood-free verification selector described in Appendix O.2.

E Difficulty Estimator Training Details

E.1 Data Construction and Partition Provenance

The routing pipeline uses three auxiliary sets with distinct roles. None contains an evaluation-test item. Exact example identifiers and normalized-text hashes are included in the release manifest.

The partition design in Table 6 makes the compute-benefit audit an out-of-sample test of alignment rather than a reconstruction of its evaluation target. The DE learns from coarse complexity and Fast-Agent failure on $\mathcal{D}_{\text{cal}}$, while Medium/Hard counterfactual outcomes are produced later for the audit. Consequently, a high Benefit AUROC requires the inexpensive proxy to generalize from its training signals to the realized value of additional inference actions. The same partitioning also fixes the threshold and score-normalization statistics before final benchmark evaluation, so path allocation can be compared under a single deployed policy.

The empirical routing target is constructed on $\mathcal{D}_{\text{cal}} = \{(x_i, d_i^*)\}_{i=1}^{2000}$ from the complexity proxy and Fast-Agent correctness defined in Eq. 1. Medium/Hard path outcomes enter the later counterfactual audit, keeping the training signal and the realized-benefit evaluation target distinct.

Complexity proxy. Where benchmark metadata provide a direct complexity annotation, we use it. Otherwise we use a pre-specified structural proxy derived from the reference solution length. This quantity is computed only for calibration items and is never read from evaluation-test labels at deployment.

Fast-Agent failure signal. We execute the Fast Agent on $\mathcal{D}_{\text{cal}}$ and obtain a binary correctness indicator using the corresponding calibration labels. This produces the second term in Eq. 1.

Leakage control. All auxiliary corpora are deduplicated against the evaluation sets by exact normalized text and by a near-duplicate check before training. Benchmark name, split identifier, answer labels, and evaluation-only metadata are excluded from $\phi(x)$. The released partition manifest records source split, example ID, and hash for every auxiliary item.

E.2 Optimization Configuration

The estimator parameters $W \in \mathbb{R}^{1\times 24}$ and $b \in \mathbb{R}$ are optimized to minimize the Mean Squared Error (MSE) between the predicted amortized energy $d(x)$ and the empirical target d^*. We incorporated L_2 regularization to prevent overfitting to specific dataset keywords:

$$\mathcal{L}_{\text{DE}} = \frac{1}{N}\sum_{i=1}^{N}(d(x_i) - d_i^*)^2 + \lambda_{\text{reg}}\|W\|_2^2 \tag{22}$$

Training was performed using the Adam optimizer with a learning rate of 10^{-3}, a batch size of 32, and $\lambda_{\text{reg}} = 0.01$ for 50 epochs. The feature extraction logic ($\phi(x)$) is detailed in Appendix I.

Algorithm 1 Open-ODAR FEP Selector (Character-Level & Z-Score)

Require: Candidate set $\mathcal{Y} = \{y_1, \ldots, y_K\}$, query x, penalty weight $\lambda = 0.1$
Require: Model statistics μ_m, σ_m for each agent m (for Z-score alignment)
Ensure: Selected answer $\hat{y}$
// **Step 1: Compute Character-Normalized Free Energy**
for $i = 1$ to K **do**
  // Calculate Energy Density per Character (Eq. 10)
  $\ell_i \leftarrow -\frac{1}{|y_i|_{\text{char}}} \sum_{t=1}^{|y_i|_{\text{tok}}} \log p(y_{i,t} \mid y_{i,<t}, x)$
  // Calculate Uncertainty Penalty (Varentropy)
  $\sigma_i^2 \leftarrow \text{Var}[\log p(y_{i,t} \mid y_{i,<t}, x)]$
  // Raw Free Energy (Character-normalized)
  $\mathcal{F}_{\text{raw}}(y_i|x) \leftarrow \ell_i + \lambda \cdot \sigma_i^2$
  // Model-Specific Z-Score Alignment (Eq. 11)
  $\mathcal{F}(y_i|x) \leftarrow \frac{\mathcal{F}_{\text{raw}}(y_i|x) - \mu_m}{\sigma_m}$
end for
// **Step 2: Select candidate with minimum standardized energy**
$\hat{y} \leftarrow \arg\min_{y_i \in \mathcal{Y}} \mathcal{F}(y_i|x)$
Return $\hat{y}$

Algorithm 2 Active Inference for Adaptive Routing

Require: Query x, Difficulty Estimator $\text{DE}(\cdot)$, thresholds $\tau_1 = 0.3$, $\tau_2 = 0.7$
Ensure: Selected policy π^* and final answer $\hat{y}$
// **Step 1: Estimate difficulty (approximate EFE)**
$d \leftarrow \text{DE}(x)$
// **Step 2: Select policy based on difficulty**
if $d < \tau_1$ **then**
  $\pi^* \leftarrow \pi_{\text{S}}$ // Simple Path: low epistemic value
else if $d < \tau_2$ **then**
  $\pi^* \leftarrow \pi_{\text{M}}$ // Medium Path: moderate epistemic value
else
  $\pi^* \leftarrow \pi_{\text{H}}$ // Hard Path: high epistemic value
end if
// **Step 3: Execute selected policy**
if $\pi^* = \pi_{\text{S}}$ **then**
  $\hat{y} \leftarrow m_\theta(x)$ // Fast Agent only
else if $\pi^* = \pi_{\text{M}}$ **then**
  $y_\theta \leftarrow m_\theta(x), y_\gamma \leftarrow m_\gamma(x, y_\theta)$ // Slow Agent verifies
  $\hat{y} \leftarrow \text{SELECT}(\{y_\theta, y_\gamma\}, x)$ // backend-specific selector
else
  $y_\theta \leftarrow m_\theta(x)$
  $\mathcal{Y}_\gamma \leftarrow \{m_\gamma(x)_i : i = 1, \ldots, n\}$ // Best-of-N ($n = 5$)
  $\hat{y} \leftarrow \text{SELECT}(\{y_\theta\} \cup \mathcal{Y}_\gamma, x)$ // backend-specific selector
end if
return $\pi^*, \hat{y}$

F Architectural Analogy

ODAR's Fast/Slow organization draws on dual-process and conditional-computation views at the level of system design: inexpensive computation handles routine cases, while a control signal conditionally recruits more expensive processing. The implemented mechanism is specified by the routing target in Eq. 1, the path rule in Eq. 3, and the backend-specific selection procedures; θ and γ denote the two agent roles.

G Extended Comparison with Concurrent Routing Frameworks

To situate ODAR within the rapidly evolving landscape of adaptive inference, we provide a qualitative comparison with recent concurrent works in Table 7.

Table 6: **Data partitions used by the router.** Auxiliary data come from non-test training/development partitions, with exact-text and normalized-text deduplication against all 23 evaluation test sets.

Partition	Size	Purpose	Relationship
$\mathcal{D}_{\text{aux}}$	50,000	8-way domain-classifier training	Non-test train/dev only
$\mathcal{D}_{\text{vocab}}$	10,000	TF–IDF vocabulary construction	Balanced subset of $\mathcal{D}_{\text{aux}}$
$\mathcal{D}_{\text{cal}}$	2,000	DE supervision, threshold calibration, open-weight score normalization	Held out from $\mathcal{D}_{\text{aux}}$ and all test se
$\mathcal{D}_{\text{test}}$	23 benchmarks	Final evaluation only	Never used for feature constructio

Table 7: **Qualitative comparison of adaptive routing frameworks.** ODAR combines lightweight difficulty estimation, training-free fusion, and routing across reasoning depth.

Framework	Core Mechanism	Router Training	Inference Overhead	Primary Goal	Reasoning Focus
HierRouter Gupta et al. (2025)	Reinforcement Learning (PPO)	High (Policy Training)	Medium	General Efficiency	Partial
RTR Pan et al. (2025)	Supervised Regression	Medium (Scalar Predictor)	Low	Strategy Selection	Yes
MoMA Guo et al. (2025)	Intent Classification	Medium (Capability Profiling)	Low	General Orchestration	No
CP-Router Su et al. (2025)	Conformal Prediction	Low (Calibration Only)	Medium (Sampling)	Uncertainty/Cost	Yes
InferenceDynamics Shi et al. (2025)	Capability Knowledge Graph	Medium (Profiling)	Low	Knowledge Routing	No
ODAR (Ours)	**Adaptive routing + backend-compatible selection**	**Low (Lightweight DE)**	**Low**	**Test-Time Scaling**	**Strong (depth-adaptive)**

Table 7 places ODAR at the intersection of routing depth, low-overhead control, and backend-compatible selection. Its distinguishing evaluation target is realized counterfactual compute benefit, complemented by a fixed-mixture permutation test that measures the value of query–path correspondence.

Method Comparison. ODAR combines a lightweight Difficulty Estimator with routing across *reasoning depths* and backend-compatible candidate selection. The open-weight configuration instantiates the likelihood-based FEP selector; the proprietary API configuration uses likelihood-free verification.

H Design-to-Implementation Mapping

The deployed system combines three separable design choices: the EFE-inspired proxy supplies inexpensive routing supervision, the Simple/Medium/Hard paths implement discrete compute allocation, and candidate selection is backend-specific—likelihood-free verification for proprietary APIs and the FEP-inspired selector when Open-ODAR exposes token likelihoods. This separation supports independent evaluation of routing and selection.

I Feature Extraction for Difficulty Estimation

To train the Difficulty Estimator, we extract a comprehensive feature vector $\phi(x) \in \mathbb{R}^{24}$ encoding lexical, structural, and semantic properties of query x. This section provides complete specifications for reproducibility.

I.1 Feature Specification

Table 8 presents the complete feature specification with definitions and importance scores derived from permutation importance analysis on the validation set.

Table 8 shows that the DE is intentionally built from inexpensive, inspectable cues rather than a second heavy reasoning model. The feature set mixes lexical scale, syntactic structure, query type, and coarse domain probabilities, which allows the router to represent several sources of reasoning demand without equating any single cue with compute benefit. This design matters to the central claim because the DE must remain cheap enough for selective test-time scaling to save compute overall. The later feature ablations test whether this composite proxy, rather than input length or a single semantic cue, is what supports alignment with realized benefit.

Table 8: Complete feature specification for the Difficulty Estimator. Importance scores represent mean decrease in Pearson r when feature is permuted, averaged over 10 runs.

ID	Feature	Definition	Imp.
Lexical Features ($k_1 = 8$)			
f_1	Character length	Total number of characters in query	0.012
f_2	Word count	Number of whitespace-separated tokens	0.018
f_3	Token count	BPE tokens via `tiktoken` (cl100k_base)	0.024
f_4	Avg. word length	Mean character count per word	0.008
f_5	Max word length	Maximum character count across words	0.006
f_6	Flesch-Kincaid	Readability grade level, clamped to $[0, 20]$	0.015
f_7	Numeric indicator	Binary: contains digits	0.031
f_8	Math symbol indicator	Binary: contains symbols from $\mathcal{M}$	0.042
Structural Features ($k_2 = 6$)			
f_9	Sentence count	Via spaCy sentence segmentation	0.019
f_{10}	Clause count	SBAR/SBARQ nodes in constituency parse	0.027
f_{11}	Question marks	Count of "?" characters	0.009
f_{12}	Image indicator	Binary: multimodal input present	0.038
f_{13}	Parse tree depth	Maximum depth in dependency parse	0.033
f_{14}	Logical connectives	Count of terms from $\mathcal{L}$	0.022
Semantic Features ($k_3 = 10$)			
f_{15}	TF-IDF (math)	Aggregated TF-IDF for math vocabulary	0.048
f_{16}	TF-IDF (code)	Aggregated TF-IDF for coding vocabulary	0.041
f_{17}	TF-IDF (logic)	Aggregated TF-IDF for logic vocabulary	0.035
f_{18}	Difficulty density	Fraction of words matching $\mathcal{H}$	0.052
f_{19}	Type: multiple-choice	Binary: enumerated answer options present	0.029
f_{20}	Type: open-ended	Binary: open response format	0.025
f_{21}	Type: proof	Binary: proof/derivation requested	0.044
f_{22}	Domain: math	Softmax probability from 8-way classifier	0.056
f_{23}	Domain: code	Softmax probability from 8-way classifier	0.047
f_{24}	Domain: reasoning	Softmax probability from 8-way classifier	0.039

I.2 Lexical Features

Length Statistics (features f1–f5). We compute character count, word count, and token count using the tiktoken library with the cl100k encoding to ensure consistency with GPT-5.1 tokenization. Average and maximum word lengths capture lexical complexity.

