Multi-Head Latent Control: A Unified Interface for LLM Agent Decision Making

Amirhosein Ghasemabadi[1,*], Ruichen Chen[1], Bahador Rashidi[2], Di Niu[1]
[1]ECE Department, University of Alberta
[2]Huawei Technologies Canada Co., Ltd.
{ghasemab, ruichen1, dniu}@ualberta.ca
rashidibahador92@gmail.com
[*]Corresponding author

Abstract

Large language models are increasingly deployed as agents, but reliable agentic behavior requires more than next-token prediction. At inference time, it is preferred that an agent can decide whether to proceed with its current reasoning, defer to a stronger model, request additional information, invoke external tools, or abstain under the given setup. Existing approaches address these decisions through prompt-level routing, external orchestration, or task-specific fine-tuning, which primarily rely on input-side signals, and are often costly and difficult to maintain as model backbones evolve. We ask whether such control decisions can be inferred directly from a model's latent generation process. We introduce **Multi-Head Latent Control**, a lightweight layer that reads hidden-state trajectories from a frozen LLM or VLM to produce deployment-time control signals. A **Capability Head** predicts whether the current model can solve the instance or should defer to a stronger collaborator, while a **Resolution Head** predicts appropriate resolution decision *Clarification*, *Tool Use*, *Abstention*, or *Direct Answering*. Both heads are trained only on latent traces from the same frozen LLM backbone, enabling post hoc adaptation without modifying the model. Across language and vision-language settings, Multi-Head Latent Control consistently improves the quality–cost tradeoff of multi-model systems, enabling early handoff from partial generations and more accurate intervention decisions. In routed execution (small + large model), it reduces large-model usage by up to 90.7% on AndroidWorld and 27–53% on average across benchmarks, while retaining most of large-model performance. Additionally, the learned control signals improve tool-use decision quality, yielding up to +158% relative score gain and 65.5% fewer missed-required tool calls.

Code: https://github.com/Amirhosein-gh98/Multi-Head-Latent-Control

1 Introduction

Foundation models are increasingly capable, but reliable deployment requires more than strong next-token prediction. In long-horizon, multi-step, and tool-augmented agentic settings, an LLM must reason not only about what to generate, but also whether it can solve the current task or should defer to a stronger model. It must further determine whether additional information is required, whether external tools should be invoked, or whether the task is infeasible under the current setup. Errors in these decisions or overly conservative strategies such as always invoking tools or using a larger LLM for tasks are costly, leading to unnecessary computation, increased latency, avoidable external calls, and compounding downstream failures.

Preprint.

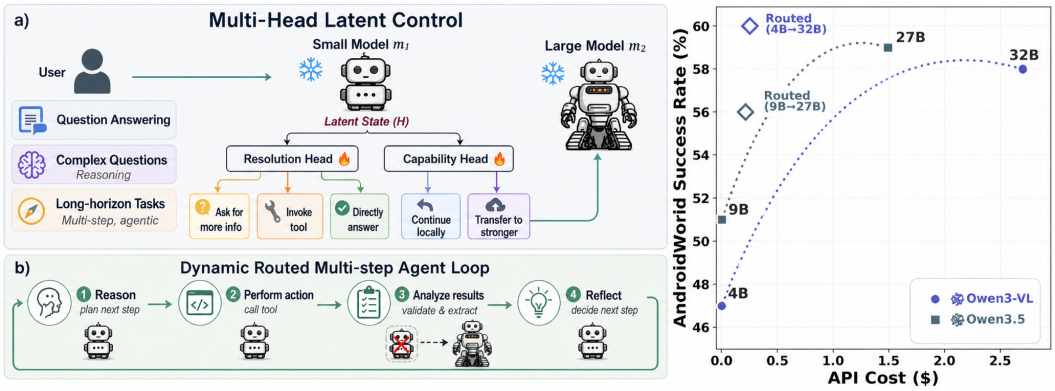

Figure 1: **Multi-head latent control as an intrinsic control interface for frozen foundation models.** **(a)** From the latent trajectory of a frozen primary model m_1, a *Capability Head* decides whether to remain with m_1 or hand off to a stronger model m_2, while a *Resolution Head* selects the appropriate resolution decision: request more information, invoke tools, abstain, or proceed directly. **(b)** Repeated use of these control signals enables dynamic routed execution in long-horizon, multi-step agent loops, allowing the system to choose the appropriate model at each step instead of relying on a large model throughout the entire trajectory. **(c)** On AndroidWorld, routed execution substantially improves the quality-cost tradeoff over single-model baselines by using the large model selectively.

This challenge is particularly evident in emerging commercial agentic systems built on frontier models (e.g., Anthropic's Claude agents), where token consumption remains high even for relatively simple tasks. This inefficiency suggests substantial untapped potential for reducing cost by enabling self-awareness in agentic LLMs allowing models to assess their own capability and act accordingly during inference.

Existing work addresses parts of this problem, but mostly through external mechanisms. Prompt- and category-level routing methods [6, 29, 7, 9] learn to assign queries to models or pipelines based on input-side signals such as prompt features or predicted task categories, improving coarse cost–quality tradeoffs but without assessing instance-level model adequacy. Multi-agent collaboration and orchestration methods [10, 15, 40, 39] improve capability through interaction among models, tools, or roles, but often rely on heavier scaffolding or dedicated end-to-end training to make models better collaborators or orchestrators. As stronger LLMs and VLMs are released at an accelerating pace, repeatedly Fine-tuning each new backbone for such roles fails to keep pace with the evolution of foundation models.

An orthogonal yet complementary line of work focuses on decoding-side methods, such as speculative decoding and reward-guided decoding strategies [19, 38, 22], which improve efficiency during generation by accelerating large-model inference once it is invoked. In contrast, our goal is to reduce how often large models are invoked by making deployment-time control decisions. Rather than replacing decoding-level optimization, we introduce a complementary layer: a lightweight intrinsic control interface that derives deployment-time decisions directly from the model's latent states.

In this paper, we show that frozen foundation models can expose a practical control interface for deployment-time decision-making without fine-tuning the backbone. We introduce *Multi-Head Latent Control*, consisting of two lightweight heads: a **Capability Head** and a **Resolution Head**, which attach to a frozen language or vision-language model and emit actionable control signals during inference. By reading hidden states from the model's own generation process, the **Capability Head** predicts whether the current model should retain control or transfer the instance to a stronger fallback model. When the decision is to remain with the primary model, the **Resolution Head** determines whether the model should request additional information, call external tools, or abstain when the task cannot be answered under current setup. This introduces an effective form of self-awareness in routed agentic settings, improving the quality–cost tradeoff of inference and strengthening resolution decision-making, while remaining lightweight enough for rapid post hoc adaptation to new backbones. Our main contributions are:

- **A post hoc self-awareness layer for frozen LLMs.** We formulate deployment-time decision-making as a latent-state control problem, introducing a two-level self-awareness mechanism based

on hidden-state trajectories. This enables a unified control interface for both model selection and intervention, without modifying or fine-tuning the backbone.

- **Capability Head for model adequacy.** We introduce a lightweight Capability Head that reads hidden-state signals to predict whether the current model is adequate to solve a given instance, enabling reliable instance-level decisions on whether to retain control or escalate to a stronger model.
- **Resolution Head for intervention decisions.** conditioned on retaining control with the primary model, we introduce a Resolution Head that determines the appropriate intervention, including requesting additional information, invoking tools, abstaining under the current setup, or proceeding directly.
- **Significant quality–cost gains with minimal adaptation.** Across model families and benchmarks, our method substantially reduces reliance on large models while preserving most of their performance. In routed small-large LLM agentic execution, it achieves up to **90.7%** reduction in paid API cost on AndroidWorld and **27–53%** average cost reduction across benchmarks. The **Resolution Head** further improves WHEN2CALL by up to **11.7** F1 points and **12.4** accuracy points, and enhances tool-use decisions on TriviaQA with up to **+158.9%** relative score gain and **65.5%** fewer missed-required tool calls.

2 Related Work

Model routers and adaptive selection. A growing line of work studies how to route queries across models or inference paths under quality, latency, and cost constraints. Methods such as *FrugalGPT* [6], *RouteLLM* [29], *RouterDC* [7], and cascade-style approaches [9] learn routers or cascades from input-level signals, such as prompt characteristics, predicted task category, or other pre-generation cues. These methods improve coarse cost-quality tradeoffs, but they do not directly ask whether the current model is adequate for the specific instance at hand. Our setting is different: we study a deployment-time control signal read directly from the model's own generation process.

Multi-model collaboration and agentic systems. Another line of work improves performance by coordinating multiple models, tools, or specialized roles. Multi-agent debate [10], role-based systems such as *MetaGPT* [15], mixture-style systems [40], and trained orchestration models [39] show that structured collaboration can improve reasoning and task performance. However, these approaches typically rely on external orchestration, fixed role assignments across models or agents, or dedicated training to make models better collaborators or orchestrators. Our approach instead asks whether a frozen model can be made into a better collaborator through a lightweight post hoc control layer.

