ACTION-INFORMED MERGING FOR AGENTIC MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic models combine reasoning with executable actions to accomplish tasks in interactive environments. Models trained for specific domains often excel within those environments, but their capabilities may not generalize to other domains. Model merging combines the parameters of separately trained models into a single model, offering a way to consolidate their domain-specific strengths into a more broadly capable agent without additional training. However, models trained for different domains may have conflicting parameter updates, making it difficult to broaden domain coverage while retaining specialist-level performance via merging. To combine complementary capabilities while limiting interference, we propose Action-Informed Merging (AIM), which combines action-relevant updates from specialist checkpoints and selects which parameters to modify and how much to change them. Specifically, AIM estimates parameter importance from squared gradients of action-token and full-response-token losses. Updates from specialist checkpoints are weighted according to parameter importance for action prediction, with less weight assigned to conflicting updates. Parameter-block selection then determines which parameters to modify and how much to change them. This selection uses full-response importance to penalize changes to sensitive parameters and constrains estimated loss increases separately for each specialist checkpoint whose capabilities should be retained. The procedure requires *neither further model training nor inference-time routing*. On tool-use benchmarks, AIM reaches a 49.11% equal-weight average of BFCL accuracy and Terminal-Bench success, compared with 47.59% for uniform averaging. Across web, mobile, desktop, and GUI grounding, its four-domain average is 50.58%, a *14.1% relative improvement* over the weighted averaging baseline.

1 INTRODUCTION

Agentic models accomplish tasks by combining reasoning with executable actions and using environmental feedback to guide subsequent decisions. Models trained from a shared pretrained backbone can develop complementary strengths in tool use, web navigation, and mobile interaction. Consolidating these strengths into a single agent would broaden its capabilities and simplify deployment. Model merging combines separately trained models into one model without further training (Wortsman et al., 2022; Ilharco et al., 2023). Figure 1 illustrates the opportunity: specialist models exhibit different performance profiles across action domains, motivating a single model that combines their strengths while limiting losses in existing capabilities.

Merging specialist checkpoints aims to broaden domain coverage while retaining each checkpoint's performance in its original domain. However, training from a common initialization can change corresponding parameters in opposite directions across domains, so merging checkpoint weights may cancel updates that contribute to domain-specific capabilities. Existing merging methods address interference through conflict resolution, parameter-importance weighting, and behavior-guided selection (Yadav et al., 2023; Matena & Raffel, 2022; Yuan et al., 2026; Shao et al., 2026). An agent response can contain both reasoning and executable actions, which play different roles in interaction. Parameter importance estimated from a full-response loss reflects sensitivity to predicting both, whereas an action-token loss measures sensitivity specifically to action prediction. This distinction motivates using action-specific sensitivity to identify contributions relevant to executable behavior, alongside full-response sensitivity to account for other response behavior. Identifying action-relevant updates is not sufficient; we must also determine which parameters to modify and how much to change them to limit performance degradation across domains.

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

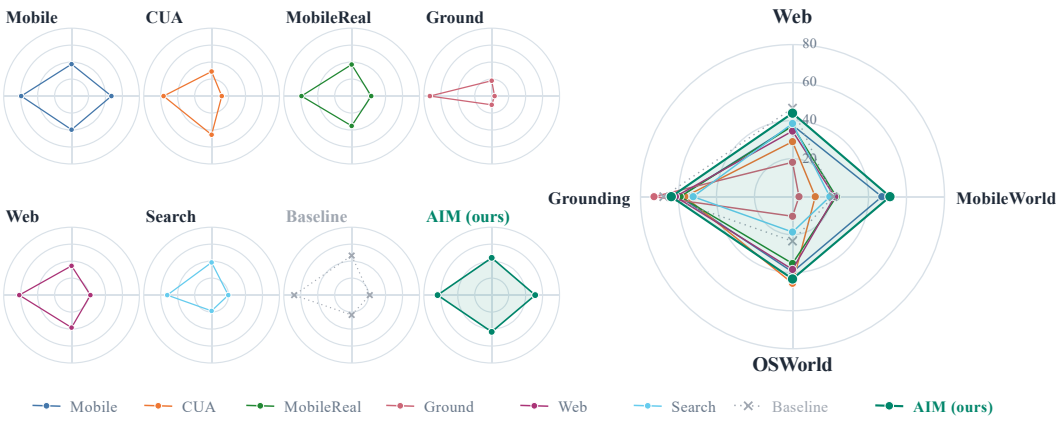

Figure 1: **Cross-domain performance of specialist checkpoints and AIM.** Small radars show the individual profiles of specialists in MobileWorld, computer use (CUA), real-device mobile interaction, grounding, web use, and deep search, alongside a prior-weighted linear baseline and AIM. The large radar overlays their scores (%) on Web, MobileWorld, OSWorld, and Grounding using a shared 0–80 scale. AIM combines strong Web and MobileWorld performance with competitive OSWorld and Grounding scores.

We propose Action-Informed Merging (AIM), a framework that combines action-relevant updates from specialist checkpoints and controls which parameters to modify and by how much. It consolidates complementary capabilities into a single model without further model training or inference-time routing.

Specifically, AIM evaluates each frozen specialist checkpoint on pre-collected trajectories containing task instructions, interaction histories, and responses. Squared gradients of action-token and full-response losses estimate parameter importance for action prediction and complete-response generation, respectively. We weight each specialist's parameter changes relative to the shared pretrained model by action importance. Starting from a weighted average of the specialist parameters, we incorporate more of their combined changes when specialists consistently increase or decrease a parameter. When some increase it and others decrease it, we keep the parameters closer to the weighted average. AIM then selects which parameter blocks to modify and how strongly to update them. For each candidate parameter block, AIM estimates how applying the proposed update would change each specialist's surrogate loss. Action importance rewards movement toward specialist parameters, while full-response importance penalizes changes away from the initial merged model. Within the selected blocks, we use linear interpolation between the initial merged parameters and those obtained by combining specialist updates. The interpolation coefficient controls how far the parameters move toward this combination, while parameters outside the selected blocks remain unchanged.