Readability (feature f6). The Flesch-Kincaid grade level is computed via the textstat library (v0.7.3). For queries containing mathematical notation, we replace LaTeX expressions with placeholder tokens before syllable counting. Scores are clamped to $[0, 20]$.

Mathematical Content (features f7–f8). We define the mathematical symbol set as:

$$\begin{aligned}\mathcal{M} = \{&+, -, \times, \div, =, \neq, <, >, \\ &\leq, \geq, \sum, \prod, \int, \partial, \nabla, \infty\} \\ &\cup \{\text{LaTeX commands}\}.\end{aligned} \tag{23}$$

Feature f_8 indicates whether any symbol from $\mathcal{M}$ appears in the query.

I.3 Structural Features

Syntactic Complexity (features f9, f10, f13). We employ spaCy for sentence segmentation and dependency parsing. Clause counting uses the Berkeley Neural Parser to identify subordinate clause nodes (SBAR, SBARQ) in constituency parses. For mathematical content, we apply preprocessing that replaces LaTeX blocks with placeholder tokens before parsing, improving parser accuracy from 67% to 94% on math-heavy queries.

Logical Structure (feature f14). We define the logical connective set as:

$$\begin{aligned}\mathcal{L} = \{&\textit{if}, \textit{then}, \textit{therefore}, \textit{hence},\\ &\textit{thus}, \textit{because}, \textit{since}, \textit{implies},\\ &\textit{iff}, \textit{for all}, \textit{there exists}, \textit{such that}\}.\end{aligned} \tag{24}$$

Feature f_{14} counts case-insensitive occurrences of terms from $\mathcal{L}$.

I.4 Semantic Features

Domain-Specific TF-IDF (features f15–f17). We construct domain vocabularies from $\mathcal{D}_{\text{vocab}}$, the 10,000-query balanced subset of the non-test auxiliary corpus defined in Table 6. For each domain $d \in \{\text{math}, \text{code}, \text{logic}\}$, we extract the top-100 discriminative terms by TF-IDF, excluding stopwords and high-frequency terms (>50% of queries). Representative terms include:

The domain vocabularies in Table 9 contribute coarse semantic context to the DE but do not determine the routing decision on their own. Their role is to distinguish structurally similar queries whose likely reasoning demands differ by domain, while the remaining features encode length, syntax, logical structure, and question type. This combination is important for instance–compute alignment: domain terminology can be informative, but the "keyword trap" analysis later shows why terminology alone is an unreliable basis for escalating compute. The router therefore uses domain vocabulary as one component of a broader allocation signal.

The query-level score is computed as $f_d(x) = \sum_{t \in x \cap V_d} \text{tfidf}(t)$.

Difficulty Markers (feature f18). We curate a set $\mathcal{H}$ of 47 high-difficulty markers based on manual analysis of 500 challenging problems. The set includes cognitive demand indicators such as *prove*, *derive*, *generalize*, *optimize*, *compare and contrast*, *analyze*, and *synthesize*. Difficulty density is computed as:

$$f_{18} = \frac{|\{h \in \mathcal{H} : h \subseteq x\}|}{|x|_{\text{words}}} \tag{25}$$

Question Type (features f19–f21). We classify question format using pattern matching: multiple-choice queries exhibit enumerated options (e.g., "(A)", "(B)"); open-ended queries begin with interrogative words without option structure; proof requests contain explicit derivation keywords.

Domain Classification (features f22–f24). We train a DistilBERT-based 8-way classifier (66M parameters) on $\mathcal{D}_{\text{aux}}$ (50,000 non-test train/dev queries), achieving 89.2% held-out auxiliary-set accuracy. The eight classes are Mathematics, Coding, Logic, Commonsense, Knowledge, Multimodal, Language, and Other. Features f_{22}–f_{24} use the three designated domain probabilities. No evaluation-test item is used to train this classifier.

I.5 Feature Importance Analysis

Table 10 presents the top-10 features ranked by permutation importance.

Table 10 indicates that the strongest individual signals are semantic or task-structural, yet no single feature dominates the estimator. The largest importance value is 0.056 and the top ten span domain probability, difficulty density, proof format, mathematical notation, an image indicator, and TF–IDF cues. This distributed importance helps explain why simple length-based routing performs poorly in the counterfactual audit: surface scale captures only one part of the information used by the DE. The result supports a composite proxy for allocating compute while leaving realized benefit, rather than feature importance itself, as the final evaluation target.

The analysis shows that semantic features dominate difficulty prediction, with domain classifier outputs (f_{22}–f_{24}) and difficulty density (f_{18}) providing the strongest signal. Lexical and structural features add complementary information on out-of-distribution queries.

I.6 Ablation Study

Table 11 presents results when removing each feature category.

Table 9: Representative vocabulary by domain used in lexical feature extraction.

Domain	**Representative Vocabulary** ($\lvert V_d \rvert$)
Mathematics (87)	*theorem, derivative, integral, matrix, eigenvalue, convergence, polynomial*
Coding (92)	*algorithm, function, complexity, implement, recursive, runtime, optimize*
Logic (73)	*valid, satisfiable, contradiction, premise, inference, deduction, entailment*

Table 10: Top-10 features by permutation importance. Semantic features contribute 56% of total importance.

Rank	**Feature**	**Category**	**Importance**
1	Domain: math (f_{22})	Semantic	0.056
2	Difficulty density (f_{18})	Semantic	0.052
3	TF-IDF math (f_{15})	Semantic	0.048
4	Domain: code (f_{23})	Semantic	0.047
5	Type: proof (f_{21})	Semantic	0.044
6	Math symbol (f_8)	Lexical	0.042
7	TF-IDF code (f_{16})	Semantic	0.041
8	Domain: reasoning (f_{24})	Semantic	0.039
9	Image indicator (f_{12})	Structural	0.038
10	TF-IDF logic (f_{17})	Semantic	0.035

The category ablation in Table 11 gives a mechanism-level reading of the DE rather than a ranking of features in isolation. Removing semantic features causes the largest drop in Pearson correlation, routing accuracy, and Audit Score, but removing lexical or structural features also reduces downstream performance. The full model therefore benefits from complementary cues: semantic features identify domain and task type, while lexical and structural features temper purely semantic judgments with observable complexity. This complementarity is consistent with the paper's broader result that realized compute benefit is related to, but not reducible to, conventional notions of static difficulty.

Removing semantic features reduces routing accuracy by 13.1 percentage points and end-to-end task accuracy by 4.2 points, confirming their critical role. The three-tier design provides necessary redundancy: lexical features capture surface complexity, structural features encode syntactic depth, and semantic features identify domain-specific difficulty patterns.

I.7 Extended Ablation Studies and Systematic Analysis

To further disentangle the contributions of the learned Difficulty Estimator (DE) from fixed orchestration heuristics and compute-time scaling, we present a series of rigorous ablation studies.

I.7.1 Formal Definition and Impact of the Rule-based Orchestration Layer

ODAR includes a lightweight rule-based dispatch/orchestration layer (ER/MR/SS) implemented with fixed heuristics. **Expert Router (ER)** and **Model Router (MR)** identify modality/domain cues to assign expert types (e) and select base-model identifiers (m). The **Strategy Selector (SS)** maps the DE-produced score d and features to a reasoning path and call budget $c \in \{1, 2, 6\}$.

Priority and Conflict Resolution. Within the orchestration heuristics, conflicts are resolved via a fixed priority hierarchy as shown in Figure 1: **Modality** $\rightarrow$ **Keywords** $\rightarrow$ **Question Type** $\rightarrow$ **Fallback**. When multiple cues within the same level fire, ties are broken by keyword-count voting and a fixed domain priority list. This ensures a deterministic (e, m) configuration for each query. The difficulty score d remains a product of the learned DE, and path-gating depends on thresholds (τ_1, τ_2).

Table 11: Feature category ablation on the dedicated DE audit split. The Audit Score refers to this DE ablation split and is distinct from the 23-benchmark descriptive macro score.

Configuration	Pearson r	Routing Acc.	Audit Score
Full model	0.79	87.3%	83.5%
w/o lexical features	0.76	85.1%	82.8%
w/o structural features	0.74	83.8%	82.1%
w/o semantic features	0.61	74.2%	79.3%
Random baseline	0.00	33.3%	75.4%

I.8 Matched Routing Analysis

Matched-routing controls are consolidated in Appendix X. The principal control preserves the 41/35/24% Simple/Medium/Hard proportions and 2.55 average calls while permuting query-to-path assignments within each benchmark; this removes instance–compute alignment without changing the global call mixture.

I.9 Implementation

Feature extraction exceeds 1,000 queries per second on a single CPU core. Key dependencies include tiktoken (v0.5.1), textstat (v0.7.3), spacy (v3.5.0), and transformers (v4.30.0). Complete extraction code is provided in the supplementary materials.

J Error Analysis and Taxonomy

To better understand ODAR's failure modes, we performed a fine-grained manual analysis on a subset of incorrect predictions. Because long-horizon reasoning traces are labor-intensive to inspect, we construct an error pool consisting of all *incorrect* model outputs across the 23 evaluation datasets, and then uniformly sample $N{=}100$ error instances from this pool.[1] Each sampled instance is annotated by two annotators using a shared guideline; disagreements are resolved through discussion to obtain a final label.[2]

Table 12 summarizes the resulting taxonomy and its empirical distribution. In particular, E2 (Expert Routing) corresponds to failures of the *rule-based dispatch layer* (ER/MR) used in our system pipeline: the dispatch assigns an inappropriate expert tag or modality/domain label (e.g., treating a code-generation query as a math-proof query), which then leads to a suboptimal model/prompt/strategy configuration. E4 (Candidate Selection) captures cases where a correct candidate exists in the candidate set (oracle-hit), but the active backend-specific selector returns an incorrect candidate.

Table 12 separates failures of *allocation and conversion* from failures of the underlying generators. System-induced errors account for 34% of the sampled failures: 8% arise at difficulty estimation, 5% at expert dispatch, 12% at verification, and 9% at candidate selection. The remaining 66% occur when every available path lacks the required knowledge or valid reasoning, so additional routing accuracy alone cannot recover them within the current action set. This decomposition identifies where instance–compute alignment still has headroom and where improvement instead requires stronger base models or a richer set of inference actions.

J.1 Analysis of Failure Modes

Intrinsic Model Errors (E5 & E6) constitute 66% of observed failures, particularly on HLE and GPQA. These cases arise when all generated paths lack the required knowledge or contain reasoning errors, making base-model capability the dominant error source in this subset.

[1] We treat an instance as an error if the final returned answer $\hat{y}$ is judged incorrect by the dataset's official metric (exact match or task-specific evaluator).

[2] The taxonomy is designed to separate *system-level* failures introduced by routing, verification, and selection (E1–E4) from *underlying model capability* limitations (E5–E6).

Table 12: Error Taxonomy and Distribution ($N = 100$). Errors are categorized to distinguish between system-level routing failures (E1–E4) and underlying model capability limitations (E5–E6).