Tool use and intervention decisions. A related literature studies when systems should call tools or switch to external resources. Work such as *Toolformer* [36], *Gorilla* [30], and *ToolLLM* [32] primarily improves tool-use behavior by training or adapting the model itself to invoke tools more accurately. Later work such as *When2Call* [35], *MeCo* [21], and related methods [45, 44] studies whether tool use is necessary and how to improve tool-calling decisions. Our setting is different: rather than training the backbone to become a better tool user, we attach a lightweight post hoc control layer to a frozen model. More broadly, our method goes beyond tool use alone by exposing a shared control interface that jointly governs transfer, clarification, tool use, abstention, and direct answering.

Decoding-time efficiency. Another adjacent line of work improves efficiency during generation. Speculative decoding and related methods accelerate inference by using a smaller draft model to speed up decoding by a larger target model [19, 5, 4, 28, 38, 22]. Their goal is to speed up large-model inference, not to reduce how often a large model is invoked. Therefore, they address a different and orthogonal problem and can naturally be combined with our method.

Reliability based on latent signals. A complementary line of work asks whether reliability can be inferred directly from model internals rather than from external judges or repeated sampling. Prior work shows that hidden activations contain useful cues about correctness and hallucination [3, 1, 49], and recent methods such as *Gnosis* [12] show that lightweight predictors over internal signals can be effective for scalar correctness estimation. In contrast, we study latent signals as a deployment-time control interface. This setting is broader than scalar reliability estimation and requires decisions over transfer, clarification, tool use, abstention, and direct answering across diverse tasks and environments.

Taken together, these lines of work do not address the central setting of this paper: equipping a frozen LLM or VLM with a practical and lightweight control interface for deployment-time decisions. Our focus is not routing alone, tool use alone, decoding acceleration, or scalar self-verification, but a unified latent control layer for deciding what to do next during agentic inference.

3 The Multi-Head Latent Control Mechanism

In this section, we formalize deployment-time control for frozen foundation models as a latent decision problem over hidden-state trajectories. Building on the overall workflow illustrated in Figure 1, we introduce the problem setup, define the latent representations used for control, and describe how these signals are decoded to guide model selection and intervention decisions at inference time.

3.1 Problem Setup

Let x denote the LLM input, and let m_1 be a frozen primary model that generates an output $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_N)$. When available, we also assume access to a stronger fallback model m_2, which can take over when m_1 is unlikely to solve the instance reliably. Let d denote the hidden size of m_1, and for any layer ℓ, let $H^{(\ell)} = [h_1^{(\ell)}; \ldots; h_N^{(\ell)}] \in \mathbb{R}^{N \times d}$ denote the sequence of hidden states aligned with the generated output tokens, where $h_t^{(\ell)}$ corresponds to token $\hat{y}_t$.

The two control heads may read different hidden-state traces from the same frozen backbone, since model adequacy and resolution decisions need not be most separable at the same depth. In our default setting, the Capability Head reads the final-layer trace $H^{\text{cap}} = H^{(L)}$, while the Resolution Head reads a selected middle-layer trace $H^{\text{res}} = H^{(\ell_{\text{res}})}$. We compress these variable-length traces into fixed-budget representations $\tilde{H}^{\text{cap}} = \Pi_{\text{cap}}(H^{\text{cap}})$ and $\tilde{H}^{\text{res}} = \Pi_{\text{res}}(H^{\text{res}})$. The layer choices are selected empirically; the corresponding layer-choice ablations for the Capability and Resolution Heads are reported in Appendix C.1 & C.2, respectively.

Multi-head latent control then produces two outputs: a scalar capability score $p_{\text{cap}} \in [0, 1]$, which indicates whether control should remain with m_1 or be transferred to m_2, and a resolution score vector $\mathbf{s}_{\text{res}} = [s_{\text{info}}, s_{\text{tool}}, s_{\text{cant}}] \in [0, 1]^3$, which governs the dominant within-model intervention when the instance remains with m_1. The backbone is frozen; only the lightweight control heads are trained.

3.2 Control Prediction from Hidden States

The two heads decode different control signals from their respective projected traces:

$$z_{\text{cap}} = e_\phi^{\text{cap}}(\tilde{H}^{\text{cap}}), \qquad z_{\text{res}} = e_\phi^{\text{res}}(\tilde{H}^{\text{res}}). \tag{1}$$

They then predict

$$p_{\text{cap}} = \sigma(h_{\text{cap}}(z_{\text{cap}}))\,, \qquad \mathbf{s}_{\text{res}} = \sigma(h_{\text{res}}(z_{\text{res}}))\,. \tag{2}$$

The **Capability Head** predicts whether the current model is adequate for the instance or should transfer control to a stronger model, while the **Resolution Head** determines the appropriate resolution action when computation remains with the current model. This decomposition enables the same latent generation process to support two complementary decisions: instance-level model adequacy and action-level intervention decision. Both heads operate on hidden-state traces from the model's completion process, using the same encoder architecture as in [12]. We restrict inputs to hidden states aligned with generated tokens, excluding the prompt and other conditioning signals, ensuring a uniform interface across text-only and vision-language settings. The encoder is kept fixed to isolate the contribution of our method to the control formulation itself—namely, the capability–resolution decomposition, head-specific supervision, and the resulting inference-time control policy.

3.3 Inference-Time Policy

As shown in Figure 1, the two heads produce control scores at inference time that determine the execution path. We first consult the Capability Head. Larger values of p_{cap} indicate that the primary model is more likely to be adequate. We therefore hand off to the stornger model m_2 only when $p_{\text{cap}} < \tau_{\text{cap}}$; otherwise, the instance remains with m_1. When computation stays with m_1, the Resolution Head predicts scores over $\mathcal{A} = \{\text{info}, \text{tool}, \text{cant}\}$. If $\max_{a \in \mathcal{A}_{\text{res}}} s_{\text{res},a} > \tau_{\text{res}}$, the system executes $a^* = \arg\max_{a \in \mathcal{A}_{\text{res}}} s_{\text{res},a}$; otherwise, it proceeds directly. The system then requests more information, invokes tools, abstains under the current setup, or proceeds directly when no intervention signal is sufficiently strong. The direct-answer case is represented implicitly by the all-zero resolution state. In this way, the control heads expose an actionable inference-time interface without modifying the backbone or introducing an additional orchestration model. Unless otherwise noted, we report

results with a fixed operating point, using a Capability threshold of 0.8 and triggering a resolution intervention only when the top Resolution score exceeds 0.5. In Appendix, Figure 2 shows that higher routing thresholds hand off more aggressively to the stronger model, increasing performance at higher cost.

3.4 Control Heads Training

We freeze the backbone and train only the two lightweight control heads. Although both heads operate on the same latent generation process, they are supervised differently to address distinct deployment-time decisions: the **Capability Head** predicts instance-level adequacy for transfer, while the **Resolution Head** predicts structured within-model intervention needs. For training, we generate model outputs using the same frozen backbone as at inference time, collect the corresponding hidden-state traces, and derive supervision targets for each head from these outputs. The heads are thus trained to infer the appropriate control decisions directly from latent representations rather than from the model's surface responses. Crucially, this allows the heads to recover correct control signals even when the model's final answer or chosen action is incorrect, as such information may still be encoded in the hidden-state trajectory.

3.4.1 Capability Head

For the **Capability Head**, we construct training data by generating responses with the frozen backbone, collecting the corresponding final-layer hidden-state traces, and assigning each instance a scalar adequacy label. The head is trained to predict whether the current model is sufficient for the instance or should transfer control to a stronger model. By default, it operates on the final-layer trace; alternative layer choices are evaluated in Appendix C.1.

We supervise the head with a scalar adequacy score $y_i^{\text{cap}} \in [0, 1]$, generated by an LLM-based judge that compares the model's output to the reference answer. Because responses may be partially correct or incomplete, scalar supervision provides a more informative signal than binary correctness labels, while aligning directly with the decision of whether the current model is sufficient. Our goal is not to learn a verifier specialized to a narrow domain, but to learn a broadly transferable adequacy signal that remains informative across domains, task types, and modalities. To induce such a signal, we train this head on a deliberately heterogeneous mixture spanning multimodal QA, visual understanding, grounding, long-form reasoning, open-domain QA, API-centric tasks, and more explicitly agentic settings; full data and scoring details are deferred to Appendix A.1.

We also study prefix-trained variants of the Capability Head to investigate how early the adequacy signal becomes reliable. The main prefix-time comparison is reported in Table 5, while a broader prefix-length ablation is deferred to Appendix C.4.

Let $\mathcal{D}_{\text{cap}} = \{(\tilde{H}_i^{\text{cap}}, y_i^{\text{cap}})\}_{i=1}^{M_{\text{cap}}}$ denote the training set. Given the prediction $p_{\text{cap}}^{(i)}$, we optimize

$$\mathcal{L}_{\text{cap}} = \frac{1}{M_{\text{cap}}} \sum_{i=1}^{M_{\text{cap}}} w_i \, \ell_{\text{reg}} \left(p_{\text{cap}}^{(i)}, y_i^{\text{cap}} \right), \tag{3}$$

where ℓ_{reg} is a regression loss and w_i reweights examples to compensate for imbalance in the training data, which contains unequal numbers of more-adequate and less-adequate responses. In our default setting, ℓ_{reg} is weighted mean squared error.

3.4.2 Resolution Head

The **Resolution Head** predicts the appropriate intervention when computation remains with the current model. We train it on WHEN2CALL, where each instance requires one of four behaviors: answer directly, invoke a tool, request additional information, or abstain under the current setup. We define the explicit action space as $\mathcal{A} = \{\text{info}, \text{tool}, \text{cant}\}$, with direct answer represented implicitly by the all-zero target vector. This design reflects the role of the Resolution Head: it activates only when intervention is required, while direct answering corresponds to the absence of such need.