We evaluate AIM on tool use and graphical user interface (GUI) interaction, covering function calling, multi-turn tool use, terminal execution, visual grounding, and task completion across web, mobile, and desktop environments. For tool use, AIM reaches 71.25% mean accuracy across the two BFCL development categories and 26.97% success on Terminal-Bench 2.1. Its equal-weight average of these two benchmark-suite scores is 49.11%, compared with 47.59% for uniform averaging. For GUI interaction, AIM reaches a four-domain average of 50.58% across Web, MobileWorld, OSWorld, and visual grounding, compared with 44.33% for weighted averaging and 45.77% for the strongest direct-merge AIM configuration. Comparisons under matched inference settings further demonstrate gains over direct merging baselines. Ablations show *complementary gains from visual and language refinements* and illustrate how interpolation strength affects the balance across domains.

Our contributions are threefold: (i) We propose AIM, a framework that combines sensitivity to action prediction with agreement among specialized models to identify compatible parameter contributions for capability consolidation. (ii) We formulate a selective update strategy that uses action and full-response sensitivity to assess candidate changes, with separate constraints on estimated loss changes for individual protected models during parameter-block selection. (iii) We evaluate AIM across tool-use and GUI benchmarks, demonstrating its effectiveness in consolidating specialized capabilities

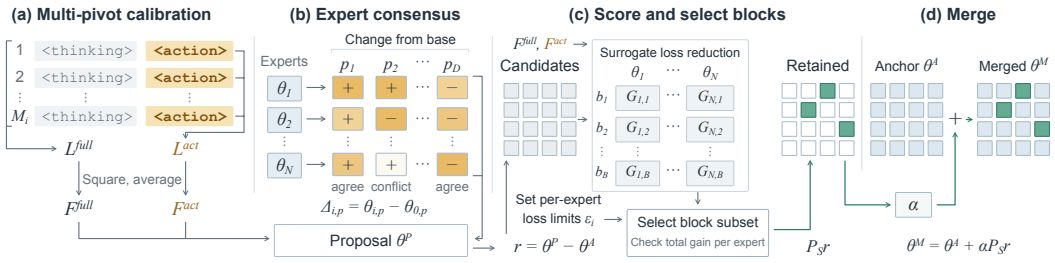

Figure 2: **Method overview of Action-Informed Merging. (a)** Estimate paired full-response and action-masked sensitivities. **(b)** Combine action importance and expert-update agreement into a consensus proposal. **(c)** Select parameter blocks under per-expert surrogate loss budgets. **(d)** Interpolate selected updates from an anchor to produce one checkpoint.

without additional model training. Ablations show complementary gains from visual and language refinements and illustrate how interpolation strength affects the balance across domains.

2 Action-Informed Merging

AIM constructs a consensus proposal using action sensitivity and agreement among specialist updates. It then selects which parameters to change and how strongly to update an anchor checkpoint (Figure 2). The procedure operates offline without additional model training.

2.1 Problem Setup

An agentic model with parameters θ maps a task x and interaction history h to a response containing reasoning and executable actions. The history includes previous responses and environment observations, such as messages, execution results, and images. Executable actions change the environment or elicit observations that inform subsequent responses. Given specialists for different action domains, our goal is to construct a single model that consolidates their complementary capabilities while preserving performance across domains, without additional model training.

Let θ_0 denote the shared pretrained parameters and $\{\theta_i\}_{i=1}^N$ the specialist checkpoints, with a common architecture and corresponding parameter entries. We write $\theta_{i,p}$ for scalar parameter p of specialist i. Each specialist is associated with calibration examples containing interaction contexts, supplied completions, and masks identifying executable action spans. Define the specialist task vectors and a linear reference merge as

$$\Delta_i = \theta_i - \theta_0, \qquad \theta_{\text{lin}} = \theta_0 + \frac{\sum_{i=1}^N w_i \Delta_i}{\sum_{i=1}^N w_i}, \tag{1}$$

where $w_i \geq 0$ and $\sum_i w_i > 0$ specify the expert mixture. Equal weights recover uniform parameter averaging. We denote by θ^A an *anchor* checkpoint whose existing capabilities we seek to retain when incorporating contributions from other specialists. The anchor may be the linear reference θ_{lin} or another compatible checkpoint. The merging procedure uses the specialist parameters and calibration data to produce a single fixed checkpoint θ^M.

2.2 Action-Informed Consensus Construction

We estimate parameter sensitivity using fixed calibration completions at frozen specialist checkpoints. Each token is evaluated given the interaction context and preceding completion tokens. For specialist i and example n, let $\ell_{i,n,t}$ be the negative log-likelihood of token t and T_n the completion

length. The mask $m_{n,t} \in \{0, 1\}$ identifies executable action tokens. We define

$$L_{i,n}^{\text{full}} = \frac{1}{T_n} \sum_{t=1}^{T_n} \ell_{i,n,t}, \qquad L_{i,n}^{\text{act}} = \frac{1}{T_n} \sum_{t=1}^{T_n} m_{n,t} \ell_{i,n,t}. \tag{2}$$

The action-masked loss includes only action tokens. Preceding reasoning and observations remain in the conditioning context. Both losses use the full completion length as the denominator, so short action spans receive no additional weight.

We compute sensitivity from squared loss gradients:

$$F_{i,p}^{b} = \frac{1}{M_i} \sum_{n=1}^{M_i} \eta_{i,n} \left(\frac{\partial L_{i,n}^{b}}{\partial \theta_{i,p}} \right)^2, \qquad b \in \{\text{act}, \text{full}\}, \tag{3}$$

where p indexes a scalar parameter, M_i is the number of calibration examples, and $\eta_{i,n} \geq 0$ is an example weight. These statistics are empirical Fisher proxies based on sequence-loss gradients (Kunstner et al., 2019).

For block selection, we normalize both estimates by the same positive scale:

$$a_{i,p} = \frac{F_{i,p}^{\text{act}}}{Z_{i,T(p)}}, \qquad h_{i,p} = \frac{F_{i,p}^{\text{full}}}{Z_{i,T(p)}}. \tag{4}$$

Here $T(p)$ is the tensor containing p, and $Z_{i,T(p)}$ is derived from its full-response sensitivity. The shared scale reduces magnitude differences across specialists while preserving the relative strength of the two signals.