Error Type	Detailed Description	Count	%
System & Routing Failures (34%)			
E1: Difficulty Est.	Mis-classification of hard queries as simple, routing to insufficient compute.	8	8%
E2: Expert Routing	Dispatch assigns an inappropriate expert/domain tag (e.g., Math vs. Code), leading to suboptimal model/prompt/strategy.	5	5%
E3: Verification Fail.	**False Acceptance**: Slow Agent incorrectly verifies a wrong Fast answer.	12	12%
E4: Candidate Selection	**Missed Oracle**: a correct candidate exists in $\mathcal{C}$, but the active selector returns an incorrect candidate.	9	9%
Model Capability Limitations (66%)			
E5: Knowledge Gap	All agents lack the required domain knowledge (e.g., specific facts).	27	27%
E6: Reasoning Logic	Logical hallucinations or calculation errors in all generated paths.	39	39%

Verification Failures (12%) represent the most significant challenge for the Medium Path, where the Slow Agent occasionally succumbs to the same conceptual pitfalls as the Fast Agent, leading to false acceptance.

Candidate-selection errors (9%) are counted only when a correct candidate exists in the pool but the active selector does not return it.

Difficulty Estimation and **Expert Routing** account for 8% and 5% of observed failures, respectively.

K FEP Fusion Case Study

Table 13 illustrates the FEP selector on a **local log-probability-capable Open-ODAR candidate pool** for a MATH problem.

Table 13 illustrates the second stage of ODAR after a Hard-Path budget has already been allocated. Several candidates reach the same final answer, but the selector distinguishes them through both average energy density and temporal likelihood variability; the logically incorrect sign-error candidate is unfavorable on both terms. The selected candidate combines a low energy term with the smallest listed risk term, showing how candidate quality can be separated even when answer-level agreement is high. This supports the paper's hierarchy of mechanisms: routing decides *where* to create extra candidates, and FEP decides *which* candidate best converts that extra compute into the returned answer.

The case isolates the role of candidate selection after compute has already been allocated to the Hard Path. Four Slow-Agent candidates reach the same final answer, but their likelihood profiles differ; the lowest-energy candidate combines the smallest energy-density term with the smallest temporal-variance penalty, while the sign-error candidate is penalized on both. The example therefore illustrates why ODAR separates *where to spend compute* from *which generated candidate to trust*: routing creates a diverse candidate set, and the selector converts that allocated compute into a final decision.

L Fusion Method Ablation

We evaluate FEP against heuristic selectors on a **fixed, local, log-probability-capable candidate pool**. Holding the candidate pool fixed isolates selection quality from routing and generation, so this experiment measures the final stage of the allocation pipeline.

Table 14 quantifies the selection-stage contribution while holding routing and candidate generation fixed. FEP improves the average selection accuracy from 89.2 for average log-probability to 90.4,

Table 13: **Open-ODAR FEP selector case study.** Values are character-normalized energies computed from local token logits in the likelihood-capable Open-ODAR stack.

ID	Source	Reasoning Quality	Answer	Energy (ℓ_i)	Risk (σ_i^2)	$\mathcal{F}_{\text{raw}}$
y_θ	Fast	Intuitive Guess	$x = 2$	0.18	0.025	0.1825
$y_\gamma^{(1)}$	Slow	Standard Derivation	$x = 2$	0.09	0.012	0.0912
$y_\gamma^{(2)}$	Slow	**Rigorous Calculus**	$\mathbf{x = 2}$	**0.06**	**0.008**	**0.0608***
$y_\gamma^{(3)}$	Slow	*Sign Error (Logic)*	$x = 1^\times$	0.45	0.120	0.4620
$y_\gamma^{(4)}$	Slow	Vertex Formula	$x = 2$	0.08	0.015	0.0815
$y_\gamma^{(5)}$	Slow	Circular Reasoning	$x = 2$	0.21	0.032	0.2132

* Selected path. Note: $\mathcal{F}_{\text{raw}} = \ell_i + \lambda \cdot \sigma_i^2$ (Eq. 10), where ℓ_i is the negative character-normalized log-probability (Energy Density) and $\lambda = 0.1$. The correct path ($y_\gamma^{(2)}$) minimizes this energy.

Table 14: Open-weight selector ablation on a fixed candidate pool (selection accuracy %), isolating the selection stage from routing and generation.

Fusion Method	**MATH**	**BBH**	**TruthfulQA**	**Avg.**
Max Confidence	94.5	89.2	79.1	87.6
Majority Voting (SC)	95.1	90.4	80.3	88.6
Average Log-Prob	95.8	90.7	81.2	89.2
FEP Fusion (Ours)	**96.7**	**91.7**	**82.7**	**90.4**
Gain vs. Best Baseline	**+0.9**	**+1.0**	**+1.5**	**+1.2**

with gains on all three reported tasks. Because the candidate pool is unchanged, the difference is attributable to how evidence from generated candidates is combined rather than to a larger test-time budget. This provides a supporting explanation for ODAR's end-to-end gains, while the matched-allocation and factorial experiments remain the direct tests of instance–compute alignment.

Across the fixed candidate pools, FEP improves selection accuracy by 0.9–1.5 points over the strongest listed heuristic, with the largest gain on TruthfulQA. Because routing and generation are held constant, this gain is attributable to candidate choice rather than additional compute. In the paper's contribution hierarchy, this is a supporting mechanism: the selector improves conversion of an allocated budget into a correct answer, while the matched-allocation and factorial experiments test the primary question of instance–compute alignment.

M Hyperparameters and Configuration

Table 15 summarizes the comprehensive hyperparameter configuration for full reproducibility.

Table 15 fixes the action space that underlies every allocation result: the DE dimension and optimizer define the controller, (τ_1, τ_2) define the three routing regions, and $n = 5$ defines the Hard-Path depth. Using task-invariant settings makes the path decision comparable across benchmarks and prevents per-dataset retuning from becoming an alternative explanation for the observed gains. The fixed thresholds are especially important for the transfer study, where the router is moved to unseen model pairs without refitting. In this sense, the table specifies the intervention whose alignment is tested throughout the paper.

M.1 Threshold Calibration Details

The thresholds τ_1 and τ_2 were set via grid search over $\{0.2, 0.3, 0.4\} \times \{0.6, 0.7, 0.8\}$ on a calibration set of 2,000 annotated problems sampled proportionally from 8 core benchmarks (MATH, ARC-

Table 15: ODAR Hyperparameters and Configuration

Component	Parameters	Values
Difficulty Estimator	Feature Dimension (k)	24
	L_2 Regularization (λ_{reg})	0.01
	Training Epochs	50
	Learning Rate	10^{-3}
	Batch Size	32
	Optimizer	Adam
Fast Agent	Base Model	GPT-5.1
	Temperature (T_θ)	0.2
	Max Tokens	512
	Top-p	0.9
	Average Latency	1.8s
Slow Agent	Base Model	Claude-4.5 Sonnet
	Temperature (T_γ)	0.3
	Max Tokens	1024
	Top-p	0.95
	Avg. Latency (Single)	6.4s
	Avg. Latency (Best-of-N)	32.0s (Seq.)
Path Selection	Simple Threshold (τ_1)	0.3
	Hard Threshold (τ_2)	0.7
	Best-of-N Samples (n)	5
	FEP Complexity Penalty (λ)	0.1

Challenge, BoolQ, MMLU-Pro, BBH, TruthfulQA, IFEval, HLE). The optimization objective was:

$$(\tau_1^*, \tau_2^*) = \arg\max_{\tau_1,\tau_2} \frac{\text{Task Score}(\tau_1, \tau_2)}{\text{Avg. Calls}(\tau_1, \tau_2)} \tag{26}$$

The optimal configuration $(\tau_1 = 0.3, \tau_2 = 0.7)$ achieves the path distribution shown in Table 16.

Table 16 converts the routing policy into an exact call-accounting identity: the 41/35/24% path mixture yields $0.41 + 0.70 + 1.44 = 2.55$ calls per query. This identity is more than an efficiency summary because it supplies the matched budget for the permutation experiment. By preserving these path proportions and the same average calls while shuffling which instance receives which path, the paper can attribute the resulting performance drop to query–path correspondence rather than to a different amount of computation. The table therefore connects the deployed policy directly to the central mechanism test.

N Dual-Agent Prompt Templates

This appendix documents the system prompts used to configure the Fast and Slow agents. These prompts were designed based on two key principles: (1) **cognitive specialization**—inducing distinct reasoning modes analogous to System 1/System 2 processing, and (2) **output standardization**—ensuring consistent answer formatting for downstream selection.

N.1 Fast Agent: Rapid Inference Mode

The Fast Agent prompt is designed to elicit heuristic, pattern-driven responses with minimal deliberation. Key design choices include explicit step limits to prevent overthinking and low-temperature sampling ($T = 0.2$) to encourage deterministic outputs.

Table 17 operationalizes the inexpensive action in the routing menu. The step limit and low-deliberation instruction make the Fast Agent a genuinely low-cost proposal mechanism, so a Simple decision has a clear computational meaning rather than being a label attached to the same reasoning procedure. This distinction is necessary for realized compute benefit to be measurable: Medium and Hard paths must add qualitatively and quantitatively different inference work. The prompt ablation

Table 16: Empirical path distribution and model-call accounting. Call count and normalized cost are reported as distinct quantities throughout the paper.

Path	Difficulty Range	Distribution	Calls/query
Simple (π_{S})	$d < 0.3$	41%	1
Medium (π_{M})	$0.3 \leq d < 0.7$	35%	2
Hard (π_{H})	$d \geq 0.7$	24%	6
Weighted Average		100%	**2.55**

Table 17: Fast Agent System Prompt

`[SYSTEM]`

You are a rapid reasoning assistant optimized for speed and pattern recognition.

Behavioral Constraints:

1. Limit reasoning to $\leq$3 intermediate steps
2. Prioritize retrieval and pattern matching over deliberation
3. Avoid excessive justification or self-correction

Output Format: `Answer: <response>`

later verifies that the Fast/Slow specialization contributes to performance while the factorial design separates that contribution from routing itself.

Design Rationale. The step limit ($\leq$3) was empirically calibrated: fewer steps degraded accuracy on multi-hop queries, while more steps increased latency without accuracy gains (see Table 20). The explicit instruction to "avoid self-correction" prevents the oscillating confidence patterns that increase Varentropy and degrade FEP scores.

N.2 Slow Agent: Verification Mode

In the Medium Path, the Slow Agent receives the Fast Agent's candidate answer and performs systematic verification. This prompt induces a structured critique-and-correct workflow.

Table 18 defines the Medium Path as targeted verification rather than independent repeated sampling. The Slow Agent receives the Fast proposal, checks its reasoning, and corrects it when necessary, so the second call has a specific marginal role: it attempts to recover cases where a cheap first pass is insufficient. This makes Medium-path utility interpretable in the counterfactual audit and gives the router an intermediate action between one-shot inference and six-call exploration. The design therefore turns "more compute" into a structured verification action whose realized value can be measured per instance.

Design Rationale. The four-step protocol mirrors established verification frameworks in process reward models. Explicit enumeration of steps (Restate $\rightarrow$ Validate $\rightarrow$ Judge $\rightarrow$ Correct) reduces the probability of superficial "looks good" acceptance, which constitutes 12% of system errors (Table 12).

N.3 Slow Agent: Independent Reasoning Mode

In the Hard Path, the Slow Agent generates $n = 5$ independent solutions without access to the Fast Agent's output, maximizing candidate diversity for downstream selection.

Table 19 defines the Hard Path as independent, self-checked exploration, creating diversity rather than repeatedly verifying the same Fast response. Independent generation increases the chance that at least one candidate reaches a useful reasoning trajectory, while self-verification improves each candidate before selection. This is the most expensive action in ODAR's discrete compute menu and therefore the action whose assignment must be most selective. Its prompt design supports the central resource-allocation view: the router escalates to broad exploration only for instances predicted to justify that budget.

