DIRECT: When and Where Should You Allocate Test-Time Compute in Embodied Planners?

Jadelynn Dao[1,*], Milan Ganai[1,*], Yasmina Abukhadra[1], Ajay Sridhar[1], Mozhgan Nasr Azadani[1,2], Katie Luo[1], Clark Barrett[1], Jiajun Wu[1], Chelsea Finn[1], Marco Pavone[1,3]

[1]Stanford University [2]University of Waterloo [3]NVIDIA

*Equal contribution.

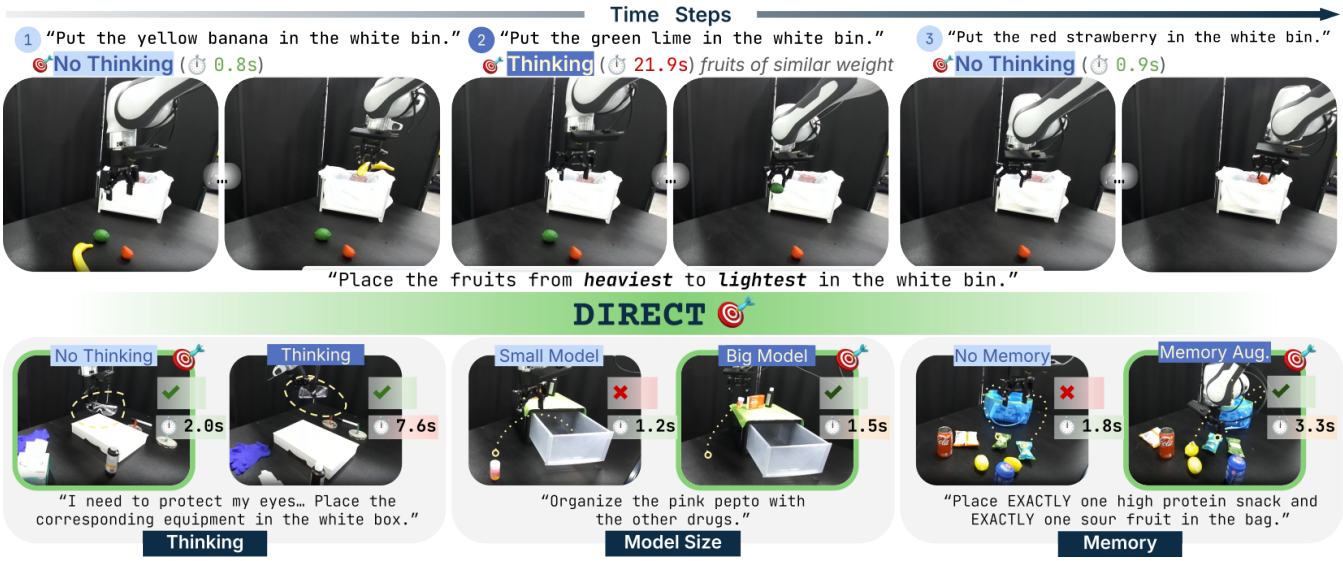


Fig. 1: **DIRECT routes test-time compute per task, matching frontier planning at a fraction of the cost. Top:** on a multi-step task, DIRECT uses a non-thinking planner on trivial steps (0.8–0.9s) and escalates to thinking (21.9s) on ambiguous steps. **Bottom:** across three axes—chain-of-thought, model size, and memory—DIRECT routes to the cheapest yet capable model per task.

***Abstract*—Vision-Language Models (VLMs) are increasingly deployed as high-level planners for embodied agents, with an emerging strategy of scaling test-time compute to improve capability. However, we observe that doing so increases latency, token usage, and FLOPs while yielding uneven, often diminishing gains in downstream success, limiting where embodied agents can be deployed. We argue that choosing when and where to spend test-time compute is central to bringing frontier performance to the real world. We introduce DIRECT, a routing framework that uses multimodal scene context to allocate compute per prompt, improving the success–cost Pareto frontier over fixed model selection. Across three dominant scaling axes, namely chain-of-thought depth, model size, and memory history, our experiments on VLABench and RoboMME show that test-time compute is not a uniform lever: different axes yield qualitatively distinct capability gains. We validate these insights on a physical Franka arm in a DROID setup spanning zero-shot manipulation and long-horizon chaining, where our router matches or exceeds a stronger model's success rate at up to 65% lower average latency. Ultimately, our results show that naively scaling test-time compute is wasteful, and that DIRECT can provide frontier-level embodied planning in robotic systems at a fraction of the cost. Project page can be found at jadee-dao.github.io/direct/.**

***Index Terms*—Hierarchical Embodied Planning, Test-Time Compute, Model Routing**

I. INTRODUCTION

Vision-Language Models (VLMs) are increasingly deployed as high-level planners that decompose abstract instructions into executable sub-skills for embodied agents [2, 16, 40, 62], enabling semantic flexibility for open-world robotic interaction. In this hierarchy, an orchestrating VLM planner generates skill plans that are enacted by a low-level policy. An emerging strategy to improve performance is scaling compute cost expended at inference [2, 44, 46]—this cost is known as test-time compute [42].

But this scaling for embodied planners creates a growing friction with real-world viability: frontier VLM latencies can reach tens of seconds. In Fig. 1 (bottom left) on the Franka, a Thinking model is 3.8× slower, yet has the same success as its non-thinking counterpart. This test-time cost narrows where such agents can be deployed, so deciding *when and where* to spend compute is critical. Model routing has emerged as one such approach for language models [9, 37] by directing queries to the cheapest capable model. In embodied planning, compute is not uniform—different techniques confer distinct capabilities, like deeper reasoning and longer-horizon memory.

Effective allocation requires routing policies that infer task demands from scene and instruction context and match them to the compute configuration with the best quality–cost tradeoff.

We propose DIRECT: **D**ynamic **I**nference **R**outer for **E**mbodied **C**ompute **T**radeoffs, a routing framework over VLM planners for hierarchical embodied planning. Our investigations show scaling test-time compute is not a uniform lever: CoT depth helps on tasks with implicit semantic, physical, or spatial constraints; VLM size governs breadth of skills a planner can command; and memory-oriented compute helps on history-dependent tasks but can hurt elsewhere. Building on this, we develop a lightweight router that conditions on multimodal context, directing each task to the planner whose capability profile matches inferred demands. Our contributions are:

1) diagnostic analyses of task difficulty, planner capability, and test-time compute tradeoffs in embodied settings;
2) our framework DIRECT that dynamically allocates test-time compute for embodied planning; and
3) experiments on VLABench, RoboMME, and Franka DROID hardware showing DIRECT can match the strongest model's performance at up to 65% lower latency.

II. Related Works

A. Hierarchical Vision-Language-Action Models

A growing line of work positions vision-language models as high-level planners that decompose abstract instructions into sub-skills for a low-level policy. Early work grounded LLM reasoning in robotic affordances [2, 25], while recent hierarchical VLAs adopt an explicit System 1–System 2 decomposition [28], where a high-level VLM emits semantic subtasks that a low-level policy enacts [6, 27, 40, 44]. Variants explore skill-centric scheduling [34] and dedicated task-management VLMs [22, 53]. Existing methods commit to a single high-level model with fixed capability and cost. In this division of labor the high-level model carries the semantic burden—parsing the instruction, resolving references in the scene, and committing to an ordered sub-skill sequence—while the low-level policy specializes in closing the perception-to-action loop; this is precisely why the choice of high-level planner, rather than the controller, dominates the cost–capability tradeoff we study. We instead treat the high-level planner as a dynamic variable, dispatching to different VLMs based on each task's inferred cognitive demands.

B. Dynamic Model Routing for Language and Multimodal Models

Dynamic routing optimizes inference by directing queries to the most efficient model in an ensemble. Frameworks like FrugalGPT [9] and RouteLLM [37] predict performance from linguistic features [1, 15]. However, these frameworks route purely on text. A nascent multimodal line [26, 30, 33, 45] shows that text features alone underestimate routing-relevant complexity. In embodied AI, difficulty further involves physical scene understanding and trajectory feasibility [58]. The closest analog in robotics, DA-SIP [12], routes *within* a low-level diffusion controller, selecting integration-step budgets by difficulty. We extend multimodal routing to the high-level planning stack, where a broad ecosystem of interchangeable candidate models span orders of magnitude in compute cost and capability. Crucially, where a question-answering router optimizes for textual answer quality, an embodied router must predict *downstream physical success*—a signal that couples instruction semantics with scene geometry and the competence of the low-level policy that ultimately executes the plan.

C. Test-Time Compute and Adaptive Inference

Scaling test-time compute can improve capability, even outperforming a 14× larger model [42], via CoT prompting [51], deliberate search [5, 54], and parallel sampling with majority voting or verifier-guided selection [50, 59]. Yet longer chains can hurt performance [23, 52] and saturate early [49]. In embodied settings, grounded multi-step reasoning improves VLA success [18, 55, 60], but its latency is prohibitive on a robot [11]; recent work amortizes this cost via thought reuse [17] or latent reasoning [4, 10, 31]. Scaling up the memory history length of the high-level planner has also been shown to improve performance on long-horizon memory-based tasks [44]. SayCan [2] shows scaling model size of the high-level LLM planner can improve task success, but performance saturates at some point. These strategies expose qualitatively different compute knobs—sequential reasoning depth, model capacity, and retained history—that scale latency, parameter count, and context length respectively, and the memory axis alone spans a wide implementation range, from perceptual token re-weighting through symbolic subgoal summaries to keyframe retrieval. We therefore treat them as three separate routing axes rather than a single scalar "more compute" dial. Since these strategies are typically applied uniformly [25, 55], we instead ask *when* compute is worth its cost, identifying high-signal cases via perception-aware classifiers conditioned on scene and instruction context.

III. Investigating How Test-Time Compute Impacts Embodied Planning

Scaling test-time compute is not free in robotics: in a hierarchical stack the planner is invoked repeatedly, so per-call costs from CoT, larger models, or denser memory compound can multiply base model latencies. Cutting compute uniformly mitigates latency but risks failure on harder tasks. We therefore study where compute actually yields gains across three axes—reasoning depth, model size, and memory history—motivating our framework in Section IV.

A. Spending Thinking Tokens: Depth vs. Delay

We first examine the effect of CoT reasoning on planning quality on VLABench [58], a multi-step reasoning benchmark, where a high-level planner decomposes abstract instructions (*e.g.*, "sort these books on the shelf so the library looks tidy") into an executable skill sequence. High-level planners that employ deliberate thinking chains can reason through

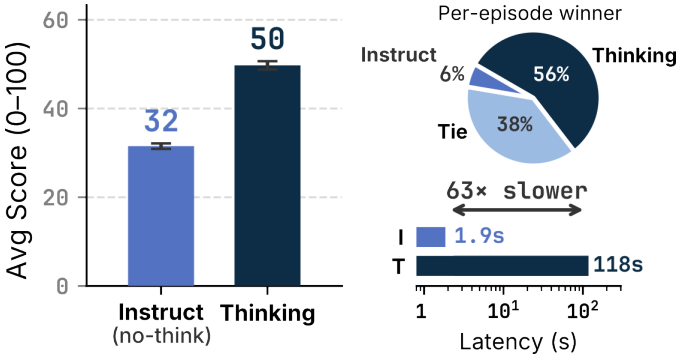


Fig. 2: **Depth vs. delay on the reasoning axis.** On VLABench [58], the Thinking planner improves mean score but is far slower; in 44% of cases Qwen3-VL 8B Instruct matches or surpasses Thinking at <2% of its latency, so the benefit of extra reasoning is concentrated on a minority of tasks rather than spread uniformly. Scores normalized to 0–100.

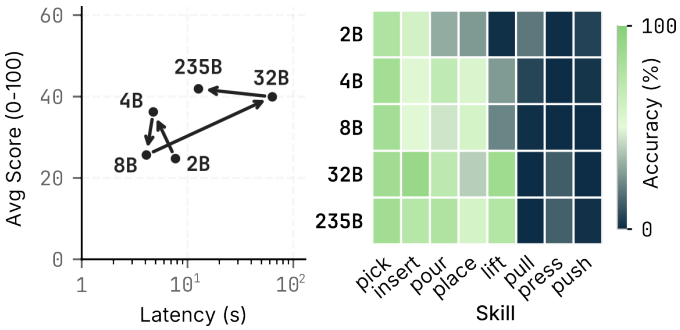


Fig. 3: Scaling Model size (2B–235B) on VLABench [58]. **Left:** score and latency scale non-monotonically with model size. **Right:** model size broadens the skill set a planner can reliably command.

subtle physical, semantic, and spatial constraints, yet this reasoning comes at a heavy cost in token generation latency. Comparing Qwen3-VL 8B Instruct versus Qwen3-VL 8B Thinking (Fig. 2), we find while thinking improves average performance, on 44% of tasks the Instruct model matches or outperforms the Thinking model with orders of magnitude faster inference (see Appendix D-A, D-B). This suggests relying on reasoning models can be wasteful as many scenes can be solved efficiently for real-time viability. Throughout, we use VLABench's non-interactive VLM-evaluation track, in which the planner emits a skill sequence that is scored by matching its directed acyclic graph against the ground-truth skill DAG (Appendix G-A).

B. Scaling Model Size: Command on More Skills

We investigate how planner performance scales with model size. Evaluating Qwen3-VL Instruct from 2B to 235B parameters [3], we find brute-force scaling traces no clean curve in either score or latency (Fig. 3 left, Appendix D-A, D-B). Score is non-monotonic, and since verbosity is not tied to parameter count, smaller variants sometimes generate too many tokens and run slower than larger ones (*e.g.*, 32B is slower than 235B). Decomposing by skill reveals size primarily governs the *breadth* of skills a planner can reliably command (Fig. 3, right). The larger model's edge is correctly commanding a broader set of skills, paying off only on tasks that demand them. Uniform deployment of the large model wastes compute on other tasks, motivating per-task allocation.

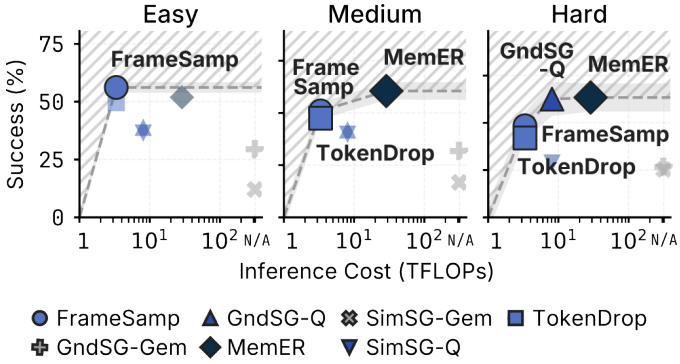


Fig. 4: **Memory is a non-uniform lever.** Memory architectures across difficulty tiers on RoboMME [13], spanning perceptual token re-weighting (`FrameSamp`, `TokenDrop`), symbolic subgoal summaries (`GroundSG`), and keyframe retrieval (`MemER`). Lightweight schemes win cheaply on easy tasks, while `MemER` and `GroundSG` lead on hard tasks, so the cost-optimal memory mechanism shifts with task difficulty rather than being fixed.

C. Varying Memory Architectures: Contextual Overhead

Long-horizon tasks can require memory, but a planner VLM can compress history in very different ways in context. `FrameSamp` and `TokenDrop` reduce raw visual tokens, `SimpleSG` and `GroundSG` summarize history as language subgoals, and `MemER` [44] recalls prior keyframes. On RoboMME [13], we observe each trades accuracy against overhead, with the most efficient choice shifting with difficulty (Fig. 4, Appendix D-A). On easy tasks needing short recall, the lightweight `FrameSamp` outperforms `MemER` at an order of magnitude fewer FLOPs; as tasks demand distant history, `MemER` and `GroundSG` have higher success. No single architecture dominates as each demonstrates benefits in particular levels of difficulty.

The Case for Adaptive Allocation. These investigations yield three trends: (1) large capability gaps separate cheap and expensive configurations; (2) the gaps are non-uniform, hinging on task-specific demands like skill command or recall horizon; and (3) lightweight models suffice on many tasks. Since any static model fixes a single latency–capability compromise, these observations motivate a framework that infers each task's demands from the scene context and allocates compute per task.

IV. Proposed Embodied Routing Framework: Direct

We introduce our routing framework Direct, which aims to learn a mapping from a task's scene and instruction to the planner best suited to solve it. Direct chooses among a fixed pool of candidates that differ in capability and cost.

A. Problem Formulation

Concretely, we formalize embodied test-time compute allocation as a selection problem over a fixed pool $\mathcal{M} =$

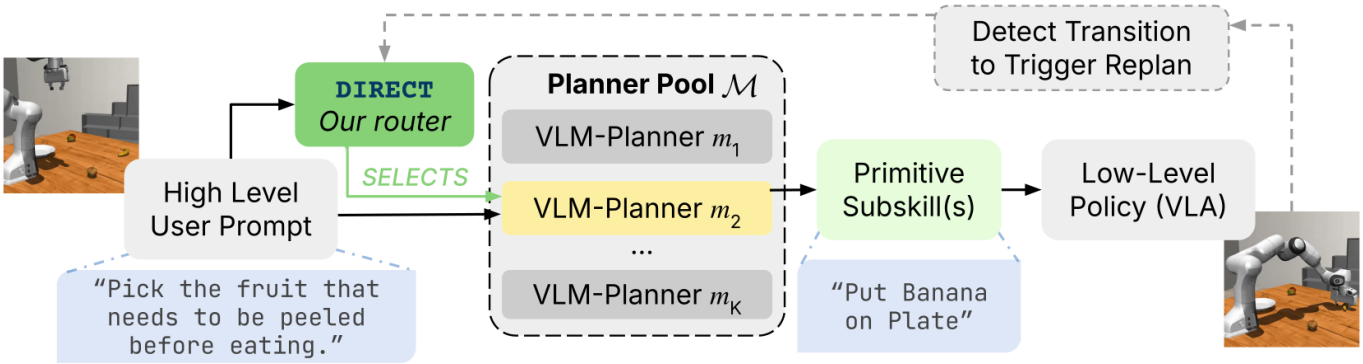


Fig. 5: Given a user instruction and scene observation, our lightweight router DIRECT selects the planner with the best quality–cost tradeoff from a fixed pool of VLM planners $\mathcal{M} = \{m_1, \ldots, m_K\}$. The chosen VLM decomposes the instruction into primitive subskills executed by a low-level VLA policy, while a transition detector triggers replanning for multi-stage tasks. By conditioning on multimodal context, DIRECT allocates test-time compute per task rather than using a single fixed-capability planner.

$\{m_1, \ldots, m_K\}$ of VLM embodied planners, each of which maps $m_k : x \mapsto (a_1, \ldots, a_T)$ from a task to a sequence of primitive sub-skills that a downstream language-conditioned policy executes on the robot. A task input $x = (I, \ell)$ pairs an initial scene observation I (an RGB image, or a short multi-view stack where available) with a natural-language instruction ℓ. Given x, a lightweight router predicts which planner offers the best quality–cost tradeoff and selects a single $m_k \in \mathcal{M}$ (Fig. 5).

B. Data Collection

For every training task $x_i \in \mathcal{D}_{\text{train}}$ and planner m_k we measure two quantities:

1) a *quality score* $q_{i,k} \in [0, 1]$: the task success rate (or progress score where available) when m_k's sub-skill sequence $(a_1, \ldots, a_T)$ is executed end-to-end; and
2) an *inference cost* $c_{i,k} \in \mathbb{R}_{>0}$, the resource measure that constrains deployment.

We use latency for the cost when planners differ along an autoregressive axis (*e.g.*, reasoning depth), since wall-clock time is the operative constraint on sequential generation, and FLOPs when they differ in parallel structure (*e.g.*, memory architectures), since latency there conflates compute with hardware-level parallelism. Collecting these across all N tasks and K planners yields $Q \in [0, 1]^{N \times K}$ and $C \in \mathbb{R}_{>0}^{N \times K}$, the quality and cost matrices over the task–planner grid. In simulation, we score rollouts directly from the benchmark.

C. Synthetic Data Generation for Hardware

On hardware, collecting data to train the router by exhaustively rolling out every planner on every physical scene is impractical. So, we synthesize tasks offline by sampling scenes disjoint from evaluation, prompting a large VLM to propose candidate instructions with reference skill decompositions, running each planner m_k on those synthetic tasks to record its sequence and latency, and scoring quality with an LLM judge against the reference. This yields Q and C training data for physical-style tasks without requiring robot hardware rollouts, and the resulting trained router is deployed zero-shot on hardware (details in Appendix G-C). Concretely, we first generate many additional feasible instructions per scene—both variants of the seed task and other plausible tasks the scene affords—indexed by routing axis and balanced across the classes that carry the routing signal. The supervision used to train and calibrate the hardware router then differs by axis:

- **Reasoning (thinking vs. no-thinking).** Every candidate planner produces a decomposition, and an LLM judge evaluates *each planner's option*, labeling it steerable if its subgoals fall within the region of simple DROID subtasks the downstream policy reliably executes.
- **Skill command and memory.** An LLM judge instead *tags each task* according to our observations from Section III—*e.g.*, whether a goal depends on unobservable history, or invokes a rigid (*put*) versus contextual (*close*, *fold*, *wipe*) skill—and we train the router to recognize these categories.

This entire pipeline runs offline without any robot execution, letting us train and calibrate the hardware router purely from synthetic data; the resulting labels become its routing targets (Fig. 6).

D. Router Architectures

The router $r(\cdot)$ maps encoded task features $\phi(x) \in \mathbb{R}^d$ to a planner index $\hat{k} \in \{1, ...K\}$, introduced at the start of the section. We construct ϕ by encoding I with a frozen SigLIP-family [48, 57] vision encoder and ℓ with a frozen BGE-M3 [35] text encoder, then concatenating the two embeddings into a single vector. Additional details and ablations over various embedding fusion strategies, vision encoders, and embedding types are reported in Appendix B-A. We evaluate many parameter-efficient router architectures, in which router inference cost and a single embedding pass remains negligible (20-50 total millisec) against any VLM planner in $\mathcal{M}$ (>1 sec). Router candidates span:

- linear models;
- k-nearest-neighbors (KNN) and pairwise-preference-aggregated KNN (PR-KNN) [61];

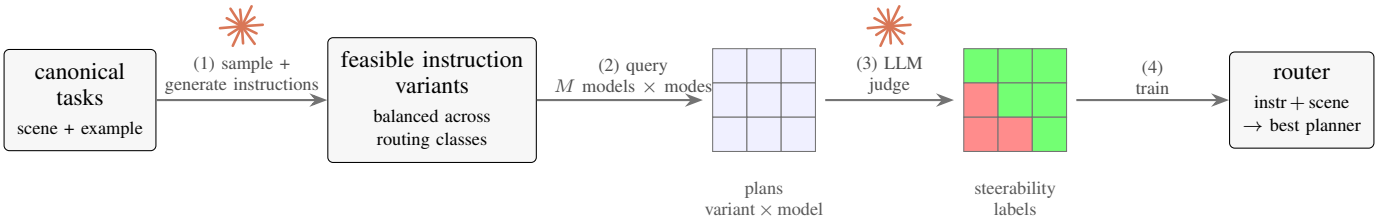


Fig. 6: **Synthetic-data pipeline for the hardware router** (detailed in Appendix G-C): (1) generate feasible instruction variants per scene, (2) decompose each with every candidate planner, (3) LLM-judge plan steerability (green = steerable, red = not), and (4) train the router on these labels.