Calibration evidence and expert agreement guide the proposal:

$$\theta_p^P = \theta_{0,p} + (1 - g_p)\Delta_p^{\text{ref}} + g_p \Delta_p^{\text{agree}}, \qquad 0 \leq g_p \leq 1. \tag{5}$$

Both updates are defined relative to the shared base θ_0. The reference update Δ^{ref} provides a fallback, while Δ^{agree} combines compatible specialist updates. The gate g_p reflects action relevance, support across specialists, and agreement among update directions. Strong support favors consensus; weak or conflicting support favors the reference.

2.3 Sensitivity-Guided Update Selection

We apply the proposal to an anchor θ^A whose capabilities we seek to retain. The anchor can differ from the reference used to construct the proposal. For a selected parameter set S, the merged checkpoint is

$$\theta^M = \mathcal{Q}\left[\theta^A + \alpha P_S(\theta^P - \theta^A)\right], \qquad 0 \leq \alpha \leq 1, \tag{6}$$

where P_S retains updates only in S. The interpolation coefficient α controls their magnitude, and $\mathcal{Q}$ converts the result to the checkpoint's storage precision. We select S and α offline. For block selection, we constrain the loss change separately for each protected expert.

Let $u = \theta^P - \theta^A$ be the difference between the stored proposal and anchor. We partition its parameters into disjoint blocks. To evaluate candidate changes, we define a quadratic surrogate:

$$\widetilde{L}_i(\theta; \theta^A) = \frac{1}{2} \sum_p \left[a_{i,p}(\theta_p - \theta_{i,p})^2 + \kappa h_{i,p}(\theta_p - \theta_p^A)^2 \right], \qquad \kappa \geq 0. \tag{7}$$

The first term favors movement toward expert i at action-sensitive parameters. The second penalizes movement away from the anchor at parameters with high full-response sensitivity. The coefficient κ controls this penalty.

Let u_b contain the update entries in block b and zeros elsewhere. Its gain for expert i is the reduction in surrogate loss:

$$\begin{aligned} G_{i,b} &= \widetilde{L}_i(\theta^A;\theta^A) - \widetilde{L}_i(\theta^A + u_b;\theta^A) \\ &= \sum_{p\in b}\left[-a_{i,p}(\theta^A_p - \theta_{i,p})u_{b,p} - \frac{1}{2}(a_{i,p} + \kappa h_{i,p})u^2_{b,p}\right]. \end{aligned} \tag{8}$$

Positive values indicate a decrease in surrogate loss. The gains add exactly across disjoint blocks because the surrogate has no cross-parameter terms. We select blocks jointly by solving

$$\max_{\{z_b\in\{0,1\}\}_{b\in\mathcal{B}}} \sum_{b\in\mathcal{B}} r_b z_b \quad \text{subject to} \quad \sum_{b\in\mathcal{B}} G_{i,b} z_b \geq -\epsilon_i, \qquad i \in \mathcal{I}. \tag{9}$$

Here $\mathcal{B}$ is the set of candidate blocks, z_b indicates selection, and r_b is the block's utility. The set $\mathcal{I}$ contains the protected experts. Each budget $\epsilon_i \geq 0$ limits the increase in expert i's surrogate loss. Gains in some blocks can offset losses in others within $\mathcal{B}$. Gains for one expert cannot compensate for losses to another.

We use two choices of utility and selection scope. To retain as many parameter changes as possible, we set $r_b = n_b$, where n_b counts changed entries in block b. We solve this objective within each layer–module group, with zero budgets for all experts. To prioritize particular expert capabilities, we instead set $r_b = \sum_i v_i G_{i,b}$, where $v_i \geq 0$ specifies expert priorities. This objective selects blocks across groups under the prescribed per-expert budgets.

The selected blocks define $S = \bigcup_{b:z_b=1} b$. We apply their updates using Equation 6. If a selected update satisfies the budgets, applying a smaller fraction to the same anchor also satisfies them, assuming no numerical rounding. These budgets constrain surrogate losses. We evaluate task performance on the final stored checkpoint.

3 Experiments

We study whether action-informed merging can combine specialist capabilities and how restricting an update affects the resulting agent. The experiments cover tool use and multimodal computer use, with separate expert pools and evaluation protocols.

3.1 Experimental setup

Models and evaluation. All specialists are derived from Qwen3.5-4B (Qwen Team, 2026). We merge three tool-use experts and six computer-use experts without optimizer updates or inference-time routing. Tool use is evaluated on function-calling and multi-turn categories of BFCL (Patil et al., 2025), including six categories evaluated after configuration selection, and on terminal execution with Terminal-Bench 2.1 (The Terminal-Bench Team, 2026). Computer use covers browser interaction (Web-L2), mobile interaction (MobileWorld; Kong et al., 2025), desktop interaction (OSWorld; Xie et al., 2024), and six visual-grounding datasets. We report all benchmark-specific scores and summarize computer use by *Avg*, the unweighted arithmetic mean of the Web, MobileWorld, OSWorld, and Grounding percentages. Grounding first averages the six localization accuracies equally, so each of the four domains receives one-quarter of the overall weight. This descriptive summary is not pooled task accuracy. Dataset definitions, task counts, calibration data, and decoding settings are in Appendix A.

3.2 Tool Use and Terminal Execution

We compare AIM with the Qwen3.5-4B base model (Qwen Team, 2026), the APIGen (Fitzsimons, 2026), AgentMercury (Minbyul, 2026), and FACET Terminal (Shi et al., 2026) specialists, uniform averaging (Wortsman et al., 2022), full-response Fisher merging (Matena & Raffel, 2022), and our action-masked Fisher variant.

Table 1 shows that AIM achieves the highest BFCL mean (71.25%) and Terminal-Bench 2.1 success rate (26.97%) among the evaluated models. The specialists exhibit complementary strengths: APIGen leads in parallel calls, whereas FACET Terminal performs better in multi-turn interaction.