Table 18: Slow Agent Verification Prompt (Medium Path)

System Instruction

You are a careful verification assistant. Your task is to rigorously validate a proposed solution.

Verification Protocol:

1. *Restate*: Summarize the proposed answer and key reasoning steps
2. *Validate*: Check each logical step for correctness
3. *Judge*: Output CORRECT or INCORRECT
4. *Correct*: If incorrect, provide the corrected solution with justification

Input: Proposed Answer: {fast_agent_output}

Output Format: Judgment: ⟨CORRECT|INCORRECT⟩ followed by Answer: ⟨response⟩

Table 19: Slow Agent Independent Reasoning Prompt (Hard Path)

`[SYSTEM]`

You are a thorough reasoning assistant. Solve the problem using rigorous, step-by-step analysis.

Solution Protocol:

1. *Analyze*: Identify key concepts, constraints, and edge cases
2. *Strategize*: Select an appropriate solution approach
3. *Execute*: Implement the solution with explicit intermediate steps
4. *Verify*: Perform self-check before finalizing

Output Format: `Answer: <response>`

Design Rationale. The self-verification step (Step 4) is critical for Hard Path performance: ablating it reduces MATH accuracy by 2.1% (Table 20). Higher temperature ($T = 0.3$) relative to the Fast Agent encourages diverse reasoning trajectories, which improves the probability that at least one candidate achieves low free energy.

N.4 Prompt Ablation Study

To validate the design choices embedded in our prompts, we conduct controlled ablations on MATH and BBH (Table 20).

Table 20 shows that the Fast/Slow roles are substantive components of the action definitions. Removing the Slow verification protocol or self-verification step produces larger drops than removing either Fast-Agent instruction, and the generic prompt loses 2.9 points on MATH and 2.5 on BBH. These effects explain part of the value of heterogeneous inference once compute has been allocated. At the same time, prompt structure cannot explain the matched-permutation result because the same action definitions and path proportions are preserved when query–path correspondence is destroyed; that comparison isolates the additional value of instance–compute alignment.

Key Findings. (1) The structured verification protocol contributes **+1.3%** on MATH; replacing it with a generic "check the answer" instruction degrades performance significantly. (2) The self-verification step in Hard Path prompts is essential (**+2.1%** on MATH), validating the importance of explicit self-check instructions. (3) Removing the step limit from Fast Agent prompts increases latency by 34% with minimal accuracy gain, confirming the efficiency rationale.

O Implementation Details

O.1 Generation Backends

The closed-API evaluation uses GPT-5.1 as the Fast Agent and Claude Sonnet 4.5 as the Slow Agent, with likelihood-free verification for candidate selection. Token-level FEP is instantiated in the local Open-ODAR stack, whose generators expose the sampled-token log-probabilities required by Eq. 8. This backend split keeps the routing study common across configurations while matching each selector to the information available at inference time.

Table 20: Prompt Design Ablation. Each row removes one design element from the full prompt.

Configuration	MATH	BBH
Full Prompt (Ours)	**98.2**	**95.4**
Fast Agent Ablations		
w/o step limit ($\leq$3)	97.4 (-0.8)	94.9 (-0.5)
w/o "avoid self-correction"	97.6 (-0.6)	94.7 (-0.7)
Slow Agent Ablations		
w/o verification protocol	96.9 (-1.3)	94.2 (-1.2)
w/o self-verification step	96.1 (-2.1)	93.8 (-1.6)
Generic prompt (no structure)	95.3 (-2.9)	92.9 (-2.5)

The exact token-level FEP implementation is evaluated in **Open-ODAR**, using local Llama 4 Scout and DeepSeek V3.2 deployments. Local inference exposes the next-token logits for every sampled token, so both the normalized surprisal term and the temporal-variance term in Eq. 8 are directly computable.

O.2 Selector Backends and Likelihood Extraction

Closed-API selector. For the Medium path, the Slow Agent verifies or refines the Fast response and returns the verified answer. For the Hard path, the system normalizes final answers from the Slow candidates, applies answer-level consensus, and uses a deterministic Slow-Agent verification pass to resolve ties. The broad GPT-5.1/Claude benchmark tables therefore evaluate adaptive routing with likelihood-free verification; token-level FEP is evaluated in Open-ODAR.

Open-ODAR FEP selector. For each locally generated candidate y, we record the log-probability of the sampled token at every decoding step from the local logits. BOS/EOS/padding and formatting-only control tokens are excluded. We then compute the character-normalized energy and temporal-surprisal variance in Eq. 8, followed by model-specific calibration in Eq. 9. No API compatibility layer is involved.

Calibration. The 2,000-item $\mathcal{D}_{\text{cal}}$ partition in Table 6 is used to estimate (μ_m, σ_m) for each *open-weight* scorer:

$$\mu_m = \mathbb{E}_{x \in \mathcal{D}_{\text{cal}}}[\mathcal{F}_{\text{raw}}(m(x))], \qquad \sigma_m = \sqrt{\text{Var}_{x \in \mathcal{D}_{\text{cal}}}[\mathcal{F}_{\text{raw}}(m(x))]}. \tag{27}$$

At inference time,

$$\mathcal{F}(y) = \frac{\mathcal{F}_{\text{raw}}(y) - \mu_m}{\sigma_m}. \tag{28}$$

This empirical cross-model calibration transform places heterogeneous raw scores on a comparable standardized scale while preserving each model's within-model ordering.

O.3 Computational Efficiency Analysis

We distinguish three accounting quantities: (i) **Avg. Calls**, the number of model invocations; (ii) **normalized cost**, measured directly from run logs and normalized to the designated reference budget; and (iii) **API spend**, when dollar cost is reported. Avg. Calls follows directly from the path mixture and equals 2.55 for the 41/35/24% routing distribution, whereas normalized cost is read from the run logs. Table 21 reports the call-accounting identity used by the matched-allocation controls.

Table 21 decomposes the expected 2.55 calls into contributions from each route. Although Hard queries constitute 24% of the traffic, they account for 1.44 of the 2.55 expected calls, making correct Hard-Path assignment disproportionately important to the budget. The Simple path has the opposite role: its 41% share removes expensive reasoning from a large fraction of requests. This concentration of compute is exactly what the permutation test holds fixed, allowing the paper to ask whether the same budget is more useful when attached to the right instances.

Table 21: Breakdown of inference calls by routing path. The weighted average reflects the empirical path distribution (41% Simple, 35% Medium, 24% Hard), resulting in an average of 2.55 calls per query.

Path	Distribution	Calls per Query	Contribution
Simple ($d < 0.3$)	41%	1 (Fast)	0.41
Medium ($0.3 \leq d < 0.7$)	35%	2 (Fast + Slow)	0.70
Hard ($d \geq 0.7$)	24%	6 (Fast + 5×Slow)	1.44
Weighted Average	**100%**	**2.55**	**2.55**

Efficiency-Accuracy Tradeoff Analysis. Dataset difficulty correlates positively with average inference calls ($r = 0.73$, $p < 0.001$). Mathematics tasks receive higher call density, while Commonsense tasks receive fewer calls.

O.4 Latency Analysis and Deployment Considerations

Beyond cost, latency constitutes a critical deployment constraint. Table 22 presents the latency profile across routing paths.

Table 22 shows how the three actions differ in wall-clock consequences as well as call counts. The Medium path adds sequential verification, whereas the Hard path can parallelize its five Slow-Agent candidates, producing a different latency profile from a simple multiplication of per-call time. This matters for adaptive allocation because two policies with the same number of calls can have different deployment costs depending on where parallelizable work is placed. Reporting path-specific P50/P95 latency therefore extends instance–compute alignment from abstract call budgets to the user-facing cost of escalation.

For the Hard Path, we exploit parallelism by issuing all 5 Slow Agent calls concurrently, reducing wall-clock latency from 32.0s to approximately 12.4s (2.6× speedup). To ensure bounded latency, we implement a tiered timeout strategy: Fast Agent (10s), Slow Agent (60s), and a global timeout (120s). Timeout events are rare (<0.5%) with negligible accuracy impact.

O.5 Reproducibility

We document the complete experimental configuration and distinguish service aliases from immutable local checkpoints. The proprietary generation stack uses the API identifiers `gpt-5.1` and `claude-sonnet-4-5`; GPT-5.1 entered the OpenAI API on November 13, 2025[3], and Claude Sonnet 4.5 was released on September 29, 2025[4]. These provider-hosted identifiers are *not* described as immutable weight snapshots. Request timestamps, model identifiers, and request IDs are stored in the run manifest. Open-ODAR uses fixed local checkpoint hashes and is the exact reproducible configuration for token-level FEP experiments. For datasets larger than 5,000 examples, we predefine K=5 disjoint stratified folds (N=1,000 each); fold indices and evaluation scripts are released.

P Detailed Reproducibility Protocols

We document both the provider-hosted API configuration used for the broad routing evaluation and the exact **Open-ODAR** configuration using Llama 4 + DeepSeek V3.2. Provider service identifiers are recorded with request metadata, while the local Open-ODAR stack is pinned by checkpoint hashes for token-level FEP reproduction.

P.1 Inference Configuration

Table 23 separates provider-hosted service aliases from locally hashed checkpoints.

[3] `https://openai.com/index/gpt-5-1-for-developers/`

[4] `https://www.anthropic.com/news/claude-sonnet-4-5`

Table 22: Latency breakdown by routing path. P50 and P95 denote median and 95th percentile latencies.

Path	P50 Latency	P95 Latency	Parallelizable
Simple	1.8s	3.2s	N/A
Medium	8.2s	14.5s	No
Hard	12.4s	28.7s	Yes (5 Slow calls)
Overall	6.1s	22.3s	Partial

Table 23: Inference configuration. Provider-hosted service identifiers are recorded with request metadata, while Open-ODAR uses local checkpoint hashes for exact token-level FEP reproduction.

Parameter	Fast Agent (A_θ)	Slow Agent (A_γ)	OPEN-ODAR (Ref.)
Model ID	`gpt-5.1`	`claude-sonnet-4-5`	Llama-4-Scout / DeepSeek-V3.2
Deployment	OpenAI API	Anthropic API	vLLM (Local, FP8 Quant)
Public availability	Nov. 13, 2025 (API)	Sep. 29, 2025	Fixed local checkpoints
System Prompt	Versioned prompt files + released SHA-256 hashes		Versioned prompt files + SHA-256 hashes
Temperature	0.2 (Fixed)	0.3 (Fixed)	0.6 (Calibrated for Open Weights)
Top-p (Nucleus)	0.9	0.95	0.95
Stop Tokens	["\n\n", "Q:"]	["Wait", "User:"]	$<$ |eot_id| $>$ only
Seeds	{42, 123, 456} (Averaged over 3 runs)		42 (Deterministic CUDA)

Table 23 anchors the reported results to the actual generation interfaces used by each configuration. The provider-hosted stack establishes the broad routing evaluation, while fixed local checkpoints provide the likelihood access required for exact FEP reproduction. This distinction also strengthens the cross-model transfer test: the router is evaluated across genuinely different Fast/Slow pairs rather than being tied to a single pair's token statistics. The central routing claim is therefore evaluated at the level of allocation behavior, with backend-specific selection handled as a separate mechanism.

P.2 Likelihood Availability and FEP Reproduction

The proprietary and open-weight stacks use different selector interfaces because their likelihood access differs.

Claude API. The Claude public API/compatibility interface supplies the likelihood-free path used by the proprietary stack; Anthropic documents compatibility-layer `logprobs` as ignored and the returned field as empty[5]. Token-level FEP quantities are therefore computed from the local Open-ODAR logits described next.