- k-means and one-versus-rest (OVR) [39] classifiers; and
- two-layer multilayer perceptrons (MLP).

Our extensive architecture sweep is detailed in Appendix B-B.

E. Routing Objective

As established in Section III, no single planner dominates across tasks, and quality and cost objectives conflict. To choose among planners, we require a scalar criterion that combines objectives into a single quantity, since the routing decision is ultimately a one-dimensional argmax over $\mathcal{M}$. We capture this through a utility function $U(q, c)$, defining the per-task target as

$$k_i^\star \;=\; \arg\max_k \;\; U(q_{i,k},\, c_{i,k})\,. \tag{1}$$

The framework is agnostic to the specific form of U, and we expose the tradeoff through a family of routing objectives, each corresponding to a different instantiation of Equation 1. One strategy is to predict $\hat{q}_{i,k}$ and $\hat{c}_{i,k}$ via regression heads against the entries of Q and C, so for $\alpha \geq 0$ weighting cost against quality, the utility can be framed as $U_{\text{reg}}(\hat{q}_{i,k},\, \hat{c}_{i,k}) = \frac{1}{\hat{c}_{i,k}} \cdot \mathbf{1}[\max_{k'} \hat{q}_{i,k'} - \hat{q}_{i,k} < \alpha]$. We detail and ablate on other utility functions and heads (see Appendix B-C). We generally found regression as the best head likely since it leverages the most expressive signal of quality and cost.

F. Evaluation Metric

Each router configuration, defined by its choice of head, hyperparameters, and architecture, realizes a different point in the quality-cost plane. We compare routers via a harmonic-mean efficiency score (η) inspired by LLM/VLM routing benchmark metrics [26, 32]:

$$\eta(\beta) = \frac{(1+\beta)\, q_s\, (1 - c_s)}{\beta\, q_s + (1 - c_s)}, \tag{2}$$

where q_s and c_s are the router's mean quality and cost min-max-scaled on a hold-out set across all routers and baselines in the comparison, as well as an oracle quality ceiling, and β weighs relative emphasis on quality. This evaluation criterion is decoupled from any specific head's objective.

G. Deployment

At inference, we observe a fresh task $x = (I, \ell)$, compute $\phi(x)$, and the router predicts a planner index $\hat{k} = r(\phi(x))$. The router is invoked once per task, or repeatedly for multi-stage tasks, with each call deciding independently on the current (I, ℓ). A routing decision costs one embedding extraction plus the router's forward pass (0.009–1.8 ms across architectures), totaling roughly 20–50 ms per query—two to three orders of magnitude below the planner latencies it arbitrates (Appendix B-E).

V. Experimental Results

We evaluate the effectiveness of DIRECT across three distinct axes of test-time compute: CoT reasoning depth, model size, and memory history. Each axis corresponds to a substantively different way in which VLMs in the pool $\mathcal{M}$ can apply more compute at test time to increase capability, and each gives rise to a routing problem whose structure we characterize empirically. We train and evaluate routers in simulation and in Franka DROID hardware (Appendix D-C, D-D1, G-B), validating how DIRECT improves test time compute scaling across all axes. In total our results contain over 270,000 simulated routing decisions and 245 hardware trajectories. Each hardware trial is scored 1 (success), 0 (failure), or 0.5 when the planner emitted a plausible subtask but the low-level policy failed to carry it out for reasons outside the planner's control, *e.g.*, an incorrect grasp (Appendix D-D).

A. Baselines

We hypothesize the benefit of test-time compute is task-contingent, so DIRECT will recover the strongest/expensive model's quality at a fraction of its cost by invoking it only when needed. In each regime we compare the learned router against:

- *Cheap* and *Expensive*: the static choices with, respectively, the lowest and highest mean cost, marking the two deployable allocation extremes;
- *Random*: allocates uniformly at random;
- *OOD detection*: tasks whose embeddings are out-of-distribution of a calibration set (detected via conformal prediction [41]) are routed to progressively more expensive variants; and
- *Oracle*: distinct from the other baselines, it selects the per-task optimal variant with full knowledge of Q and C.

We report results for the router with the optimal configuration and hyperparameters in the sweeps for *OOD detection* and DIRECT. The *oracle* is not deployable, but bounds the achievable frontier given the pool; the gap to any policy quantifies the headroom available to selective allocation.

TABLE I: **VLABench routing comparison** ($N = 900$). Comparing learned routing against baselines on VLABench. Columns report mean success score (Q, %) and average planning inference latency (Lat, s); routing methods additionally report routing efficiency (η, %) where $\beta = 0.1$. More extensive results can be found in Appendix C. Pair labels: I/T = Instruct/Thinking, M/T = Minimal/Thinking. †These models were evaluated on a 20% subsample due to substantially higher inference time.

	Cheap		*Expensive*		*Random*			*OOD Detection*			DIRECT		
Pair (Cheap / Expensive)	Q	Lat	Q	Lat	Q	Lat	η	Q	Lat	η	Q	Lat	η
Homogeneous Open-Weight Pairs													
GLM-4.6 I / 4.1 T	27.24	2.11	54.02	45.10	40.02	22.39	46.33	49.83	39.60	54.52	51.79	31.82	**75.26**
GLM-4.6 I / T	27.24	2.11	46.09	24.08	35.97	13.24	43.68	42.73	20.85	55.34	44.69	16.97	**74.92**
Qwen3-VL 8B I / T †	31.64	1.85	48.28	120.76	40.62	64.40	48.48	46.89	106.22	54.20	47.48	81.72	**74.78**
Homogeneous Closed-Weight Pairs													
Gemini 3 Flash M / T	61.22	1.88	64.22	15.86	62.11	8.82	15.53	61.83	6.70	10.81	64.83	8.47	**57.99**
Gemini-ER 1.6 M / T	62.88	2.31	62.62	7.02	62.46	4.67	0.0	63.19	3.13	19.68	63.92	4.72	**37.09**
Heterogeneous Model Pairs													
Cosmos-R2 / Qwen3-VL 8B T †	29.56	2.02	48.28	120.76	38.93	64.49	45.03	37.32	49.37	38.47	46.98	86.36	**71.16**
RynnBrain-Plan / GLM 4.1 T	45.25	1.67	54.02	45.10	49.44	22.17	32.51	52.56	32.72	50.42	54.33	27.50	**63.94**

B. Routing on the Chain-of-Thought Test-Time Compute Axis

Simulation Results. We evaluate routing over No-Thinking (*e.g.*, instruct), Thinking model pairs drawn from both open-weight and closed-weight families, along with heterogeneous open-weight model pairs. As seen in Table I, DIRECT achieves the highest routing efficiency η in every configuration. Open-weight pairs reveal substantial CoT headroom that DIRECT exploits most aggressively. On the three homogeneous open-weight pairs, the cheap-to-expensive quality gap is large, and DIRECT recovers near-expensive quality while cutting latency by around 30%, with $\eta \approx 75\%$ across all three pairs. On closed-weight pairs, DIRECT still achieves the best routing efficiency η, and furthermore DIRECT's success score *exceeds both* the cheap and expensive static choices, evidence that our routing framework extracts complementary strengths even when neither variant dominates on average. The advantage persists under heterogeneous pools, where cheap and expensive are drawn from different model families. DIRECT recovers most of the expensive variant's quality at ∼30-40% lower latency, indicating the learned signal is not tied to within-family stylistic cues. Together, these results indicate that only a subset of VLABench's language-conditioned manipulation tasks benefit from CoT, and DIRECT can identify these tasks from the initial scene and instruction across model families and capability gaps.

Physical Validation. We validate these findings on physical robot tasks using Qwen3.5-9B with and without CoT reasoning, evaluating each variant across five tasks spanning three cognitive categories: Common Sense / World Knowledge (selecting objects that require background knowledge to identify), Spatial reasoning (placing objects under relational constraints), and Perceptual understanding (identifying objects by their properties). As shown in Fig. 7, the non-thinking planner is fast but low in success, while CoT reasoning substantially raises success at a 20× latency cost. Across our five tasks, the non-thinking planner matches the Thinking variant on two tasks, while thinking yields a significant gain on the remaining three. Because this benefit is task-contingent, DIRECT escalates to the thinking model only where it helps, matching or exceeding the Thinking variant's success at significantly lower latency (the best router reaches 60.9% success versus the Thinking planner's 52.2% at less than half the latency; details in Appendix Table XXIII). DIRECT achieves close to the Oracle on both success and latency, and outperforms both OOD- and random-based routing.

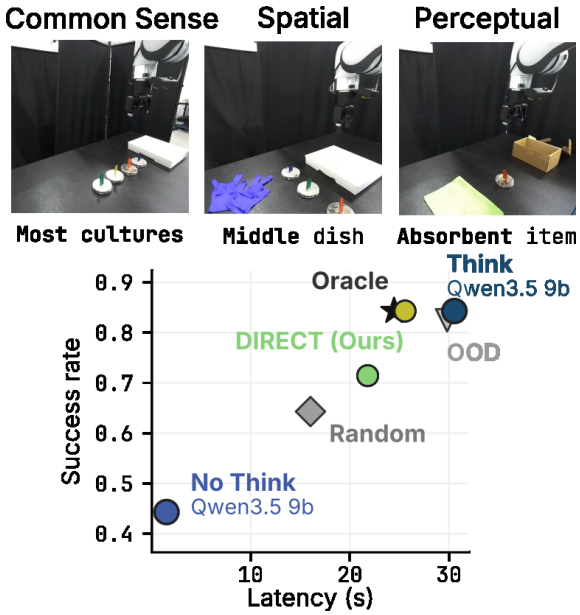


Fig. 7: **Reasoning axis on hardware.** Qwen3.5-9B Instruct vs. Think as high-level planners on the physical Franka DROID suite. DIRECT matches or exceeds the Thinking planner's task success at a fraction of its latency by escalating to chain-of-thought only on the ambiguous tasks that need it and defaulting to the fast no-think planner elsewhere.

TABLE II: **Multi-step grocery bagging (heaviest to lightest).** After each completed step, DIRECT routes to the appropriate model to plan the next, matching or exceeding the Thinking planner's success at lower latency.

Planner	Success (%)	Lat (s)
Qwen3.5-VL 9B I	47.62	2.19
Qwen3.5-VL 9B T	90.48	19.58
DIRECT	**95.24**	6.85

We further evaluate DIRECT on a multi-step grocery-

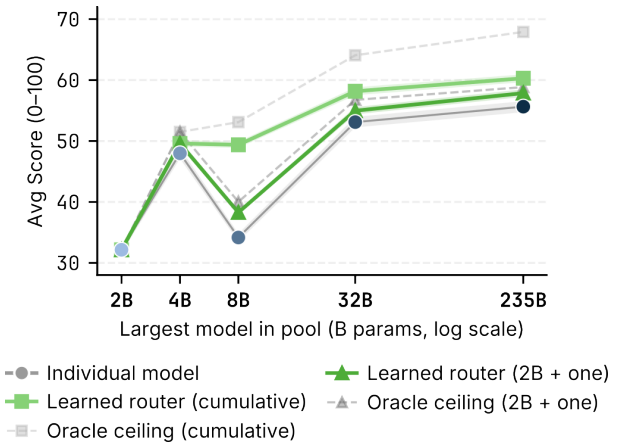


Fig. 8: Per-task routing across Qwen3-VL Instruct 2B–235B on VLABench [58].

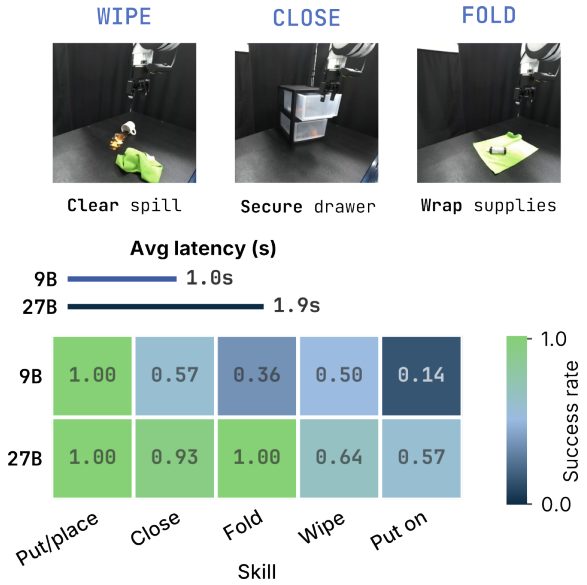


Fig. 9: Qwen3.5-VL 9B vs. 27B on DROID. Apart from `Put/Place`, the 27B planner raises success on every skill (heatmap) at $\sim 2\times$ latency.

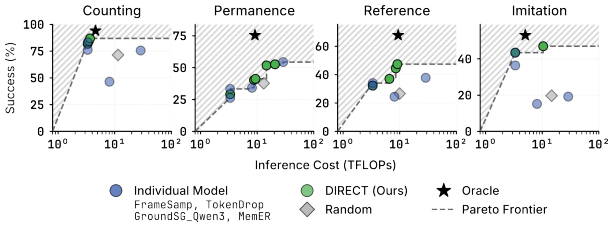


Fig. 10: RoboMME memory routing across difficulty suites. Success rate versus inference cost (TFLOPs) for individual memory architectures and DIRECT's routed Pareto frontier (dashed). In Reference the routed frontier outperforms the strongest single architecture at far lower cost than the most expensive specialist (`MemER`), evidence that no single model is optimal suite-wide.

bagging task that bags items from heaviest to lightest, where each completed step exposes a new scene, and the planner is re-invoked to plan the next (see Fig. 1, top). Since steps vary in reasoning demand, this experiment tests per-step allocation: DIRECT decides at each step whether routing to the thinking model is worth its cost. As shown in Table II, DIRECT, which routes step-wise, attains the highest progress success (95.24%, exceeding the Thinking model's 90.48%) at much lower latency, while the non-thinking model alone generally fails to complete the task successfully.

C. Routing on the Model Size Test-Time Compute Axis

Simulation Results. Because scale helps unevenly across skills, simply deploying the largest planner overspends on the many tasks where it adds little benefit. We use DIRECT to route over Qwen3-VL Instruct variants from 2B to 235B on VLABench, comparing per-task routing against the individual models and an oracle ceiling (Fig. 8). Consistent with Section III, individual-model performance is non-monotonic in scale. *Pairwise* routing between the 2B variant and a single larger model (Fig. 8) improves over either constituent at each size but inherits the pool's non-uniformity: with only a weak large model to escalate to, the 8B dip persists, though the curve still tracks its oracle ceiling. *Cumulative* routing over all sizes up to each point instead becomes monotonic: since each added variant only enlarges the collective skill set, the router never loses an earlier capability, maintaining 49.3% at 8B by falling back on the 4B model while reserving larger ones for tasks that demand them. For the most expensive model (32B), the cumulative router (2B+4B+8B+32B) improves performance by 5.1 points while reducing average latency by 32.4 seconds. DIRECT thus converts an erratic scaling curve into monotonic improvement, recovering each model's distinct skill coverage while discarding its weaknesses.

Physical Validation. We validate our observation from Section III on scaling model size using Qwen3.5-VL at 9B and 27B Instruct as embodied planners on the Franka DROID hardware across five household skills. As seen in Fig. 9, the 27B planner improves command success on every skill except for `Put/Place` at double the latency of the smaller model. This mirrors the simulation finding that model size primarily broadens the set of skills a planner can reliably command: while a 9B model could command basic skills like `Put/Place` (1.00) and is mediocre at `Wipe` (0.50), improved command on a broader set of skills including `Fold` (0.36 $\rightarrow$ 1.00) and `Close` (0.57 $\rightarrow$ 0.93) depend almost entirely on the larger 27B model. Even where the 27B helps, the gains are not uniform: `Wipe` only climbs to 0.64 and `Put on` to 0.57, while `Fold` and `Close` jump to near-saturation. The benefit of a larger planner is therefore concentrated on the specific skills a task demands rather than spread uniformly across the skill set (see Appendix Table XXIV).

D. Routing on the Memory Context Test-Time Compute Axis

Simulation Results. We price memory-experiment cost in FLOPs (two per model parameter per token), as each mechanism's planner base cost plus its memory overhead, yielding an $\approx 8.6\times$ spread from `FrameSamp` to `MemER` (Appendix G-D). The per-suite Pareto comparison (Fig. 10) shows that routing's value is itself non-uniform across memory families. In each suite, DIRECT generally achieves a success–cost point

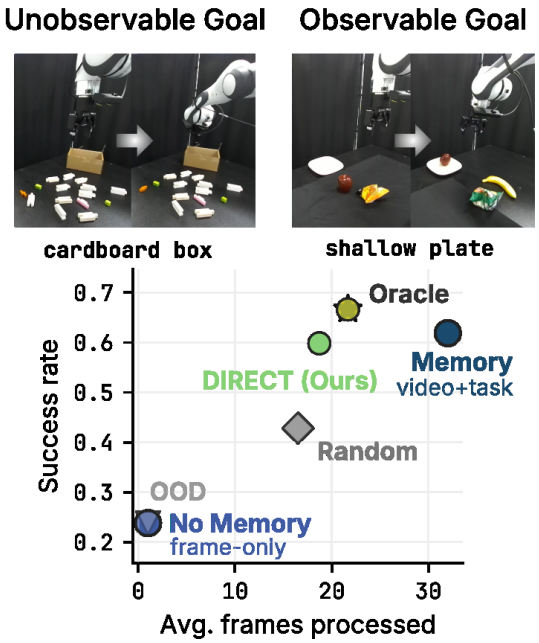


Fig. 11: **Memory axis on hardware.** Routing between a memory-free and a memory-augmented planner on the physical Franka DROID suite. DIRECT achieves close to oracle performance, invoking the costlier memory planner only on the history-dependent tasks where it pays off and avoiding its overhead on tasks observable from the current frame.

better than the most performant architecture, at a fraction of the most expensive specialist's (MemER) cost. This is only possible when a family decomposes into sub-types with different optimal recall structures: if one memory model were optimal family-wide, the router could at best match it by always selecting it. DIRECT's margin over the best specialist is therefore evidence that it has discovered *multiple successful decompositions* within a family. A residual gap to the oracle persists across families, whose source we leave open: it may be *unlearnable noise* that no router could predict, or *learnable structure* that current routers in DIRECT miss, implying a stronger framework could recover it. Resolving this is left for future work on the learning theory of embodied routing.

Physical Validation. We test memory routing on the physical Franka DROID setup across two task types that stress recall differently: *Unobservable Goal* tasks, whose target is specified only by task history no longer visible in the current frame, and *Observable Goal* tasks, whose constraints can be resolved from the present scene. The non-memory model is Gemini-ER 1.6 with a single frame and task as context. We construct a memory planner by providing Gemini-ER 1.6 with both a 40-frame video history along with a summary of task list histories. As shown in Fig. 11, DIRECT recovers near-oracle success rate while operating at substantially lower average frames processed than the full memory-augmented planner and outperforms the memory-free variant as well as the OOD and random routing baselines (details in Appendix Table XXV). This confirms the simulation finding in physical execution: memory-oriented compute is only worth its cost when a task genuinely depends on distant history. Furthermore, DIRECT can allocate test-time compute in the form of historical context selectively rather than paying the recall overhead on every task.

VI. Conclusion

We studied when and where test-time compute pays off in VLM embodied planning, showing across reasoning depth, model size, and memory history that its benefit is task-contingent rather than uniform: each axis yields a qualitatively distinct capability gain, and cheap configurations already suffice on a large fraction of tasks. Building on this, DIRECT routes each task to a planner whose capability profile matches its inferred demands, recovering near-frontier success at substantially lower cost on VLABench, RoboMME, and on Franka DROID hardware experiments. Our synthesis suggests we can infer task difficulty from scene and instruction context, suggesting practitioners can build effective routers from lightweight, embedding-based features. On the VLM embodied planners and benchmarks evaluated, these results suggest that treating the high-level planner as a dynamic variable is a practical path to recovering frontier-level success at meaningfully lower cost.

A. Limitations

The router is trained offline over a fixed pool, matching the standard deployment setting where candidate planners are known in advance. Adapting to a changing pool requires recollecting Q, C and retraining, and by construction the router allocates among pool members rather than synthesizing new capabilities. We treat cost as a deployment-bound proxy (*e.g.*, latency or FLOPs), so reported crossover points are representative of these settings rather than universal constants. Our per-call formulation conditions on features $\phi(x)$ from the initial observation, to keep router overhead negligible relative to any VLM planner, so we do not model cross-stage dependencies.

Acknowledgments

The authors would like to thank Zen Yasukawa and Steve Cousins for help with hardware setup. JD's work is supported by NSF Graduate Research Fellowship Program. MG's work is supported by NASA University Leadership Initiative, Stanford Center for Automated Reasoning, Stanford Google Cloud Platform, and Thinking Machines Lab. YA is currently employed by Wing; however, this research was conducted independently and not in connection with her employment. AS's work is supported by NSF Graduate Research Fellowship Program. MNA's work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC)'s Postdoctoral Fellowship and Mitacs Globalink Research Award. KL's work is supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement Number 00011869. This project is also supported by Stanford Robotics Center, Cornell University Gemini Academic Program, ONR grant N00014-22-1-2621, DoD High Performance Computing Modernization

Program, Stanford Marlowe GPU Cluster [29], the National Artificial Intelligence Research Resource Pilot and the Anvil supercomputer [43] (award NSF-OAC 2005632). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any aforementioned supporting entity.

References

[1] Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*, 2023.

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[3] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

[4] Shuanghao Bai, Jing Lyu, Wanqi Zhou, Zhe Li, Dakai Wang, Lei Xing, Xiaoguang Zhao, Pengwei Wang, Zhongyuan Wang, Cheng Chi, et al. Latent reasoning vla: Latent thinking and prediction for vision-language-action models. *arXiv preprint arXiv:2602.01166*, 2026.

[5] Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.

[6] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 27–30 Sep 2025.

[7] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.010.

[8] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multilinguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.137. URL https://aclanthology.org/2024.findings-acl.137/.

[9] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models more efficiently while reducing cost and latency. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

[10] William Chen, Suneel Belkhale, Suvir Mirchandani, Oier Mees, Danny Driess, Karl Pertsch, and Sergey Levine. Training strategies for efficient embodied reasoning. *arXiv preprint arXiv:2505.08243*, 2025.

[11] William Chen, Suneel Belkhale, Suvir Mirchandani, Karl Pertsch, Danny Driess, Oier Mees, and Sergey Levine. Training strategies for efficient embodied reasoning. In Joseph Lim, Shuran Song, and Hae-Won Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 365–391. PMLR, 27–30 Sep 2025.

[12] Inkook Chun, Seungjae Lee, Michael Albergo, Saining Xie, and Eric Vanden-Eijnden. Dynamic test-time compute scaling in control policy: Difficulty-aware stochastic interpolant policy. *Advances in Neural Information Processing Systems*, 38:51441–51462, 2026.

[13] Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. Robomme: Benchmarking and understanding memory for robotic generalist policies. *arXiv preprint arXiv:2603.04639*, 2026.

[14] Ronghao Dang, Jiayan Guo, Bohan Hou, Sicong Leng, Kehan Li, Xin Li, Jiangpin Liu, Yunxuan Mao, Zhikai Wang, Yuqian Yuan, et al. Rynnbrain: Open embodied

foundation models. *arXiv preprint arXiv:2602.14979*, 2026.