For each training example, we generate the backbone's output, collect the corresponding hidden-state trace, and pair it with the correct resolution label. The model's explicit action is not always correct—for example, it may invoke a tool when clarification is required. The Resolution Head is therefore trained to recover the appropriate decision from the latent trajectory rather than from the model's surface behavior. Details of label construction are provided in Appendix A.2.

System	Score	m_1 Stats			m_2 Stats			API Cost
		In Tok.	Out Tok.	#Calls	In Tok.	Out Tok.	#Calls	
m_1: Qwen3-VL-4B	0.47	14.15M	0.47M	3250	0	0	0	0
m_2: Qwen3-VL-32B	0.58	0	0	0	14.74M	0.53M	3595	$2.70
Routed: Qwen3-VL-4B $\rightarrow$ 32B	0.60	15.14M	0.56M	3667	1.39M	0.05M	300	$0.25 ($\downarrow$ 90.7%)
m_1: Qwen3.5-9B	0.51	9.86M	0.36M	2282	0	0	0	0
m_2: Qwen3.5-27B	0.59	0	0	0	12.75M	0.57M	2854	$1.49
Routed: Qwen3.5-9B $\rightarrow$ 27B	0.56	10.2M	0.37M	2170	1.9M	0.07M	470	$0.21 ($\downarrow$ 85.8%)

Table 1: AndroidWorld success rate and backend usage for single-model and routed runs. The smaller model is assumed to run locally for free, so reported API cost reflects only large-model usage. Dynamic Routed execution improves success while sharply reducing paid large-model calls.

Let $\mathcal{D}_{\text{res}} = \{(\tilde{H}_i^{\text{res}}, \mathbf{y}_i^{\text{res}})\}_{i=1}^{M_{\text{res}}}$, where $\mathbf{y}_i^{\text{res}} \in \{0,1\}^3$. Given the prediction $\mathbf{s}_{\text{res}}^{(i)}$, we optimize

$$\mathcal{L}_{\text{res}} = \frac{1}{M_{\text{res}}} \sum_{i=1}^{M_{\text{res}}} \sum_{a \in \mathcal{A}} \ell_{\text{BCE}}\left(s_a^{(i)}, y_{i,a}^{\text{res}}\right). \tag{4}$$

4 Experiments and Results

We evaluate Multi-Head Latent Control across two complementary dimensions: *capability prediction* and *resolution decision-making*, each designed to test a distinct aspect of deployment-time control. For **capability-guided routing** (§4.1), we study routed collaboration between a smaller primary model and a stronger fallback model across multiple backbone families and benchmark suites, focusing on score–cost tradeoffs. We further include AndroidWorld as a long-horizon agentic case study to assess real-world cost and usage behavior under sequential decision-making. For **resolution decision-making** (§4.2), we evaluate the Resolution Head on WHEN2CALL, where each instance requires selecting among clarification, tool use, abstention, or direct answering. This setting isolates the quality of intervention decisions by comparing the backbone's native behavior against the same model augmented with our control layer. We also study two deployment-oriented variants: (i) a web-augmented TriviaQA setting (§4.3) to evaluate tool-escalation decisions under incomplete knowledge, and (ii) a prefix-time setting (§4.4) to test whether adequacy signals can be recovered from partial generations. Additional ablations on layer choice, prefix length, and training mixture are provided in §4.5. Across all settings, we assess whether lightweight heads attached to frozen backbones produce control signals with direct system value, including improved score–cost tradeoffs, more accurate intervention decisions, better tool-escalation behavior, and reliable early adequacy prediction.

Backbones. We evaluate three backbone families: **Qwen3-VL** [2], **Qwen3.5** [33], and **Gemma** [13]. Across these families, our routed systems span model sizes from 2B to 32B and include both thinking and non-thinking variants, allowing us to test whether the proposed control layer transfers across different model lines, scales, and inference modes rather than a single narrow setup.

Training/eval pipeline and infrastructure. We train lightweight control heads on top of Qwen3-VL-2B/4B/32B, Qwen3-VL-2B-Thk/4B-Thk, Qwen3.5-4B/9B, and Gemma-4B/4B-Thk backbones, *Thk* denotes thinking mode. Training labels are constructed by comparing each model generation with the ground-truth answer using Qwen3vl 30B-A3B as judge model. We optimize the heads with Adam at a learning rate of 1×10^{-4}. As backbone remains frozen, the full pipeline is lightweight: in a representative 9B-scale Qwen setting, data generation and training for both heads completes in under one day on a single high-end GPU, and head training itself fits within 16 GB of GPU memory.

Benchmarks. We evaluate across a deliberately diverse set of tasks and deployment settings rather than a single benchmark type. For capability collaboration, we report results on **SimpleVQA** [8], **ScreenSpot-Pro** [20], **CharXiv-Reasoning** [42], **MathVerse** [50], **MathVista** [24], and **MMLU-Pro** [41], covering multimodal perception, GUI grounding, chart and document reasoning, mathematical reasoning, and broad knowledge tasks. We additionally include **AndroidWorld** [34] as a long-horizon, multi-step agentic case study with backend usage statistics, implemented using the Mobile-Agent-v3.5 framework [46]. For resolution control, we evaluate structured intervention prediction on WHEN2CALL [35]. Finally, we study **TriviaQA** [17] as a concrete external-action setting for web-search decision quality. Taken together, these benchmarks test generalization across backbone families, task formats, modalities, and deployment behaviors.

Family	Setting	CharXiv *score / cost*	MathVerse *score / cost*	MathVista *score / cost*	ScreenSpot-Pro *score / cost*	SimpleVQA *score / cost*	MMLU-Pro *score / cost*	Overall *score / cost*
m_1: Qwen3.5-4B m_2: Qwen3.5-27B-Thk	m_1:	0.63 /	0.86 /	0.84 /	0.36 /	0.37 /	0.68 /	0.63 /
	m_2:	0.73 / $3.50	0.89 / $4.66	0.87 / $3.06	0.65 / $7.59	0.42 / $2.68	0.78 / $5.88	0.72 / $27.37
	Routed	0.70 / $2.09	0.90 / $1.31	0.86 / $1.30	0.64 / $5.79	0.41 / $1.48	0.78 / $3.21	0.72 / $15.17
		↓40.3%	↓71.9%	↓57.5%	↓23.7%	↓44.8%	↓45.4%	↓44.6%
m_1: Qwen3.5-9B m_2: Qwen3.5-27B-Thk	m_1:	0.68 /	0.88 /	0.85 /	0.47 /	0.41 /	0.71 /	0.67 /
	m_2:	0.73 / $3.50	0.89 / $4.66	0.87 / $3.06	0.65 / $7.59	0.42 / $2.68	0.80 / $5.85	0.73 / $27.35
	Routed	0.73 / $2.48	0.91 / $1.24	0.87 / $1.00	0.64 / $4.57	0.42 / $1.25	0.78 / $2.30	0.72 / $12.85
		↓29.1%	↓73.4%	↓67.3%	↓39.8%	↓53.4%	↓60.7%	↓53.0%
m_1: Qwen3-VL-2B-Thk m_2: Qwen3-VL-32B-Thk	m_1:	0.38 /	0.74 /	0.72 /	0.27 /	0.33 /	0.54 /	0.50 /
	m_2:	0.64 / $1.33	0.86 / $2.75	0.84 / $1.42	0.52 / $6.48	0.41 / $0.68	0.72 / $2.53	0.67 / $15.19
	Routed	0.62 / $1.19	0.85 / $1.45	0.84 / $0.96	0.49 / $4.84	0.39 / $0.40	0.72 / $2.22	0.65 / $11.06
		↓10.5%	↓47.3%	↓32.4%	↓25.3%	↓41.2%	↓12.3%	↓27.2%
m_1: Qwen3-VL-4B m_2: Qwen3-VL-32B-Thk	m_1:	0.41 /	0.79 /	0.78 /	0.54 /	0.35 /	0.63 /	0.58 /
	m_2:	0.64 / $1.33	0.86 / $2.75	0.84 / $1.42	0.52 / $6.48	0.41 / $0.68	0.73 / $2.52	0.67 / $15.17
	Routed	0.63 / $1.16	0.86 / $1.49	0.83 / $0.70	0.57 / $2.62	0.39 / $0.43	0.71 / $1.58	0.67 / $7.98
		↓12.8%	↓45.8%	↓50.7%	↓59.6%	↓36.8%	↓37.3%	↓47.4%
m_1: Qwen3-VL-4B-Thk m_2: Qwen3-VL-32B-Thk	m_1:	0.53 /	0.82 /	0.78 /	0.37 /	0.37 /	0.65 /	0.59 /
	m_2:	0.64 / $1.33	0.86 / $2.75	0.84 / $1.42	0.52 / $6.48	0.41 / $0.68	0.73 / $2.47	0.67 / $15.13
	Routed	0.63 / $0.88	0.86 / $0.98	0.84 / $0.92	0.49 / $3.88	0.41 / $0.31	0.72 / $1.43	0.66 / $8.40
		↓33.8%	↓64.4%	↓35.2%	↓40.1%	↓54.4%	↓42.1%	↓44.5%
m_1: Gemma-4B m_2: Gemma-31B-Thk	m_1:	0.41 /	0.65 /	0.69 /	–	0.30 /	0.61 /	0.53 /
	m_2:	0.63 / $0.62	0.87 / $1.34	0.80 / $0.81	–	0.41 / $0.39	0.78 / $1.05	0.70 / $4.20
	Routed	0.58 / $0.49	0.87 / $1.02	0.79 / $0.62	–	0.41 / $0.24	0.75 / $0.68	0.68 / $3.05
		↓21.0%	↓23.9%	↓23.5%	–	↓38.5%	↓35.2%	↓27.4%
m_1: Gemma-4B-Thk m_2: Gemma-31B-Thk	m_1:	0.42 /	0.65 /	0.67 /	–	0.31 /	0.62 /	0.53 /
	m_2:	0.63 / $0.62	0.87 / $1.34	0.80 / $0.81	–	0.41 / $0.39	0.78 / $1.05	0.70 / $4.21
	Routed	0.59 / $0.43	0.84 / $1.02	0.76 / $0.44	–	0.41 / $0.29	0.76 / $0.71	0.67 / $2.89
		↓30.6%	↓23.9%	↓45.7%	–	↓25.6%	↓32.4%	↓31.4%