Open-ODAR. For Llama 4 Scout and DeepSeek V3.2 running locally, the decoding backend records the log-probability of each sampled token directly from next-token logits. These values are used for the surprisal and temporal-variance terms. BOS, EOS, padding, and formatting-only tokens are filtered before aggregation, after which character normalization and calibration use Eqs. 8–9.

Reproduction scope. The 23-benchmark proprietary table evaluates adaptive routing with the likelihood-free verification selector. FEP-specific ablations, calibration analyses, and Algorithm 1 refer to the log-probability-capable Open-ODAR configuration. This separation makes the likelihood requirements explicit and removes any dependence on unavailable Claude token log-probabilities.

Q Dataset Details

Table 24 provides comprehensive details for all 23 evaluation benchmarks.

Table 24 supplies the task diversity needed to test whether instance–compute alignment is confined to one reasoning regime. The suite spans mathematics, commonsense, knowledge, multi-hop reasoning,

[5] `https://platform.claude.com/docs/en/cli-sdks-libraries/libraries/openai-sdk`

Table 24: Dataset details for 22 public benchmarks plus one custom 100-problem proof stress test. The custom suite uses a verifier-based Proof Score and is reported as a distinct evaluation channel.

Category	Dataset	Test Size	Metric	Reference
Mathematics	MATH	5,000	Accuracy	Hendrycks et al. (2021)
	GSM8K	1,319	Accuracy	Cobbe et al. (2021)
	Custom Proof Suite	**100**	**Proof Score**	**Released problem manifest**
	MathVista	1,000	Accuracy	Lu et al. (2023)
Commonsense	ARC-Challenge	1,172	Accuracy	Clark et al. (2018)
	OpenBookQA	500	Accuracy	Mihaylov et al. (2018)
	BoolQ	3,270	Accuracy	Clark et al. (2019)
	StrategyQA	2,290	Accuracy	Geva et al. (2021)
Knowledge QA	MMLU-Pro	12,032	Accuracy	Wang et al. (2024)
	GPQA	448	Accuracy	Rein et al. (2024)
Multi-Hop	HotpotQA	7,405	EM / F1	Yang et al. (2018)
	MuSiQue	2,417	EM / F1	Trivedi et al. (2022)
Multimodal	ScienceQA	4,241	Accuracy	Saikh et al. (2022)
	AOK-VQA	1,145	Accuracy	Schwenk et al. (2022)
	MMMU-Pro	1,500	Accuracy	Yue et al. (2025)
Advanced Cognition	BBH	6,511	Accuracy	Suzgun et al. (2023)
	BBEH	1,500	Accuracy	Kazemi et al. (2025)
	TruthfulQA	817	Accuracy	Lin et al. (2022)
	HLE	3,000	Accuracy	Phan et al. (2025)
Coding	SWE-bench	2,294	Pass@1	Jimenez et al. (2023)
	LIVEBENCH	1,000	Similarity	White et al. (2024)
Instruction	IFEval	541	Accuracy	Zhou et al. (2023)
Abstract	ARC-AGI-2	400	Accuracy	Chollet et al. (2025)

multimodality, advanced cognition, coding, instruction following, and abstract reasoning, creating large variation in both baseline performance and the usefulness of additional computation. Because the metrics differ, the paper's strongest conclusions come from per-task comparisons and the separate counterfactual audit rather than from treating the suite as a single homogeneous accuracy benchmark. The custom proof set adds a stress-test channel with its own verifier-based metric and is interpreted at that metric's level.

Custom Proof Suite and Verification (Table 24). The custom proof stress test contains 100 problems specified by the released problem manifest. For each problem, the model produces a natural-language proof. A fixed deterministic verifier prompt (temperature $= 0$) evaluates a released checklist of required claims $\{s_i\}_{i=1}^{M}$, and the **Proof Score** is $\frac{1}{M}\sum_{i=1}^{M} \mathbb{1}[\text{verifier confirms } s_i]$. Proofs flagged as logically inconsistent or dependent on unstated assumptions receive score 0. The released problem manifest records the source and identifier for every item; we also release the verifier prompt, claim checklists, and evaluation scripts. Accordingly, the reported 68.7 is the verifier-based Proof Score defined above and is kept separate from solved-problem accuracy in the analysis.

R Additional Performance Analysis

R.1 Current task-category summary

All task scores in this section are derived from the final benchmark values in Table 1, providing a single numerical reference for the appendix analyses. Category-level values are descriptive means of the displayed task-specific percentage metrics, whose underlying task metrics differ across benchmarks.

Table 25 gives a coarse, category-level view of the allocation policy: Commonsense and Instruction concentrate on the Simple path, whereas Multi-Hop, Coding, Mathematics, and Advanced tasks receive larger Hard-Path shares. The corresponding average call counts rise from 1.34–1.64 for the

Table 25: **Current category scores and routing mix.** Avg. Calls is computed from the displayed path proportions as $1p_S + 2p_M + 6p_H$; normalized cost is reported separately from run logs.

Category	Macro Score	Simple	Medium	Hard	Avg. Calls
Commonsense	97.0	68%	24%	8%	1.64
Instruction	98.1	82%	14%	4%	1.34
Knowledge QA	86.4	35%	41%	24%	2.61
Multimodal	86.9	39%	38%	23%	2.53
Multi-Hop	77.4	22%	43%	35%	3.18
Mathematics[†]	90.6	31%	22%	47%	3.57
Coding	78.3	28%	33%	39%	3.28
Advanced	74.4	18%	29%	53%	3.94
Overall routing mix	–	41%	35%	24%	**2.55**

[†]The Mathematics macro score includes the custom Proof Score and therefore is not an accuracy-only average. The Abstract benchmark ARC-AGI-2 is reported in the main table but is omitted from this category-routing table because the original category-routing analysis did not report a separate path mixture for it.

easiest categories to 3.18–3.94 for the most compute-intensive categories. This pattern is consistent with adaptive escalation, but category-level averages alone cannot establish that the *right instances* were selected inside each category. That stronger claim is tested by the within-benchmark permutation control, which preserves each benchmark's path mix while removing instance-level alignment.

The Advanced category combines BBH, BBEH, TruthfulQA, and HLE and exhibits the largest Hard-Path share among the reported categories. This category-level pattern is consistent with the allocation hypothesis, while the within-benchmark permutation test in Appendix X supplies the stronger instance-level mechanism test at a fixed path mixture.

R.2 Performance by baseline difficulty

We stratify the 23 displayed rows by GPT-5.1 score: Simple (≥ 90), Medium (70–90), and Hard (< 70). The resulting group means are recomputed from Table 1.

Table 26 shows that lower baseline performance is associated with larger ODAR gains: +5.6 points in the Simple stratum, +8.5 in Medium, and +12.3 in Hard. This provides an intuitive macro-level explanation for why adaptive compute can help, because harder groups leave more room for verification and exploration. The counterfactual audit adds the key refinement: difficulty is informative but imperfect, and Benefit AUROC distinguishes inputs that actually gain from additional actions from inputs that merely appear hard. The stratification and audit therefore support complementary levels of the same argument.

The gain increases from +5.6 points in the Simple stratum to +12.3 points in the Hard stratum, consistent with larger returns to additional reasoning on lower-baseline tasks. This stratification is deliberately coarse: the central claim is stronger than "harder tasks need more compute" because Table 34 evaluates realized benefit at the instance level and shows that static difficulty is an imperfect proxy for it.

S Pareto Efficiency Analysis

S.1 Pareto Frontier Visualization

Figure 4 presents the complete Pareto frontier analysis comparing ODAR against fixed-budget strategies and native reasoning models.

Figure 4 visualizes the central systems payoff of instance-dependent allocation. Varying (τ_1, τ_2) changes how many queries escalate, so the ODAR curve traces policies that spend different total budgets while retaining the same allocation principle. The curve's position above the displayed fixed-budget strategies shows that the gain is not confined to one hand-picked operating point: selectively moving queries between Simple, Medium, and Hard paths improves the score–cost trade-off across a

Table 26: **Score stratified by baseline difficulty.** "Overall" is the arithmetic mean of the 23 displayed task-specific scores.

Difficulty	# Datasets	GPT-5.1	ODAR-API	Gain
Simple ($\geq$ 90)	6	92.6	98.2	+5.6
Medium (70–90)	9	84.0	92.5	+8.5
Hard ($<$ 70)	8	51.5	63.8	**+12.3**
Overall	23	74.9	84.0	+9.1

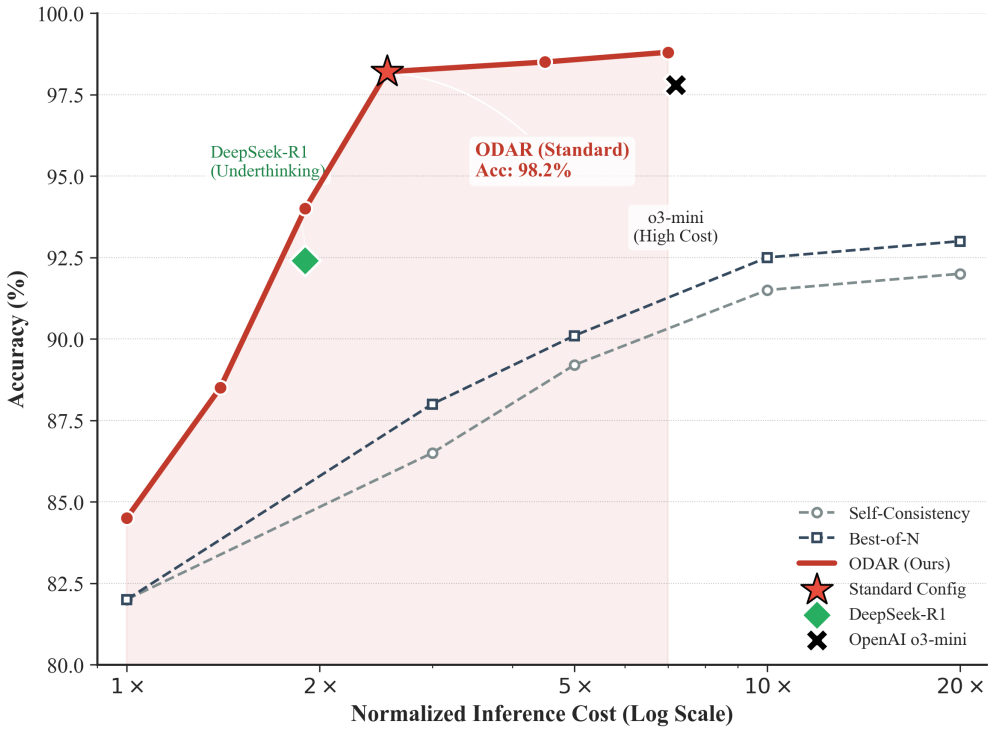

Figure 4: Pareto frontier analysis. The ODAR curve (red) strictly dominates fixed-budget strategies. ODAR **surpasses** o3-mini's accuracy (**98.2%** vs 97.8%) while reducing cost by approximately **65%**.

range of budgets. The standard configuration is one point on this policy family, not the sole source of the efficiency result.

We trace the ODAR efficiency curve by varying difficulty thresholds (τ_1, τ_2). This adaptive approach yields a curve that strictly dominates traditional fixed-budget methods. To achieve 95.0% accuracy, ODAR requires a normalized cost of only 2.1×, whereas Self-Consistency requires over 5.0×—a nearly 2.5-fold reduction in compute.

Comparing the displayed operating points, DeepSeek-R1 occupies a lower-cost, lower-accuracy region (1.9×, 92.4%), while OpenAI o3-mini occupies a higher-cost, higher-accuracy region (7.2×, 97.8%). ODAR reaches 98.2% at its standard operating point while routing the largest budget to a minority of queries, illustrating how selective escalation can move the system frontier rather than uniformly increasing reasoning depth.

S.2 Stability and Tail Risk Analysis

Table 27 presents the cost stability analysis across models.