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

[16] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: an embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[17] Zhekai Duan, Yuan Zhang, Shikai Geng, Gaowen Liu, Joschka Boedecker, and Chris Xiaoxuan Lu. Fast ecot: Efficient embodied chain-of-thought via thoughts reuse. *arXiv preprint arXiv:2506.07639*, 2025.

[18] Milan Ganai, Katie Luo, Jonas Frey, Clark Barrett, and Marco Pavone. Self-supervised bootstrapping of action-predictive embodied reasoning. *arXiv preprint arXiv:2602.08167*, 2026.

[19] Google DeepMind. Gemini 3 flash. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf, 2025. Model card.

[20] Google DeepMind. Gemini robotics-er 1.6. https://deepmind.google/models/model-cards/gemini-robotics-er-1-6/, 2025. Model card.

[21] Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40041–40076, 2026.

[22] Songhao Han, Boxiang Qiu, Yue Liao, Siyuan Huang, Chen Gao, Shuicheng Yan, and Si Liu. Robocerebra: A large-scale benchmark for long-horizon robotic manipulation evaluation, 2025. URL https://arxiv.org/abs/2506.06677.

[23] Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. Don't overthink it. preferring shorter thinking chains for improved llm reasoning. *arXiv preprint arXiv:2505.17813*, 2025.

[24] Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.5 v and glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*, 2025.

[25] Wenlong Huang, Pieter Abbeel, Deepak Tamane, Karol Hausman, Kuan Fang, Dhruv Shah, Igor Mordatch, Andy Zaldivar, Sergio Guadarrama, and Fei Lu. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[26] Zhehao Huang, Baijiong Lin, Jingyuan Zhang, Jingying Wang, Yuhang Liu, Ning Lu, Tao Li, and Xiaolin Huang. Vl-routerbench: A benchmark for vision-language model routing, 2025. URL https://arxiv.org/abs/2512.23562.

[27] Yuheng Ji, Huajie Tan, Jiayu Shi, Xiaoshuai Hao, Yuan Zhang, Hengyuan Zhang, Pengwei Wang, Mengdi Zhao, Yao Mu, Pengju An, et al. Robobrain: A unified brain model for robotic manipulation from abstract to concrete. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1724–1734, 2025.

[28] Daniel Kahneman. Thinking, fast and slow. *Farrar, Straus and Giroux*, 2011.

[29] Craig Kapfer, Kurt Stine, Balasubramanian Narasimhan, Christopher Mentzel, and Emmanuel Candes. Marlowe: Stanford's gpu-based computational instrument, 2025.

[30] Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. Adaptive vision-language model routing for computer use agents. *arXiv preprint arXiv:2603.12823*, 2026.

[31] Zhuoyang Liu, Jiaming Liu, Hao Chen, Jiale Yu, Ziyu Guo, Chengkai Hou, Chenyang Gu, Xiangju Mi, Renrui Zhang, Kun Wu, et al. Last _{0}: Latent spatio-temporal chain-of-thought for robotic vision-language-action model. *arXiv preprint arXiv:2601.05248*, 2026.

[32] Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. Routerarena: An open platform for comprehensive comparison of llm routers. *arXiv preprint arXiv:2510.00202*, 2025.

[33] Haoxuan Ma, Guannan Lai, and Han-Jia Ye. Mmr-bench: A comprehensive benchmark for multimodal llm routing. *arXiv preprint arXiv:2601.17814*, 2026.

[34] Weixin Mao, Weiheng Zhong, Zhou Jiang, Dong Fang, Zhongyue Zhang, Zihan Lan, Haosheng Li, Fan Jia, Tiancai Wang, Haoqiang Fan, et al. Robomatrix: A skill-centric hierarchical framework for scalable robot task planning and execution in open-world. *arXiv preprint arXiv:2412.00171*, 2024.

[35] Multi-Linguality Multi-Functionality Multi-Granularity. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024.

[36] NVIDIA. Cosmos-reason2. https://huggingface.co/nvidia/Cosmos-Reason2-8B, 2025. Model card.

[37] Isaac Ong, Lingjiao Chen, Shiyi Huang, Matei Zaharia, and James Zou. RouteLLM: Learning to route LLMs with open models. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[38] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features

without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024.

[39] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of machine learning research*, 5(Jan):101–141, 2004.

[40] Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

[41] Rohan Sinha, Amine Elhafsi, Christopher Agia, Matt Foutter, Edward Schmerling, and Marco Pavone. Real-Time Anomaly Detection and Reactive Planning with Large Language Models. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.114.

[42] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[43] X. Carol Song, Preston Smith, Rajesh Kalyanam, Xiao Zhu, Eric Adams, Kevin Colby, Patrick Finnegan, Erik Gough, Elizabett Hillery, Rick Irvine, Amiya Maji, and Jason St. John. Anvil - system architecture and experiences from deployment and early user operations. In *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You*, PEARC '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391610. doi: 10.1145/3491418.3530766. URL https://doi.org/10.1145/3491418.3530766.

[44] Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Memer: Scaling up memory for robotic control via experience retrieval. In *The Fourteenth International Conference on Learning Representations*, 2025.

[45] Xin Tang, Youfang Han, Fangfei Gou, Wei Zhao, Xin Meng, Yang Yu, Jinguo Zhang, Yuanchun Shi, Yuntao Wang, and Tengxiang Zhang. Ecvl-router: Scenario-aware routing for vision-language models. *arXiv preprint arXiv:2510.27256*, 2025.

[46] Gemini Robotics Team, Abbas Abdolmaleki, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Ashwin Balakrishna, Nathan Batchelor, Alex Bewley, Jeff Bingham, et al. Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025.

[47] Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[48] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*, 2025.

[49] Xi Wang, James McInerney, Lequn Wang, and Nathan Kallus. Entropy after </think> for reasoning model early exiting. *arXiv preprint arXiv:2509.26522*, 2025.

[50] 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.

[51] Jason Wei, Xuezhi Wang, Dale Schuurman, Maarten Bazarewski, Brian Ichter, Ricardo Liu, Denny Zhou, and Quoc V Le. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[52] Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. Towards thinking-optimal scaling of test-time compute for llm reasoning. *Advances in Neural Information Processing Systems*, 38:43605–43631, 2026.

[53] Yi Yang, Jiaxuan Sun, Siqi Kou, Yihan Wang, and Zhijie Deng. Lohovla: A unified vision-language-action model for long-horizon embodied tasks. *arXiv preprint arXiv:2506.00411*, 2025.

[54] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[55] Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024.

[56] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[57] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11975–11986, 2023.

[58] Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11142–11152, October 2025.

[59] Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. *arXiv preprint arXiv:2502.01839*, 2025.

[60] Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-

of-thought reasoning for vision-language-action models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1702–1713, 2025.

[61] Yandong Zheng, Hui Zhu, Rongxing Lu, Yunguo Guan, Songnian Zhang, Fengwei Wang, Jun Shao, and Hui Li. Efficient and privacy-preserving aggregated reverse knn query over crowd-sensed data. *IEEE Transactions on Information Forensics and Security*, 18:4285–4299, 2023.

[62] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

DIRECT: When and Where Should You Allocate Test-Time Compute in Embodied Planners?

Appendix

APPENDIX A
NOTATION AND MODEL REFERENCES

A. Notation

Table III (below) collects the central symbols used in the main paper and this supplement; minor, section-local notation is defined where it appears.

Symbol	Meaning
Routing formulation (Sec. 4 of the main paper)	
$\mathcal{M}$	Planner pool: the K candidate model variants the router selects among.
N	Number of tasks / evaluation samples.
$x = (I, \ell)$	Task input: scene observation I + language instruction ℓ.
Q, C	Per-(task, variant) quality and cost matrices ($N \times K$); entries $q_{i,k} \in [0, 1]$, $c_{i,k} \geq 0$.
$\bar{c}_k$	Training-time average cost of variant k.
U	Preference (utility) function.
$k_i^\star$	Routing target for task i: $\arg\max_k U(q_{i,k}, c_{i,k})$.
λ	Quality tolerance defining the classification target.
$\phi(x)$	Fused multimodal embedding of x; the router's input features.
$r_\theta(x)$	Variant the trained router selects for input x.
$\hat{q}_{i,k}$	Regression head's predicted quality for sample i and planner k.
$\hat{c}_{i,k}$	Regression head's predicted cost for sample i and planner k.
Evaluation metric	
$\eta(\beta)$	Routing efficiency: preference-weighted harmonic mean of scaled quality q_s and cost-savings $(1 - c_s)$.
β	Cost-emphasis weight in η *and* also re-used for the rank-score cost-emphasis (in App. B-D and App. F-A) .
α	Regression tie window (Base ϵ, Tie $0.1 + \epsilon$).

Conventions. Scores and η values in the ablation tables are $\times 100$; cost is the VLM Planner's latency in seconds unless otherwise noted (ex. given in TFLOPs).

B. Model References

Throughout the paper we refer to models by short colloquial names; Table IV maps each to its reference and its model id (Hugging Face repo, GitHub project, or API name) for reproducibility.

TABLE IV: Models used in the paper: reference and model ID. Open-weight rows give the Hugging Face repo (or GitHub project); API rows are closed-weight and accessed via the listed endpoint.

Name in text	Ref.	Model id
Planner VLMs		
Qwen3-VL-{2B,8B}-Instruct	[3]	`Qwen/Qwen3-VL-{2B,8B}-Instruct`
Qwen3-VL-{4B,8B,33B,235B}-FP8-Instruct	[3]	`Qwen/Qwen3-VL-{4B,...,235B}-Instruct-FP8`
Qwen3-VL-8B-Thinking	[3]	`Qwen/Qwen3-VL-8B-Thinking`
Qwen3.5-{9B,27B}	[47]	`Qwen/Qwen3.5-{9B, 27B}`
GLM-4.1V-Thinking	[24]	`zai-org/GLM-4.1V-9B-Thinking`
GLM-4.6V-Flash	[24]	`zai-org/GLM-4.6V-Flash`
Cosmos-Reason2	[36]	`nvidia/Cosmos-Reason2-8B`
RynnBrain-8B	[14]	`Alibaba-DAMO-Academy/RynnBrain-8B`
RynnBrain-8B-Plan	[14]	`Alibaba-DAMO-Academy/RynnBrain-Plan-8B`
Gemini-3-Flash*	[19]	`gemini-3-flash-preview`
Gemini-Robotics-ER 1.6**	[20]	`gemini-robotics-er-1.6-preview`
Encoders (router embeddings)		
SigLIP-2 (large)	[48]	`google/siglip2-large-patch16-512`
SigLIP-2 (so400m)	[48]	`google/siglip2-so400m-patch14-384`
SigLIP (large)	[56]	`google/siglip-large-patch16-384`
DINOv2 (large)	[38]	`facebook/dinov2-large`
BGE-M3 (text)	[8]	`BAAI/bge-m3`
π_0 (VLA policy embedding)	[7]	`lerobot/pi0_base`
Low-level VLA policy		
$\pi_{0.5}$ (DROID-finetuned)	[6]	`Physical-Intelligence/openpi` (`openpi05`)

* Set thinking level to `MINIMAL` or use default.
** Set thinking budget to 0 (thinking off) vs default budget.

APPENDIX B
ROUTER DESIGN DECISIONS

We build on the VL-RouterBench [26] structure for training and evaluating a collection of diverse routers. This includes making use of their scaffolding for the router architectures and the classification-headed routers, as well as their cost/quality matrix and embedding-generation pipelines, and training/evaluation steps. We introduce our own pipeline adapted to our input dataset and scoring structure and modify the existing methods to fit our project's unique requirements. Beyond classification routing, we repurpose the existing router architectures to perform *regression* directly on cost and score, and define a set of test-time decision rules (utility functions) that select the VLM planner from the predicted score and cost. We also extended the vision embedding-generation pipeline with newer vision encoders (i.e., two from the SigLIP-2 family).

A. Embeddings and Fusion Strategies

The router input is an embedding of the task. We use a **text** embedding of the language instruction (text encoder) and a **vision** embedding of the scene observation (vision encoder); a *fusion method* specifies how these two are combined into the single vector the router consumes. We additionally consider a **VLA** (policy) embedding from the π_0 VLA, which enters only through the policy-embedding strategy below. We ablate:

- `concat`: sequence concatenation of the text and image embeddings.
- `normalize_concat`: the same after ℓ_2-normalizing the text and image embeddings.
- `average`: mean of the text and image embeddings.
- `weighted_average`: text-weighted mean of the text and image embeddings (text weight $\in \{0.25, 0.75\}$, denoted `w0.25`/`w0.75` or `weighted_average_ text025`/`weighted_average_text075`).
- *text embedding*: the instruction (text) embedding alone.
- *vision embedding*: the scene (image) embedding alone.
- *policy embedding*: Following the feature extraction strategy of SAFE [21], we use the π_0 VLA policy's prefix hidden states (image+text), masked-mean-pooled into a single vector.

Quantitative comparison appears in the embedding ablations in Appendix C. We additionally ablate both the fusion strategy and the vision encoder for every routing model pair; the top-performing results are also reported in Appendix C.

B. Router Architectures

All base router architectures were already implemented by VL-RouterBench [26]; we adapt their training signal and inference rule (Appendix B-C). We summarize each architecture below; the chosen hyperparameters are listed in Table VI.

- **Linear**: a single linear layer (`nn.Linear`). Despite the "soft-label" naming, for classification it reduces to a hard-label objective, since our binarized scores yield exactly one correct planner.

- **MLP**: a small two-hidden-layer network.
- **KNN**: distance-weighted vote over the k nearest training embeddings.
- **PR-KNN** [61]: preference-ranking KNN, aggregating neighbors by Copeland-score voting; often identical to KNN in our settings.
- **K-means**: a nearest-centroid classifier with one prototype per planner, the mean embedding of the training samples that planner is the labeled winner, routing each query to the most cosine-similar prototype.
- **OVR** [39]: one-vs-rest, one logistic regressor per planner.

The above describes base behavior, modifications to behavior for regression are described in B-D.

C. Router Heads

We consider two router-head families. Let $\phi(x)$ denote the fused embedding of input x, $\mathcal{M}$ the planner pool, Q the per-variant quality matrix, and C the per-variant cost matrix.

Classification. Given $\phi(x)$, the head directly predicts a planner choice from $\mathcal{M}$ based on training labels of optimal planner choice per sample, as described below (B-D).

Regression. Given $\phi(x)$, regression heads predict cost and quality, trained on the cost and quality of individual samples from Q and C. These predicted cost and quality values are then used to choose a planner from $\mathcal{M}$ based on the decision functions defined below (B-D).

D. Preference and Test-Time Decision Functions

Given predicted score and cost, the chosen planner is determined by utility rule:

$$k_i^\star \;=\; \arg\max_k \;\; U(q_{i,k},\, c_{i,k}) \tag{3}$$

Label definition (classification). Training labels are determined by the highest-scoring planner, with ties broken by lowest latency (cost), this is the same as generating labels with the following utility rule with $\alpha = \epsilon$:

$$U_{\text{class}}(q_{i,k},\, c_{i,k}) \;=\; \frac{1}{c_{i,k}} \cdot \mathbf{1}[\max_{k'} q_{i,k'} - q_{i,k} < \alpha\,] \tag{4}$$

We then train the routers to directly predict a model based on these training labels.

Test-time selection (regression).

The example utility function given in the main paper is:

$$U_{\text{reg}}(\hat{q}_{i,k},\, \hat{c}_{i,k}) \;=\; \frac{1}{\hat{c}_{i,k}} \cdot \mathbf{1}[\max_{k'} \hat{q}_{i,k'} - \hat{q}_{i,k} < \alpha\,], \tag{5}$$

which is used for the following **Base** and **Tie** implementations.

Here are example decision methods used:

- **Base** ($U_{\text{reg}}(\hat{q}_{i,k},\, \hat{c}_{i,k})$ with $\alpha = \epsilon$): highest predicted score, ties broken by lowest latency.
- **Tie** ($U_{\text{reg}}(\hat{q}_{i,k},\, \hat{c}_{i,k})$ with $\alpha = 0.1 + \epsilon$): all predicted scores within 0.1 of the maximum are treated as equal, ties broken by lowest latency.
- **Rank**: uses the planner that maximizes the VL-RouterBench-derived rank-score $S(\beta)$ [26]. This utility score is computed for each sample i and planner k using the predicted quality $\hat{q}_{ik}$ and cost $\hat{c}_{ik}$, with costs log-normalized against the minimum and maximum values observed in the training set. We use tradeoff parameter $\beta = 0.1$.

$$C_{ik}^{\text{norm}} = \frac{\log_2(c_{\max} + \varepsilon) - \log_2\big(\,\text{clip}(\hat{c}_{i,k},\, c_{\min},\, c_{\max}) + \varepsilon\big)}{\log_2(c_{\max} + \varepsilon) - \log_2(c_{\min} + \varepsilon) + \varepsilon} \tag{6}$$

$$S_{i,k}(\beta) = \frac{(1+\beta)\,\hat{q}_{i,k}\, C_{i,k}^{\text{norm}}}{\beta\,\hat{q}_{i,k} + C_{i,k}^{\text{norm}} + \varepsilon} \tag{7}$$

$$k_i^\star = \arg\max_k \;\; S_{i,k}(\beta) \tag{8}$$

Regression-based score and cost prediction. In every regression variant, the router predicts for each planner, a continuous quality score (regressed on the ground-truth scores) and a cost (regressed on the ground-truth costs). In all cases the multi-output predictors fit each planner independently (sharing only the input features). The two predictors are:

- **OVR**: *score* — one `Ridge` regressor per planner, each fit to that planner's scores; *cost* — the same per-planner `Ridge`, here expressed as a `MultiOutputRegressor` over all planners' costs.

- **KNN**: *score* — a k-NN regressor (distance-weighted mean of the k nearest training neighbors' scores, per planner); *cost* — a second k-NN regressor over the same neighbors, predicting each planner's cost.
- **PR-KNN**: *score* — distance-weighted mean of the k neighbors' scores per planner; *cost* — distance-weighted mean of those same neighbors' costs.
- **K-means**: each prototype stores a per-planner *score* lookup (the mean score of the samples assigned to that planner) and a per-planner *cost* lookup (the mean cost of that group); a query's predicted score and cost are the similarity-weighted combinations of these lookups across prototypes.
- **Linear**: *score* — a linear model with one output per planner regressed on ground-truth scores (per-planner `Ridge` on CPU, or a linear layer with MSE loss on GPU); *cost* — a separate per-planner `Ridge` fit to costs.
- **MLP**: *score* — a multilayer perceptron with one output per planner trained as a regressor (MSE) on ground-truth scores; *cost* — a separate per-planner `Ridge` fit to costs.

Given the predicted per-planner score $\hat{q}_{i,k}$ and cost $\hat{c}_{i,k}$, the planner is then chosen by the selection rule (base, tie, or rank).

E. Routing Cost and Latency

A routing decision costs the embedding extraction plus the router's own forward pass, and both are negligible relative to the planner (seconds to tens of seconds per query), so the estimates here are deliberately coarse. Router inputs are formed from a text encoder (BGE-M3) [8] over the instruction, a vision encoder over the observation (we ablate SigLIP [56], two SigLIP-2 [48] variants, and DINOv2 [38]); extraction dominates and is itself dominated by the vision encoder, with text embeddings amortizing to a few milliseconds per sample and a single vision embedding estimated on the order of $\sim$10–25 ms if optimized (observed $\sim$25–45 ms amortized in our un-optimized batched runs; a π_0-based VLA encoder [7], which we include only for ablation studies over fusion method, adds additional overhead). The trained routers themselves are extremely light— on the order of 0.0088–1.82 ms (amortized) per sample across architectures (derived from latency profiling for GLM 4.1V Thinking / GLM 4.6V Flash routing decisions over all router architectures, ablated over the 4 vision encoders using the `normalize_concat` fusion method. The GPU was pre-warmed and a batch of size 16 was run 50 times per router/ablation). For fused text and vision embeddings, a routing decision totals roughly $\sim$20–50 ms amortized per query, two to three orders of magnitude below planner latency; embeddings could additionally be cached and reused across repeated queries on the same observation for further optimization.

F. Per-Experiment Configuration

Table V records the vision encoder and fusion method used for each experiment in the main paper and appendix.

TABLE V: Vision encoder and fusion method per experiment.

Experiment	Fusion	Vision encoder
Chain-of-Thought (main Table 1)	`normalize_concat`	SigLIP-2 (large)
Qwen size scaling	`normalize_concat`	SigLIP (large)
RoboMME (memory)	`concat`	SigLIP (large)
Hardware (Franka/DROID)	`normalize_concat`	SigLIP (large)

APPENDIX C
ROUTER ABLATIONS: ARCHITECTURE, FUSION, VISION ENCODER, AND HEADS

For each routing model pair we ablate the router design space along three axes — the router *architecture* (linear, MLP, KNN, PR-KNN, K-means, OVR; Appendix B-B), the embedding *fusion* method (Appendix B-A), and the *vision encoder* — and present two tables per pair. The first is a **router-architecture ablation**: fixing the embedding configuration to the deployed setting (`normalize_concat` fusion with the SigLIP-2 (large) encoder; Appendix B-F), it reports the best router per architecture by η, the routing-efficiency metric of the main paper (the preference-weighted harmonic mean of scaled quality and cost, reported at $\beta = 0.1$). The second is a combined **embedding ablation** that sweeps (i) the fusion method (vision encoder fixed) and (ii) the vision encoder (fusion fixed), reporting the best router per setting by average score and by η; the decision *head* (classification vs. the regression variants; Appendix B-C) and the architecture are encoded in each router's name (convention below). All tables also list the fixed baselines and oracles (Appendix F); average score and η are multiplied by 100 and cost is in seconds. Router inference hyperparameters are in Appendix C-A, and the complete per-router sweep for a representative pair is in Table XXIX (Appendix I).

Router naming convention. Each trained router is specified by its configuration: {`architecture`} {`fusion`} {`head`}:

- *architecture*: `linear`, `mlp` (with hidden sizes, e.g., `h2048_1024`), `knn` (with neighbor count, e.g., `k5`), `prknn`, `kmeans`, or `ovr` (Appendix B.2).

- *fusion*: the embedding fusion strategy (Appendix B.1): `normalize_concat`, `concat`, `average`, `weighted_average_w0.25_w0.75`, `only_text`, `only_image`, or `only_vla`. This duplicates the "Fusion Method" / "Vision Encoder" column of each table.
- *head*: the router head / decision rule (Appendix B.3): no suffix denotes classification; `regression`, `rank_regression`, and `tie_regression` denote the regression variants; `soft_`λ`0.1` denotes the soft-label linear router with $\lambda = 0.1$.

A. Router Inference Hyperparameters

During inference, standard classification routers employ conventional decision rules: $\arg\max$ over probabilities or logits for Linear, MLP, and OVR models, and majority voting for KNN. For alternative formulations, PR-KNN utilizes Copeland-score voting to aggregate pairwise preferences, while K-means assigns inputs to the nearest centroid to retrieve its learned planner preference. Note that regression variants (`--regression`, `--rank_regression`, `--tie_regression`) share these base architectures but modify the inference rule to suit their training signals (detailed in Appendix B-D).

Table VI lists the core configuration and hyperparameters for each base router architecture.

TABLE VI: Router architecture hyperparameters (base classification configuration).