Table 2: Score–cost tradeoff across model families and benchmarks. m_1 is the local model, m_2 the always-large baseline, and Routed switches between them using the Capability Head. Each cell reports score and estimated paid API cost. Green indicates reduction in large-model cost relative to the m_2 baseline; m_1 is assumed to run locally at no cost.

Metrics. For multi-agent experiments, we report each benchmark's native task score alongside estimated API cost. In Table 2, each entry shows *score / estimated paid cost*, with the Overall column reporting average score and total cost across benchmarks. In Table 1, we further report backend usage statistics—including input tokens, output tokens, and number of calls for both primary and fallback models—to characterize changes in large-model usage under routed execution.

In settings where we want to compare the quality of the Capability signal directly, we report ROC-AUC, AUPR-C, AUPR-I, and ECE. ROC-AUC measures how well the adequacy signal ranks more-adequate above less-adequate cases, AUPR-C and AUPR-I measure precision–recall quality under the two complementary positive classes, and ECE measures calibration. For the Resolution Head on WHEN2CALL, Table 3 reports F1 score and accuracy. For the TriviaQA web-search analysis, Table 4 reports model's score together with the number of web calls, tool-call precision, and missed-needed web calls. Estimated API cost is reported as a deployment-oriented proxy based on token usage; full details are deferred to Appendix D.

4.1 Efficient Multi-Model Collaboration

We evaluate the **Capability Head** in the primary deployment setting: collaboration between a weaker local model and a stronger fallback model. Table 2 reports score–cost tradeoffs across benchmarks and model families, while Table 1 presents AndroidWorld as a long-horizon agentic case study. The key question is whether a lightweight adequacy signal can recover the benefits of a stronger model without incurring its cost on every instance. Across settings, routed systems consistently outperform the smaller-model baselines while remaining substantially cheaper than always using the stronger model. For example, routed Qwen3.5 systems reduce paid cost by approximately 45–53% while retaining most of the large-model performance, and routed Qwen3-VL systems achieve similar savings with comparable performance preservation.

AndroidWorld highlights this effect clearly. It is a long-horizon benchmark where the model is invoked repeatedly over multi-step trajectories involving planning, acting, tool use, and reflection. In our routed setup, the Capability Head decides at each step whether to remain local or escalate control, rather than relying on a large model throughout or fixed role assignment. This yields substantial

Method	Qwen-VL-2B		Qwen-VL-4B		Qwen-VL-2B-Thk		Qwen3.5-4B		Gemma-2B		Gemma-2B-Thk	
	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc
MNM 4B When2Call-SFT [35]						F1 = 48.1		Acc = 67.8				
MNM 8B When2Call-SFT [35]						F1 = 49.4		Acc = 68.2				
Backbone Choice	37.3	52.7	48.7	63.9	55.1	73.6	43.5	57.9	40.1	56.5	54.2	70.1
Backbone + Resolution Head	49.0	65.1	52.7	70.1	54.5	69.5	50.0	63.6	55.2	71.2		

Table 3: Resolution decision prediction on WHEN2CALL. We compare each frozen backbone's native action choice against the same backbone augmented with the Resolution Head; First two rows show full-LLM WHEN2CALL baselines fine-tuned directly on the benchmark.

savings while preserving or improving performance: *Qwen3-VL-4B* $\rightarrow$ *32B* improves score from 0.47 to 0.60 with a 90.7% reduction in paid API cost, and *Qwen3.5-9B* $\rightarrow$ *27B* improves score from 0.51 to 0.56 with an 85.8% cost reduction.

Appendix B.2 and Appendix B.3 evaluate two natural alternatives to latent-control routing. The first considers a self-abstention strategy, where the smaller model is prompted to defer when uncertain; however, this leads to severe under-escalation and fails to recover the performance gains of latent-control routing on ScreenSpot-Pro and MMLU-Pro. The second examines token-level confidence as a proxy for adequacy: on Qwen3.5-9B over ScreenSpot-Pro, confidence scores poorly distinguish correct from incorrect predictions, whereas the Capability Head provides a significantly more discriminative signal.

4.2 Resolution Control on WHEN2CALL

We next isolate the Resolution Head to evaluate whether hidden-state representations can reliably support within-model intervention decisions.

Table 3 evaluates the Resolution Head on WHEN2CALL across several frozen backbones. For each model, we compare the backbone's native action choice, obtained from its surface response, against the same backbone augmented with the Resolution Head. We also include the task-specialized WHEN2CALL baselines reported by the benchmark as reference points. Unlike our lightweight head-based adaptation, these baselines are full LLMs fine-tuned directly on WHEN2CALL.

The main result is that latent control consistently improves resolution decision quality over the backbone's explicit behavior. The gains are often substantial: for example, on Qwen-VL-2B, F1 improves from 37.3 to 49.0 and accuracy from 52.7 to 65.1, while on Qwen3.5-4B, F1 improves from 43.5 to 54.5 and accuracy from 57.9 to 69.5. Even when the model's surface action is wrong, its hidden-state contains information for the head to recover the correct resolution decision. These results show that lightweight control heads can improve structured intervention decisions without modifying the backbone.

4.3 Web-Tool Decision Quality on TriviaQA

We next study a concrete deployment variant of Capability Head: using the Capability Head to decide whether the current model should answer locally or escalate to web search. TriviaQA is a useful testbed for this setting because retrieval is often helpful, but not every instance requires an external search. Table 4 compares the backbone's native web-search behavior against the same backbone augmented with the Capability Head. Here, the control signal is used in the same spirit as routed collaboration, except that inadequate cases are escalated to a tool call rather than to a stronger fallback model. The question is therefore not whether the head simply suppresses tool use, but whether it improves the quality of the decision to use external retrieval.

Across backbones, the Capability Head consistently improves task score and reduces missed-needed web calls, indicating better recovery of when web search is actually required. In several cases, this comes with more web calls rather than fewer, so the main effect is not blanket suppression of tool use. Instead, the adequacy signal improves the quality of the escalation decision. This effect is visible even for very strong large models: on *Qwen3-VL-32B*, adding the control head improves tool-call precision from 72.7% to 75.5%, raises task score from 0.672 to 0.778, and reduces missed-needed web calls from 328 to 222.

Backbone	No-Tool ↑	Backbone Choice				Backbone + Capability Head			
		Score ↑	Calls	Precision (%) ↑	Missed ↓	Score ↑	Calls	Precision (%) ↑	Missed ↓
Qwen3-VL-4B	0.231	0.292	67	91.0	708	0.756 (+158.9%)	518 (+451)	87.1 (-3.9)	244 (-464)
Qwen3-VL-32B	0.656	0.672	22	72.7	328	0.778 (+15.8%)	163 (+141)	75.5 (+2.8)	222 (-106)
Qwen3.5-9B	0.593	0.624	39	79.5	376	0.858 (+37.5%)	357 (+318)	75.4 (-4.1)	142 (-234)
Gemma-4B	0.427	0.862	608	71.5	138	0.921 (+6.8%)	620 (+12)	77.1 (+5.6)	79 (-59)
Gemma-4B-Thk	0.447	0.811	457	79.6	189	0.902 (+11.2%)	570 (+113)	80.7 (+1.1)	98 (-91)

Table 4: TriviaQA web-search decision quality. We compare each backbone's native web-search behavior against a Capability-Head variant that triggers web search only when the model is unlikely to answer correctly from parametric knowledge alone.

Train	Inference	Qwen3-VL-2B-Thk				Qwen3-VL-4B-Thk				Gemma-4B-Thk			
		ROC-AUC ↑				AUPR-C ↑		AUPR-I ↑		ECE ↓			
Full	Full	**0.85**	**0.73**	**0.87**	**0.20**	**0.86**	**0.91**	**0.75**	**0.14**	**0.84**	**0.74**	**0.88**	**0.16**
Full	Prefix-200	0.77	0.65	0.81	0.33	0.78	0.80	0.70	0.20	0.78	0.69	0.81	0.24
Prefix-200	Prefix-200	0.80	0.69	0.82	**0.20**	0.80	0.82	0.73	0.18	0.82	0.71	0.84	**0.16**

Table 5: Prefix-time evaluation of the Capability Head on top of each backbone. Metrics are averaged over the benchmarks in Table 2. We measure the quality of the adequacy signal itself rather than end-to-end routed performance. Applying a head trained on full completions to prefixes degrades performance, while training directly on prefixes recovers stronger prefix-time quality.