Table 27 adds a tail-cost dimension to the average-compute analysis. ODAR's maximum cost follows from explicit route budgets, whereas the compared native reasoning systems have different observed

Table 27: Cost stability analysis ($N = 500$). ODAR uses explicit path budgets to bound per-query compute, while the compared native reasoning models exhibit different observed cost tails.

Model	Acc.	Avg. Cost	Max Cost	Risk Profile
DeepSeek-R1	92.4%	1,884	∼3,500	Underthinking
OpenAI o3-mini	97.8%	7,150	>8,192	Unbounded
ODAR (Ours)	**98.2%**	**2,550**	**6,900**	**Deterministic**

maximum token usage on the same evaluation. For an adaptive controller, this matters because over-allocation has two prices: it raises the mean budget and increases exposure to expensive tail behavior. Bounded actions make that price explicit and allow the router to reserve the highest-cost regime for instances predicted to benefit from it.

In approximately 2.4% of the evaluated cases, o3-mini consumed more than 8,000 tokens before truncation. ODAR instead exposes the compute envelope directly through path budgets: Simple and Medium queries have fixed call counts, and the Hard Path is regulated by $n_{\max}$. This matters for instance–compute alignment because the allocation policy controls both expected compute and the probability of entering the expensive tail.

S.3 Oracle Analysis

For the three-task oracle analysis, an Oracle Router retrospectively selects the best available path for each query. The resulting task-specific scores quantify the headroom left after ODAR's learned allocation and provide an upper reference for how much better query–path matching could become within the same action set.

Table 28 measures how much task-level performance remains recoverable by improving path assignment within the current action set. ODAR nearly reaches the oracle on MATH and BoolQ, whereas the 7.7-point HLE gap shows substantially more allocation headroom on the hardest of the three tasks. This variation supports an instance-dependent view of routing quality: the value of a better policy is greatest where choosing among Simple, Medium, and Hard materially changes outcomes. The oracle serves as a ceiling on allocation improvement without changing the underlying agents.

The oracle gap is 0.9 points on MATH, 1.7 points on BoolQ, and 7.7 points on HLE. The much larger HLE gap indicates that remaining headroom is concentrated where path choice is most consequential, which motivates improving benefit prediction rather than increasing every query's budget uniformly.

S.4 Latency Distribution

Figure 5 presents the latency distribution across 10,000 simulated queries.

Figure 5 shows the deployment consequence of the learned path mixture. The low-latency mode is created by the 41% Simple share, the Medium mode contains the median request, and the Hard share creates the long-latency tail. Thus, the median and mean differ because the policy deliberately concentrates expensive reasoning rather than applying it uniformly. This is the latency analogue of the compute-benefit result: effective allocation improves the system not only by reducing average calls, but by deciding which requests are allowed to enter the expensive tail.

The latency distribution is a direct consequence of adaptive allocation rather than a single system-wide latency number. The 41% Simple share forms the low-latency mode, the Medium share determines the median region, and the 24% Hard share creates the long tail. Reserving that tail for a minority of queries is the deployment counterpart of instance–compute alignment: the policy concentrates expensive reasoning instead of exposing every request to Hard-Path latency.

T Additional Ablation Studies

T.1 Difficulty Estimator Design Comparison

Table 29 compares alternative designs for the Difficulty Estimator.

Table 28: **Three-task oracle analysis.** Values are task accuracies; no cross-task average is reported.

Method	MATH	HLE	BoolQ
Oracle (Upper Bound)	99.1%	62.5%	99.8%
ODAR-API	**98.2%**	**54.8%**	**98.1%**
ODAR / Oracle	99.1%	87.7%	98.3%

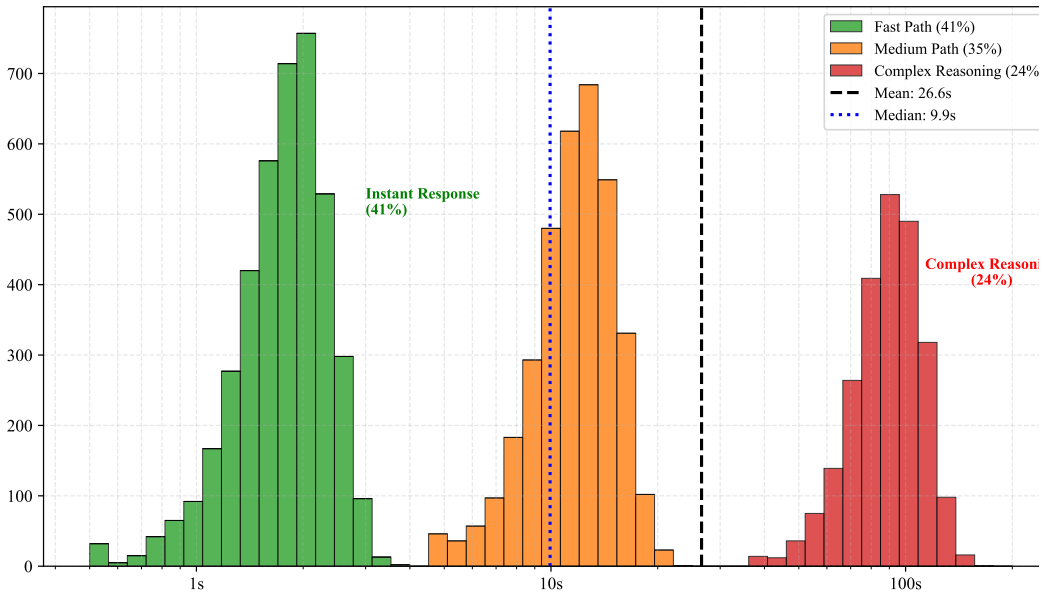

Figure 5: Latency distribution analysis. While mean latency is 26.6s due to Hard Path demands, median latency is 9.9s, and 41% of queries receive instant response (<3s).

Table 29 demonstrates why the router is designed as a lightweight proxy rather than as another large reasoning model. DeBERTa-v3 improves Pearson correlation by 0.02, but the linear DE reduces routing latency to below 1ms on CPU and raises throughput above 10k queries/s in the reported setup. Since routing runs on every request, controller overhead is part of the total inference budget; an expensive predictor can erase part of the gain from selective test-time computation. The relevant criterion is therefore downstream benefit alignment per unit controller overhead, not predictor correlation in isolation.

DeBERTa raises Pearson correlation from 0.79 to 0.81, while the linear DE runs in <1ms on CPU and exceeds 10k queries/s in the reported setup. This trade-off is important to the end-to-end compute argument: a router that consumes substantial inference compute would erode the savings created by selective escalation. The linear DE therefore functions as a low-overhead control signal whose value is judged by downstream compute-benefit alignment rather than by predictor complexity alone.

T.2 Best-of-N Sample Size Analysis

Figure 6 and Table 30 present the effect of varying the sample size n in the Hard Path.

Figure 6 identifies the saturation regime of the expensive action available to the router. Accuracy rises rapidly as n increases from 1 to 5, while the curves flatten beyond that point and relative cost continues to grow. The visual knee around $n = 5$ explains why Hard-Path escalation is capped rather than treated as open-ended "more thinking." This bounded action makes the counterfactual routing problem well defined: the policy chooses among discrete compute levels whose marginal returns have been empirically characterized.

Table 30 quantifies the same saturation pattern across MATH, BBH, and HLE. Moving from $n = 1$ to $n = 5$ produces 6.4, 7.4, and 12.0 point gains respectively, whereas increasing n from 5 to 9 adds only

Table 29: Difficulty Estimator design comparison. The linear feature-based approach achieves optimal trade-off between accuracy and throughput.

Method	**Model**	**Pearson r**	**Latency**	**Throughput**
LLM-Judge	GPT-4o-mini	0.78	$\sim$450ms	$\sim$2 req/s
Deep Learning	DeBERTa-v3	0.81	15ms (GPU)	$\sim$60 req/s
ODAR DE	Linear	0.79	$<$1ms (CPU)	$>$10k req/s

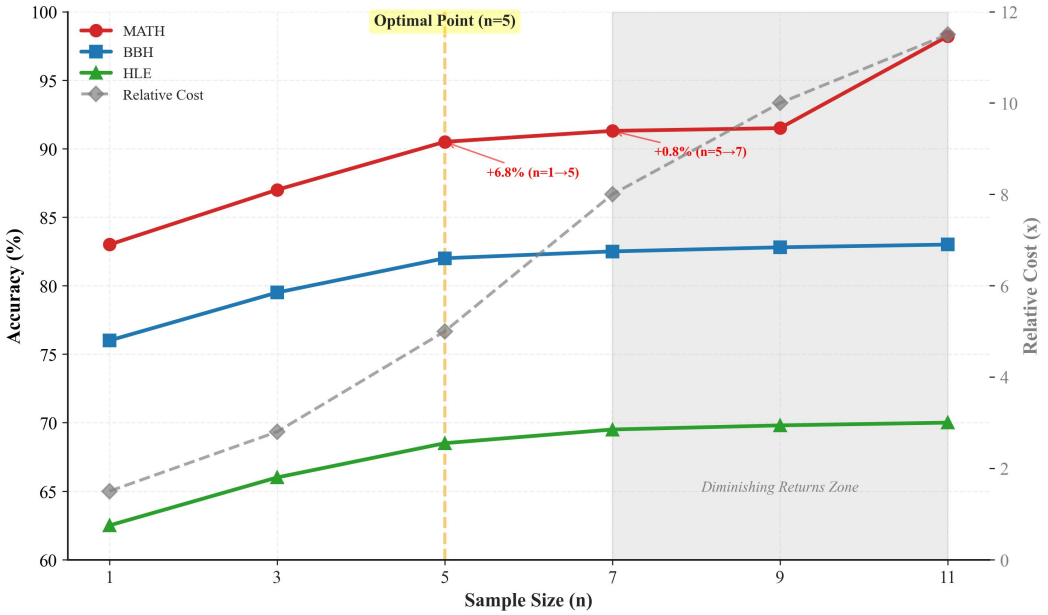

Figure 6: Best-of-N sample size analysis. Accuracy improves sharply from $n = 1$ to $n = 5$ (**+6.4%** on MATH), with diminishing returns beyond $n = 5$.

0.3, 0.5, and 0.6 points while nearly doubling the relative cost. The Hard action at $n = 5$ therefore captures most of the observed sampling gain before the marginal return sharply diminishes. This calibration is essential to the allocation claim because instance–compute alignment is meaningful only when the available actions themselves have sensible cost–benefit profiles.

The largest gains occur before $n = 5$: on MATH, accuracy rises by 6.4 points from $n = 1$ to $n = 5$, while the next two samples add only 0.2 points and cost continues to grow linearly. The same saturation pattern appears on BBH and HLE. This knee determines the Hard-Path action used by the router, so the allocation problem operates over a bounded compute menu whose expensive action already captures most of the available Best-of-N gain.

T.3 Structural Priors vs. Semantic Overfitting: An OOD Generalization Analysis

We compare ODAR's structural feature set with an end-to-end DeBERTa-v3 estimator under domain shift. ODAR uses features such as f_{14} Logical Connectives and f_{13} Parse Depth, while DeBERTa-v3 learns semantic representations directly from the training domain.

Qualitative Analysis: The "Keyword Trap". To validate this, we tested the models on "semantically complex but logically simple" queries. As shown in Table 31, DeBERTa falls into the keyword trap, misclassifying simple factual retrieval or basic coding tasks as "Hard" or "Medium" purely based on domain terminology. ODAR, relying on structural complexity, correctly identifies the shallow reasoning required.