Router	Hyperparameters
KNN	$k = 5$, fusion `concat`
Linear	penalty `l2`, $C = 1.0$, `train_lambda`$= 0.1$, seed $= 42$
MLP	hidden $= [2048, 1024]$, `relu`, $\alpha = 10^{-4}$ (L2), seed $= 42$
OVR	one `LogisticRegression` / planner (sklearn default: L2, $C = 1.0$)
PR-KNN	$k = 5$, metric `cosine`, weights `distance`
K-means	`normalize`=True (L2), `lambda_cost`$= 0.0$

B. GLM-4.1V-Thinking vs. GLM-4.6V-Flash

TABLE VII: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
Linear	soft-label (λ=0.1)	51.79	31.82	75.26	Trained Router
MLP [2048,1024]	classification	50.19	30.13	73.12	Trained Router
KNN (k=5)	classification	51.27	31.99	73.83	Trained Router
PR-KNN (k=5)	classification	51.27	31.99	73.83	Trained Router
K-means	classification	48.51	26.20	71.41	Trained Router
OVR	tie regression	49.57	31.61	70.08	Trained Router
expensive global	—	54.02	45.10	0.00	Fixed Baseline
strongest global	—	54.02	45.10	0.00	Fixed Baseline
strongest per category	—	54.02	45.10	0.00	Fixed Baseline
random	—	40.02	22.39	46.33	Fixed Baseline
cheapest global	—	27.24	2.11	0.00	Fixed Baseline
ood baseline	—	49.83	39.60	54.52	OOD Baseline
accuracy oracle	—	55.17	27.96	87.94	Oracle
η oracle	—	55.17	33.32	80.59	Oracle

TABLE VIII: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	MLP [2048,1024]	regression	53.47	38.52	64.02	Trained Router
average	best η	Linear	tie regression	49.51	25.83	74.46	Trained Router
concat	best avg score	MLP [2048,1024]	regression	53.43	38.26	64.91	Trained Router
concat	best η	Linear	soft-label (λ=0.1)	51.79	31.82	75.26	Trained Router
normalize_concat	best avg score	MLP [2048,1024]	regression	53.63	38.80	63.20	Trained Router
normalize_concat	best η	Linear	soft-label (λ=0.1)	51.79	31.82	75.26	Trained Router
vision embedding	best avg score	MLP [2048,1024]	rank regression	53.88	39.41	61.02	Trained Router
vision embedding	best η	MLP [2048,1024]	classification	51.47	31.68	74.68	Trained Router
text embedding	best avg score	MLP [2048,1024]	regression	53.90	42.83	37.41	Trained Router
text embedding	best η	Linear	soft-label (λ=0.1)	51.82	31.86	75.29	Trained Router
policy embedding	best avg score	KNN (k=5)	rank regression	53.74	43.41	30.67	Trained Router
policy embedding	best η	KNN (k=5)	classification	50.94	32.34	72.60	Trained Router
weighted_average_text025	best avg score	KNN (k=5)	rank regression	53.51	42.93	36.20	Trained Router
weighted_average_text025	best η	Linear	soft-label (λ=0.1)	52.17	33.23	74.20	Trained Router
weighted_average_text075	best avg score	MLP [2048,1024]	regression	53.63	43.12	34.11	Trained Router
weighted_average_text075	best η	Linear	soft-label (λ=0.1)	51.69	32.07	74.72	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	MLP [2048,1024]	regression	53.95	44.09	20.76	Trained Router
dinov2-large	best η	Linear	tie regression	49.91	27.20	74.71	Trained Router
siglip-large-patch16-384	best avg score	MLP [2048,1024]	regression	53.47	37.95	66.02	Trained Router
siglip-large-patch16-384	best η	Linear	tie regression	49.92	26.96	74.92	Trained Router
siglip2-large-patch16-512	best avg score	MLP [2048,1024]	regression	53.63	38.80	63.20	Trained Router
siglip2-large-patch16-512	best η	Linear	soft-label (λ=0.1)	51.79	31.82	75.26	Trained Router
siglip2-so400m-patch14-384	best avg score	KNN (k=5)	regression	53.56	38.40	64.59	Trained Router
siglip2-so400m-patch14-384	best η	Linear	tie regression	49.77	26.81	74.59	Trained Router
—	—	strongest per category	—	54.02	45.10	0.00	Fixed Baseline
—	—	expensive global	—	54.02	45.10	0.00	Fixed Baseline
—	—	strongest global	—	54.02	45.10	0.00	Fixed Baseline
—	—	random	—	40.02	22.39	46.33	Fixed Baseline
—	—	cheapest global	—	27.24	2.11	0.00	Fixed Baseline
—	—	ood baseline (average)	—	54.02	45.10	0.00	OOD Baseline
—	—	ood baseline (concat)	—	49.83	39.60	54.52	OOD Baseline
—	—	ood baseline (normalize_concat)	—	49.83	39.60	54.52	OOD Baseline
—	—	ood baseline (vision embedding)	—	28.63	3.36	5.44	OOD Baseline
—	—	ood baseline (text embedding)	—	37.28	22.40	37.03	OOD Baseline
—	—	ood baseline (policy embedding)	—	54.02	45.10	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	54.02	45.10	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	28.13	3.92	3.50	OOD Baseline
—	—	ood baseline (dinov2-large)	—	49.27	39.65	53.52	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	49.11	38.96	55.64	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	49.83	39.60	54.52	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	49.74	39.46	54.92	OOD Baseline
—	—	accuracy oracle	—	55.17	27.96	87.94	Oracle
—	—	η oracle	—	55.17	33.32	80.59	Oracle

C. GLM-4.1V-Thinking vs. RynnBrain-8B-Plan

TABLE IX: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	**Selection Head**	**Avg Score**	**Avg Cost (s)**	η	**Category**
Linear	regression	54.33	27.50	63.94	Trained Router
MLP [2048,1024]	regression	54.11	26.20	63.18	Trained Router
KNN (k=5)	regression	53.23	27.16	57.33	Trained Router
PR-KNN (k=5)	regression	53.23	27.16	57.33	Trained Router
K-means	classification	52.23	27.91	50.66	Trained Router
OVR	regression	53.21	29.06	56.31	Trained Router
strongest per category	—	54.29	42.16	37.19	Fixed Baseline
expensive global	—	54.02	45.10	0.00	Fixed Baseline
strongest global	—	54.02	45.10	0.00	Fixed Baseline
random	—	49.44	22.17	32.51	Fixed Baseline
cheapest global	—	45.25	1.67	0.00	Fixed Baseline
ood baseline	—	52.56	32.72	50.42	OOD Baseline
accuracy oracle	—	58.64	17.65	94.98	Oracle
η oracle	—	58.60	22.90	91.79	Oracle

TABLE X: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	Linear	regression	53.93	27.98	61.25	Trained Router
average	best η	MLP [2048,1024]	regression	53.70	24.47	61.30	Trained Router
concat	best avg score	Linear	regression	54.33	27.49	63.94	Trained Router
concat	best η	Linear	regression	54.33	27.49	63.94	Trained Router
normalize_concat	best avg score	Linear	regression	54.33	27.50	63.94	Trained Router
normalize_concat	best η	Linear	regression	54.33	27.50	63.94	Trained Router
vision embedding	best avg score	MLP [2048,1024]	regression	53.71	39.50	46.66	Trained Router
vision embedding	best η	Linear	regression	53.24	26.84	57.49	Trained Router
text embedding	best avg score	MLP [2048,1024]	regression	54.32	34.49	58.37	Trained Router
text embedding	best η	Linear	regression	53.68	27.45	59.98	Trained Router
policy embedding	best avg score	Linear	regression	53.57	32.57	56.27	Trained Router
policy embedding	best η	KNN (k=5)	regression	53.08	25.78	56.88	Trained Router
weighted_average_text025	best avg score	MLP [2048,1024]	regression	54.32	33.22	59.73	Trained Router
weighted_average_text025	best η	Linear	regression	53.93	28.29	61.09	Trained Router
weighted_average_text075	best avg score	Linear	regression	53.83	27.81	60.72	Trained Router
weighted_average_text075	best η	Linear	regression	53.83	27.81	60.72	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	MLP [2048,1024]	regression	54.04	30.74	60.26	Trained Router
dinov2-large	best η	Linear	regression	53.94	28.24	61.16	Trained Router
siglip-large-patch16-384	best avg score	Linear	regression	54.10	27.38	62.61	Trained Router
siglip-large-patch16-384	best η	Linear	regression	54.10	27.38	62.61	Trained Router
siglip2-large-patch16-512	best avg score	Linear	regression	54.33	27.50	63.94	Trained Router
siglip2-large-patch16-512	best η	Linear	regression	54.33	27.50	63.94	Trained Router
siglip2-so400m-patch14-384	best avg score	Linear	regression	54.34	27.35	64.04	Trained Router
siglip2-so400m-patch14-384	best η	Linear	regression	54.34	27.35	64.04	Trained Router
—	—	strongest per category	—	54.29	42.16	37.19	Fixed Baseline
—	—	expensive global	—	54.02	45.10	0.00	Fixed Baseline
—	—	strongest global	—	54.02	45.10	0.00	Fixed Baseline
—	—	random	—	49.44	22.17	32.51	Fixed Baseline
—	—	cheapest global	—	45.25	1.67	0.00	Fixed Baseline
—	—	ood baseline (average)	—	45.25	1.67	0.00	OOD Baseline
—	—	ood baseline (concat)	—	52.56	32.72	50.42	OOD Baseline
—	—	ood baseline (normalize_concat)	—	52.56	32.72	50.42	OOD Baseline
—	—	ood baseline (vision embedding)	—	45.25	1.67	0.00	OOD Baseline
—	—	ood baseline (text embedding)	—	45.25	1.67	0.00	OOD Baseline
—	—	ood baseline (policy embedding)	—	54.02	45.10	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	45.25	1.67	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	45.25	1.67	0.00	OOD Baseline
—	—	ood baseline (dinov2-large)	—	47.82	11.29	20.63	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	52.48	31.60	50.64	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	52.56	32.72	50.42	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	52.48	31.99	50.41	OOD Baseline
—	—	accuracy oracle	—	58.64	17.65	94.98	Oracle
—	—	η oracle	—	58.60	22.90	91.79	Oracle

D. GLM-4.6V-Flash-Thinking vs. GLM-4.6V-Flash

TABLE XI: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	**Selection Head**	**Avg Score**	**Avg Cost (s)**	η	**Category**
Linear	regression	44.69	16.97	74.92	Trained Router
MLP [2048,1024]	tie regression	43.59	15.89	73.08	Trained Router
KNN (k=5)	tie regression	44.15	16.44	74.14	Trained Router
PR-KNN (k=5)	tie regression	44.15	16.44	74.14	Trained Router
K-means	classification	43.53	16.00	72.68	Trained Router
OVR	tie regression	42.22	14.84	69.27	Trained Router
expensive global	—	46.09	24.08	0.00	Fixed Baseline
strongest per category	—	46.09	24.08	0.00	Fixed Baseline
strongest global	—	46.09	24.08	0.00	Fixed Baseline
random	—	35.97	13.24	43.68	Fixed Baseline
cheapest global	—	27.24	2.11	0.00	Fixed Baseline
ood baseline	—	42.73	20.85	55.34	OOD Baseline
accuracy oracle	—	47.47	13.37	91.28	Oracle
η oracle	—	47.29	16.00	84.94	Oracle

TABLE XII: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	KNN (k=5)	regression	45.65	19.44	69.94	Trained Router
average	best η	KNN (k=5)	tie regression	43.83	15.60	74.39	Trained Router
concat	best avg score	KNN (k=5)	rank regression	45.73	22.68	41.31	Trained Router
concat	best η	Linear	regression	44.69	16.97	74.92	Trained Router
normalize_concat	best avg score	KNN (k=5)	rank regression	45.73	22.68	41.31	Trained Router
normalize_concat	best η	Linear	regression	44.69	16.97	74.92	Trained Router
vision embedding	best avg score	MLP [2048,1024]	regression	45.95	20.57	64.44	Trained Router
vision embedding	best η	KNN (k=5)	tie regression	44.24	16.21	74.88	Trained Router
text embedding	best avg score	MLP [2048,1024]	regression	45.38	22.82	38.45	Trained Router
text embedding	best η	Linear	regression	44.86	17.56	74.05	Trained Router
policy embedding	best avg score	KNN (k=5)	regression	45.92	19.85	68.62	Trained Router
policy embedding	best η	KNN (k=5)	tie regression	43.83	16.03	73.73	Trained Router
weighted_average_text025	best avg score	KNN (k=5)	rank regression	45.74	22.66	41.51	Trained Router
weighted_average_text025	best η	KNN (k=5)	tie regression	44.04	16.08	74.37	Trained Router
weighted_average_text075	best avg score	KNN (k=5)	regression	45.40	19.46	69.19	Trained Router
weighted_average_text075	best η	KNN (k=5)	tie regression	43.88	15.63	74.55	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	KNN (k=5)	regression	45.71	19.59	69.41	Trained Router
dinov2-large	best η	Linear	regression	45.02	17.31	75.22	Trained Router
siglip-large-patch16-384	best avg score	KNN (k=5)	regression	45.47	19.38	69.72	Trained Router
siglip-large-patch16-384	best η	Linear	regression	44.91	16.93	75.75	Trained Router
siglip2-large-patch16-512	best avg score	KNN (k=5)	rank regression	45.73	22.68	41.31	Trained Router
siglip2-large-patch16-512	best η	Linear	regression	44.69	16.97	74.92	Trained Router
siglip2-so400m-patch14-384	best avg score	MLP [2048,1024]	regression	46.10	23.13	32.45	Trained Router
siglip2-so400m-patch14-384	best η	Linear	regression	44.69	16.97	74.93	Trained Router
—	—	strongest global	—	46.09	24.08	0.00	Fixed Baseline
—	—	expensive global	—	46.09	24.08	0.00	Fixed Baseline
—	—	strongest per category	—	46.09	24.08	0.00	Fixed Baseline
—	—	random	—	35.97	13.24	43.68	Fixed Baseline
—	—	cheapest global	—	27.24	2.11	0.00	Fixed Baseline
—	—	ood baseline (average)	—	46.09	24.08	0.00	OOD Baseline
—	—	ood baseline (concat)	—	42.73	20.85	55.34	OOD Baseline
—	—	ood baseline (normalize_concat)	—	42.73	20.85	55.34	OOD Baseline
—	—	ood baseline (vision embedding)	—	28.07	2.63	4.51	OOD Baseline
—	—	ood baseline (text embedding)	—	34.47	13.06	36.70	OOD Baseline
—	—	ood baseline (policy embedding)	—	46.09	24.08	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	46.09	24.08	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	28.00	3.16	4.11	OOD Baseline
—	—	ood baseline (dinov2-large)	—	42.76	21.39	51.86	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	42.31	20.57	55.89	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	42.73	20.85	55.34	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	42.70	20.77	55.73	OOD Baseline
—	—	accuracy oracle	—	47.47	13.37	91.28	Oracle
—	—	η oracle	—	47.29	16.00	84.94	Oracle

E. Gemini-3-Flash vs. Gemini-3-Flash-MinimalThinking

TABLE XIII: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	**Selection Head**	**Avg Score**	**Avg Cost (s)**	η	**Category**
Linear	regression	64.78	8.76	57.04	Trained Router
MLP [2048,1024]	regression	64.83	8.47	57.99	Trained Router
KNN (k=5)	regression	64.57	7.25	54.86	Trained Router
PR-KNN (k=5)	regression	64.57	7.25	54.86	Trained Router
K-means	classification	64.28	11.63	46.82	Trained Router
OVR	regression	64.07	9.55	46.10	Trained Router
cheapest global	—	61.22	1.88	0.00	Fixed Baseline
ood baseline	—	61.83	6.70	10.81	OOD Baseline
strongest per category	—	64.58	12.24	49.51	Fixed Baseline
expensive global	—	64.22	15.86	0.00	Fixed Baseline
strongest global	—	64.22	15.86	0.00	Fixed Baseline
random	—	62.11	8.82	15.53	Fixed Baseline
accuracy oracle	—	67.39	4.49	97.96	Oracle
η oracle	—	67.31	6.23	94.97	Oracle

TABLE XIV: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	Linear	regression	64.95	8.60	59.60	Trained Router
average	best η	Linear	regression	64.95	8.60	59.60	Trained Router
concat	best avg score	MLP [2048,1024]	regression	64.96	9.04	59.28	Trained Router
concat	best η	MLP [2048,1024]	regression	64.96	9.04	59.28	Trained Router
normalize_concat	best avg score	MLP [2048,1024]	regression	64.83	8.47	57.99	Trained Router
normalize_concat	best η	MLP [2048,1024]	regression	64.83	8.47	57.99	Trained Router
vision embedding	best avg score	Linear	regression	64.76	10.50	54.82	Trained Router
vision embedding	best η	Linear	regression	64.76	10.50	54.82	Trained Router
text embedding	best avg score	Linear	regression	64.84	8.66	57.92	Trained Router
text embedding	best η	Linear	regression	64.84	8.66	57.92	Trained Router
policy embedding	best avg score	Linear	regression	64.58	10.16	52.83	Trained Router
policy embedding	best η	Linear	regression	64.58	10.16	52.83	Trained Router
weighted_average_text025	best avg score	Linear	regression	64.85	9.09	57.71	Trained Router
weighted_average_text025	best η	Linear	regression	64.85	9.09	57.71	Trained Router
weighted_average_text075	best avg score	Linear	regression	64.91	8.75	58.84	Trained Router
weighted_average_text075	best η	Linear	regression	64.91	8.75	58.84	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	Linear	regression	64.76	8.42	57.02	Trained Router
dinov2-large	best η	Linear	regression	64.76	8.42	57.02	Trained Router
siglip-large-patch16-384	best avg score	Linear	regression	64.75	8.60	56.69	Trained Router
siglip-large-patch16-384	best η	KNN (k=5)	regression	64.72	7.63	56.85	Trained Router
siglip2-large-patch16-512	best avg score	MLP [2048,1024]	regression	64.83	8.47	57.99	Trained Router
siglip2-large-patch16-512	best η	MLP [2048,1024]	regression	64.83	8.47	57.99	Trained Router
siglip2-so400m-patch14-384	best avg score	Linear	regression	64.90	9.01	58.40	Trained Router
siglip2-so400m-patch14-384	best η	Linear	regression	64.90	9.01	58.40	Trained Router
—	—	cheapest global	—	61.22	1.88	0.00	Fixed Baseline
—	—	ood baseline (average)	—	61.22	1.88	0.00	OOD Baseline
—	—	ood baseline (concat)	—	61.83	6.70	10.81	OOD Baseline
—	—	ood baseline (normalize_concat)	—	61.83	6.70	10.81	OOD Baseline
—	—	ood baseline (vision embedding)	—	61.22	1.88	0.00	OOD Baseline
—	—	ood baseline (text embedding)	—	61.22	1.88	0.00	OOD Baseline
—	—	ood baseline (policy embedding)	—	64.22	15.86	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	61.22	1.88	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	61.22	1.88	0.00	OOD Baseline
—	—	ood baseline (dinov2-large)	—	61.23	1.89	0.20	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	61.44	6.13	3.94	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	61.83	6.70	10.81	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	61.58	6.34	6.37	OOD Baseline
—	—	strongest per category	—	64.58	12.24	49.51	Fixed Baseline
—	—	strongest global	—	64.22	15.86	0.00	Fixed Baseline
—	—	expensive global	—	64.22	15.86	0.00	Fixed Baseline
—	—	random	—	62.11	8.82	15.53	Fixed Baseline
—	—	accuracy oracle	—	67.39	4.49	97.96	Oracle
—	—	η oracle	—	67.31	6.23	94.97	Oracle

F. Gemini-Robotics-ER vs. Gemini-Robotics-ER-ThinkingOff

TABLE XV: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	**Selection Head**	**Avg Score**	**Avg Cost (s)**	η	**Category**
Linear	regression	63.92	4.72	37.09	Trained Router
MLP [2048,1024]	rank regression	63.66	3.71	31.40	Trained Router
KNN (k=5)	rank regression	63.58	4.13	29.23	Trained Router
PR-KNN (k=5)	rank regression	63.58	4.13	29.23	Trained Router
K-means	classification	63.03	4.90	15.17	Trained Router
OVR	regression	63.01	4.51	14.70	Trained Router
cheapest global	—	62.88	2.31	11.39	Fixed Baseline
ood baseline	—	63.19	3.13	19.68	OOD Baseline
strongest global	—	62.88	2.31	11.39	Fixed Baseline
strongest per category	—	63.58	5.82	27.57	Fixed Baseline
random	—	62.46	4.67	0.00	Fixed Baseline
expensive global	—	62.62	7.02	0.00	Fixed Baseline
accuracy oracle	—	66.48	2.90	98.72	Oracle
η oracle	—	66.42	3.45	97.10	Oracle

TABLE XVI: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	MLP [2048,1024]	regression	63.98	4.67	38.60	Trained Router
average	best η	MLP [2048,1024]	regression	63.98	4.67	38.60	Trained Router
concat	best avg score	Linear	regression	63.92	4.72	37.09	Trained Router
concat	best η	Linear	regression	63.92	4.72	37.09	Trained Router
normalize_concat	best avg score	Linear	regression	63.92	4.72	37.09	Trained Router
normalize_concat	best η	Linear	regression	63.92	4.72	37.09	Trained Router
vision embedding	best avg score	MLP [2048,1024]	regression	63.89	4.16	36.91	Trained Router
vision embedding	best η	MLP [2048,1024]	regression	63.89	4.16	36.91	Trained Router
text embedding	best avg score	Linear	regression	64.15	4.56	42.67	Trained Router
text embedding	best η	Linear	regression	64.15	4.56	42.67	Trained Router
policy embedding	best avg score	Linear	rank regression	63.74	4.59	32.92	Trained Router
policy embedding	best η	Linear	rank regression	63.74	4.59	32.92	Trained Router
weighted_average_text025	best avg score	Linear	regression	63.91	4.69	36.90	Trained Router
weighted_average_text025	best η	Linear	regression	63.91	4.69	36.90	Trained Router
weighted_average_text075	best avg score	Linear	regression	63.92	4.67	37.30	Trained Router
weighted_average_text075	best η	Linear	regression	63.92	4.67	37.30	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	Linear	regression	63.86	4.74	35.77	Trained Router
dinov2-large	best η	Linear	regression	63.86	4.74	35.77	Trained Router
siglip-large-patch16-384	best avg score	Linear	regression	63.88	4.62	36.26	Trained Router
siglip-large-patch16-384	best η	Linear	regression	63.88	4.62	36.26	Trained Router
siglip2-large-patch16-512	best avg score	Linear	regression	63.92	4.72	37.09	Trained Router
siglip2-large-patch16-512	best η	Linear	regression	63.92	4.72	37.09	Trained Router
siglip2-so400m-patch14-384	best avg score	MLP [2048,1024]	regression	64.05	4.37	40.59	Trained Router
siglip2-so400m-patch14-384	best η	MLP [2048,1024]	regression	64.05	4.37	40.59	Trained Router
—	—	strongest global	—	62.88	2.31	11.39	Fixed Baseline
—	—	cheapest global	—	62.88	2.31	11.39	Fixed Baseline
—	—	ood baseline (average)	—	62.88	2.31	11.39	OOD Baseline
—	—	ood baseline (concat)	—	63.19	3.13	19.68	OOD Baseline
—	—	ood baseline (normalize_concat)	—	63.19	3.13	19.68	OOD Baseline
—	—	ood baseline (vision embedding)	—	62.88	2.31	11.39	OOD Baseline
—	—	ood baseline (text embedding)	—	62.88	2.31	11.39	OOD Baseline
—	—	ood baseline (policy embedding)	—	62.62	7.02	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	62.88	2.31	11.39	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	62.88	2.31	11.39	OOD Baseline
—	—	ood baseline (dinov2-large)	—	62.88	2.31	11.39	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	63.12	3.32	17.72	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	63.19	3.13	19.68	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	63.17	3.04	18.96	OOD Baseline
—	—	strongest per category	—	63.58	5.82	27.57	Fixed Baseline
—	—	random	—	62.46	4.67	0.00	Fixed Baseline
—	—	expensive global	—	62.62	7.02	0.00	Fixed Baseline
—	—	accuracy oracle	—	66.48	2.90	98.72	Oracle
—	—	η oracle	—	66.42	3.45	97.10	Oracle