Table 1: **Tool-use and terminal-execution results (%).** Each BFCL category contains 200 development examples; its mean averages only those two categories. TB 2.1 denotes Terminal-Bench 2.1. Overall Avg. equally weights the BFCL Avg. and TB 2.1 success rate, treating the two benchmark suites as separate domains. Bold and underlining mark the best and second-best reported values in each column.

Model	BFCL			TB 2.1	Overall
	Parallel calls	Multi-turn	Avg.	Success	Avg.
Qwen3.5-4B (base)	74.50	60.50	67.50	18.43	42.97
APIGen	**85.00**	54.00	69.50	8.05	38.78
AgentMercury	74.50	54.50	64.50	16.28	40.39
FACET Terminal	72.00	61.00	66.50	13.48	39.99
Uniform averaging	80.00	60.00	<u>70.00</u>	25.17	<u>47.59</u>
Fisher (full-response)	69.50	**65.50**	67.50	<u>26.52</u>	47.01
Fisher (action-masked, Ours)	71.50	<u>62.50</u>	67.00	25.84	46.42
AIM (Ours)	<u>83.00</u>	59.50	**71.25**	**26.97**	**49.11**

Uniform averaging already benefits from this complementarity, outperforming every specialist on BFCL mean. Both Fisher variants, however, fall below this baseline: their higher multi-turn accuracy comes with substantially lower parallel-call accuracy. This suggests that parameter-importance weighting alone does not ensure effective capability consolidation. AIM achieves a more favorable balance, improving parallel-call accuracy over uniform averaging while retaining nearly the same multi-turn accuracy. Its stronger terminal performance further suggests that the benefits extend beyond structured function calling. The equal-weight average of the BFCL mean and Terminal-Bench success rate is 49.11%, a 3.2% relative improvement over uniform averaging (47.59%). On six additional BFCL categories evaluated after configuration selection, pooled accuracy also increases from 34.74% to 35.77%.

3.3 Cross-Domain Computer Use

Table 2 compares the cross-domain performance of direct merges and anchor refinements, including weighted averaging (Wortsman et al., 2022) and full-response Fisher merging (Matena & Raffel, 2022) as baselines. Among direct merges with matched settings, AIM (consensus) achieves the highest four-domain average of 45.77%. Relative to weighted averaging, it improves mobile performance by 4.27 percentage points, with smaller gains on desktop and grounding and a 0.09-point decrease on web tasks. Its main benefit is therefore strengthening the weakest domain while largely retaining performance elsewhere. AIM (consensus) also has the highest grounding score in this direct-merge group (68.79%), while full-response Fisher has the lowest desktop success rate (31.21%), illustrating that localization accuracy alone does not determine interactive performance.

Within the anchor-refinement group, AIM (balanced) improves all three interaction domains over both anchors. Compared with the vision-repaired anchor, it gains 3.16, 0.85, and 1.79 percentage points on web, mobile, and desktop tasks, respectively, with a 0.20-point decrease in grounding. Its average reaches 50.58%, a 2.8% relative improvement, showing gains across interaction domains with a small grounding trade-off. The average also exceeds weighted averaging (44.33%) and action-normalized Fisher (44.44%) by 14.1% and 13.8% in relative terms, respectively.

3.4 GUI Grounding

Table 3 reports point-in-target accuracy on six GUI-grounding benchmarks. The specialist checkpoints provide a reference for the capability profiles, while the base and merge methods in the lower block are evaluated on matched task lists. Among the merging methods, AIM (consensus) obtains the highest six-benchmark average (68.79%), compared with 67.83% for full-response Fisher and 67.21% for action-masked Fisher. Figure 3 shows the per-benchmark profiles, contrasting the

Table 2: **GUI domain results. (%)** All rows have complete coverage in the same evaluation batch. MW and OSW denote MobileWorld and OSWorld; Ground. is the mean of six grounding datasets. Overall is the equal mean of the four domain scores. Bold marks the best score within each block. All scores are percentages.

Method	WEB-L2	MW	OSW	Ground.	Overall
Direct merge comparisons					
Weighted averaging	46.22	19.66	43.77	67.68	44.33
Fisher (full-response)	43.98	20.51	31.21	67.83	40.88
Fisher (action-normalized) (Ours)	**47.85**	17.95	**44.44**	67.52	44.44
AIM (consensus) (Ours)	46.13	**23.93**	44.23	**68.79**	**45.77**
Anchor refinements					
Mobile-balanced anchor (Ours)	40.74	45.30	40.78	**64.21**	47.76
Vision-repaired anchor (Ours)	40.78	50.43	41.51	64.02	49.19
AIM (balanced) (Ours)	**43.94**	**51.28**	**43.30**	63.82	**50.58**

Table 3: **Grounding accuracy of specialists, baselines, and AIM variants.** Scores are point-in-target accuracy (%); columns are MMBench grounding (MMB-G), OS-G, OS-G Refine (OS-G-R), ScreenSpot-Pro (SS-Pro), ScreenSpot-V2 (SS-V2), and UI-Vision (UI-V). Overall is the unweighted mean of the six scores. The upper block contains complete historical specialist references; all models in the lower block use the same complete task lists and evaluation protocol. Cross-block comparisons are descriptive. Bold and underline indicate the best and second-best merging methods in each column, respectively; specialist scores are contextual references.

Model	MMB-G	OS-G	OS-G-R	SS-Pro	SS-V2	UI-V	Overall
Specialist checkpoints							
MW expert	76.77	51.06	59.40	52.56	90.88	25.52	59.36
CUA expert	76.24	49.11	56.03	44.02	89.07	25.42	56.65
Mobile-real expert	75.65	54.61	60.82	48.07	91.19	24.58	59.15
Web expert	79.27	54.08	64.01	52.37	91.82	28.25	61.63
DeepSearch expert	63.08	39.18	56.56	48.32	87.34	19.31	52.30
Grounding expert	**87.90**	**64.72**	**70.39**	**67.62**	**95.28**	**51.47**	**72.90**
Base and merge methods							
Qwen3.5-4B base	73.57	51.42	64.01	56.29	91.82	24.68	60.30
Full-response Fisher	<u>84.70</u>	60.46	<u>67.73</u>	61.29	93.71	<u>39.08</u>	<u>67.83</u>
Action-masked Fisher (Ours)	82.61	**61.70**	67.38	<u>61.99</u>	<u>94.50</u>	35.12	67.21
AIM (Ours)	**85.48**	<u>61.35</u>	**68.09**	**63.44**	**94.58**	**39.81**	**68.79**

grounding-focused merge with the specialists and with the base and merging baselines. Evaluation protocols, task counts, and the interpretation of the calibration overlap are described in Appendix A.3.