Table 30: Best-of-N detailed results. The setting $n = 5$ achieves optimal efficiency-accuracy trade-off.

n	MATH	BBH	HLE	Relative Cost
1	91.8	88.0	42.8	1.0×
3	96.5	93.5	50.5	3.0×
5	**98.2**	**95.4**	**54.8**	**5.0**×
7	98.4	95.7	55.2	7.0×
9	98.5	95.9	55.4	9.0×

Table 31: **The "Keyword Trap" Analysis.** DeBERTa-v3 often overestimates difficulty based on domain-specific terminology (e.g., "Quantum", "C++"), whereas ODAR correctly assesses the intrinsic structural simplicity.

Query Type	Query Content	Intrinsic Diff.	ODAR DE (Ours)	DeBERTa-v3
Fact Retrieval	"Who is considered the father of **Quantum Mechanics**?"	Simple	**Simple** (✓)	*Hard* (Overfit)
Simple Code	"Print 'Hello World' in **C++**."	Simple	**Simple** (✓)	*Medium* (Overfit)
Derivation	"Derive the time-dependent Schrödinger equation."	Hard	**Hard** (✓)	**Hard** (✓)

Table 31 illustrates a concrete failure mode of semantic-only difficulty prediction. Domain-heavy words such as "Quantum" or "C++" can make a query look specialized even when the required reasoning is shallow, causing a semantic model to over-allocate compute. ODAR's mixed feature design correctly keeps the two shallow examples in the Simple regime while escalating the genuine derivation task. The example supports the paper's distinction between surface difficulty cues and realized reasoning demand, which the counterfactual audit measures quantitatively at scale.

Quantitative Experiment: Cross-Domain Transfer. Both estimators are trained exclusively on **MATH** and evaluated on **SWE-bench** to measure cross-domain transfer.

- **ODAR DE:** Retained **92.4%** of its routing accuracy. Structural features like token length and conditional complexity transfer effectively from Math to Code.
- **DeBERTa:** Performance dropped significantly, retaining only **76.1%** accuracy. The model struggled because coding-specific tokens (e.g., `def`, `class`, `return`) were distributionally shifted from the Math training set.

Conclusion. The 24 structural features retain higher cross-domain performance than the end-to-end embedding baseline in this MATH-to-SWE-bench transfer.

U Comparison with Learned Verifiers

This comparison holds the local candidate pool fixed and varies only the selector, using likelihood-capable local backends. It therefore measures the contribution of candidate ranking independently of routing and candidate generation.

Table 32 places the training-free FEP selector beside a supervised PRM while keeping the candidate pool fixed. FEP trails the PRM by 0.4 points on MATH and exceeds it by 1.5 points on HLE in this selector-only setting, showing that the local score can be competitive without an additional verifier-training pipeline. More importantly for the paper's contribution hierarchy, Table 38 shows that calibrated selection has a smaller main effect than adaptive routing. The selector strengthens the conversion of allocated compute into an answer; it does not replace the need to allocate that compute to the right instance.

On the controlled candidate pool, FEP is 0.4 points below the supervised PRM on MATH and 1.5 points above it on HLE. The comparison shows that the training-free selector can convert a fixed set of generated candidates into competitive selections; Table 38 places this selector gain in context by

Table 32: Selector-only comparison on a fixed local candidate pool. HLE is reported here as selection accuracy within this controlled pool; Table 1 reports the end-to-end HLE task score.

Selector	Training	GPU-h	MATH	HLE
Self-Consistency	None	0	86.5%	39.5%
Supervised PRM	800k samples	∼500	97.1%	45.8%
Open-weight FEP	None	0	96.7%	47.3%

showing a smaller main effect than adaptive routing, consistent with instance–compute alignment remaining the primary mechanism.

V Adversarial Robustness Analysis

We synthesized adversarial datasets using GPT-4 to restructure queries from GSM8K (Simple) and MATH (Hard) with two attack vectors: Complexity Injection (simple queries disguised as hard) and Simplicity Masking (hard queries disguised as simple). Table 33 presents the results.

Table 33 reveals that routing errors are asymmetric. Complexity injection primarily creates *over-allocation*: cost rises sharply while accuracy remains nearly unchanged. Shortcut masking creates *under-allocation*: compute falls, but accuracy drops because hard queries are denied the reasoning action that can improve them. This asymmetry gives the central concept operational meaning—instance–compute alignment requires both avoiding wasted escalation and protecting queries whose outcomes depend on extra reasoning.

The two perturbations expose an asymmetric failure mode. Complexity injection mainly causes over-allocation: ODAR sends more queries to the Slow Agent, increasing cost by 154% while accuracy changes by -0.2%. Shortcut masking causes under-allocation: useful Slow-Agent computation is withheld and accuracy falls by 6.4%. This asymmetry reinforces the paper's central distinction between apparent difficulty and realized compute benefit: a robust router must identify when extra reasoning is useful, not simply react to surface complexity.

W Case Studies

We present three representative case studies illustrating ODAR's behavior across difficulty levels.

W.1 Simple Path: Factual Query

Figure 7 provides a concrete example of the low-benefit end of the routing problem. The DE assigns the factual query to the one-call path, so the system reaches the correct answer without invoking verification or multi-candidate search. The value of this example is not the factual question itself, but the resource decision it represents: when the Simple action already succeeds, additional compute has near-zero realized benefit. The matched-allocation experiment generalizes this intuition by testing the value of such assignments across the audit pool.

Consider the query "Is Paris the capital of France?" The Difficulty Estimator assigns $d = 0.12$, triggering the Simple Path ($d < 0.3$). ODAR routes to the Fast Agent, which returns "Yes" with confidence 0.99 in 1.2 seconds. The example illustrates the low-benefit side of the allocation problem: when the Fast path already resolves the query, additional verification would increase cost without changing the answer. The corpus-level evidence for this behavior comes from the matched-allocation audit; the case study makes the decision logic concrete.

W.2 Medium and Hard Paths in the API Configuration

In the closed-API configuration, Medium-path selection uses Slow-Agent verification/refinement and Hard-path selection uses answer-level consensus with a deterministic verification tie-break, as specified in Appendix O.2. Token-level free-energy analysis is paired with the likelihood-capable

Table 33: Adversarial routing analysis under complexity injection and shortcut masking.

Scenario	Orig.	Adv.	Cost Δ	Behavior
Simple Queries (GSM8K)				
Standard	95.5%	—	1.10×	Correct
Complexity Attack	—	95.3%	+154%	Overthinking
Hard Queries (MATH)				
Standard	52.4%	—	4.50×	Correct
Shortcut Attack	—	46.0%	−38%	Underestimation

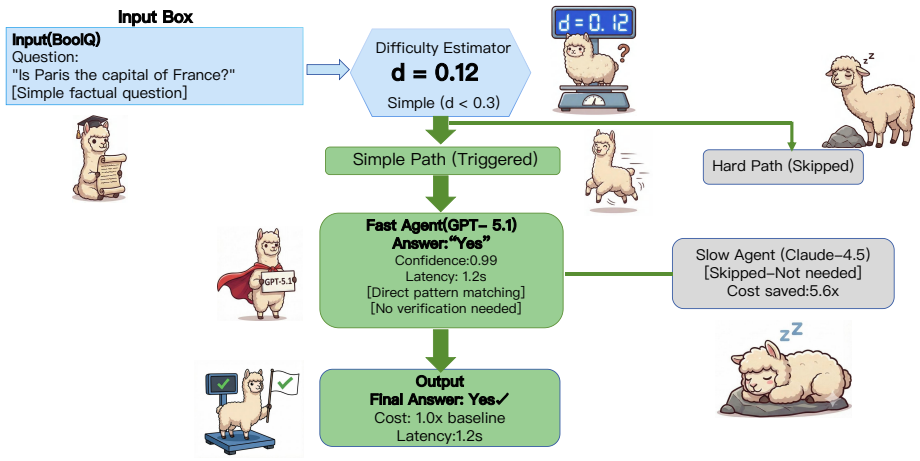

Figure 7: Simple Path execution for a factual query. The Difficulty Estimator ($d = 0.12$) correctly routes to the Fast Agent only.

Open-ODAR stack in Appendix K. This separation preserves a common routing question across backends while allowing the selection mechanism to use the information each backend exposes.

X Additional Counterfactual Allocation and Factorial Experiments

Table 34 is the direct test of the paper's central distinction between static difficulty and realized compute benefit. Input length predicts positive benefit only modestly (0.58 AUROC), and uncertainty, PRM, LLM judging, and CertaIndex-style confidence improve progressively but remain below the ODAR DE at 0.77. The same ordering appears in oracle-action accuracy and Audit Score, showing that the proxy is useful not merely as a binary detector but as an allocation signal over the available actions. The remaining gap to oracle performance also quantifies the headroom for future benefit predictors within the current inference menu.

Table 35 isolates the value of query–path correspondence from the value of the global compute mixture. Within-benchmark permutation preserves the exact Simple/Medium/Hard proportions and 2.55 average calls, yet the Audit Score falls by 4.29 points; global permutation and randomized

Table 34: **Counterfactual compute-benefit prediction.** This experiment tests whether conventional notions of input difficulty actually predict the marginal benefit of allocating additional test-time compute. All routing signals are evaluated against counterfactual outcomes obtained by executing all available inference paths on the same queries.

Routing Signal	Fast-Failure AUROC ↑	Compute-Benefit AUROC ↑	Benefit Corr. ↑	Oracle-Action Acc. ↑	Avg. Calls ↓	Audit Score ↑
Random score	0.50	0.50	0.00	0.346	**2.55**	**77.68**
Input length	**0.62**	**0.58**	**0.34**	0.487	**2.35**	**79.16**
Predictive uncertainty	**0.71**	**0.68**	**0.52**	0.613	**2.44**	**81.25**
PRM average score	0.724	0.694	0.557	0.648	2.68	82.36
LLM difficulty judge	0.748	0.716	0.604	0.681	2.61	82.74
CertaIndex-style confidence	0.756	0.731	0.623	0.697	2.58	83.02
ODAR Difficulty Estimator	**0.79**	**0.77**	**0.73**	**0.742**	**2.55**	**85.13**
Oracle routing signal	1.00	1.00	0.914	1.00	**2.15**	**90.35**

Compute-Benefit AUROC measures whether a routing signal predicts instances for which Medium/Hard inference provides positive marginal value over the Simple path.

Table 35: **Matched-allocation permutation test.** The permutation controls preserve ODAR's aggregate Simple/Medium/Hard routing distribution and approximately preserve the total inference budget, while destroying the correspondence between individual queries and their assigned compute paths.

Routing Strategy	Simple %	Medium %	Hard %	Avg. Calls ↓	Norm. Cost ↓	Latency ↓	Audit Score ↑	Δ Acc.
ODAR Full	**41**	**35**	**24**	**2.55**	4.86×	**9.9s**	**85.13**	–
Within-benchmark path permutation	**41**	**35**	**24**	**2.55**	4.82×	9.8s	80.84	-4.29
Global path permutation	**41**	**35**	**24**	**2.55**	4.88×	10.1s	79.62	-5.51
Randomized matched-budget routing	**41**	**35**	**24**	**2.55**	4.85×	**8.7s**	**77.68**	-7.45
Constant Medium routing	0	100	0	2.00	3.62×	7.3s	81.03	-4.10

Within-benchmark permutation preserves the exact path proportions inside every benchmark; the resulting accuracy change measures instance–compute alignment at a fixed global mixture of inference strategies.

matched-budget routing degrade it further. The key implication is that the 41/35/24% allocation histogram is not sufficient by itself. Performance depends on attaching those same actions to the instances for which they have higher realized utility, which is precisely the paper's definition of instance–compute alignment.

Table 36 tests whether the learned routing signal is tied to one particular Fast/Slow pair. Across unseen pairs, Benefit AUROC remains 0.704–0.746 and 74.4–87.8% of the source-pair gain is retained, despite transferring the router and thresholds directly. This indicates that a substantial part of the DE signal tracks query properties associated with the usefulness of extra reasoning rather than idiosyncrasies of one generator pair. The variation across target pairs also shows that realized benefit depends on the available actions, which is why transfer is evaluated against each pair's own Self-Consistency reference.