G. Qwen3-VL-8B-Thinking vs. Cosmos-Reason2

TABLE XVII: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	**Selection Head**	**Avg Score**	**Avg Cost (s)**	η	**Category**
Linear	rank regression	46.49	85.04	70.16	Trained Router
MLP [2048,1024]	classification	46.26	82.39	70.44	Trained Router
KNN (k=5)	tie regression	46.98	86.36	71.16	Trained Router
PR-KNN (k=5)	tie regression	46.98	86.36	71.16	Trained Router
K-means	tie regression	48.28	97.62	67.49	Trained Router
OVR	classification	45.01	72.34	68.80	Trained Router
strongest per category	—	48.28	112.34	43.53	Fixed Baseline
expensive global	—	48.28	120.76	0.00	Fixed Baseline
strongest global	—	48.28	120.76	0.00	Fixed Baseline
random	—	38.93	64.49	45.03	Fixed Baseline
cheapest global	—	29.56	2.02	0.00	Fixed Baseline
ood baseline	—	37.32	49.37	38.47	OOD Baseline
accuracy oracle	—	50.47	70.83	88.87	Oracle
η oracle	—	48.32	55.90	88.76	Oracle

TABLE XVIII: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	KNN (k=5)	regression	48.46	97.32	68.19	Trained Router
average	best η	KNN (k=5)	tie regression	47.74	85.89	73.79	Trained Router
concat	best avg score	KNN (k=5)	regression	48.46	98.26	67.31	Trained Router
concat	best η	KNN (k=5)	tie regression	46.98	86.36	71.16	Trained Router
normalize_concat	best avg score	KNN (k=5)	regression	48.46	98.26	67.31	Trained Router
normalize_concat	best η	KNN (k=5)	tie regression	46.98	86.36	71.16	Trained Router
vision embedding	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
vision embedding	best η	Linear	rank regression	47.62	88.74	71.95	Trained Router
text embedding	best avg score	KNN (k=5)	regression	48.60	94.40	71.02	Trained Router
text embedding	best η	PR-KNN (k=5)	classification	46.72	80.46	72.70	Trained Router
policy embedding	best avg score	KNN (k=5)	regression	48.70	101.90	63.88	Trained Router
policy embedding	best η	KNN (k=5)	classification	45.53	76.75	69.67	Trained Router
weighted_average_text025	best avg score	KNN (k=5)	regression	48.46	98.26	67.31	Trained Router
weighted_average_text025	best η	KNN (k=5)	tie regression	47.60	87.28	72.67	Trained Router
weighted_average_text075	best avg score	KNN (k=5)	regression	48.46	95.18	70.03	Trained Router
weighted_average_text075	best η	KNN (k=5)	classification	46.44	78.00	72.55	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	KNN (k=5)	regression	48.46	98.26	67.31	Trained Router
dinov2-large	best η	MLP [2048,1024]	classification	47.00	82.04	73.05	Trained Router
siglip-large-patch16-384	best avg score	KNN (k=5)	regression	48.46	97.94	67.62	Trained Router
siglip-large-patch16-384	best η	KNN (k=5)	tie regression	47.88	88.84	72.68	Trained Router
siglip2-large-patch16-512	best avg score	KNN (k=5)	regression	48.46	98.26	67.31	Trained Router
siglip2-large-patch16-512	best η	KNN (k=5)	tie regression	46.98	86.36	71.16	Trained Router
siglip2-so400m-patch14-384	best avg score	KNN (k=5)	regression	48.39	96.94	68.36	Trained Router
siglip2-so400m-patch14-384	best η	KNN (k=5)	rank regression	48.16	92.17	71.46	Trained Router
—	—	strongest per category	—	48.28	112.34	43.53	Fixed Baseline
—	—	strongest global	—	48.28	120.76	0.00	Fixed Baseline
—	—	expensive global	—	48.28	120.76	0.00	Fixed Baseline
—	—	random	—	38.93	64.49	45.03	Fixed Baseline
—	—	cheapest global	—	29.56	2.02	0.00	Fixed Baseline
—	—	ood baseline (average)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (concat)	—	37.32	49.37	38.47	OOD Baseline
—	—	ood baseline (normalize_concat)	—	37.32	49.37	38.47	OOD Baseline
—	—	ood baseline (vision embedding)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (text embedding)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (policy embedding)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (dinov2-large)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	36.91	42.57	36.69	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	37.32	49.37	38.47	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	36.91	46.48	36.60	OOD Baseline
—	—	accuracy oracle	—	50.47	70.83	88.87	Oracle
—	—	η oracle	—	48.32	55.90	88.76	Oracle

H. Qwen3-VL-8B-Thinking vs. Qwen3-VL-8B-Instruct

TABLE XIX: Router-architecture ablation (embedding fusion fixed to `normalize_concat`, vision encoder SigLIP-2 — the deployed configuration, App. B-F). Per architecture we report the best router by η, with the fixed baselines and oracles. avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Architecture	**Selection Head**	**Avg Score**	**Avg Cost (s)**	η	**Category**
Linear	tie regression	44.77	61.66	68.36	Trained Router
MLP [2048,1024]	tie regression	47.48	81.72	74.78	Trained Router
KNN (k=5)	rank regression	47.56	90.90	70.55	Trained Router
PR-KNN (k=5)	rank regression	47.56	90.90	70.55	Trained Router
K-means	tie regression	47.11	94.06	67.08	Trained Router
OVR	classification	44.67	67.93	66.94	Trained Router
strongest per category	—	48.28	112.33	43.62	Fixed Baseline
expensive global	—	48.28	120.76	0.00	Fixed Baseline
strongest global	—	48.28	120.76	0.00	Fixed Baseline
random	—	40.62	64.40	48.48	Fixed Baseline
cheapest global	—	31.64	1.85	0.00	Fixed Baseline
ood baseline	—	46.89	106.22	54.20	OOD Baseline
accuracy oracle	—	50.12	62.72	91.30	Oracle
η oracle	—	48.46	50.06	91.05	Oracle

TABLE XX: Embedding ablation: (i) embedding-fusion method (vision encoder fixed to SigLIP-2) and (ii) vision encoder (fusion fixed to `normalize_concat`). Best router per setting by average score and by η. *text* / *vision* / *policy embedding* are the single-modality fusions (instruction, scene, and π_0 VLA embeddings). avg_score / η $\times 100$; avg_cost in seconds; η cells shaded red to green.

Fusion / Encoder	Selected by	Model Architecture	Selection Head	Avg Score	Avg Cost (s)	η	Category
(i) Embedding fusion — vision encoder fixed (SigLIP-2)							
average	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
average	best η	KNN (k=5)	classification	45.75	71.73	70.88	Trained Router
concat	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
concat	best η	KNN (k=5)	rank regression	47.56	90.90	70.55	Trained Router
normalize_concat	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
normalize_concat	best η	MLP [2048,1024]	tie regression	47.48	81.72	74.78	Trained Router
vision embedding	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
vision embedding	best η	Linear	rank regression	45.58	81.53	67.54	Trained Router
text embedding	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
text embedding	best η	KNN (k=5)	tie regression	47.40	82.75	74.04	Trained Router
policy embedding	best avg score	MLP [2048,1024]	regression	48.31	120.22	4.75	Trained Router
policy embedding	best η	KNN (k=5)	rank regression	46.63	91.77	66.95	Trained Router
weighted_average_text025	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
weighted_average_text025	best η	OVR	classification	46.00	67.85	72.76	Trained Router
weighted_average_text075	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
weighted_average_text075	best η	KNN (k=5)	classification	46.45	70.54	74.08	Trained Router
(ii) Vision encoder — fusion fixed (`normalize_concat`)							
dinov2-large	best avg score	MLP [2048,1024]	regression	48.28	119.56	10.01	Trained Router
dinov2-large	best η	KNN (k=5)	rank regression	48.16	91.41	72.19	Trained Router
siglip-large-patch16-384	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
siglip-large-patch16-384	best η	KNN (k=5)	tie regression	47.33	84.88	72.87	Trained Router
siglip2-large-patch16-512	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
siglip2-large-patch16-512	best η	MLP [2048,1024]	tie regression	47.48	81.72	74.78	Trained Router
siglip2-so400m-patch14-384	best avg score	K-means	regression	48.28	120.76	0.00	Trained Router
siglip2-so400m-patch14-384	best η	KNN (k=5)	tie regression	47.12	85.98	71.62	Trained Router
—	—	strongest per category	—	48.28	112.33	43.62	Fixed Baseline
—	—	strongest global	—	48.28	120.76	0.00	Fixed Baseline
—	—	expensive global	—	48.28	120.76	0.00	Fixed Baseline
—	—	random	—	40.62	64.40	48.48	Fixed Baseline
—	—	cheapest global	—	31.64	1.85	0.00	Fixed Baseline
—	—	ood baseline (average)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (concat)	—	46.89	106.22	54.20	OOD Baseline
—	—	ood baseline (normalize_concat)	—	46.89	106.22	54.20	OOD Baseline
—	—	ood baseline (vision embedding)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (text embedding)	—	33.52	31.63	11.06	OOD Baseline
—	—	ood baseline (policy embedding)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text025)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (weighted_average_text075)	—	48.28	120.76	0.00	OOD Baseline
—	—	ood baseline (dinov2-large)	—	46.20	107.06	51.46	OOD Baseline
—	—	ood baseline (siglip-large-patch16-384)	—	46.20	102.54	57.24	OOD Baseline
—	—	ood baseline (siglip2-large-patch16-512)	—	46.89	106.22	54.20	OOD Baseline
—	—	ood baseline (siglip2-so400m-patch14-384)	—	46.62	103.46	57.25	OOD Baseline
—	—	accuracy oracle	—	50.12	62.72	91.30	Oracle
—	—	η oracle	—	48.46	50.06	91.05	Oracle

I. Comparing Router Architectures

Across pairs, the linear router with a soft-label or tie-regression head is most often selected on η, while the `mlp` regression head tends to win on raw average score at higher cost; hyperparameters for each architecture are in Appendix C-A.

APPENDIX D
ADDITIONAL RESULTS

A. Expanded Main-Paper Result Figures

For completeness, this section further details main-paper result figures with ± 1 standard-error-of-the-mean (SEM) error bars and other notable details. Fig. 12 gives the per-axis results with SEM error bars for the intelligence/thinking, skill/model-size, and memory axes.

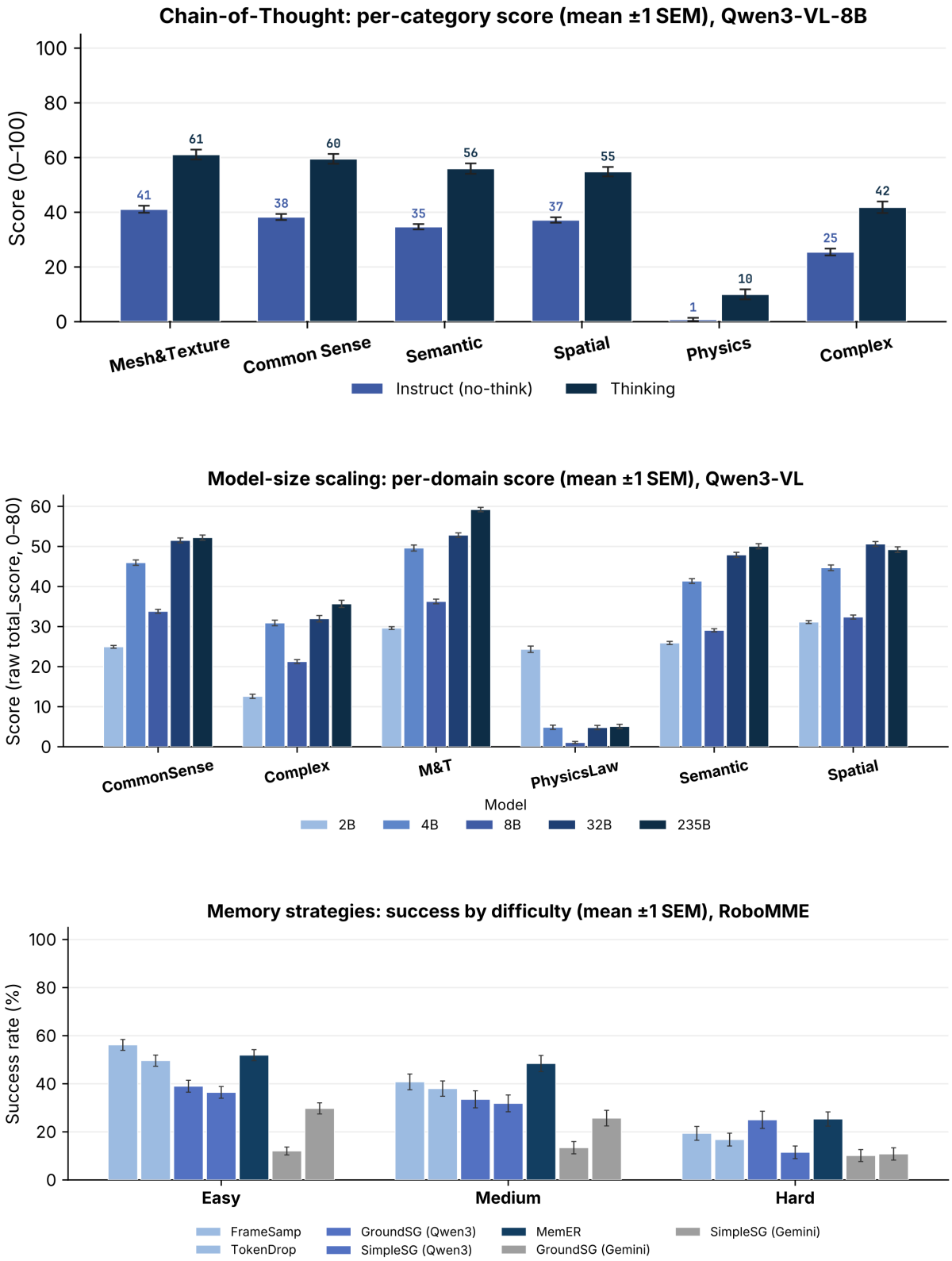


Fig. 12: **Main-paper results with ±1 SEM error bars**, by test-time-compute axis. **Top — intelligence / thinking:** score for No-Think vs. Think planners. **Middle — skill command / model size:** score for different-sized planners from the same model family. **Bottom — memory:** success for various memory-augmented VLA. Error bars denote ±1 SEM.

Fig. 13 gives the enlarged RoboMME memory-routing Pareto frontiers, with each memory architecture labeled: the top row breaks the routed success–cost frontier out by difficulty tier and the bottom row by task suite. Across both breakdowns DIRECT's routed frontier tracks the oracle and sits at or above the best individual memory architecture, at a fraction of MemER's cost.

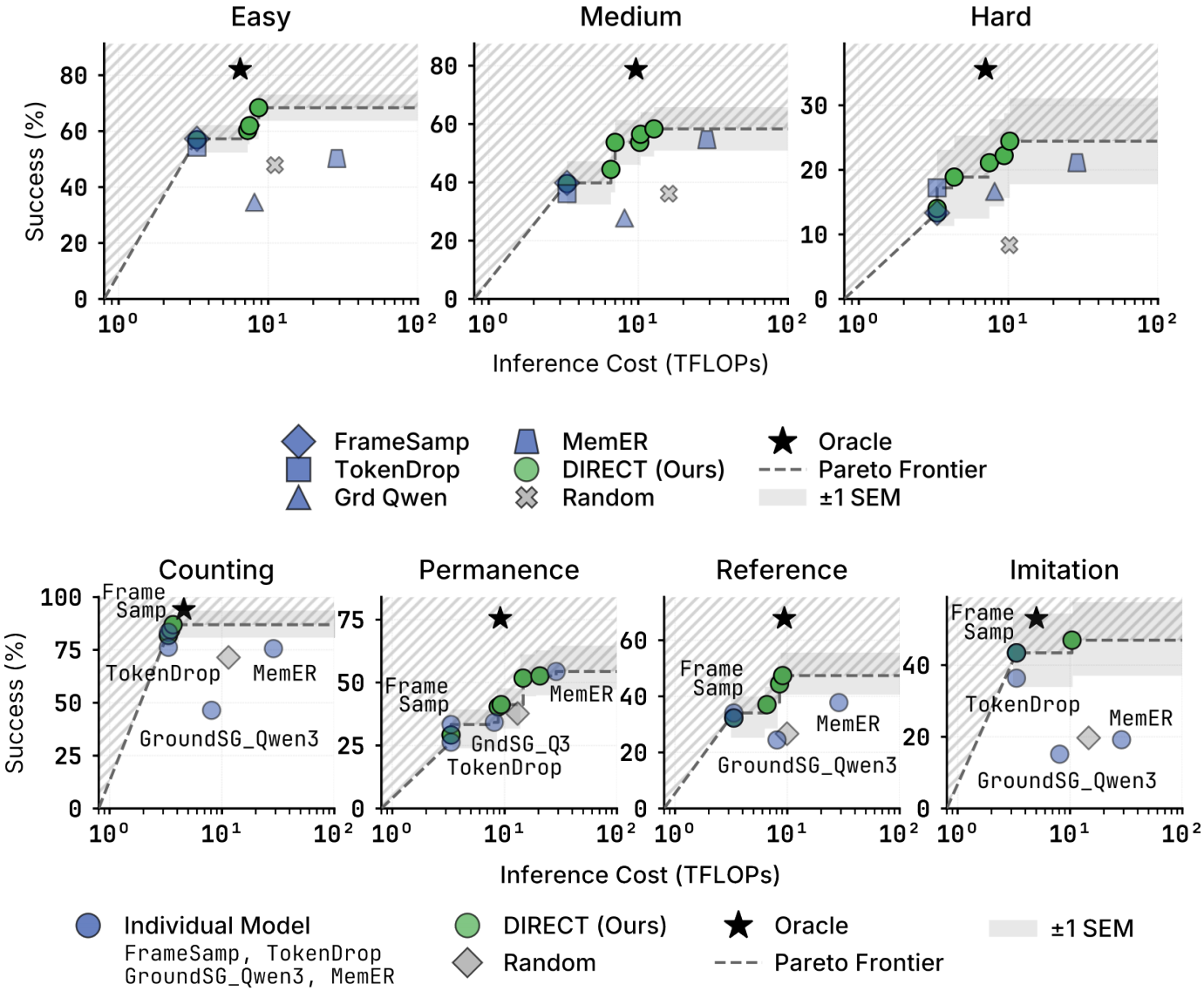


Fig. 13: **Enlarged RoboMME memory-routing Pareto frontiers**, success (%) vs. inference cost (TFLOPs, log scale), labeled by memory architecture. Points are the individual architectures (FrameSamp, TokenDrop, GroundSG-Qwen3, MemER), DIRECT's routed frontier (dashed), and the random and oracle references. **Top:** split by difficulty tier (easy / medium / hard), with ± 1 SEM error bars. **Bottom:** split by task suite (counting, permanence, reference, imitation).

B. Individual-Planner Simulation Performance

We report the individual-planner performance underlying the routing pools, before any routing. Table XXI gives the VLABench results broken out by task category and planner latency; Table XXII gives the RoboMME memory-planner results broken out both by difficulty tier and by task suite, against the $2NT$ inference-cost estimate. All entries are mean success ($\pm$ SEM), and the per-planner *overall* column is shaded red (low) to green (high) so the capability ordering is visible at a glance. Latencies for closed-weight (API) planners are full round-trip and noisier than the locally-measured open-weight ones (Appendix E).

TABLE XXI: **Per-planner performance on VLABench, by task category.** Cells are mean success (%) ± SEM. Unmarked rows use the full 100 episodes/task ($n \approx 4500$); ‡ marks the 20-episode/task (20%) subsample ($n \approx 900$) used for the pairwise routing comparisons in the main paper (a model may appear as both). *Think* marks chain-of-thought planners. Color runs red to green : *Overall* (mean across categories) is higher = green, while the resource columns *Size* and *Lat.* are inverted (smaller / faster = green, log-scaled); API planners have round-trip latencies (Appendix E) and a grey *Size* cell.