4.4 Prefix-Time Capability Prediction

The default collaboration setup waits for the weaker model to finish its response before applying the Capability Head. While effective, this can still waste compute when the local trajectory is already heading toward failure. We therefore ask whether the adequacy signal can be recovered earlier from only a short prefix of the generated answer. Table 5 evaluates the quality of the Capability signal itself under three regimes: *Full / Full*, *Full / Prefix-200*, and *Prefix-200 / Prefix-200*. As expected, prefix-time prediction is weaker than full-trajectory prediction, since the head observes only a partial hidden-state trace. However, training directly in the prefix regime recovers substantially stronger prefix-time signal quality than naively applying a full-trajectory-trained head to partial generations.

This signal-level difference also translates to end-to-end routed behavior. Appendix Table 6 shows that training directly on prefixes better recovers the routed performance of full-trajectory control than simply applying a full-trained head at 200 tokens, while still preserving substantial reductions in paid API cost relative to always using the large model. These results show that model adequacy can often be detected early enough to support useful routed inference, and that prefix-time control is best learned in the same regime in which it is applied.

4.5 Ablations

Additional analyses in the appendix support the main design choices of the method. Appendix C.1 ablates the **hidden-state layer used by the Capability Head** and shows that the **final layer** provides the **strongest overall adequacy signal**. Appendix C.2 performs the corresponding study for the **Resolution Head** and shows that a **middle-layer trace** is most effective for **resolution decision prediction**. Appendix C.3 tests the **breadth of the capability-head training mixture** and shows that **narrow visual-math-only training transfers substantially worse** than the **full mixed-data setting** on **ScreenSpot Pro**, supporting our goal of learning a **broadly transferable adequacy signal**. Appendix C.4 studies how the **Capability signal changes with available prefix length**, while Table 6 reports the corresponding **end-to-end routed score–cost tradeoff** under the same **prefix-time regimes**. Appendix B.2 includes a **prompt-level self-switching baseline**, showing that the model **rarely escalates to the stronger model when needed** compared with the **Capability Head**.

5 Conclusion

We introduced multi-head latent control, a lightweight deployment-time control layer that equips frozen foundation models with a practical control interface for deciding what to do next during

inference. Rather than modifying the backbone, the method trains small control heads that read hidden-state traces to predict model adequacy and resolution decisions. Across multi-model collaboration, long-horizon agentic execution, structured resolution decision-making, tool-use decisions, and prefix-time prediction, the results show that these signals can improve deployment behavior while preserving the efficiency and reusability of frozen models. More broadly, this suggests that hidden-state traces provide a scalable substrate for lightweight control interfaces that can be rapidly attached to new foundation models as they are deployed.

References

[1] A. Azaria and T. Mitchell. The internal state of an llm knows when it's lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 967–976, 2023.

[2] S. Bai, Y. Cai, R. Chen, K. Chen, X. Chen, Z. Cheng, L. Deng, W. Ding, C. Gao, C. Ge, W. Ge, Z. Guo, Q. Huang, J. Huang, F. Huang, B. Hui, S. Jiang, Z. Li, M. Li, M. Li, K. Li, Z. Lin, J. Lin, X. Liu, J. Liu, C. Liu, Y. Liu, D. Liu, S. Liu, D. Lu, R. Luo, C. Lv, R. Men, L. Meng, X. Ren, X. Ren, S. Song, Y. Sun, J. Tang, J. Tu, J. Wan, P. Wang, P. Wang, Q. Wang, Y. Wang, T. Xie, Y. Xu, H. Xu, J. Xu, Z. Yang, M. Yang, J. Yang, A. Yang, B. Yu, F. Zhang, H. Zhang, X. Zhang, B. Zheng, H. Zhong, J. Zhou, F. Zhou, J. Zhou, Y. Zhu, and K. Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[3] C. Burns, H. Ye, D. Klein, and J. Steinhardt. Discovering latent knowledge in language models without supervision. *arXiv preprint arXiv:2212.03827*, 2022.

[4] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

[5] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[6] L. Chen, M. Zaharia, and J. Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[7] S. Chen, W. Jiang, B. Lin, J. Kwok, and Y. Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024.

[8] X. Cheng, W. Zhang, S. Zhang, J. Yang, X. Guan, X. Wu, X. Li, G. Zhang, J. Liu, Y. Mai, et al. Simplevqa: Multimodal factuality evaluation for multimodal large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4637–4646, 2025.

[9] J. Dekoninck, M. Baader, and M. Vechev. A unified approach to routing and cascading for llms. *arXiv preprint arXiv:2410.10347*, 2024.

[10] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.

[11] EvolvingLMMs-Lab. open-r1-multimodal. `https://github.com/EvolvingLMMs-Lab/open-r1-multimodal`, 2025. GitHub repository; includes the multimodal-open-r1-8k-verified dataset.

[12] A. Ghasemabadi and D. Niu. Can llms predict their own failures? self-awareness via internal circuits. *arXiv preprint arXiv:2512.20578*, 2025.

[13] Google. Gemma 4 model overview. `https://ai.google.dev/gemma/docs/core`, 2026. Official model documentation.

[14] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913, 2017.

[15] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

[16] Y.-C. Hsiao, F. Zubach, G. Baechler, S. Sunkara, V. Cărbune, J. Lin, M. Wang, Y. Zhu, and J. Chen. Screenqa: Large-scale question-answer pairs over mobile app screenshots. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 9427–9452, 2025.

[17] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, 2017.

[18] A. Kembhavi, M. Salvato, E. Kolve, M. Seo, H. Hajishirzi, and A. Farhadi. A diagram is worth a dozen images. In *European conference on computer vision*, pages 235–251. Springer, 2016.

[19] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[20] K. Li, Z. Meng, H. Lin, Z. Luo, Y. Tian, J. Ma, Z. Huang, and T.-S. Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786, 2025.

[21] W. Li, D. Li, K. Dong, C. Zhang, H. Zhang, W. Liu, Y. Wang, R. Tang, and Y. Liu. Adaptive tool use in large language models with meta-cognition trigger. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13346–13370, 2025.

[22] B. Liao, Y. Xu, H. Dong, J. Li, C. Monz, S. Savarese, D. Sahoo, and C. Xiong. Reward-guided speculative decoding for efficient llm reasoning. *arXiv preprint arXiv:2501.19324*, 2025.

[23] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in neural information processing systems*, 35:2507–2521, 2022.

[24] P. Lu, H. Bansal, T. Xia, J. Liu, C. Li, H. Hajishirzi, H. Cheng, K.-W. Chang, M. Galley, and J. Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

[25] A. Masry, X. L. Do, J. Q. Tan, S. Joty, and E. Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pages 2263–2279, 2022.

[26] M. Mathew, D. Karatzas, and C. Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

[27] M. Mathew, V. Bagal, R. Tito, D. Karatzas, E. Valveny, and C. Jawahar. Infographicvqa. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1697–1706, 2022.

[28] H. Narasimhan, W. Jitkrittum, A. S. Rawat, S. Kim, N. Gupta, A. K. Menon, and S. Kumar. Faster cascades via speculative decoding. *arXiv preprint arXiv:2405.19261*, 2024.

[29] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

[30] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

[31] A. Prabhakar, Z. Liu, M. Zhu, J. Zhang, T. Awalgaonkar, S. Wang, Z. Liu, H. Chen, T. Hoang, J. C. Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025.

[32] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[33] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

[34] C. Rawles, S. Clinckemaillie, Y. Chang, J. Waltz, G. Lau, M. Fair, A. Li, W. Bishop, W. Li, F. Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

[35] H. Ross, A. S. Mahabaleshwarkar, and Y. Suhara. When2call: When (not) to call tools. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3391–3409, 2025.

[36] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

[37] D. Schwenk, A. Khandelwal, C. Clark, K. Marino, and R. Mottaghi. A-okvqa: A benchmark for visual question answering using world knowledge. In *European conference on computer vision*, pages 146–162. Springer, 2022.

[38] Z. Shen, H. Lang, B. Wang, Y. Kim, and D. Sontag. Learning to decode collaboratively with multiple language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12974–12990, 2024.

[39] H. Su, S. Diao, X. Lu, M. Liu, J. Xu, X. Dong, Y. Fu, P. Belcak, H. Ye, H. Yin, et al. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

[40] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

[41] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[42] Z. Wang, M. Xia, L. He, H. Chen, Y. Liu, R. Zhu, K. Liang, X. Wu, H. Liu, S. Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *Advances in Neural Information Processing Systems*, 37:113569–113697, 2024.

[43] L. Wiedmann, O. Zohar, A. Mahla, X. Wang, R. Li, T. Frere, L. von Werra, A. R. Gosthipaty, and A. Marafioti. Finevision: Open data is all you need. *arXiv preprint arXiv:2510.17269*, 2025.

[44] H. Xu, Z. Zhu, L. Pan, Z. Wang, S. Zhu, D. Ma, R. Cao, L. Chen, and K. Yu. Reducing tool hallucination via reliability alignment. *arXiv preprint arXiv:2412.04141*, 2024.