3.5 ABLATION STUDIES

We examine how update construction, parameter scope, and interpolation strength affect the merged model. Table 4 isolates the contributions of module-specific updates after anchor construction, with evaluation and inference settings held fixed. Both updates interpolate toward the full-response Fisher merge within their respective parameter subsets. The 10.00% visual update raises MobileWorld success from 45.30% to 50.43%, while changing web and grounding scores by less than 0.20 points. Starting from this vision-updated anchor, we compare language-update strengths of 2.50% and 5.00%. The 2.50% update improves web, mobile, and desktop performance by 3.16, 0.85, and 1.79 points, respectively, yielding AIM (balanced). Doubling its strength lowers these scores

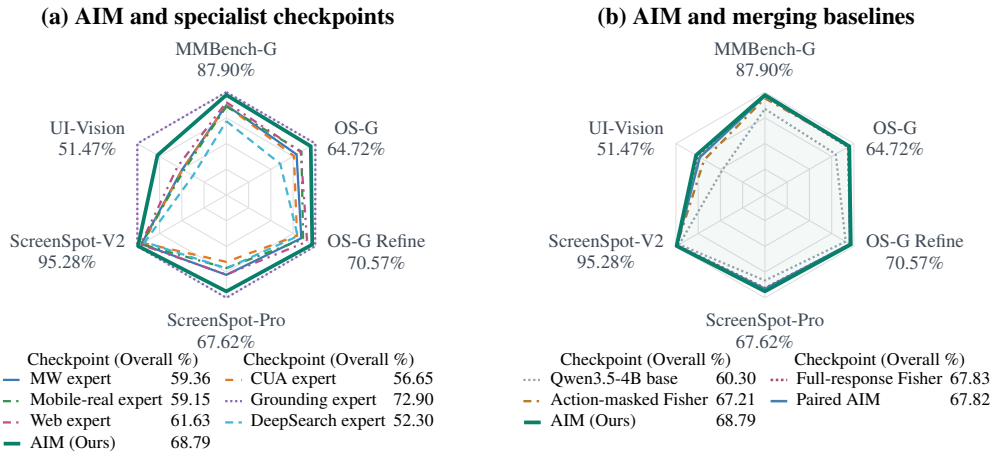

Figure 3: **Grounding accuracy across six visual-grounding benchmarks.** Each polygon shows the model's score relative to the highest displayed score on that dataset; the corresponding maximum is printed beside each axis. Panel (a) compares our method with the six domain specialists. Panel (b) compares the base model and several baseline methods.

Table 4: **Ablation of visual and language refinements.** Starting from the balanced anchor, we apply a visual update, then compare two language-update strengths from the same vision-updated anchor. Parenthesized percentages denote interpolation strengths. MW, OSW, and Ground. denote MobileWorld, OSWorld, and the six-benchmark grounding mean. Scores are percentages; bold marks the best in each column.

Construction	WEB-L2	MW	OSW	Ground.
Balanced anchor (Ours)	40.74	45.30	40.78	**64.21**
+ vision update (10.00%, Ours)	40.78	50.43	41.51	64.02
+ language update (2.50%, Ours)	**43.94**	**51.28**	**43.30**	63.82
+ language update (5.00%, Ours)	42.65	47.86	39.73	63.81

by 1.29, 3.42, and 3.57 points, while grounding differs by only 0.01 points. These results highlight two complementary decisions in applying a merge: visual and language updates contribute differently across domains, and a modest language update preserves a better balance than stronger interpolation. Thus, identifying a useful update direction is not sufficient; its parameter scope and interpolation strength also shape the resulting capabilities.

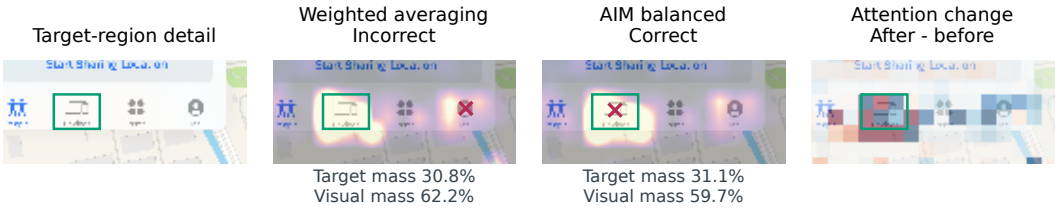

Figure 4: **Visual attention during action generation.** From left to right: the input crop, weighted averaging, AIM, and their attention difference (AIM minus weighted averaging). Green boxes identify the target; magenta crosses show each model's generated click. The attention heatmaps share a color scale, with brighter regions indicating greater attention. In the difference map, red indicates increased attention and blue indicates decreased attention. *Target mass* is the fraction of image-token attention assigned to the target; *visual mass* is the fraction of total attention assigned to image tokens.

3.6 Visualizing Attention During Action Generation

Figure 4 compares AIM and weighted averaging on a navigation example with matched inputs and preprocessing. We visualize attention to image tokens immediately before the first action-coordinate token, averaged across all heads in the eight full-attention layers. For the instruction "check device location," AIM clicks the correct *Devices* control, while weighted averaging clicks the neighboring *Me* tab. The difference map shows increased attention around *Devices* and reduced attention around *Me*. In this example, the correct action coincides with a shift in visual attention toward the intended target. Appendix F provides layerwise maps and additional desktop examples.