Model	Size	Think	Common	Complex	M&T	Physics	Semantic	Spatial	Lat. (s)	Overall
Gemini 3 Flash	API	✓	77.06 ±0.78	56.43 ±1.21	85.78 ±0.58	1.83 ±0.38	75.92 ±0.77	76.25 ±0.67	15.57 ±0.22	66.11 ±0.5
Gemini Robotics-ER	API	✓	74.44 ±0.8	45.45 ±1.2	84.08 ±0.66	3.25 ±0.5	72.44 ±0.8	75.16 ±0.65	6.99 ±0.03	63.38 ±0.5
Gemini 3 Flash (min-think)	API	–	73.22 ±0.77	56.31 ±1.22	80.92 ±0.67	0.0 ±0.0	69.64 ±0.79	75.81 ±0.65	1.87 ±0.01	62.86 ±0.49
Gemini Robotics-ER (think-off)	API	–	73.03 ±0.79	46.5 ±1.23	83.81 ±0.63	1.25 ±0.32	69.67 ±0.83	77.47 ±0.63	2.31 ±0.01	62.78 ±0.5
Qwen3-VL-235B-Instruct (FP8)	235B	–	65.25 ±0.8	44.58 ±1.13	74.0 ±0.7	6.33 ±0.68	62.56 ±0.81	61.5 ±0.87	12.19 ±0.08	55.64 ±0.46
GLM-4.1V-Thinking	9B	✓	66.28 ±0.84	37.88 ±1.04	70.61 ±0.82	1.08 ±0.3	61.42 ±0.82	65.81 ±0.82	44.93 ±0.53	54.24 ±0.48
Qwen3-VL-32B-Instruct (FP8)	32B	–	64.38 ±0.77	39.96 ±1.0	66.03 ±0.69	6.0 ±0.66	59.86 ±0.8	63.25 ±0.77	58.64 ±0.88	53.11 ±0.44
Qwen3-VL-8B-Thinking‡	8B	✓	59.53 ±1.8	41.83 ±2.11	61.11 ±1.8	10.0 ±1.83	55.97 ±1.91	54.84 ±1.72	117.7 ±2.03	49.73 ±0.94
Qwen3-VL-4B-Instruct (FP8)	4B	–	57.44 ±0.84	38.63 ±0.88	62.0 ±0.94	6.08 ±0.67	51.69 ±0.75	55.84 ±0.87	4.53 ±0.21	47.98 ±0.44
GLM-4.6V-Flash-Thinking	9B	✓	55.06 ±0.87	32.37 ±0.98	64.25 ±0.81	0.25 ±0.14	52.03 ±0.81	60.12 ±0.82	23.77 ±0.2	47.36 ±0.45
RynnBrain-8B-Plan	8B	–	56.91 ±0.91	43.18 ±1.02	64.81 ±0.86	0.0 ±0.0	47.47 ±0.9	55.34 ±0.91	1.68 ±0.01	47.21 ±0.46
Qwen3-VL-8B-Instruct (FP8)	8B	–	42.25 ±0.61	26.57 ±0.61	45.33 ±0.72	1.33 ±0.33	36.33 ±0.49	40.47 ±0.63	3.89 ±0.08	34.17 ±0.32
Qwen3-VL-2B-Instruct	2B	–	31.19 ±0.43	15.73 ±0.66	37.03 ±0.43	30.42 ±1.0	32.39 ±0.49	38.91 ±0.44	6.21 ±0.24	32.15 ±0.25
Qwen3-VL-8B-Instruct‡	8B	–	38.3 ±1.1	25.5 ±1.3	41.1 ±1.3	0.8 ±0.6	34.7 ±1.0	37.2 ±1.0	1.85 ±0.02	31.5 ±0.6
Qwen3-VL-8B-Instruct	8B	–	38.06 ±0.47	24.82 ±0.55	39.69 ±0.48	0.75 ±0.25	34.81 ±0.43	37.59 ±0.48	1.89 ±0.02	31.21 ±0.27
Cosmos-Reason2	8B	–	31.62 ±0.39	27.48 ±0.63	35.39 ±0.41	5.25 ±0.63	32.08 ±0.39	36.97 ±0.44	2.07 ±0.01	29.44 ±0.24
Cosmos-Reason2‡	8B	–	30.9 ±0.9	26.0 ±1.4	34.7 ±0.9	5.8 ±1.5	30.8 ±0.8	38.0 ±1.0	2.05 ±0.03	29.0 ±0.5
GLM-4.6V-Flash	9B	–	31.38 ±0.52	17.02 ±0.58	38.22 ±0.58	0.0 ±0.0	28.42 ±0.48	37.22 ±0.56	2.18 ±0.03	27.41 ±0.28
RynnBrain-8B	8B	–	26.25 ±0.52	7.06 ±0.54	31.28 ±0.55	0.0 ±0.0	16.53 ±0.6	33.47 ±0.51	1.36 ±0.02	20.96 ±0.28

TABLE XXII: **Per-planner RoboMME success (%) ± SEM**, broken out by difficulty tier and by task suite. Color scale red to green : *Overall* is higher = green, while *Cost* (TFLOPs, Appendix G-D) is *inverted* (lower = green, log-scaled). The Gemini-predictor variants have unknown parameter counts, so their FLOPs are not computed (grey, —). *Overall* is the same under either breakdown; rows sorted by overall score.

		By difficulty			By task suite				
Planner	Cost (TFLOPs)	Easy	Medium	Hard	Counting	Permanence	Reference	Imitation	Overall
MemER	28.761	51.9 ±2.28	48.42 ±3.36	25.34 ±2.96	71.83 ±3.25	53.47 ±3.27	37.86 ±2.92	29.5 ±3.0	45.55 ±1.66
FrameSamp	3.338	56.15 ±2.26	40.78 ±3.27	19.37 ±2.88	79.06 ±3.28	25.55 ±2.71	36.68 ±3.01	50.83 ±3.23	44.72 ±1.68
TokenDrop	3.338	49.62 ±2.33	37.99 ±3.19	16.78 ±2.66	73.0 ±3.51	26.4 ±2.79	35.18 ±3.0	38.0 ±3.19	39.93 ±1.66
GroundSG (Qwen3)	8.091	38.97 ±2.47	33.52 ±3.54	25.0 ±3.57	55.37 ±4.54	39.09 ±3.49	31.66 ±3.31	21.0 ±2.89	34.73 ±1.78
SimpleSG (Qwen3)	8.091	36.41 ±2.44	31.84 ±3.49	11.49 ±2.63	61.16 ±4.45	17.77 ±2.73	24.62 ±3.06	29.0 ±3.22	30.13 ±1.71
SimpleSG (Gemini)	API	29.74 ±2.32	25.7 ±3.25	10.81 ±2.56	56.2 ±4.53	14.21 ±2.47	23.12 ±3.0	18.0 ±2.72	24.83 ±1.61
GroundSG (Gemini)	API	12.05 ±1.65	13.41 ±2.55	10.14 ±2.49	17.36 ±3.46	16.24 ±2.63	11.56 ±2.27	5.0 ±1.54	11.99 ±1.21

C. Full Hardware Results

The full physical Franka/DROID results are given in three parts: per-task individual-planner success (Fig. 14), the test-set Pareto frontiers used to select the routers (Fig. 15), and the trained routers run on the robot (Tables XXIII–XXV).

D. Individual-Planner Hardware Performance

Fig. 14 reports each individual planner's per-task success on the physical suite (Appendix G-B), grouped by within-axis category (Table XXVII). These planner-level numbers underlie the aggregated results in the main paper and show which tasks drive each axis's cheap-vs-expensive gap.

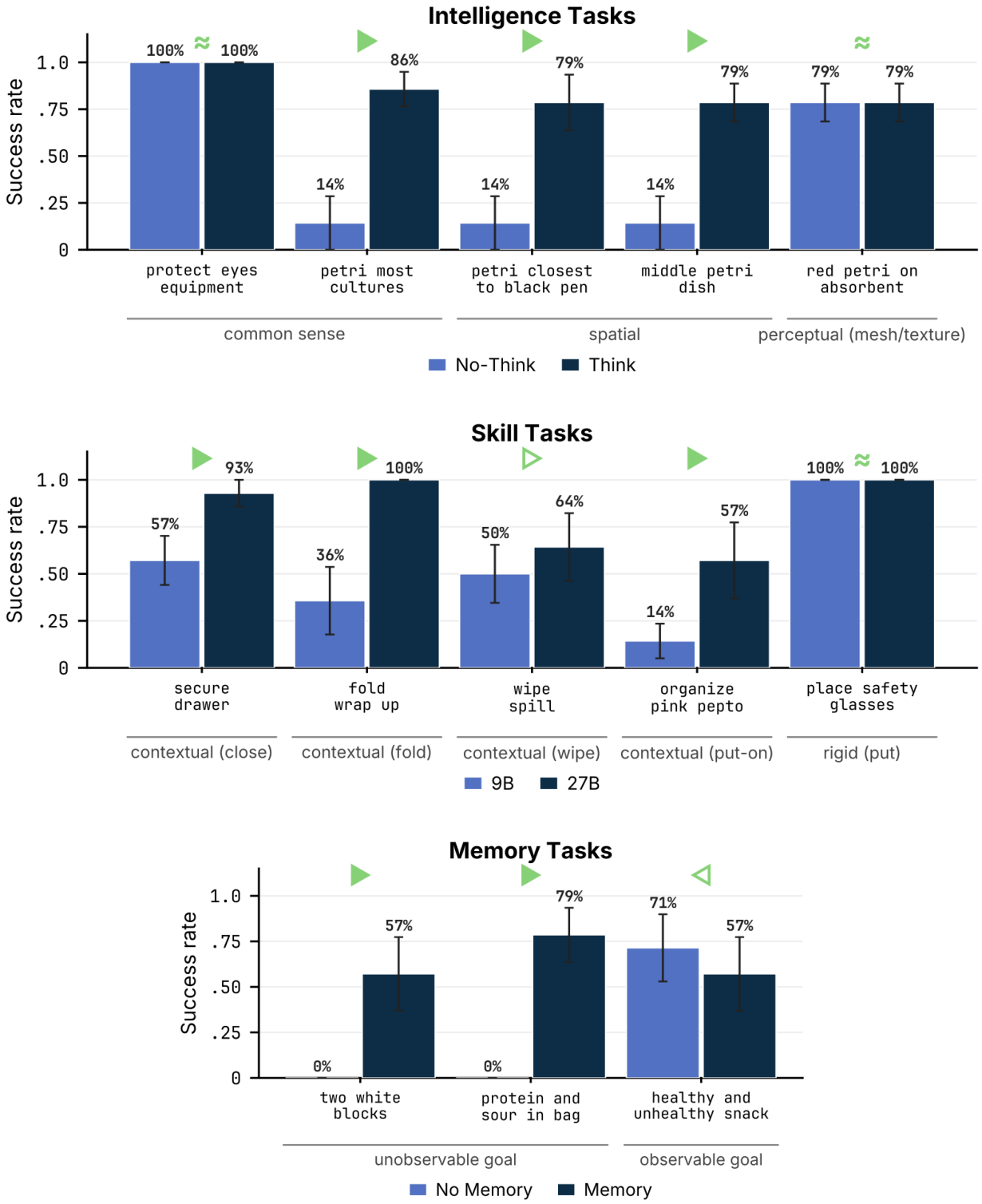


Fig. 14: **Per-task success of individual planners on the physical Franka/DROID suite**, by test-time-compute axis. Each trial is scored 1 (success), 0 (failure), or 0.5 when the high-level planner emitted a plausible subtask but the low-level policy failed to carry it out for reasons outside the planner's control (*e.g.*, an incorrect grasp); bars are the mean over trials. Symbols above each task mark the gap between the two planners in units of the standard error of the mean (SEM): $\approx$ denotes a gap ≤ 0.5 SEM (effectively tied); $\triangleleft / \triangleright$ a gap of 0.5–1 SEM; and ◀ / ▶ a gap > 1 SEM, with the triangle pointing toward the stronger planner.

1) Test-Set Pareto Frontiers: Since physical rollouts are expensive, routers are selected on the held-out test set of the synthetic routing data (Appendix G-C) before robot deployment. Fig. 15 shows this test-set cost–quality frontier, marking the

selected routers alongside the fixed cheap/expensive planners and the oracle ceiling.

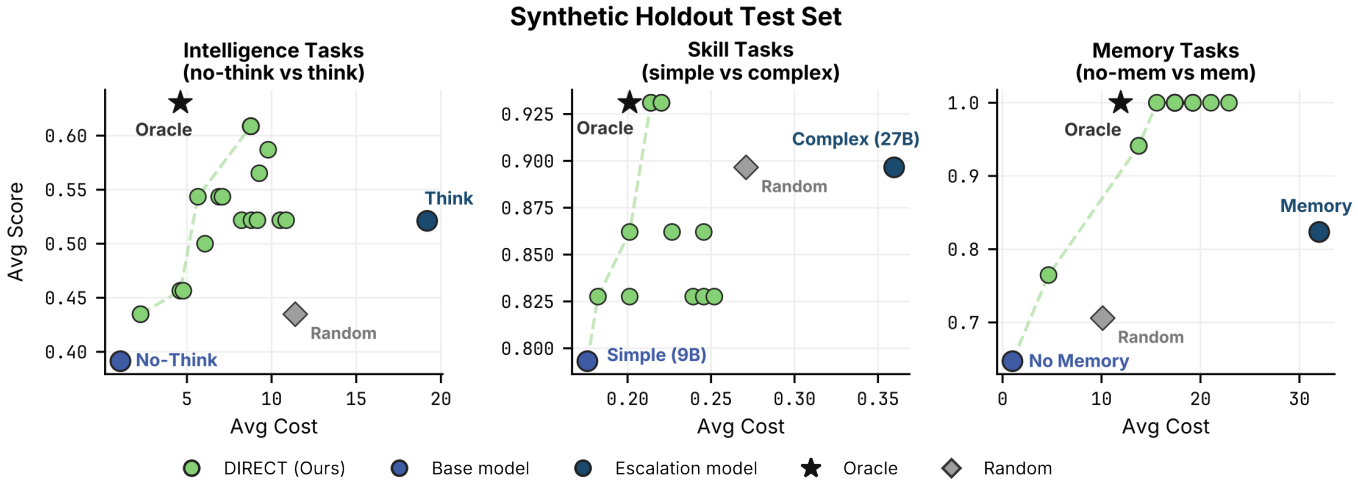


Fig. 15: **Held-out test-set Pareto frontiers.** Selected routers, fixed cheap/expensive planners, and the oracle ceiling are marked.

2) Trained Routers on the Physical Suite: We evaluate the selected routers directly on the physical tasks, reporting each axis separately in the same format as the simulation ablations (Appendix C): the score column is *physical* success rate and there is a single fixed input configuration per axis (Table V).

Intelligence / thinking axis. Table XXIII routes between the No-Think and Think planners. The best router by η (KNN/PR-KNN, regression) reaches 60.9% success, exceeding the Think planner (52.2%) and approaching the oracle (63.0%) while cutting average latency more than half (8.8 s vs. 19.2 s); cheaper configurations (*e.g.*, K-means classification) trade success for lower cost, tracing the cost–quality trade-off between the two planners.

TABLE XXIII: **Physical routing results: intelligence / thinking axis** (No-Think vs. Think planner). Best trained router per architecture (by routing efficiency η), with the fixed cheap/expensive planners, the random and OOD baselines, and the success oracle. Q is physical task success rate (%); η is routing efficiency (%, $\beta = 0.1$); cost is planner latency (s). η cells are shaded red (low) to green (high).

Method	Architecture	Head	Q (%)	Lat (s)	η (%)
Trained Router	KNN / PR-KNN (k=5)	regression	60.87	8.78	86.34
Trained Router	Linear	regression	58.70	9.80	77.73
Trained Router	OVR	rank regression	54.35	6.90	64.00
Trained Router	MLP [2048,1024]	tie regression	52.17	8.23	55.04
Trained Router	K-means	classification	52.17	10.86	53.64
Fixed Baseline	Think planner	—	52.17	19.19	0.00
Fixed Baseline	random	—	43.48	11.40	19.19
Fixed Baseline	No-Think planner	—	39.13	1.07	0.00
OOD Baseline	OOD baseline	—	50.00	17.65	32.57
Oracle	accuracy oracle	—	63.04	4.62	97.84

Skill command / model-size axis. Table XXIV routes between the 9B and 27B planners. The top router by η (KNN/PR-KNN, tie regression) reaches 93.1% success—matching the oracle and exceeding the 27B planner (89.7%)—at near-cheap latency (0.21 s vs. the 27B planner's 0.36 s), while the remaining routers fall between the 9B and 27B planners.

TABLE XXIV: **Physical routing results: skill command / model-size axis** (9B vs. 27B planner). Best trained router per architecture (by routing efficiency η), with the fixed cheap/expensive planners, the random and OOD baselines, and the success oracle. Q is physical task success rate (%); η is routing efficiency (%, $\beta = 0.1$); cost is planner latency (s). η cells are shaded red (low) to green (high).

Method	Architecture	Head	Q (%)	Lat (s)	η (%)
Trained Router	KNN / PR-KNN (k=5)	tie regression	93.10	0.21	97.68
Trained Router	Linear	tie regression	86.21	0.20	51.98
Trained Router	K-means	classification	86.21	0.23	51.45
Trained Router	MLP [2048,1024]	tie regression	82.76	0.20	26.72
Trained Router	OVR	tie regression	82.76	0.24	26.49
Fixed Baseline	27B planner	—	89.66	0.36	0.00
Fixed Baseline	random	—	89.66	0.27	71.41
Fixed Baseline	9B planner	—	79.31	0.18	0.00
OOD Baseline	OOD baseline	—	82.76	0.18	26.81
Oracle	accuracy oracle	—	93.10	0.20	98.57

Memory axis. Table XXV routes between the memory-free and memory planners (here cost is an input-frame proxy). The strongest router (Linear, soft-label) attains 100% success—matching the oracle and exceeding the always-memory planner (82.4%)—while processing roughly half its frames (15.6 vs. 32); cheaper routers cut frames further at lower success.

TABLE XXV: **Physical routing results: memory axis** (memory-free vs. memory planner). Best trained router per architecture (by routing efficiency η), with the fixed cheap/expensive planners, the random and OOD baselines, and the success oracle. Q is physical task success rate (%); η is routing efficiency (%, $\beta = 0.1$); cost is an input-frame proxy. η cells are shaded red (low) to green (high).

Method	Architecture	Head	Q (%)	Frames	η (%)
Trained Router	Linear	soft-label (λ=0.1)	100.00	15.6	92.52
Trained Router	KNN / PR-KNN (k=5)	classification	100.00	17.4	90.72
Trained Router	MLP [2048,1024]	tie regression	100.00	17.4	90.72
Trained Router	OVR	tie regression	100.00	17.4	90.72
Trained Router	K-means	classification	100.00	19.2	88.51
Fixed Baseline	memory planner	—	82.35	32.0	0.00
Fixed Baseline	random	—	70.59	10.1	30.21
Fixed Baseline	memory-free planner	—	64.71	1.0	15.49
OOD Baseline	OOD baseline	—	58.82	8.3	0.00
Oracle	accuracy oracle	—	100.00	11.9	95.28

APPENDIX E
EXPERIMENTAL AND HARDWARE SETUP

A. Evaluation Hardware

Open-weight planners were evaluated locally, on hardware that differs by experiment due to model-memory requirements:

- *Chain-of-thought* (think vs. no-think) experiments: a single NVIDIA RTX 4090 GPU.
- *Model-size scaling* experiments (Qwen3-VL Instruct 2B–235B): NVIDIA H100 GPU(s).

Because absolute latency is hardware-dependent, latencies are comparable within an experiment but not directly across the RTX 4090 and H100 settings. RoboMME results are taken from the public leaderboard (Appendix G-D) rather than re-run locally, and physical DROID planner latency is measured from cloud API endpoints (Appendix G-B).

Latency for each planner is reported as `latency_s`, the wall-clock time of the `evaluate()` call, which includes both per-model preprocessing and generation time (see Appendix G-A).

Closed-weight / API model latency. Inference latency measurements for closed-weight models are derived via cloud API endpoints. These values are subject to significant variance compared to local open-weight deployment due to request batching, dynamic server-side load balancing, and network-induced overhead, and the reported figures therefore include the full round-trip HTTP request.[1]

B. Inference Hyperparameters for the VLM Planners

Decoding hyperparameters for each planner's `model.generate()` call were set according to the corresponding model card recommendations when available.

[1]Denoted † in the result tables.

Embedding / sequence concatenation. Throughout, the fusion strategy we report in the main paper (`concat` or `normalize_concat`) refers to *sequence* concatenation of the per-modality embeddings (concatenating along the token/sequence axis), as opposed to *channel* concatenation (concatenating along the feature axis). See Appendix B-A for the full set of fusion strategies we ablated.

Appendix F
Baselines, Ceilings, and the AUC Metric

We use several existing VL-RouterBench baselines (random, strongest-global, cheapest-global, standard oracle) and contribute our own baseline / ceiling sweeps: the out-of-distribution (OOD) baseline and a budgeted oracle (modified from the original to trace a spread of cost–accuracy tradeoffs).

A. Baselines and Ceilings

Random. Route each sample to a planner chosen uniformly at random.
OOD (calibration). We detect out-of-distribution (OOD) embeddings via conformal prediction [41]. Given a success threshold γ: for each planner, all training-set samples with score $> \gamma$ form its calibration set. The per-planner percentile for pairwise calibration is $T = 5.0$. At inference, a test sample is routed to the first (cheapest) planner whose calibration set contains a neighbor with cosine similarity at least the T-percentile threshold; if no planner achieves this, we route to the most expensive planner. We sweep γ from 0 to 1 in increments of 0.1. Each γ represents a different cost–accuracy tradeoff: a higher γ shrinks the calibration set, making the router more likely to escalate to a more expensive planner in pursuit of a higher score.
Strongest per category. Route to the strongest planner for each task category.
Strongest global. Always route to the planner with the best average score.
Cheapest global. Always route to the cheapest average planner.
Budgeted oracle. Given a cost budget of $x\%$ of the dataset-wide maximum cost $c_{\max}$ (a single scalar representing the largest cost entry across all samples and planners), pick the most accurate planner within budget on the current sample, breaking exact ties by lower cost; if no planner fits, fall back to the cheapest planner for that sample. We sweep $x \in \{0, 5, \ldots, 95, 100\}$. As an oracle over the true scores, this traces an approximate theoretical upper bound on the routing cost–accuracy Pareto frontier.
Accuracy oracle. Picks the most accurate planner, breaking exact ties by lower cost.
Rank-score oracle. Picks the planner with the highest rank-score based on true score and cost, used as the representation of the cost-quality tradeoff on a sample level. Uses the per-sample rank-score equation taken from VL-RouterBench [26] (see section B-D), replacing predicted per-sample cost ($\hat{c}_{ik}$) and predicted per-sample score ($\hat{q}_{ik}$) with the true per-sample cost c_{ik} and score q_{ik}. The minimum and maximum cost across planners and samples are taken over the test set for log normalization of cost.

B. AUC Metric

We summarize each router / baseline / oracle by points of (scaled average quality q, scaled average cost c), which together form Pareto frontiers. To obtain a single AUC per method, we anchor each frontier with the individual-model points (cheapest / strongest) and compute the area under a piecewise-linear curve over $[\text{min_cost}, \text{max_cost}]$, or $[0, 1]$ in the scaled space. If a curve does not span the full domain, we extrapolate flat to the left (held at $y = 0$) and right (held at the last y-value).

We compute five specific curves, anchoring each Pareto frontier with the individual cheapest and strongest models:

1) **Router Pareto** — Pareto frontier of all trained routers, plus the anchors.
2) **OOD sweep** — Pareto frontier of all OOD threshold points, plus the anchors.
3) **Random baseline** — Pareto frontier of the random baseline routing point, plus the anchors.
4) **Strongest-per-category baseline** — Pareto frontier of the strongest-per-category point, plus the anchors.
5) **Oracle ceiling** — Formed from the points of the budgeted oracle (described in F-A). The budget-oracle curve is not a Pareto frontier: it is simply the budget-sweep points connected in order of budget percentage, with no model anchors.

Appendix G
Benchmark Details

A. VLABench

VLABench [58] is a language-conditioned manipulation benchmark. We use its *non-interactive VLM evaluation track* (the `VLABench/vlm_evaluation_v1.0` dataset), which is designed specifically for evaluating VLMs as high-level planners. In this track the VLM is given the task instruction together with the segmented scene images and must generate an *action sequence* in the format of our skill sequence. The generated sequence is then scored by matching its directed acyclic graph (DAG) against the ground-truth skill DAG, yielding an overall score. This non-interactive, skill-sequence formulation is exactly the high-level decomposition our router targets, which is why we adopt it.

TABLE XXVI: **VLABench routing AUC metric results** ($N = 900$). Pair labels: I/T = Instruct/Thinking. †These models were evaluated on a 20% subsample due to substantially higher inference time.

Pair (Cheap / Expensive)	Ours DIRECT	*Baselines* OOD	Str./cat.	Random	*Ceiling* Oracle
Homogeneous Open-Weight Pairs					
GLM-4.6 I / 4.1 T	**0.5634**	0.4207	0.4795	0.4822	0.7699
GLM-4.6 I / T	**0.5547**	0.4095	0.4660	0.4457	0.8177
Qwen3-VL 8B I / T †	**0.5957**	0.4427	0.4822	0.4564	0.8192
Heterogeneous Model Pairs					
Cosmos-R2 / Qwen3-VL 8B T †	**0.5732**	0.4575	0.4794	0.4362	0.7997
RynnBrain-Plan / GLM 4.1 T	**0.4632**	0.3279	0.3605	0.3295	0.8580

Score normalization. The VLABench scoring utility (`VLABench/evaluation/utils.py`, `get_final_score`) computes a weighted average but includes only two of the sub-scores and weights each of them by 0.4, so the maximum attainable score is 80 rather than 100. For ease of comprehension and consistency, we re-normalize all VLABench scores from the $[0, 80]$ range to $[0, 100]$ (equivalently $[0, 1]$).