[45] H. Xu, Z. Wang, Z. Zhu, L. Pan, X. Chen, S. Fan, L. Chen, and K. Yu. Alignment for efficient tool calling of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17787–17803, 2025.

[46] H. Xu, X. Zhang, H. Liu, J. Wang, Z. Zhu, S. Zhou, X. Hu, F. Gao, J. Cao, Z. Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

[47] Y. Xu, Z. Wang, J. Wang, D. Lu, T. Xie, A. Saha, D. Sahoo, T. Yu, and C. Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

[48] Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[49] A. Zhang, Y. Chen, J. Pan, C. Zhao, A. Panda, J. Li, and H. He. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*, 2025.

[50] R. Zhang, D. Jiang, Y. Zhang, H. Lin, Z. Guo, P. Qiu, A. Zhou, P. Lu, K.-W. Chang, Y. Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pages 169–186. Springer, 2024.

[51] L. Zheng, Z. Huang, Z. Xue, X. Wang, B. An, and S. Yan. Agentstudio: A toolkit for building general virtual agents. *arXiv preprint arXiv:2403.17918*, 2024.

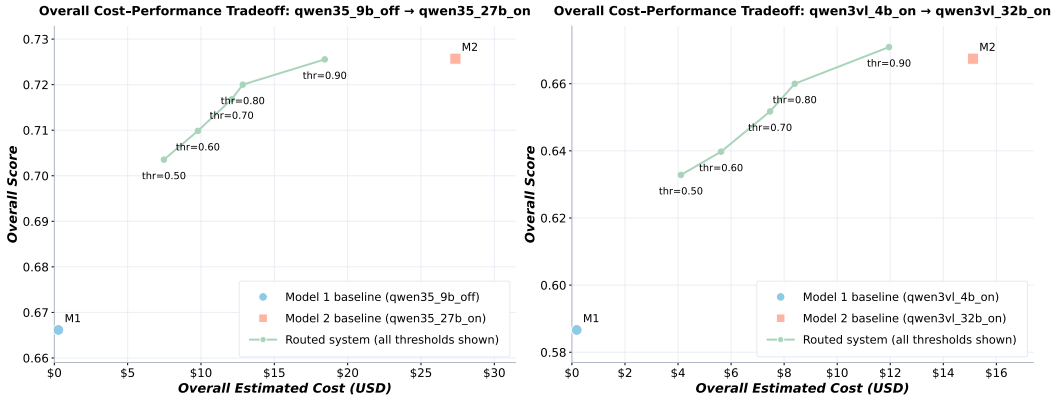

Figure 2: Overall cost–performance tradeoff for two routed systems: **Qwen3.5-9B** $\rightarrow$ **Qwen3.5-27B-Thk** (left) and **Qwen3-VL-4B-Thk** $\rightarrow$ **Qwen3-VL-32B-Thk** (right). Each panel compares the local small model (m_1:), the always-large baseline (m_2:), and the routed system across routing thresholds. Each routed point corresponds to a threshold value, shown directly below the marker. The routed point at $\mathrm{thr} = 0.80$ is aligned with the values reported in the table 2. Lower cost and higher score are preferred.

A Training Data and Label Construction

A.1 Capability Head Data

The **Capability Head** is trained on a 120K-example mixture designed to induce a control signal that generalizes across modalities, topics, and task styles. Our goal is not to learn a verifier specialized to a narrow regime such as math reasoning or hallucination detection. Rather, we want a lightweight head that reads the hidden-state trajectory of a frozen model and estimates whether that model is adequate for the current instance under the current setup. For this signal to be useful in practice, it must remain informative across both vision and text, across reasoning and parametric knowledge, and across question answering, grounding, tool use, and more agentic interaction patterns.

To this end, we train on a deliberately mixed dataset spanning visual question answering, science and diagram reasoning, chart and document understanding, screen understanding, UI grounding, multimodal and text-only reasoning, open-domain factual QA, and multi-turn tool-use data. Much of the visual portion is accessed through *FineVision* [43], which provides a unified interface over heterogeneous multimodal sources. This breadth is intentional. By training on a mixed distribution rather than a single task family, we encourage the head to learn a broader adequacy signal from hidden-state traces rather than a domain-specific notion of confidence. As we show in Appendix C.3, this breadth is important for transfer: narrowing the training distribution yields a substantially less reliable control signal on a deployment-oriented benchmark.

For each prompt, we query the frozen backbone model, record its generated output and aligned hidden-state trajectory, and score the generation against the task reference using an external LLM-based evaluator. The resulting scalar score in $[0, 1]$ is used as the supervision target for the Capability Head.

Training sources. The mixture includes the following datasets.

- **VQAv2** [14]: broad visual question answering over natural images, providing general grounded QA examples.
- **ScienceQA** [23]: science-oriented multimodal question answering, adding visually grounded scientific and educational reasoning.
- **ChartQA** [25]: question answering over charts and plots, contributing structured visual reading and lightweight quantitative reasoning.
- **DocVQA** [26]: document question answering, contributing OCR-heavy and layout-sensitive visual understanding.

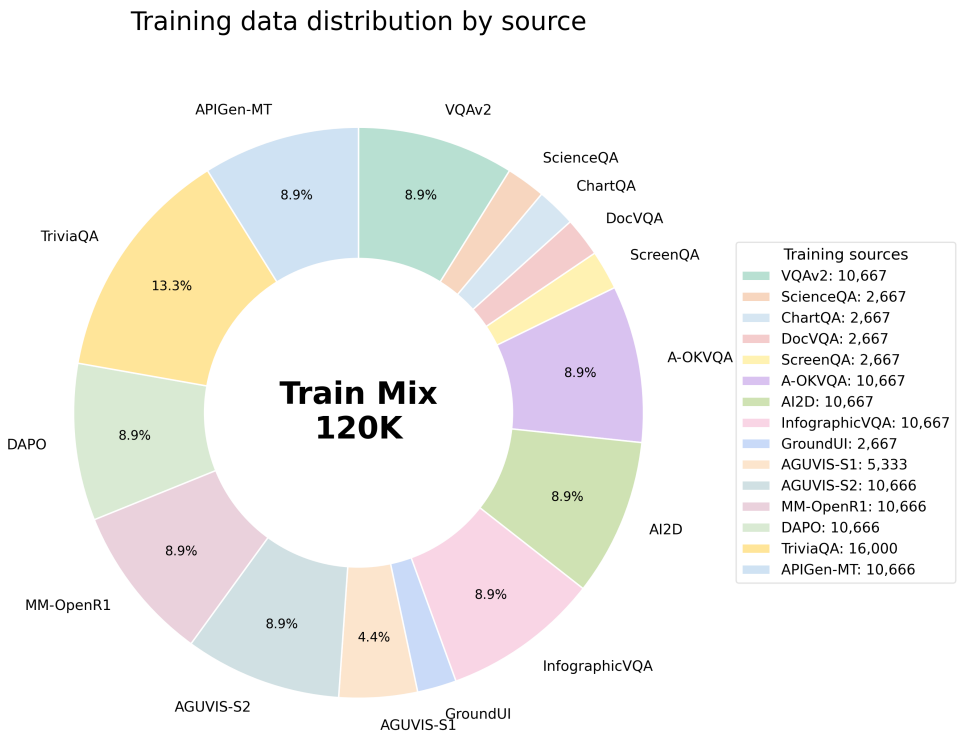

Figure 3: Composition of the 120K training mixture for the Capability Head. The mixture intentionally spans visual QA, reasoning, parametric knowledge, grounding, tool use, and agentic interaction across both vision and text.

- **ScreenQA** [16]: question answering over screen content, bringing screen-centered visual QA closer to deployment-oriented settings.
- **A-OKVQA** [37]: knowledge-intensive visual question answering, requiring both image understanding and parametric or external world knowledge.
- **AI2D** [18]: diagram understanding and reasoning, adding a distinct form of structured visual interpretation beyond natural images.
- **InfographicVQA** [27]: question answering over infographics, combining OCR, layout understanding, and visually embedded factual reasoning.
- **GroundUI** [51]: UI element grounding, contributing fine-grained spatial grounding over interface elements.
- **AGUVIS Stage 1** [47]: grounded GUI action prediction, introducing action-oriented visual control in interface environments.
- **AGUVIS Stage 2** [47]: richer GUI interaction data with more agentic structure, extending grounding toward multi-step interface behavior.
- **MM-OpenR1** [11]: verified multimodal reasoning data, contributing harder visual reasoning traces.
- **DAPO-Math** [48]: text-only mathematical reasoning, adding long-form reasoning trajectories beyond the visual domain.
- **TriviaQA** [17]: open-domain factual question answering, contributing text-only parametric knowledge and broad topic coverage.
- **APIGen-MT-5k** [31]: multi-turn tool-use data, adding tool-calling and agentic task structure beyond standard QA.

Figure 3 summarizes the composition of the training mixture by source.