4 Related Work

4.1 Model Merging

Model merging combines specialized models into a single checkpoint. Model soups average fine-tuned parameters, while task arithmetic combines updates relative to a shared initialization (Wortsman et al., 2022; Ilharco et al., 2023). Model Stock estimates a central model from a small number of fine-tuned checkpoints (Jang et al., 2024). To reduce interference, TIES removes small updates and resolves sign conflicts (Yadav et al., 2023), while DARE randomly drops and rescales updates (Yu et al., 2024). Task Singular Vectors and Iso-CTS address the same problem through low-rank task representations and the balancing of shared and task-specific components, respectively (Gargiulo et al., 2025; Marczak et al., 2025).

Data-informed methods use calibration evidence to guide merging. Fisher merging estimates parameter importance from gradients (Matena & Raffel, 2022), RegMean uses activation statistics (Jin et al., 2023), and functionality-oriented merging incorporates model behavior through a geometric formulation (Wang et al., 2026a). AdaMerging and LiNeS further control expert contributions by adapting merging coefficients or rescaling updates across layers (Yang et al., 2024; Wang et al., 2025). Recent approaches extend these ideas to agentic models. RAM separates shared and task-specific updates (Yuan et al., 2026). ARM uses role-conditioned activations to transfer expert neurons while protecting neurons important to other tasks (Feng et al., 2026). AgentPatch identifies behaviors to repair and preserve from interaction trajectories, then applies targeted updates while excluding protected neurons (Shao et al., 2026). Building on importance-aware merging and behavior-guided parameter selection, AIM jointly considers action-specific and full-response sensitivity to construct consensus updates across specialists.

4.2 Agent Training

Agentic models use reasoning and executable actions to interact with external environments. ReAct interleaves reasoning and actions with environment observations (Yao et al., 2023), while Agent-Tuning develops general agent capabilities through training on interaction trajectories and general instructions (Zeng et al., 2024). Distillation also enables capability consolidation: MAGA emphasizes structured executable actions during distillation (Yan et al., 2026), and UI-MOPD transfers knowledge from platform-specific teachers on states visited by the student (Lian et al., 2026). These approaches acquire or consolidate capabilities through model training. AIM addresses the complementary setting in which specialist checkpoints are already available and their capabilities are combined without additional model training.

5 Conclusion

We introduced AIM, a training-free framework for consolidating capabilities from specialist checkpoints across action domains. The framework uses parameter sensitivity to action-masked and full-response losses, together with agreement in update directions, to combine specialist updates. It then selects which parameters to modify and controls the magnitude of these changes through interpolation with an anchor checkpoint. Experiments on tool use and computer use demonstrate that AIM combines complementary capabilities while limiting performance degradation across domains. Ablations further show that controlling both the scope and magnitude of updates helps retain the anchor's existing capabilities.

AI USE STATEMENT

Generative AI tools were used for literature retrieval, language editing, LaTeX assistance, and assistance with writing some visualization scripts. The authors take full responsibility for all methods, implementations, experimental results, and scientific claims presented in this manuscript, as well as all AI-assisted text and code.

REFERENCES

Zhuoka Feng, Kang Chen, Sihan Zhao, Kai Xiong, Yaoning Wang, Minshen Yu, Junjie Nian, Changyi Xiao, Yixin Cao, and Yugang Jiang. ARM: Role-conditioned neuron transplantation for training-free generalist LLM agent merging. *arXiv preprint arXiv:2601.07309*, 2026. URL https://arxiv.org/abs/2601.07309.

Colin Fitzsimons. Qwen3.5-4B fine-tuned on APIGen-MT-5k for edge tool use. Hugging Face model repository, 2026. URL https://huggingface.co/CFitzsimons/Qwen3.5-4B-APIGen-MT-5k-F16-GGUF. Checkpoint revision 3c60a476.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2412.00081. arXiv:2412.00081.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2212.04089. arXiv:2212.04089.

Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. Model stock: All we need is just a few fine-tuned models. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2403.19522. arXiv:2403.19522.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2212.09849. arXiv:2212.09849.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and MCP-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025. URL https://arxiv.org/abs/2512.19432.

Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1905.12558. arXiv:1905.12558.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Ruslan Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=5298fKGmv3. arXiv:2412.05467.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8778–8786. Association for Computing Machinery, 2025. doi: 10.1145/3746027.3755688. URL https://doi.org/10.1145/3746027.3755688.

Niu Lian, Tongbo Chen, Zhehao Yu, Chengzhen Duan, Fazhan Liu, Hui Liu, Pei Fu, Jian Luan, Heng Qu, Shu-Tao Xia, and Jinpeng Wang. UI-MOPD: Multi-platform on-policy distillation for unified GUI agents. *arXiv preprint arXiv:2607.04425*, 2026. URL https://arxiv.org/abs/2607.04425.

Daniel Marczak, Simone Magistri, Sebastian Cygert, Bartłomiej Twardowski, Andrew D Bagdanov, and Joost van de Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2502.04959. arXiv:2502.04959.

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *Advances in Neural Information Processing Systems*, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/70c26937fbf3d4600b69a129031b66ec-Abstract-Conference.html. arXiv:2111.09832.

Minbyul. AgentMercury-Qwen3.5-4B model card. Hugging Face model repository, 2026. URL https://huggingface.co/Minbyul/AgentMercury-Qwen3.5-4B. Checkpoint revision a392d4a6.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A. Rodriguez, Montek Kalsi, Nicolas Chapados, M. Tamer Özsu, Aishwarya Agrawal, David Vazquez, Christopher Pal, Perouz Taslakian, Spandana Gella, and Sai Rajeswar. UI-Vision: A desktop-centric GUI benchmark for visual perception and interaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 45817–45851. PMLR, 2025. URL https://proceedings.mlr.press/v267/nayak25a.html.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning*, pp. 48371–48392, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Qwen Team. Qwen3.5-4B model card. Hugging Face model repository, 2026. URL https://huggingface.co/Qwen/Qwen3.5-4B. Shared initialization; model revision 851bf6e8.

Zibo Shao, Baochen Xiong, Chengdong Xu, Linhui Xiao, Kaichen Li, Haoran Gong, Yan Li, Yaguang Song, and Xiaoshan Yang. AgentPatch: Coarse-to-fine weak-task repair for merging agentic multimodal large language models. *arXiv preprint arXiv:2608.06699*, 2026. URL https://arxiv.org/abs/2608.06699.