Additional metrics tracked. We extended the evaluator to record, per model:

- `latency_s` — wall-clock seconds for the `evaluate()` call (preprocessing + generation). For API models this includes the full round-trip HTTP request and is therefore noisier.
- `power_mean_w` / `power_peak_w` — mean and peak GPU power draw.
- `flops_estimate` — estimated as $2 \cdot \text{num_params} \cdot (\text{input_tokens} + \text{output_tokens})$.
- `thinking_tokens` — best-effort count of thinking tokens in the output, accounting for per-model differences in thinking-token representation.

In this paper, we focus on VLABench's `latency_s`, given its strong correlation with the other recorded metrics.

Other modifications. We made the evaluator robust to per-model output differences (generalizing the skill-sequence JSON parsing).

B. Physical Setup: Franka / DROID

Hardware and low-level policy. We evaluate on a physical Franka arm in the DROID setup. In the physical experiments the high-level planner is queried through OpenRouter API calls, while the low-level $\pi_{0.5}$ VLA policy [6] (finetuned on DROID) executes the emitted subskills locally on an RTX 5090. Observations are captured from a wrist camera and a left third-person camera.

Planners. The high-level planners vary per experiment: Qwen3.5-VL-9B with and without CoT for the reasoning axis, Qwen3.5-VL-9B vs. 27B for the size axis, and Gemini-Robotics-ER 1.6 [20] for the memory axis (a single-frame memory-free planner vs. a memory planner given a 40-frame video history plus a task-list summary).

Latency measurement. Planner latency is measured from the OpenRouter standard inference endpoint and is subject to small request-to-request variance.

Task suite. Table XXVII lists the physical task suite, grouped by the test-time compute axis each task targets, and Figure 16 shows the starting frame for every task. The grouping follows the domain-bucket→axis correspondence of the synthetic-data pipeline (Appendix G-C).

TABLE XXVII: Physical Franka/DROID task suite (instructions given to the planner), grouped by test-time compute axis and by within-axis category. For skill command, *rigid* verbs (*e.g.*, *put*) map directly to an action, whereas *contextual* verbs (*e.g.*, *close*, *fold*, *wipe*, *put-on*) require grounding the verb in the scene.

Category	Task ID	Instruction
(1) Intelligence form / thinking (CoT reasoning) — science-themed tasks		
common sense	`protect_eyes_equipment`	I need to protect my eyes during my next experiment. Place the corresponding equipment in the white box.
common sense	`petri_most_cultures`	Place the petri dish with the most cultures in the white box.

continued on next page

Category	Task ID	Instruction
spatial	`petri_closest_to_black_pen`	Place the petri dish that is closest to the black pen into the white box.
spatial	`middle_petri_dish`	Place the middle petri dish into the white box.
perceptual (mesh/texture)	`red_petri_on_absorbent`	Place the red petri dish on the absorbent item.
(2) Skill command / model size — household-themed tasks		
contextual (close)	`secure_drawer`	I'm done packing my drawer. Please secure the drawer.
contextual (fold)	`fold_wrap_up`	I am packing up my first aid supplies, wrap up.
contextual (wipe)	`wipe_spill`	Clear away the spill with the towel.
contextual (put-on)	`organize_pink_pepto`	Organize the pink Pepto with the other drugs.
rigid (put)	`place_safety_glasses_white_box`	The safety glasses go in the white box.
(3) Memory — factory-themed tasks		
unobservable goal	`two_white_blocks`	Place exactly two white blocks in the brown box.
observable goal	`healthy_and_unhealthy_snack`	I want exactly one healthy snack and exactly one unhealthy snack on the plate.
unobservable goal	`protein_and_sour_in_bag`	Place exactly one high-protein snack and exactly one sour fruit in the bag.

Intelligence

(a) `protect_eyes_equipment`: *I need to protect my eyes during my next experiment. Place the corresponding equipment in the white box.*

(b) `petri_most_cultures`: *Place the petri dish with the most cultures in the white box.*

(c) `petri_closest_to_black_pen`: *Place the petri dish that is closest to the black pen into the white box.*

(d) `middle_petri_dish`: *Place the middle petri dish into the white box.*

(e) `red_petri_on_absorbent`: *Place the red petri dish on the absorbent item.*

Skill

(f) `secure_drawer`: *I'm done packing my drawer. Please secure the drawer.*

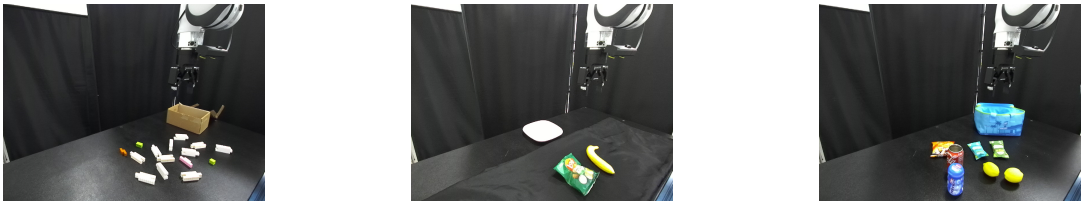
(g) `fold_compact_towel`: *I am packing up my first aid supplies, wrap up.*

(h) `wipe_spill`: *Clear away the spill with the towel.*

(i) `organize_pink_pepto`: *Organize the pink Pepto with the other drugs.*

(j) `lab_goggles_to_box`: *The safety glasses go in the white box.*

Memory



(k) `two_white_blocks`: *Place exactly two white blocks in the brown box.*

(l) `healthy_and_unhealthy_snack`: *I want exactly one healthy snack and exactly one unhealthy snack on the plate.*

(m) `protein_and_sour_in_bag`: *Place exactly one high-protein snack and exactly one sour fruit in the bag.*

Fig. 16: Per-task starting frames for the physical evaluation suite, grouped by test-time-compute axis: **Intelligence** (top, 5 tasks), **Skill** (middle, 5 tasks), and **Memory** (bottom, 3 tasks). Each card shows the starting-frame scene image, with the task ID and its instruction (*italicized*) below; see Table XXVII for the same tasks in tabular form.

C. Synthetic-Data Pipeline for the Hardware Router

Physical rollouts are expensive, so we train the DROID router on synthetic routing data rather than robot executions. At a high level, the pipeline (Fig. 6, shown in the main text) follows these steps:

Generate variants. Each canonical task is a representative scene paired with one example instruction. For each scene we generate many additional feasible instructions (the example serves only as a seed), indexed by routing axis (`INTELLIGENCE_FORM` / `SKILL_COMMAND` / `MEMORY`) and balanced across classes. The classes are chosen to *carry* the routing signal: `MEMORY` spans the memory cells (memory-needed / neither / -hurts), and `SKILL_COMMAND` contrasts rigid verbs (*put*, *pick*) with contextual ones (*close*, *fold*, *wipe*, *put-on*).

Query planners. For each variant we query every candidate planner (*e.g.*, 9B vs. 27B, think vs. no-think, memory vs. memory-free) for its subgoal decomposition against the grounded frame. Nothing runs on the robot; the output is a per-(model, variant) grid of plans.

Judge steerability. With no execution, an LLM judge proxies success: a decomposition is labeled a success iff its subgoals fall in the region of simple DROID tasks the downstream policy reliably executes, and a failure if they are too complex, abstract, or out-of-distribution. This yields per-(model, variant) steerability labels.

Train the router. The steerability labels are the routing targets: from instruction + scene, the router selects the planner whose plan stays in the high-steerability region — *e.g.*, memory vs. memory-free, or a larger vs. smaller planner for complex vs. simple skill commands.

D. RoboMME

RoboMME [13] benchmarks memory-augmented robotic generalist policies: a suite of long-horizon manipulation tasks evaluated over a family of memory-augmented VLA (MME-VLA) variants built on a $\pi_{0.5}$ backbone [6]. We use the benchmark unmodified; all results are drawn from the RoboMME leaderboard test set (the public evaluation results in `Yinpei/selected_robomme_eval_results`).

Difficulty suites. In addition to memory type, tasks are stratified into *easy*, *medium*, and *hard* levels based on scene clutter, horizon length, and environmental dynamics [13]; our per-suite Pareto analysis (Fig. 3 of the main paper) routes within each level.

Routed memory variants. From the RoboMME variant set we route over five selected memory architectures: `FrameSamp` and `TokenDrop` (perceptual memory that sub-samples / drops past visual tokens, adding no parameters), `SimpleSG` and `GroundSG` (symbolic memory that summarizes history as language subgoals via an auxiliary planner pass), and `MemER` [44] (retrieval over recalled keyframe history).

Cost metric (TFLOPs). We report inference cost via the standard $2NT$ scaling-law approximation (two FLOPs per parameter per token), $C = 2 \cdot N_{\text{params}} \cdot (T_{\text{in}} + T_{\text{out}})$. Each memory method is costed as the π_0 planner *base* cost plus the compute of its memory mechanism:

$$C_{\text{method}} = \underbrace{C_{\text{base}}}_{\text{vision + VLM prefix + 10-step diffusion decode}} + \underbrace{2\, N_{\text{aux}}\, T_{\text{mem}}}_{\text{memory overhead}} . \tag{9}$$

The constants are read from our stack: the π_0 backbone [7] is a `gemma_2b` VLM ($N_{\text{vlm}} = 2.114$B) plus a `gemma_300m` action expert ($N_{\text{exp}} = 0.302$B), and the base cost ($C_{\text{base}} \approx 3.34$ TFLOPs) is vision + the VLM prefix (176 tokens) + 10 denoising steps over the 51-token suffix. The symbolic and retrieval methods invoke an auxiliary Qwen3-VL-4B-Instruct planner ($N_{\text{aux}} \approx 4.0$B) [3]; the perceptual methods only re-weight existing visual tokens (sample / drop) and add no parameters, so their cost $\approx C_{\text{base}}$. Per-method values then follow from the token counts (Table XXVIII), giving an $\approx 8.6\times$ spread from `FrameSamp` to `MemER`.

TABLE XXVIII: Memory-cost derivation for our reported figure. Every method is the π_0 planner base cost plus a memory overhead. TFLOPs reproduce via $2NT$.

Method	**Accounting**	T_{mem}	**TFLOPs**
`FrameSamp` / `TokenDrop`	base only (token re-weighting)	—	3.338
`SimpleSG` / `GroundSG`	base + Qwen3-VL-4B subgoal pass	∼594	8.091
`MemER`	base + Qwen3-VL-4B keyframe retrieval	∼3178	28.761

The token budgets are estimates: ≈ 594 tokens for the symbolic pass (one frame + subgoal text) and ≈ 3178 for `MemER`'s ≈ 12 recalled keyframes.

Closed-weight variants and the physical cost proxy. The Gemini-predictor symbolic variants are omitted from the FLOPs accounting: their parameter count is unknown, so $2NT$ cannot be evaluated. This is also why the physical memory experiment falls back to *input-frame count* as its cost proxy rather than FLOPs (main paper, Sec. 5.3).

Relation to RoboMME's reported cost. Our reported FLOPs differ from RoboMME [13, Fig. 4] (3.75 / 6.26 / 10.15 / 15.37) because of four accounting choices. (i) *Base policy*: we price the π_0 planner (3.34 TFLOPs) whereas RoboMME prices $\pi_{0.5}$ (3.75). (ii) *Perceptual memory*: we treat `FrameSamp`/`TokenDrop` as token re-weighting with no added parameters (cost $\approx$ base, 3.34), whereas RoboMME charges the modulation memory-token overhead through the VLM ($+\sim 2.5 \rightarrow 6.26$).

(iii) *Retrieval depth*: our MemER recalls a longer keyframe history (≈ 12 frames, 28.76) than RoboMME's configuration (≈ 5 frames, 15.37). All use the $2NT$ estimate, so relative orderings hold even where magnitudes differ, and the largest gap (MemER) is almost entirely choice (iii).

APPENDIX H
PROMPTS

A. Synthetic-Pipeline Prompts (Task Generation and LLM-as-Judge)

The synthetic-data pipeline (Appendix G-C) uses two prompts: a task-generation prompt that proposes the next subtask, and an LLM-as-Judge prompt that labels/scores the candidate plans. We give the task-generation prompt first, followed by the judge.

1) Task-Generation Prompt (Synthetic Data and Hardware): The prompt below drives the synthetic-data pipeline and the hardware task generation: given the user request and the scene image, a large VLM proposes the immediate next subtask in the plain imperative style the downstream VLA policy can execute. <vlm_input> is replaced with the user request at generation time.

```
You give a tabletop robot understandable tasks so a vision-language manipulation
policy can follow them unambiguously. Generate the immediate next subtask that
aligns with the user's intent and the image. Subtasks are single instructions that
the robot VLA can execute. Examples of subtasks include:
- place the [COLOR] [ITEM] on the [COLOR] [ITEM]
- put the [COLOR] [ITEM] in the [COLOR] [ITEM]
- move the [COLOR] [ITEM] to the [COLOR] [ITEM]
- reach for the [COLOR] [ITEM]
- open the [COLOR] [ITEM]
- close the [COLOR] [ITEM]
- fold the [COLOR] [ITEM]
- wipe the [COLOR] [ITEM] with the [COLOR] [ITEM]
- remove the [COLOR] [ITEM] from the [COLOR] [ITEM]

DO NOT generate a sequence of subtasks. Generate only the immediate next subtask.
Use colors that distinguish objects, so if all the objects are the same color, use
a color that unambiguously distinguishes the target object from the other objects.
Use concrete colors and identifiable, concise object nouns inferred from what the
user said and the image (e.g. "red apple", "blue plate"). Clear imperative sentence
using the same plain style (object, color, spatial relation). Respond with only
that sentence -- no quotes, bullets, labels, or extra text.

USER_REQUEST: <vlm_input>
```

2) Synthetic LLM-as-Judge Prompts: The synthetic-data pipeline (Appendix G-C) uses two LLM-as-Judge prompts. The first labels whether a planner's predicted single-step subtask would drive the *same physical action* as the ground-truth primitive—i.e. whether it lands in the steerable, groundable primitive distribution rather than merely string-matching. The second labels whether a task requires *memory* across steps or can instead be re-prompted step-by-step from the scene alone. Both return strict JSON; we show condensed versions below (the full synonym lists and few-shot anchors are omitted for space), with fields in $\{\{\ldots\}\}$ interpolated per call.

Subtask-correctness judge. Decides whether the predicted subtask grounds to the same action as the ground truth, given the scene.

```
You are a strict but fair judge deciding whether a vision-language planner
produced a correct single-step subtask for a tabletop manipulation policy
(DROID / pi-0.5). The policy is not a reasoner: it executes only concrete,
groundable primitives. Decide whether the PREDICTION would drive the same
physical action as the GROUND_TRUTH given the scene -- not whether it
string-matches.

You are given:
- INSTRUCTION (high-level user request): {{vlm_input}}
- GROUND_TRUTH (canonical primitive(s)): {{vla_ground_truth}}
- PREDICTION (planner's emitted subtask): {{prediction}}
- SCENE (image): provided alongside this message

Return CORRECT only if ALL of the following hold; otherwise INCORRECT.

1. VERB. Accept verbs in the canonical grammar and same-class synonyms
   (place/put/move/set/drop/slide -> "put"; reach for/pick up -> "pick up";
   wipe/clean/mop -> "wipe"; close/shut -> "close"). A verb outside any class
   is INCORRECT only if it would change the VLA's behavior.
```

```
2. OBJECT. The referent must be correct and groundable. Color must match GT or
   be camera-confusable (clear/white/transparent; red/pink/orange;
   purple/violet); categorically different colors are INCORRECT. Noun synonyms
   are fine when only one such item is present (dish/petri dish; goggles/safety
   glasses; block/brick). Dropping a brand word is fine; dropping a
   disambiguating color is not.

3. DESTINATION. If GT requires placement ("put X in/on Y"), a prediction that
   only picks up drops the step -> INCORRECT. A bare pickup matched by "pick up
   X" is CORRECT. Destination color/type must match (or be camera-confusable).

4. DISAMBIGUATION. If multiple candidates exist and the prediction drops the
   distinguishing attribute -> INCORRECT. In singular scenes a missing color
   is fine.

5. ABSTRACTION. If the instruction contains an abstraction the VLA cannot
   ground (color-by-analogy, density / world-knowledge, exclusion reasoning),
   the prediction must RESOLVE it to the concrete color/object in GROUND_TRUTH;
   parroting the abstraction -> INCORRECT. Simple spatial location
   (leftmost / rightmost / beside) is observable, not an abstraction.

6. Off-distribution color words the VLA likely won't bind ("lime") -> INCORRECT.
7. Empty prediction -> INCORRECT.

Reason about the scene first, then decide. Output ONLY this JSON:
{"llm_judge_match": true|false, "reason": "<one concise sentence>"}
```

Memory-need / routing judge. Decides whether the task needs cross-step memory or is observable step-by-step from the scene (gripper state and source-object disappearance).

```
You are routing a tabletop manipulation task. Decide whether executing it
correctly requires MEMORY across steps, or whether the scene alone (gripper
state and which source objects have disappeared) determines the next action at
every step. The planner is re-called fresh at each step with the SAME
instruction and the CURRENT scene; it has no conversation history.

- MEMORY IS NEEDED when the same scene state recurs across steps, or the scene
  does not uniquely determine which step the task is on.
- NO MEMORY when progress is observable: source objects disappear as they are
  placed, the held object is visible in the gripper, or the destination fills.

You are given:
- INSTRUCTION: {{instruction}}
- GROUND_TRUTH sequence of primitives: {{vla_ground_truth}}
- SCENE INVENTORY (optional): which objects are abundant / duplicated, and
  whether the destination is opaque: {{scene_inventory}}

Apply these rules in order; the first that matches decides:
R1. Single step -> NO memory (no progress to track).
R2. Repeat of the same action on an abundant / duplicated source -> MEMORY
    (cannot tell from the scene how many are already done).
R3. Repeat on a singleton: repeat-wipe -> MEMORY (the spill may not visibly
    change); repeat-pickup -> NO memory (held-vs-not-held is visible).
R4. Cycle / toggle over 3+ steps where a state recurs (open->close->open) ->
    MEMORY (a step counter is needed to know when to stop).
R5. Pickup-then-place of a singleton -> NO memory (held object visible in the
    gripper after the pickup).
R6. Ordered placement of distinct singletons, even into an opaque destination
    -> NO memory (source-side disappearance tracks order).
R7. Default: if the scene uniquely determines the next action -> NO memory;
    otherwise -> MEMORY.

Reason about whether the scene reveals progress, then decide. Output ONLY:
{"needs_memory": true|false, "reason": "<one sentence>", "rule": "<R1-R7>"}
```

B. VLABench Planning Prompt

The system prompt below is given to every VLABench planner. It defines nine skills (`pick`, `place`, `press`, `open_door`, `insert`, `pull`, `pour`, `push`, `lift`) with their JSON call formats, describes the two-image input (unlabeled views + numbered views), and asks the model to emit a JSON skill-call sequence matching one of the allowed sub-skill patterns (e.g. `["pick","place"]`, `["pick","pour","place"]`). We use this English prompt; an equivalent Chinese version (`eval_vlm_zh.txt`) is also available.

In Skill Lab, we provide a series of robot skills to support efficient automated operations for various tasks. Each skill has a specific call format, which includes the skill name and corresponding parameters.

The available skills are as follows:

1. pick: Used to grasp and pick up a target object.
   - Call format:
     {
       "name": "pick",
       "params": { "target_entity_name": Target Number }
     }

2. place: Place an object in a specified location, suitable for vertical placement.
   - Call format:
     {
       "name": "place",
       "params": { "target_container_name": Target Number }
     }

3. press: Press a specified location or button.
   - Call format:
     {
       "name": "press",
       "params": { "target_entity_name": Target Number }
     }

4. open_door: Open a door.
   - Call format:
     { "name": "open_door", "params": { } }

5. insert: Insert an item into a target location.
   - Call format:
     {
       "name": "insert",
       "params": { "target_container_name": Target Number }
     }

6. pull: Pull the robotic arm horizontally.
   - Call format:
     { "name": "pull", "params": { } }

7. pour: Pour a liquid or granular substance.
   - Call format:
     {
       "name": "pour",
       "params": { "target_container_name": Target Number }
     }

8. push: Push a target object horizontally.
   - Call format:
     {
       "name": "push",
       "params": { "target_container_name": Target Number }
     }

9. lift: Lift the robotic arm vertically.
   - Call format:
     { "name": "lift", "params": { } }

These call formats ensure that each skill operation has clearly defined parameters, allowing the system to accurately execute the specified automated tasks.

You will receive the following input:

1. Image input: Two images
   - The first image shows four different perspectives of objects (without labels).
   - The second image contains the same four perspectives of objects, but each view is labeled with a number (representing each object’s identifier).

2. Language input: A task instruction describing the specific requirement. Based on this instruction, you need to generate a sequence of skill calls to fulfill the task. Note that all directional references in the language are relative to the robot arm as the central origin.

Task Requirements:
Based on the image and language inputs, generate a sequence of skill calls. Each skill call sequence should contain the skill name (extracted from the task instruction), the skill operation parameters (if the skill requires parameters), and the target entity or container number (obtained from the labeled image).

Generation Steps:
1. Extract Task Instruction: Identify operation requirements (e.g., adjust,

```
inspect, move) and the target object number or view from the language input.
2. Combine with Image Information: Using the labeled image, match the objects
described in the task instruction with their corresponding identifiers to determine
the target number.
3. Generate Skill Call Sequence: Based on the task instruction, use the extracted
operation requirements as skill names and assign them the appropriate number
information.
4. Output Format: Generate a skill call sequence in the following structure:
[
    { "name": "Skill Name 1", "params": { "parameter": "value" } },
    { "name": "Skill Name 2", "params": { "parameter": "value" } }
]

Since the evaluation process will extract patterns from the skill sequence to build
an operation graph, the output skill call sequence should ideally match one or more
of the following patterns to ensure correctness:

Sub-skill sequence patterns:
- ["pick", "place"]
- ["pick", "insert"]
- ["pick", "pour", "place"]
- ["pick", "pour"]
- ["pick", "pull"]
- ["pick", "lift"]
- ["pick", "push", "pull"]
- ["pick", "push", "place"]
- ["pick", "push"]
- ["pick", "open_door"]
- ["press"]

For example, if the output only includes a single "pick" skill, it will be
considered incorrect during the evaluation.
Please pay close attention to spatial information and ensure accurate pattern
selection. For example, if the object needs to be picked up and displayed, use the
pattern ["pick", "lift"]. If the object needs to be pulled out and displayed, use
the pattern ["pick", "pull"]. If ["pick", "place"] is used incorrectly in place of
these patterns, it will be considered an error.
```

APPENDIX I
COMPLETE ROUTER SWEEP (REPRESENTATIVE PAIR)

The per-pair ablation tables in Appendix C are summaries: for each embedding-fusion method they list only the single best router under each selection criterion (best average score and best η). For completeness, Table XXIX gives the *full* sweep for one representative pair (GLM-4.1V-Thinking vs. GLM-4.6V-Flash) — every trained router (architecture $\times$ fusion method $\times$ head), followed by the fixed baselines, oracles, the complete OOD sweep (every fusion method $\times$ γ), and the full budget-oracle sweep — so the reader can see the complete option space behind the summarized tables. The same sweep is available for every pair in our released results.