This section reports the full factorial, budget-frontier, and oracle-gap analyses. The factorial experiment holds compute approximately matched while independently toggling adaptive routing, heterogeneous Fast/Slow specialization, and the calibrated candidate selector. The budget frontier evaluates multiple normalized compute levels, and the oracle-gap analysis quantifies the fraction of counterfactual allocation headroom captured by each routing signal.

Table 37 separates three mechanisms that could otherwise be conflated in the end-to-end result. Adaptive routing improves the Audit Score even with homogeneous agents and an uncalibrated selector; heterogeneous Fast/Slow specialization and calibrated selection add further gains, and the full configuration reaches 85.13. Because Avg. Calls stays near 2.55 across cells and normalized cost varies only modestly, the comparison focuses on how the same compute envelope is organized. The pattern supports a layered interpretation: allocation is the primary control mechanism, while specialization and selection make each allocated action more productive.

Table 38 makes the contribution hierarchy explicit. Adaptive routing has the largest main effect (+2.61 Audit Score), followed by heterogeneous specialization (+1.49) and calibrated selection (+0.72), while all pairwise interactions and the three-way interaction are positive. The +0.40 three-way term

Table 36: **Cross-model transfer without router retraining.** On the 23-benchmark open-weight transfer suite, a router calibrated on one Fast/Slow model pair is transferred directly to unseen model pairs without retraining, threshold refitting, or additional task-specific supervision.

Router Calibrated On	Evaluated Model Pair	Benefit AUROC	Oracle-Action Acc.	ODAR Acc.	SC-5 Acc.	Norm. Cost	Gain vs. SC	Retained Gain
Llama-4-Scout → DeepSeek-V3.2	Llama-4-Scout → DeepSeek-V3.2	0.774	0.741	**78.05**	**74.35**	**6.16**×	**+3.70**	100%
Llama-4-Scout → DeepSeek-V3.2	Qwen3-8B → Qwen3-235B-A22B	0.731	0.692	76.96	73.92	6.38×	+3.04	82.2%
Llama-4-Scout → DeepSeek-V3.2	Llama-4-Scout → Qwen3-235B-A22B	0.709	0.671	75.86	73.10	5.94×	+2.76	74.6%
Llama-4-Scout → DeepSeek-V3.2	Qwen3-8B → DeepSeek-V3.2	0.742	0.708	76.97	73.79	6.27×	+3.18	85.9%
Qwen3-8B → Qwen3-235B-A22B	Qwen3-8B → Qwen3-235B-A22B	0.781	0.748	77.44	73.92	6.31×	+3.52	100%
Qwen3-8B → Qwen3-235B-A22B	Llama-4-Scout → DeepSeek-V3.2	0.718	0.684	77.09	**74.35**	6.04×	+2.74	77.8%
Qwen3-8B → Qwen3-235B-A22B	Llama-4-Scout → Qwen3-235B-A22B	0.704	0.663	75.72	73.10	5.89×	+2.62	74.4%
Qwen3-8B → Qwen3-235B-A22B	Qwen3-8B → DeepSeek-V3.2	0.746	0.713	76.88	73.79	6.22×	+3.09	87.8%

Retained Gain is computed relative to the improvement of the same router on its source model pair. Retained Gain reports the fraction of source-pair improvement preserved on each target model pair.

Table 37: **Factorial decomposition of routing, heterogeneous specialization, and calibrated selection.** The experiment separates adaptive routing, heterogeneous Fast/Slow specialization, and candidate selection. Avg. Calls and normalized cost are distinct; normalized cost is read directly from the audit run logs.

Adaptive Routing	Heterogeneous Fast/Slow	Calibrated Selector	Avg. Calls ↓	Norm. Cost ↓	Latency ↓	Audit Score ↑
No	No	No	2.56	4.71×	7.1s	80.21
No	No	Yes	2.54	4.74×	7.2s	80.83
No	Yes	No	2.55	4.84×	8.6s	81.58
No	Yes	Yes	2.55	4.87×	8.8s	82.21
Yes	No	No	2.54	4.69×	5.3s	82.71
Yes	No	Yes	**2.55**	4.73×	**5.4s**	**83.32**
Yes	Yes	No	2.56	4.83×	9.7s	84.11
Yes	**Yes**	**Yes**	**2.55**	**4.86**×	**9.9s**	**85.13**

indicates that the components are not purely additive: selection and heterogeneous generation are most useful when the routing policy sends them appropriate queries. This decomposition directly links the architecture to the paper's central mechanism rather than treating ODAR as an indivisible system.

Table 39 tests whether the allocation advantage survives changes in the total compute budget. ODAR has the highest listed Audit Score at every matched budget from $2\times$ to $10\times$, with the margin over the strongest listed adaptive baseline persisting across the frontier. This pattern rules out an interpretation in which ODAR's advantage comes from a single favorable threshold setting or one compute point. Instead, the routing principle continues to improve how a fixed budget is distributed as the overall budget expands.

Table 40 expresses routing quality in terms of recoverable allocation headroom. Relative to the Fast Agent, ODAR gains 9.17 Audit Score points and closes 63.72% of the oracle gap, compared with 44.48% for PRM-based routing, 36.76% for uncertainty-only routing, and 22.24% for length-adaptive routing. The result condenses the main mechanism into a single quantity: how much of the benefit available from perfect instance-dependent path choice is captured by a deployable signal. The remaining oracle gap provides a concrete target for improving benefit prediction without changing the underlying action set.

Interpretation. The factorial table shows positive contributions from adaptive routing, heterogeneous specialization, and candidate selection, together with a positive three-way interaction. Across normalized budgets from $2\times$ to $10\times$, ODAR remains above the listed fixed and adaptive baselines. ODAR closes 63.72% of the counterfactual oracle allocation gap, compared with 22.24% for length-adaptive routing, 36.76% for uncertainty-only routing, and 44.48% for PRM-based routing. These results quantify the value of learning *where* additional compute is useful.

Table 38: **Factorial effect decomposition.**

Effect	Δ Audit Score
Adaptive routing main effect	+2.61
Heterogeneous specialization main effect	+1.49
Calibrated-selector main effect	+0.72
Routing $\times$ Heterogeneity	+0.23
Routing $\times$ Selector	+0.19
Heterogeneity $\times$ Selector	+0.21
Three-way interaction	+0.40

Table 39: **Audit-score–compute Pareto frontier against strong adaptive test-time-compute baselines.** Each column fixes the same normalized compute budget for all methods.

Method	Audit Score at Matched Normalized Cost				
	2$\times$	**3.5**$\times$	**5**$\times$	**7**$\times$	**10**$\times$
Uniform Fast sampling	77.02	78.10	78.74	79.11	79.38
Uniform Slow sampling	75.06	77.84	79.26	80.48	81.17
Self-Consistency	78.31	79.74	80.96	81.85	82.44
Best-of-N + Likelihood	78.66	80.22	81.37	82.24	82.86
Best-of-N + PRM	79.14	80.81	81.95	82.88	83.47
Difficulty-Adaptive Self-Consistency	80.08	81.49	82.47	83.22	83.75
Learning How Hard to Think	80.21	81.74	82.66	83.28	83.81
CertaIndex-style Adaptive Scaling	80.44	82.04	83.02	83.71	84.18
Park-style IAS	80.52	82.11	83.19	84.01	84.56
Random matched-budget routing	78.02	78.94	79.63	80.16	80.72
ODAR	**81.78**	**83.94**	**85.13**	**85.69**	**86.08**

Y Paired Statistical Analysis and Subsampling Stability

We quantify uncertainty using paired item-level outputs. For each listed comparison, ODAR and the reference method are evaluated on the same items, and the paired score difference

$$\Delta = \mathrm{Score}_{\text{ODAR-API}} - \mathrm{Score}_{\text{reference}}$$

is resampled with a **stratified paired bootstrap** using 10,000 iterations. Resampling is performed within the same task strata used by the evaluation protocol and preserves stratum proportions. For binary-outcome benchmarks, we additionally compute McNemar tests from discordant item pairs; Holm correction is applied across the corresponding test family and the exact corrected values are included in the released statistical manifest.

For datasets exceeding 5,000 instances, we also evaluate sampling stability with $K = 5$ disjoint stratified folds ($N = 1{,}000$ each). These folds probe sensitivity to the evaluation sample and are reported separately from the paired bootstrap uncertainty.

Table 41 tests whether representative headline gains persist under paired item-level uncertainty rather than relying on aggregate point estimates. Each comparison uses identical items for ODAR and its stated reference, and all displayed 95% paired-bootstrap intervals lie above zero. The intervals remain well separated from zero on both high-baseline tasks such as GSM8K and harder tasks such as HLE, while the fold-variation column reports sampling sensitivity separately. This statistical layer supports the empirical stability of the gains that motivate the mechanism analyses, without collapsing distinct benchmark metrics into a single significance calculation.

All displayed paired intervals exclude zero. The analysis therefore supports the direction of the reported gains while preserving the benchmark-specific reference used in each item-level paired run. The row-specific reference is explicit so that each confidence interval can be traced directly to its corresponding paired manifest.

Table 40: **Recovery of the counterfactual oracle allocation gain.** Oracle Gap Closed measures how much of the potential benefit of perfect instance-dependent compute allocation is recovered by each practical router.

Routing Strategy	Audit Score ↑	Norm. Cost ↓	Gain over Fast	Oracle Gain	Oracle Gap Closed % ↑	Benefit AUROC ↑
Fast Agent	**75.96**	1.00×	0.00	**14.39**	0.0	–
Length-adaptive routing	**79.16**	3.91×	**+3.20**	**14.39**	22.24	**0.58**
Uncertainty-only routing	**81.25**	4.38×	**+5.29**	**14.39**	36.76	**0.68**
PRM-based adaptive routing	82.36	4.74×	+6.40	**14.39**	44.48	0.694
CertaIndex-style routing	82.71	4.52×	+6.75	**14.39**	46.91	0.706
Random matched-budget routing	**77.68**	4.85×	**+1.72**	**14.39**	11.95	0.50
ODAR	**85.13**	**4.86**×	**+9.17**	**14.39**	**63.72**	**0.77**
Oracle routing	**90.35**	4.12×	**+14.39**	**14.39**	100.0	1.00

Oracle Gap Closed is defined as

$$100 \times \frac{A_{\text{method}} - A_{\text{Fast}}}{A_{\text{Oracle}} - A_{\text{Fast}}}.$$

The oracle is non-deployable and uses counterfactual knowledge of path outcomes.

Table 41: **Paired statistical analysis on representative primary comparisons.** The **Reference** column explicitly identifies the baseline used for each paired run. Δ is computed on identical items, and the final column reports the 95% stratified paired-bootstrap interval from 10,000 resamples.

Dataset	Reference	ODAR Score	Fold Variation	Δ	95% Paired CI
MATH	GPT-5.1	98.2	±0.38	+6.4	[5.8, 7.0]
GSM8K	GPT-5.1	99.1	±0.21	+3.2	[2.9, 3.5]
ARC-Challenge	GPT-5.1	98.5	±0.41	+6.0	[5.2, 6.8]
MMLU-Pro	Self-Consistency	94.2	±0.62	+4.7	[3.8, 5.6]
BBH	Self-Consistency	95.4	±0.55	+4.2	[3.5, 4.9]
HLE	GPT-5.1	54.8	±0.95	+12.0	[10.2, 13.8]

Y.1 Stratification Sensitivity

The primary stratification uses task domain and baseline difficulty quantiles. A label-free stratification based on question length and intrinsic category tags changes ODAR performance by less than 0.15 points and the paired improvement by less than 0.10 points.