Kou Shi, Zun Wang, Qisheng Su, Shiting Huang, Ziao Zhang, Zhen Fang, Qingnan Ren, Jin Liu, Yu Zeng, Yiming Zhao, Lin Chen, Zehui Chen, and Feng Zhao. FACET: Preserving source intent and executable state in terminal task synthesis. *arXiv preprint arXiv:2608.18580*, 2026. URL https://arxiv.org/abs/2608.18580.

The Terminal-Bench Team. Terminal-Bench 2.1. Official benchmark release announcement, 2026. URL https://www.tbench.ai/news/terminal-bench-2-1.

Jiayu Wang, Zuojun Ye, and Wenpeng Yin. Functionality-oriented LLM merging on the fisher–rao manifold. *arXiv preprint arXiv:2603.04972*, 2026a. URL https://arxiv.org/abs/2603.04972.

Ke Wang, Nikolaos Dimitriadis, Alessandro Favero, Guillermo Ortiz-Jimenez, François Fleuret, and Pascal Frossard. LiNeS: Post-training layer scaling prevents forgetting and enhances model merging. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.17146. arXiv:2410.17146.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, Weiyun Wang, Xiangyu Zhao, Jixuan Chen, Haodong Duan, Tianbao Xie, Chenyu Yang, Shiqian Su, Yue Yu, Yanting Zhang, Xiangyu Yue, Weijie Su, Xizhou Zhu, Wei Shen, Jifeng Dai, and Wenhai Wang. MMBench-GUI: A unified hierarchical evaluation framework for multi-platform GUI agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6239–6248, 2026b. URL https://openaccess.thecvf.com/content/CVPR2026/papers/Wang_MMBench-GUI_A_Unified_Hierarchical_Evaluation_Framework_for_Multi-Platform_GUI_Agents_CVPR_2026_paper.pdf.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pp. 23965–23998, 2022. URL https://proceedings.mlr.press/v162/wortsman22a.html.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-ATLAS: A foundation action model for generalist GUI agents. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=n9PDaFNi8t.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094, 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. Scaling computer-use grounding via user interface decomposition and synthesis. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0817. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/22c868099177ee278eb7baccec649f35-Abstract-Datasets_and_Benchmarks_Track.html.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.01708. arXiv:2306.01708.

Hang Yan, Zhangxuan Gu, Beitong Zhou, Jiaxuan Chen, Runze Li, Yusong Hu, Shuheng Shen, and Changhua Meng. MAGA: Multi-platform self-fusion of GUI agents via structured action distillation. *arXiv preprint arXiv:2607.29320*, 2026. URL https://arxiv.org/abs/2607.29320.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. AdaMerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.02575. arXiv:2310.02575.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629. arXiv:2210.03629.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2311.03099. arXiv:2311.03099.

Xiangchi Yuan, Dachuan Shi, Chunhui Zhang, Zheyuan Liu, Shenglong Yao, Soroush Vosoughi, and Wenke Lee. Behavior knowledge merge in reinforced agentic models. *arXiv preprint arXiv:2601.13572*, 2026. URL https://arxiv.org/abs/2601.13572.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

Shuyan Zhou, Frank Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=zlsj9akpaa.

A Datasets and Evaluation Protocols

A.1 Specialist checkpoints

All specialists share the Qwen3.5-4B initialization and architecture. The tool-use pool contains APIGen, AgentMercury, and FACET Terminal, specialized in function calling, multi-turn interaction, and terminal execution, respectively. All three share the same pretrained initialization and receive equal merging weights.

The computer-use pool contains six specialists: MobileWorld, desktop, real-device mobile, grounding, Web, and Deep Search. Weighted averaging uses priors proportional to $20/20/20/20/10/10$ in this order. The MobileWorld and real-device checkpoints are separate specialists within the mobile domain. Uniform averaging assigns equal weights to all six specialists.

A.2 Tool-use benchmarks

Function calling. BFCL development data comprise parallel calls, parallel calls with multiple functions, and multi-turn tool use, with 200 examples per category. After configuration selection, the checkpoint is fixed and evaluated on six additional categories: missing functions, missing parameters, long context, and three memory settings. The first three contain 200 examples each; each memory setting contains 155 examples, giving 1,065 post-selection examples. The two-category BFCL mean averages parallel-call and multi-turn accuracy only.

Terminal execution. Terminal-Bench 2.1 evaluates task completion in terminal environments (The Terminal-Bench Team, 2026). Success rate is $100n/N$, where n is the number of successful tasks and N is the scored-task count for that run.

Conversational retail. We use τ-bench version 1.0.1 with a Qwen3.5-4B user simulator at temperature zero. The 32-task initial set, 34-task additional set, and eight-task preliminary set are mutually disjoint. Tasks with nonempty natural-language judge assertions are excluded. Success is determined by the evaluator's state and communication checks; reference tool actions are not exposed to the agent. Each task is evaluated three times, and success is averaged over trials. Empty responses count as failures. Only infrastructure failures are eligible for retry.

A.3 Computer-use benchmarks

Interactive tasks. WEB-L2 is an internally constructed subset of our mocked web-app environment, rather than a public benchmark or the WorkArena Level-2 split. Following the goal-driven, stateful interaction setting used in web-agent benchmarks (Zhou et al., 2023; Le Sellier de Chezelles et al., 2025), each item specifies an initial app state, a natural-language request, and structured checks on the resulting app state and response. We froze six task-disjoint split files (L2b, L2d, L2e, L2f, L2g, and L2h) before the corresponding evaluations. Each split contains 30 tasks, one from each of the same 30 simulated apps; the six splits therefore provide 180 unique tasks. The "L2" prefix is an internal split name, not a published difficulty tier. Each task is run twice, and we average repeated rewards within each task before averaging across tasks. MOBILEWORLD uses a fixed subset of 117 mobile GUI tasks and measures binary task success. OSWORLD uses 361 desktop tasks, excluding Google Drive tasks, and measures mean task reward. All metrics are presented as percentages. Incomplete evaluations are not extrapolated to full task lists.