Family	System	Train	Infer	CharXiv	MathVerse	MathVista	ScreenSpot-Pro	SimpleVQA	MMLU-Pro	Overall
m_1: Qwen3-VL-2B-Thk m_2: Qwen3-VL-32B-Thk	m_1:	–	–	0.38 / $0.00	0.74 / $0.00	0.72 / $0.00	0.27 / $0.00	0.33 / $0.00	0.54 / $0.00	0.50 / $0.00
	m_2:	–	–	0.64 / $1.33	0.86 / $2.75	0.84 / $1.42	0.52 / $6.48	0.41 / $0.68	0.72 / $2.53	0.67 / $15.19
	Routed	Full	Full	0.62 / $1.19 (↓3%, ↓11%)	0.85 / $1.45 (↓1%, ↓47%)	0.84 / $0.96 (→0%, ↓32%)	0.49 / $4.84 (↓6%, ↓25%)	0.39 / $0.40 (↓5%, ↓41%)	0.72 / $2.22 (→0%, ↓12%)	0.65 / $11.06 (↓3%, ↓27%)
	Routed	Full	Prefix-200	0.45 / $0.30 (↓30%, ↓77%)	0.82 / $0.80 (↓5%, ↓71%)	0.79 / $0.57 (↓6%, ↓60%)	0.45 / $4.31 (↓13%, ↓34%)	0.38 / $0.37 (↓7%, ↓46%)	0.68 / $0.96 (↓6%, ↓62%)	0.59 / $7.31 (↓12%, ↓52%)
	Routed	Prefix-200	Prefix-200	0.59 / $1.09 (↓8%, ↓18%)	0.85 / $1.75 (↓1%, ↓36%)	0.83 / $1.10 (↓1%, ↓23%)	0.48 / $5.26 (↓8%, ↓19%)	0.38 / $0.40 (↓7%, ↓41%)	0.77 / $2.03 (↑7%, ↓20%)	0.65 / $11.63 (↓3%, ↓23%)
m_1: Qwen3-VL-4B-Thk m_2: Qwen3-VL-32B-Thk	m_1:	–	–	0.53 / $0.00	0.82 / $0.00	0.78 / $0.00	0.37 / $0.00	0.37 / $0.00	0.65 / $0.00	0.59 / $0.00
	m_2:	–	–	0.64 / $1.33	0.86 / $2.75	0.84 / $1.42	0.52 / $6.48	0.41 / $0.68	0.73 / $2.47	0.67 / $15.13
	Routed	Full	Full	0.63 / $0.88 (↓2%, ↓34%)	0.86 / $0.98 (→0%, ↓64%)	0.84 / $0.92 (→0%, ↓35%)	0.49 / $3.88 (↓6%, ↓40%)	0.41 / $0.31 (→0%, ↓54%)	0.72 / $1.43 (↓1%, ↓42%)	0.66 / $8.40 (↓1%, ↓45%)
	Routed	Full	Prefix-200	0.55 / $0.12 (↓14%, ↓91%)	0.85 / $0.58 (↓1%, ↓79%)	0.82 / $0.84 (↓2%, ↓41%)	0.49 / $4.68 (↓6%, ↓28%)	0.41 / $0.34 (→0%, ↓50%)	0.77 / $1.31 (↑5%, ↓47%)	0.65 / $7.86 (↓3%, ↓48%)
	Routed	Prefix-200	Prefix-200	0.57 / $0.46 (↓11%, ↓65%)	0.84 / $1.08 (↓2%, ↓61%)	0.83 / $0.93 (↓1%, ↓35%)	0.50 / $5.00 (↓4%, ↓23%)	0.40 / $0.31 (↓2%, ↓54%)	0.78 / $1.60 (↑7%, ↓35%)	0.65 / $9.38 (↓3%, ↓38%)
m_1: Gemma-4B-Thk m_2: Gemma-31B-Thk	m_1:	–	–	0.42 / $0.00	0.65 / $0.00	0.67 / $0.00	–	0.31 / $0.00	0.62 / $0.00	0.53 / $0.00
	m_2:	–	–	0.63 / $0.62	0.87 / $1.34	0.80 / $0.81	–	0.41 / $0.39	0.78 / $1.05	0.70 / $4.21
	Routed	Full	Full	0.59 / $0.43 (↓6%, ↓31%)	0.84 / $1.02 (↓3%, ↓24%)	0.76 / $0.44 (↓5%, ↓46%)	–	0.41 / $0.29 (→0%, ↓26%)	0.76 / $0.71 (↓3%, ↓32%)	0.67 / $2.89 (↓4%, ↓31%)
	Routed	Full	Prefix-200	0.50 / $0.21 (↓21%, ↓66%)	0.78 / $0.77 (↓10%, ↓43%)	0.72 / $0.31 (↓10%, ↓62%)	–	0.38 / $0.22 (↓7%, ↓44%)	0.76 / $0.27 (↓3%, ↓74%)	0.63 / $1.78 (↓10%, ↓58%)
	Routed	Prefix-200	Prefix-200	0.58 / $0.49 (↓8%, ↓21%)	0.84 / $1.11 (↓3%, ↓17%)	0.74 / $0.39 (↓8%, ↓52%)	–	0.41 / $0.30 (→0%, ↓23%)	0.82 / $0.59 (↑5%, ↓44%)	0.68 / $2.88 (↓3%, ↓32%)

Table 6: **End-to-end routed performance under the three prefix-time regimes from Table 5.** Table 5 evaluates the quality of the Capability signal itself; this appendix table shows the corresponding effect on routed task score and paid API cost. *Full / Full* trains and applies the Capability Head on full trajectories, *Full / Prefix-200* applies a full-trajectory-trained head to only the first 200 generated tokens, and *Prefix-200 / Prefix-200* trains and applies the head directly on 200-token prefixes. Each cell reports task score / estimated paid API cost. For routed systems, the inline parenthesis reports the relative change in score and the relative reduction in paid API cost with respect to the corresponding always-large baseline (Model 2).

A.2 Resolution Head Labels from WHEN2CALL

WHEN2CALL is designed to evaluate tool-calling decision-making, including when to make a tool call, when to ask a follow-up question, and when the query cannot be answered under the current tool setup. However, its released training data is preference-oriented and does not directly provide the exact per-example labels required by our Resolution Head.

We therefore apply an external judge LLM offline to derive a gold resolution decision label for each example in the action space $\mathcal{A} = \{\text{info}, \text{tool}, \text{cant}\}$. We then prompt the frozen backbone on the same query, record its generated output and hidden-state trajectory, and train the Resolution Head against the derived gold label regardless of whether the backbone's explicit action choice is itself correct. This construction is intentional: it allows the head to recover the correct resolution decision from the model's hidden-state dynamics even when the model's textual behavior is wrong.

B Additional Experiments

B.1 End-to-End Effect of Prefix-Time Capability Prediction

Table 6 extends Table 5 from signal-quality evaluation to end-to-end routed behavior. While Table 5 measures the quality of the Capability Control signal itself, here we report the resulting task score and paid API cost when that signal is used for actual routing. A weaker prefix-time routing signal can lead to poorer transfer decisions and, in practice, may increase paid API cost by escalating more cases to the large model. Training directly on partial trajectories improves the prefix-time adequacy signal and can yield a better early-routing quality–cost tradeoff than naively applying a full-trajectory-trained head to partial generations.

B.2 Prompt-Level Self-Switching as a Baseline

We also compare our latent-control routing against a simple prompt-level self-switching baseline. In this baseline, the smaller model is explicitly instructed not to answer when it is unsure; if no answer is returned, control is transferred to the stronger fallback model. This tests whether the smaller model can trigger useful handoff through its own surface behavior, without an auxiliary latent control signal.

Table 7 reports results for the **Qwen3.5-9B → Qwen3.5-27B-Thk** setting on two representative benchmarks: **ScreenSpot-Pro** and **MMLU-Pro**. We compare the prompt-level self-switching baseline against the corresponding single-model and latent-routed systems from the main paper. The self-switching baseline triggers escalation only very rarely—just **4 / 1581** cases (**0.25%**) on ScreenSpot-Pro and **28 / 1000** cases (**2.8%**) on MMLU-Pro. As a result, its performance remains essentially at the small-model level and falls well short of latent-control routing.

This comparison highlights an important distinction. The issue is not whether the smaller model can sometimes express uncertainty in text, but whether it can do so reliably enough to drive useful

System	ScreenSpot-Pro *action acc.*	MMLU-Pro *accuracy*
m_1: (**Qwen3.5-9B**)	0.47	0.71
m_2: (**Qwen3.5-27B-Thk**)	0.65	0.80
Prompt-level self-switching	0.4719	0.7030
Latent routed	0.64	0.78

Table 7: **Prompt-level self-switching versus latent-control routing for Qwen3.5-9B → Qwen3.5-27B-Thk.** All entries are benchmark scores. Prompt-level self-switching remains near the small-model baseline on both benchmarks, while latent-control routing recovers substantially more of the stronger model's performance.