TABLE XXIX: **Complete router sweep for a representative pair (GLM-4.1V-Thinking vs. GLM-4.6V-Flash), sorted by η.** Every trained router, i.e. each (architecture $\times$ embedding-fusion method $\times$ head) combination, ordered highest-to-lowest η. The *Fusion Method* sets how the router-input embedding is formed: `concat` / `normalize_concat` / `average` / `weighted_average` combine the `text` (instruction) and `image` (scene; SigLIP-2) embeddings, while `only_text` / `only_image` use one of them alone and `only_vla` uses the π_0 VLA policy embedding alone (full definitions in App. B-A). Avg score and η are $\times 100$; avg cost in seconds; η cells shaded red (low) to green (high). Fixed baselines, oracles, the OOD sweep, and the budget-oracle frontier are listed in full at the foot of the table.

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
only_text	linear_only_text_softλ0.1	51.82	31.86	75.29
concat	linear_concat_softλ0.1	51.79	31.82	75.26
normalize_concat	linear_normalize_concat_softλ0.1	51.79	31.82	75.26
normalize_concat	linear_normalize_concat_tie_regression	49.89	26.52	75.12
concat	linear_concat_tie_regression	49.89	26.57	75.08
weighted_average_text075	linear_weighted_average_softλ0.1	51.69	32.07	74.72
only_image	mlp_only_image_h2048_1024	51.47	31.68	74.68
average	linear_average_tie_regression	49.51	25.83	74.46
weighted_average_text075	linear_weighted_average_tie_regression	49.48	25.75	74.41
only_text	mlp_only_text_h2048_1024	50.99	30.94	74.36
weighted_average_text075	mlp_weighted_average_h2048_1024	51.13	31.29	74.32
average	linear_average_softλ0.1	51.89	32.79	74.22

continued on next page

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
weighted_average_text025	linear_weighted_average_softλ0.1	52.17	33.23	74.20
only_text	linear_only_text_tie_regression	49.54	26.62	74.06
weighted_average_text025	linear_weighted_average_tie_regression	49.57	26.78	74.05
concat	mlp_concat_h2048_1024	50.80	30.82	74.01
weighted_average_text025	mlp_weighted_average_h2048_1024_tie_regression	51.99	33.16	73.90
concat	knn_k5_concat	51.27	31.99	73.83
concat	prknn_k5_concat	51.27	31.99	73.83
normalize_concat	knn_k5_normalize_concat	51.27	31.99	73.83
normalize_concat	prknn_k5_normalize_concat	51.27	31.99	73.83
weighted_average_text025	knn_k5_weighted_average_w0.25	51.06	31.55	73.83
weighted_average_text025	prknn_k5_weighted_average	51.06	31.55	73.83
average	mlp_average_h2048_1024	50.37	29.93	73.77
only_image	linear_only_image_tie_regression	49.61	27.42	73.73
average	mlp_average_h2048_1024_tie_regression	52.62	34.19	73.60
average	knn_k5_average	51.15	31.96	73.58
average	prknn_k5_average	51.15	31.96	73.58
weighted_average_text025	mlp_weighted_average_h2048_1024	50.42	30.31	73.55
weighted_average_text075	kmeans_weighted_average	49.80	28.62	73.38
only_image	linear_only_image_softλ0.1	51.37	32.58	73.30
average	kmeans_average	49.53	27.80	73.27
weighted_average_text025	linear_weighted_average_regression	52.58	34.35	73.23
only_text	ovr_only_text	49.71	28.59	73.16
normalize_concat	mlp_normalize_concat_h2048_1024	50.19	30.13	73.12
normalize_concat	mlp_normalize_concat_h2048_1024_tie_regression	52.17	33.90	73.12
only_text	ovr_only_text_tie_regression	50.66	31.39	73.05
weighted_average_text075	ovr_weighted_average_w0.75	49.40	27.63	73.03
weighted_average_text075	knn_k5_weighted_average_w0.75	50.83	31.77	73.01
weighted_average_text075	prknn_k5_weighted_average	50.83	31.77	73.01
only_image	linear_only_image_regression	52.77	34.69	73.00
weighted_average_text075	ovr_weighted_average_w0.75_tie_regression	50.75	31.62	73.00
only_text	kmeans_only_text	49.80	29.14	72.98
concat	mlp_concat_h2048_1024_tie_regression	51.54	33.12	72.93
weighted_average_text075	knn_k5_weighted_average_w0.75_tie_regression	52.00	33.86	72.83
weighted_average_text075	prknn_k5_weighted_average_tie_regression	52.00	33.86	72.83
concat	linear_concat_regression	52.38	34.35	72.81
normalize_concat	linear_normalize_concat_regression	52.38	34.35	72.81
concat	knn_k5_concat_tie_regression	52.20	34.18	72.74
concat	prknn_k5_concat_tie_regression	52.20	34.18	72.74
normalize_concat	knn_k5_normalize_concat_tie_regression	52.20	34.18	72.74
normalize_concat	prknn_k5_normalize_concat_tie_regression	52.20	34.18	72.74
weighted_average_text025	ovr_weighted_average_w0.25_tie_regression	51.13	32.60	72.71
weighted_average_text075	mlp_weighted_average_h2048_1024_tie_regression	52.37	34.41	72.68
average	ovr_average_tie_regression	51.36	33.04	72.63
average	knn_k5_average_tie_regression	52.13	34.17	72.60
average	prknn_k5_average_tie_regression	52.13	34.17	72.60
only_image	ovr_only_image_tie_regression	52.37	34.45	72.60
only_vla	knn_k5_only_vla	50.94	32.34	72.60
only_vla	prknn_k5_only_vla	50.94	32.34	72.60

continued on next page

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
only_text	mlp_only_text_h2048_1024_tie_regression	52.61	34.73	72.59
average	linear_average_regression	52.61	34.74	72.57
only_text	knn_k5_only_text	50.51	31.49	72.55
only_text	prknn_k5_only_text_tie_regression	52.05	34.12	72.50
only_text	prknn_k5_only_text	50.40	31.29	72.49
weighted_average_text025	knn_k5_weighted_average_w0.25_tie_regression	52.24	34.43	72.38
weighted_average_text025	prknn_k5_weighted_average_tie_regression	52.24	34.43	72.38
only_vla	mlp_only_vla_h2048_1024_tie_regression	50.79	32.23	72.37
only_vla	mlp_only_vla_h2048_1024	50.62	32.05	72.19
only_text	knn_k5_only_text_tie_regression	52.10	34.41	72.11
average	ovr_average	48.88	27.28	71.82
only_vla	knn_k5_only_vla_tie_regression	52.45	34.99	71.75
only_vla	prknn_k5_only_vla_tie_regression	52.45	34.99	71.75
only_image	knn_k5_only_image	50.54	32.40	71.56
only_image	prknn_k5_only_image	50.54	32.40	71.56
weighted_average_text025	ovr_weighted_average_w0.25	48.74	27.12	71.52
concat	kmeans_concat	48.51	26.20	71.41
normalize_concat	kmeans_normalize_concat	48.51	26.20	71.41
only_image	knn_k5_only_image_tie_regression	52.29	35.11	71.19
only_image	prknn_k5_only_image_tie_regression	52.29	35.11	71.19
only_vla	ovr_only_vla_tie_regression	52.58	35.42	71.15
only_vla	ovr_only_vla	48.82	28.29	70.98
weighted_average_text075	linear_weighted_average_regression	52.78	35.73	70.86
weighted_average_text025	kmeans_weighted_average	48.28	26.20	70.74
only_vla	linear_only_vla_tie_regression	48.72	28.46	70.57
only_text	linear_only_text_regression	52.80	35.94	70.44
only_image	ovr_only_image	48.30	27.37	70.13
concat	ovr_concat_tie_regression	49.57	31.61	70.08
normalize_concat	ovr_normalize_concat_tie_regression	49.57	31.61	70.08
normalize_concat	ovr_normalize_concat	48.12	27.25	69.69
concat	ovr_concat	47.98	27.18	69.33
only_image	kmeans_only_image	47.60	25.65	69.07
concat	mlp_concat_h2048_1024_rank_regression	53.33	37.01	68.68
normalize_concat	mlp_normalize_concat_h2048_1024_rank_regression	53.50	37.52	67.47
only_vla	linear_only_vla_softλ0.1	47.21	27.45	66.97
concat	knn_k5_concat_regression	53.34	37.75	66.46
concat	prknn_k5_concat_regression	53.34	37.75	66.46
normalize_concat	knn_k5_normalize_concat_regression	53.34	37.75	66.46
normalize_concat	prknn_k5_normalize_concat_regression	53.34	37.75	66.46
average	knn_k5_average_regression	53.30	37.76	66.38
average	prknn_k5_average_regression	53.30	37.76	66.38
only_text	knn_k5_only_text_regression	53.25	37.75	66.34
weighted_average_text075	knn_k5_weighted_average_w0.75_regression	53.43	37.84	66.34
weighted_average_text075	prknn_k5_weighted_average_regression	53.43	37.84	66.34
only_text	prknn_k5_only_text_regression	53.31	37.84	66.15
only_image	mlp_only_image_h2048_1024_tie_regression	53.36	37.90	66.02
weighted_average_text025	knn_k5_weighted_average_w0.25_regression	53.40	38.10	65.42
weighted_average_text025	prknn_k5_weighted_average_regression	53.40	38.10	65.42
concat	mlp_concat_h2048_1024_regression	53.43	38.26	64.91
only_vla	linear_only_vla_regression	52.84	38.00	64.86
only_vla	mlp_only_vla_h2048_1024_rank_regression	52.09	37.63	64.72
average	mlp_average_h2048_1024_rank_regression	53.29	38.34	64.41

continued on next page

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
average	mlp_average_h2048_1024_regression	53.47	38.52	64.02
concat	ovr_concat_rank_regression	51.22	37.39	63.89
normalize_concat	ovr_normalize_concat_rank_regression	51.22	37.39	63.89
concat	ovr_concat_regression	51.55	37.62	63.84
normalize_concat	ovr_normalize_concat_regression	51.55	37.62	63.84
only_vla	knn_k5_only_vla_regression	53.46	38.60	63.71
only_vla	prknn_k5_only_vla_regression	53.46	38.60	63.71
only_image	knn_k5_only_image_regression	53.15	38.49	63.66
only_vla	mlp_only_vla_h2048_1024_regression	52.37	38.18	63.50
normalize_concat	mlp_normalize_concat_h2048_1024_regression	53.63	38.80	63.20
only_image	prknn_k5_only_image_regression	53.35	38.69	63.20
only_vla	kmeans_only_vla_regression	46.88	32.17	62.71
only_text	kmeans_only_text_regression	46.27	30.36	62.53
weighted_average_text075	kmeans_weighted_average_regression	46.17	30.41	62.21
concat	linear_concat_rank_regression	52.52	38.81	61.52
normalize_concat	linear_normalize_concat_rank_regression	52.52	38.81	61.52
only_image	mlp_only_image_h2048_1024_rank_regression	53.88	39.41	61.02
average	kmeans_average_regression	45.37	29.65	60.49
only_image	mlp_only_image_h2048_1024_regression	53.86	39.65	59.84
weighted_average_text075	ovr_weighted_average_w0.75_rank_regression	52.18	39.16	59.68
weighted_average_text025	ovr_weighted_average_w0.25_regression	52.90	39.49	59.35
only_vla	kmeans_only_vla	44.03	22.76	59.29
weighted_average_text025	ovr_weighted_average_w0.25_rank_regression	52.53	39.39	59.27
average	linear_average_rank_regression	52.51	39.39	59.22
average	ovr_average_rank_regression	52.41	39.36	59.19
only_image	linear_only_image_rank_regression	52.83	39.59	58.78
weighted_average_text025	linear_weighted_average_rank_regression	52.64	39.59	58.53
only_text	ovr_only_text_rank_regression	52.58	39.76	57.70
average	ovr_average_regression	52.73	39.88	57.34
weighted_average_text075	ovr_weighted_average_w0.75_regression	52.44	39.87	56.99
only_text	ovr_only_text_regression	52.72	40.02	56.63
weighted_average_text075	linear_weighted_average_rank_regression	52.61	40.20	55.64
only_vla	ovr_only_vla_rank_regression	53.25	40.40	55.34
only_text	linear_only_text_rank_regression	52.56	40.38	54.63
only_vla	ovr_only_vla_regression	53.51	40.65	54.23
concat	kmeans_concat_regression	42.79	26.96	54.12
normalize_concat	kmeans_normalize_concat_regression	42.79	26.96	54.12
only_image	ovr_only_image_regression	53.53	40.96	52.37
only_vla	kmeans_only_vla_rank_regression	41.65	23.93	51.39
weighted_average_text025	kmeans_weighted_average_regression	41.66	25.97	50.89
only_image	kmeans_only_image_regression	41.00	24.75	49.10
weighted_average_text025	mlp_weighted_average_h2048_1024_rank_regression	53.00	41.40	48.97
only_text	knn_k5_only_text_rank_regression	53.18	41.47	48.66
only_text	prknn_k5_only_text_rank_regression	53.18	41.47	48.66
only_image	ovr_only_image_rank_regression	53.35	41.55	48.29
only_vla	linear_only_vla_rank_regression	52.96	41.56	47.83
only_text	kmeans_only_text_rank_regression	40.11	19.41	47.07
average	kmeans_average_rank_regression	40.08	19.51	46.94
weighted_average_text075	kmeans_weighted_average_rank_regression	39.91	18.94	46.45
weighted_average_text075	knn_k5_weighted_average_w0.75_rank_regression	53.20	41.82	46.15
weighted_average_text075	prknn_k5_weighted_average_rank_regression	53.20	41.82	46.15
only_text	mlp_only_text_h2048_1024_rank_regression	53.61	41.96	45.35

continued on next page

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
only_image	knn_k5_only_image_rank_regression	53.20	42.02	44.50
only_image	prknn_k5_only_image_rank_regression	53.20	42.02	44.50
weighted_average_text075	mlp_weighted_average_h2048_1024_rank_regression	53.44	42.15	43.62
average	knn_k5_average_rank_regression	53.29	42.16	43.45
average	prknn_k5_average_rank_regression	53.29	42.16	43.45
concat	knn_k5_concat_rank_regression	53.35	42.41	41.28
concat	prknn_k5_concat_rank_regression	53.35	42.41	41.28
normalize_concat	knn_k5_normalize_concat_rank_regression	53.35	42.41	41.28
normalize_concat	prknn_k5_normalize_concat_rank_regression	53.35	42.41	41.28
weighted_average_text025	kmeans_weighted_average_rank_regression	38.32	17.76	41.08
only_image	kmeans_only_image_rank_regression	38.26	18.29	40.83
concat	kmeans_concat_rank_regression	38.23	17.67	40.77
normalize_concat	kmeans_normalize_concat_rank_regression	38.23	17.67	40.77
weighted_average_text025	mlp_weighted_average_h2048_1024_regression	53.51	42.79	37.63
only_text	mlp_only_text_h2048_1024_regression	53.90	42.83	37.41
weighted_average_text025	knn_k5_weighted_average_w0.25_rank_regression	53.51	42.93	36.20
weighted_average_text025	prknn_k5_weighted_average_rank_regression	53.51	42.93	36.20
weighted_average_text075	mlp_weighted_average_h2048_1024_regression	53.63	43.12	34.11
only_vla	knn_k5_only_vla_rank_regression	53.74	43.41	30.67
only_vla	prknn_k5_only_vla_rank_regression	53.74	43.41	30.67
only_text	kmeans_only_text_tie_regression	29.02	3.36	6.96
weighted_average_text075	kmeans_weighted_average_tie_regression	28.96	3.34	6.74
average	kmeans_average_tie_regression	28.93	3.39	6.63
only_vla	kmeans_only_vla_tie_regression	28.89	4.33	6.47
only_image	kmeans_only_image_tie_regression	27.65	7.00	1.64
weighted_average_text025	kmeans_weighted_average_tie_regression	27.49	6.73	0.98
concat	kmeans_concat_tie_regression	27.40	6.26	0.66
normalize_concat	kmeans_normalize_concat_tie_regression	27.40	6.26	0.66
Fixed baselines and oracles				
—	expensive_global	54.02	45.10	0.00
—	strongest_global	54.02	45.10	0.00
—	strongest_per_category	54.02	45.10	0.00
—	random	40.02	22.39	46.33
—	cheapest_global	27.24	2.11	0.00
—	accuracy_oracle	55.17	27.96	87.94
—	η oracle	55.17	33.32	80.59
OOD baseline sweep — all fusion methods $\times\ \gamma \in \{0, 0.1, \ldots, 1\}$				
concat	ood_baseline_success_000	28.29	8.59	4.14
concat	ood_baseline_success_010	28.29	8.59	4.14
concat	ood_baseline_success_020	28.42	8.77	4.64
concat	ood_baseline_success_030	34.03	16.52	25.81
concat	ood_baseline_success_040	37.04	21.70	36.28
concat	ood_baseline_success_050	49.83	39.60	54.52
concat	ood_baseline_success_060	49.83	39.60	54.52
concat	ood_baseline_success_070	49.83	39.60	54.52
concat	ood_baseline_success_080	53.66	43.75	26.01
concat	ood_baseline_success_090	53.66	43.75	26.01
concat	ood_baseline_success_100	54.02	45.10	0.00
normalize_concat	ood_baseline_success_000	28.29	8.59	4.14
normalize_concat	ood_baseline_success_010	28.29	8.59	4.14
normalize_concat	ood_baseline_success_020	28.42	8.77	4.64
normalize_concat	ood_baseline_success_030	34.03	16.52	25.81
normalize_concat	ood_baseline_success_040	37.04	21.70	36.28
normalize_concat	ood_baseline_success_050	49.83	39.60	54.52
normalize_concat	ood_baseline_success_060	49.83	39.60	54.52

continued on next page

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
normalize_concat	ood_baseline_success_070	49.83	39.60	54.52
normalize_concat	ood_baseline_success_080	53.66	43.75	26.01
normalize_concat	ood_baseline_success_090	53.66	43.75	26.01
normalize_concat	ood_baseline_success_100	54.02	45.10	0.00
average	ood_baseline_success_000	27.24	2.11	0.00
average	ood_baseline_success_010	27.24	2.11	0.00
average	ood_baseline_success_020	27.24	2.11	0.00
average	ood_baseline_success_030	27.24	2.11	0.00
average	ood_baseline_success_040	27.24	2.11	0.00
average	ood_baseline_success_050	27.24	2.11	0.00
average	ood_baseline_success_060	27.24	2.11	0.00
average	ood_baseline_success_070	27.24	2.11	0.00
average	ood_baseline_success_080	27.24	2.11	0.00
average	ood_baseline_success_090	27.24	2.11	0.00
average	ood_baseline_success_100	54.02	45.10	0.00
weighted_average_text025	ood_baseline_success_000	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_010	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_020	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_030	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_040	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_050	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_060	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_070	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_080	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_090	27.24	2.11	0.00
weighted_average_text025	ood_baseline_success_100	54.02	45.10	0.00
weighted_average_text075	ood_baseline_success_000	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_010	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_020	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_030	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_040	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_050	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_060	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_070	27.24	2.11	0.00
weighted_average_text075	ood_baseline_success_080	28.13	3.92	3.50
weighted_average_text075	ood_baseline_success_090	28.13	3.92	3.50
weighted_average_text075	ood_baseline_success_100	54.02	45.10	0.00
only_text	ood_baseline_success_000	27.24	2.11	0.00
only_text	ood_baseline_success_010	27.24	2.11	0.00
only_text	ood_baseline_success_020	27.24	2.11	0.00
only_text	ood_baseline_success_030	27.24	2.11	0.00
only_text	ood_baseline_success_040	27.24	2.11	0.00
only_text	ood_baseline_success_050	27.24	2.11	0.00
only_text	ood_baseline_success_060	27.24	2.11	0.00
only_text	ood_baseline_success_070	27.24	2.11	0.00
only_text	ood_baseline_success_080	37.28	22.40	37.03
only_text	ood_baseline_success_090	37.28	22.40	37.03

continued on next page

Fusion Method	Router (arch_fusion_head)	Avg Score	Avg Cost (s)	η
only_text	ood_baseline_success_100	54.02	45.10	0.00
only_image	ood_baseline_success_000	27.24	2.11	0.00
only_image	ood_baseline_success_010	27.24	2.11	0.00
only_image	ood_baseline_success_020	27.24	2.11	0.00
only_image	ood_baseline_success_030	27.24	2.11	0.00
only_image	ood_baseline_success_040	27.24	2.11	0.00
only_image	ood_baseline_success_050	27.24	2.11	0.00
only_image	ood_baseline_success_060	27.24	2.11	0.00
only_image	ood_baseline_success_070	27.24	2.11	0.00
only_image	ood_baseline_success_080	28.63	3.36	5.44
only_image	ood_baseline_success_090	28.63	3.36	5.44
only_image	ood_baseline_success_100	54.02	45.10	0.00
only_vla	ood_baseline_success_000	54.02	45.10	0.00
only_vla	ood_baseline_success_010	54.02	45.10	0.00
only_vla	ood_baseline_success_020	54.02	45.10	0.00
only_vla	ood_baseline_success_030	54.02	45.10	0.00
only_vla	ood_baseline_success_040	54.02	45.10	0.00
only_vla	ood_baseline_success_050	54.02	45.10	0.00
only_vla	ood_baseline_success_060	54.02	45.10	0.00
only_vla	ood_baseline_success_070	54.02	45.10	0.00
only_vla	ood_baseline_success_080	54.02	45.10	0.00
only_vla	ood_baseline_success_090	54.02	45.10	0.00
only_vla	ood_baseline_success_100	54.02	45.10	0.00
Budget-oracle sweep — $x \in \{0, 5, \ldots, 100\}\%$				
—	accuracy_oracle_budget_000pct	27.24	2.11	0.00
—	accuracy_oracle_budget_005pct	28.34	2.52	4.30
—	accuracy_oracle_budget_010pct	45.46	13.97	65.83
—	accuracy_oracle_budget_015pct	51.99	21.26	84.04
—	accuracy_oracle_budget_020pct	53.75	24.05	87.45
—	accuracy_oracle_budget_025pct	54.56	25.79	88.36
—	accuracy_oracle_budget_030pct	54.73	26.41	88.27
—	accuracy_oracle_budget_035pct	55.10	27.58	88.16
—	accuracy_oracle_budget_040pct	55.10	27.58	88.16
—	accuracy_oracle_budget_045pct	55.17	27.96	87.94
—	accuracy_oracle_budget_050pct	55.17	27.96	87.94
—	accuracy_oracle_budget_055pct	55.17	27.96	87.94
—	accuracy_oracle_budget_060pct	55.17	27.96	87.94
—	accuracy_oracle_budget_065pct	55.17	27.96	87.94
—	accuracy_oracle_budget_070pct	55.17	27.96	87.94
—	accuracy_oracle_budget_075pct	55.17	27.96	87.94
—	accuracy_oracle_budget_080pct	55.17	27.96	87.94
—	accuracy_oracle_budget_085pct	55.17	27.96	87.94
—	accuracy_oracle_budget_090pct	55.17	27.96	87.94
—	accuracy_oracle_budget_095pct	55.17	27.96	87.94
—	accuracy_oracle_budget_100pct	55.17	27.96	87.94