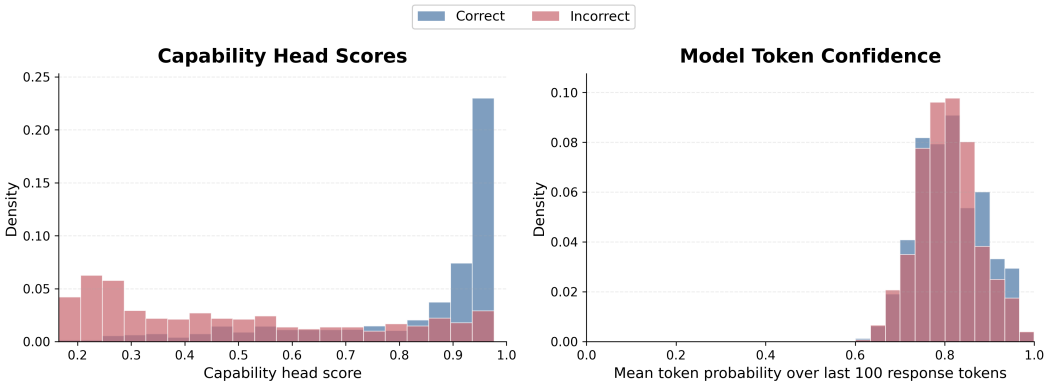

Figure 4: **Capability Head scores versus model token confidence on Qwen3.5-9B over ScreenSpot-Pro.** The left panel shows the distribution of Capability Head scores for correct and incorrect predictions. The right panel shows the corresponding distribution of the model's mean token probability over the last 100 response tokens. Token confidence exhibits substantial overlap between correct and incorrect cases, making it a weak adequacy signal. In contrast, the Capability Head provides much stronger separation, with correct cases concentrated at high scores and incorrect cases distributed much more broadly.

deployment-time control. In these experiments, relying on prompt-level abstention leads to severe under-escalation, whereas the Capability Head produces a much more effective transfer signal and recovers substantially more of the stronger model's performance.

B.3 Model Token Confidence vs. Latent Adequacy Signal

We also analyze whether the model's own token-level confidence can serve as a useful substitute for the Capability Head. Figure 4 compares, on **Qwen3.5-9B** over **ScreenSpot-Pro**, the distribution of **Capability Head** scores against the model's mean token probability over the last 100 response tokens, separating **correct** and **incorrect** predictions. The contrast is clear. The model's token confidence shows substantial overlap between correct and incorrect cases, indicating that surface-level confidence is a weak signal for deciding whether the current model is actually adequate for the instance. By contrast, the Capability Head produces much stronger separation: correct cases concentrate at high head scores, while incorrect cases are distributed much more broadly and extend far more heavily into lower-score regions. This suggests that the latent adequacy signal decoded by the head is substantially more informative for deployment-time control than the model's own token confidence. More broadly, the result supports the core premise of the paper: useful control signals need not be reliably expressed in the model's surface probabilities, but can still be recovered from its hidden-state trajectory.

Layer Used	ROC-AUC ↑	AUPR-C ↑	AUPR-I ↑	ECE ↓
Layer 5	0.85	0.88	0.68	0.21
Middle	0.87	0.89	0.72	0.16
Final	0.86	0.91	0.75	0.14

Table 8: Layer-choice ablation for the Capability Head on Qwen3-VL-4B-Thk. We report average ROC-AUC, AUPR-C, AUPR-I, and ECE across all evaluation benchmarks. The final layer is the best overall choice and is therefore used by default.

Layer Used	F1 ↑	Acc ↑
Layer 5	49.1	64.2
Middle	52.07	70.1
Final	50.2	66.0

Table 9: Layer-choice ablation for the Resolution Head on Qwen3-VL-4B evaluated on WHEN2CALL. The middle layer provides the strongest signal for resolution intervention prediction and is therefore used by default.

C Ablations

C.1 Layer Choice for the Capability Head

We ablate which hidden-state layer provides the strongest adequacy signal for the Capability Head. Using **Qwen3-VL-4B-Thk**, we train the same Capability Head on hidden-state traces taken from three representative layers: an early layer (Layer 5), a middle layer, and the final layer. We report the average ROC-AUC, AUPR-C, AUPR-I, and ECE across all evaluation benchmarks. This ablation shows that the final-layer trace provides the strongest overall signal for model adequacy prediction, and motivates the default choice used in the main paper.

C.2 Layer Choice for the Resolution Head

We ablate which hidden-state layer provides the strongest signal for the Resolution Head. Using **Qwen3-VL-4B** on WHEN2CALL, we train the same Resolution Head on traces taken from three representative layers: an early layer (Layer 5), a middle layer, and the final layer. We report F1 and accuracy on the validation split. This ablation shows that a middle-layer trace provides the strongest resolution decision signal, and motivates the default choice used in the main paper.

C.3 Training-Data Breadth

We next examine whether the breadth of the Capability Head training mixture is genuinely necessary, or whether a useful control signal can be learned from a much narrower domain. This question matters because the goal of the Capability Head is not to predict correctness for one task family, but to estimate *model adequacy* in a way that transfers across various tasks. For that reason, the full 120K training mixture is intentionally constructed to span distinct decision regimes, including visual question answering, grounded perception, reasoning, parametric knowledge, grounding, tool use, and more agentic interaction patterns, rather than serving as a random aggregation of datasets.

To isolate the effect of training-data breadth, we train two Capability Heads on the same frozen **Qwen3-VL-4B** backbone using the same head architecture and optimization setup. The only difference is the training distribution. One head is trained on the full 120K mixed dataset described in Appendix A.1. The other is trained only on **visual-math data**, yielding a substantially narrower signal concentrated on a single reasoning regime.

We evaluate both heads on **ScreenSpot Pro**, which lies outside that narrow visual-math regime and instead emphasizes visually grounded understanding with grounding- and action-oriented structure. This makes it a useful transfer test: if the Capability Head merely learns a task-specific confidence heuristic tied to visual-math trajectories, transfer should be weak. If, instead, it learns a broader ade-

Evaluation dataset: ScreenSpot Pro				
Training data for Capability Head	**ROC-AUC ↑**	**AUPR-Correct ↑**	**AUPR-Incorrect ↑**	**ECE ↓**
Visual-math only	0.62	0.58	0.61	0.59
Full 120K mixed data	0.84	0.85	0.83	0.31

Table 10: **Training-data breadth ablation on ScreenSpot Pro.** Ablation for the Capability Head on the frozen Qwen3-VL-4B backbone. Both heads use the same backbone, latent encoder architecture, and optimization setup; only the training distribution changes. Narrow visual-math-only training transfers poorly to **ScreenSpot Pro**, while the full mixed-data setting yields a substantially stronger and better-calibrated adequacy signal.

Prefix Length	**ROC-AUC ↑**	**AUPR-C ↑**	**AUPR-I ↑**	**ECE ↓**
50 tokens	0.66	0.61	0.53	0.32
200 tokens	0.80	0.82	0.73	0.18
1000 tokens	0.82	0.85	0.72	0.17
Full length	0.86	0.91	0.75	0.14

Table 11: Prefix-length ablation for the Capability Head on Qwen3-VL-4B-Thk. Each row trains and evaluates the head at the same prefix length. We report average ROC-AUC, AUPR-C, AUPR-I, and ECE across benchmarks. Longer prefixes yield stronger adequacy signals and better calibration.

quacy signal from hidden-state trajectories, then the mixed-data head should generalize substantially better.

Table 10 reports results on **ScreenSpot Pro**. The gap is clear. The head trained only on visual-math data transfers poorly, with substantially weaker discrimination and much worse calibration. In contrast, the head trained on the full mixed dataset remains far more reliable. This result directly supports the logic of the mixture design. Broad training exposure is important not because more data is always better in the abstract, but because the deployment problem itself spans heterogeneous behaviors. Training on a deliberately structured mixture encourages the Capability Head to learn a transferable notion of *when the current model is adequate under the current setup*, rather than a narrow notion of confidence tied to a single task family.

C.4 Prefix-Length Ablation for the Capability Head

We study how the quality of the Capability signal depends on how much of the generated answer is observed. To this end, we train prefix-based variants of the Capability Head on **Qwen3-VL-4B-Thk**, using only the first K generated tokens, with $K \in \{50, 200, 1000, \text{full}\}$. We evaluate each variant on the same benchmark suite and report the average ROC-AUC, AUPR-C, AUPR-I, and ECE across benchmarks. This ablation measures how early model adequacy becomes reliably recoverable from hidden-state traces.

D Cost Estimation Details

We estimate API cost from the observed input and output token counts of each model call. For **Qwen3-VL** and **Qwen3.5**, we use the official Alibaba pricing for the corresponding family, matched by model scale and thinking/non-thinking mode. For **Gemma**, since comparable public API pricing is not available for the models used in our experiments, we use the Qwen3-VL pricing table as a proxy, again matched by scale and inference mode. We then compute cost from the corresponding per-1M-token input and output prices. Since the smaller primary models are run locally in our routed setting, the reported paid API cost reflects only fallback-model usage.

E Limitations

The control heads studied here already provide useful deployment-time signals, but their quality still directly affects overall system efficiency. In long-horizon settings, even modest improvements in adequacy prediction or intervention prediction can compound over many steps, leading to better routing decisions, fewer unnecessary escalations, and stronger quality–cost tradeoffs. An important

direction for future work is therefore to further improve the quality, robustness, and calibration of these control signals.

F Broader View of Latent Control

The two heads studied in the main paper should be viewed as concrete instances rather than an exhaustive set. Our broader claim is that hidden-state trajectories provide a reusable substrate for lightweight inference-time control. Prior work has mostly used internal signals for narrow diagnostic goals such as hallucination or correctness estimation. We instead view them as a basis for control.

Once such signals can be read reliably, additional heads can be trained for different deployment needs, user preferences, and application settings, including escalation, clarification, tool use, abstention, confidence shaping, safety filtering, and application-specific routing. In this sense, multi-head latent control is not only a method for the two heads considered here, but also a step toward a broader direction in which frozen models expose lightweight, trainable control interfaces for real-world deployment.