Visual grounding. We evaluate MMBench grounding (Wang et al., 2026b), OS-G and OS-G Refine (Xie et al., 2025), ScreenSpot-Pro (Li et al., 2025), ScreenSpot-V2 (Wu et al., 2025), and UI-Vision (Nayak et al., 2025). MMBench grounding follows the evaluation runner's localization protocol, rather than standard multiple-choice MMBench evaluation. The grounding aggregate is the unweighted mean of the six dataset accuracies. It is not pooled-example accuracy. ScreenSpot-V2 shares a source with calibration and therefore contributes development evidence rather than an independent test measurement. The four-domain average assigns equal weight to Web, MobileWorld, OSWorld, and the grounding aggregate.

A.4 Inference and model selection

Evaluation uses independent single-accelerator replicas on H100-80GB devices. MobileWorld and OSWorld use temperature zero, a maximum of 50 actions, and at most 4,096 generated tokens per response. MobileWorld retains three images with a 2,073,600-pixel cap. OSWorld retains five history steps and uses 1920×1080 screenshots, with action coordinates scaled to $[0, 1000]$. Grounding uses the same structured-action runner across models.

Tool-use configuration selection uses the three BFCL development categories. The six post-selection categories do not inform this choice. The balanced computer-use configuration is selected by the four-domain average among complete development evaluations. The computer-use benchmarks therefore serve as development data.

When estimating uncertainty from repeated task observations, we use tasks as bootstrap clusters and retain all repetitions within each cluster. Paired sign-flip tests likewise operate at the task level. These procedures require matched tasks and do not account for model selection or variation between evaluation batches.

Complete evaluation prompt templates are provided in Appendix G.

B Calibration and Consensus Proposals

B.1 Calibration data

Each tool-use specialist contributes eight calibration examples. APIGen and AgentMercury use prompts from the live-parallel tool-call source with their own generated completions. FACET Terminal contributes terminal-command completions. The calibration sets are expert-specific, and all specialist weights remain fixed during sensitivity estimation.

Computer-use calibration uses 151 action-aware pivots spanning visual grounding, pointer actions, keyboard input, and structured Web interactions. Additional pivots are used for corroboration and anchor construction.

B.2 Squared-gradient sensitivity

The model-expectation Fisher is

$$\mathcal{I}(\theta) = \mathbb{E}_x \mathbb{E}_{y \sim p_\theta(\cdot|x)} \left[\nabla_\theta \log p_\theta(y \mid x) \nabla_\theta \log p_\theta(y \mid x)^\top \right]. \tag{10}$$

Our estimator instead uses fixed completions and the squared gradients of averaged sequence losses. It retains only coordinatewise statistics. Squaring a sequence gradient introduces cross-token terms, so the estimate differs from averaging squared token gradients. Token gradients can cancel; consequently, action-masked sensitivity can exceed full-response sensitivity. We use these quantities as sensitivity proxies, not exact curvature estimates (Kunstner et al., 2019).

B.3 Sensitivity and grounding comparisons

Parameter sensitivity measures the local response of a calibration loss to changes in model parameters. The diagonal empirical-Fisher proxy is formed from squared sequence-loss gradients; it is neither parameter magnitude nor an exact Hessian and does not represent interactions between parameters.

Table 3 reports point-in-target accuracy on six grounding benchmarks. The specialist rows provide capability references, whereas the base and merging methods in the lower block use matched task lists and evaluation protocols. AIM reaches 68.79% overall, compared with 67.83% for full-response Fisher. The subgroup results in Table 5 expand all specialist, base, and merging-method rows from the grounding table, together with the additional baselines and anchor refinements in Table 2. Each overall score is the unweighted mean of the six benchmark accuracies; sample counts and protocols are given in Appendix A.3. ScreenSpot-V2 overlaps with calibration sources, so we treat it as development evidence. Figure 3 visualizes the benchmark-level profiles.

Table 5: **Grounding accuracy by benchmark subgroup (%).** Expanded results for the specialists, baselines, and AIM variants in Tables 3 and 2. n denotes subgroup size; Overall equally averages the six benchmark accuracies. The averaging and Fisher baselines follow Wortsman et al. (2022) and Matena & Raffel (2022). OS-G-R denotes OS-G Refine; Adv., Dev., and Sci. denote Advanced, Development tools, and Scientific applications.

	MMBench-G		**OS-G**	**OS-G-R**	**ScreenSpot-Pro**						**ScreenSpot-V2**			**UI-Vision**			
Model	Basic	Adv.	All	All	CAD	Creative	Dev.	Office	OS	Sci.	Desktop	Mobile	Web	Button	Input	Menu	**Overall**
n	1,787	1,807	564	564	261	341	299	230	196	254	334	501	437	3,099	1,703	677	–
Specialist checkpoints																	
MW expert	81.65	71.94	51.06	59.40	44.44	47.80	52.84	63.91	51.53	57.48	90.42	92.81	89.02	18.13	30.77	46.09	59.36
CUA expert	83.38	69.18	49.11	56.03	31.80	38.12	41.81	59.13	47.45	50.79	86.83	92.22	87.19	16.55	30.30	53.77	56.65
Mobile-real expert	79.29	72.05	54.61	60.82	37.55	43.99	47.16	63.04	43.88	55.12	89.82	92.81	90.39	16.42	30.48	47.12	59.15
Web expert	85.51	73.10	54.08	64.01	41.00	49.27	53.18	68.70	52.04	52.76	91.92	93.41	89.93	20.10	33.06	53.47	61.63
DeepSearch expert	60.83	65.30	39.18	56.56	36.78	43.70	47.83	59.57	48.98	56.30	86.83	89.02	85.81	14.00	21.90	37.08	52.30
Grounding expert	93.62	82.24	64.72	70.39	65.52	62.17	68.90	80.43	70.41	61.81	96.41	95.81	93.82	45.37	55.08	70.31	72.90
Base and merging methods (grounding comparison)																	
Qwen3.5-4B base	75.71	71.44	51.42	64.01	52.49	49.85	55.18	68.70	53.57	61.02	91.32	93.21	90.62	17.84	28.60	46.09	60.30
Full-response Fisher	90.60	78.86	60.46	67.73	56.70	56.30	59.53	73.48	60.20	64.57	94.31	94.41	92.45	30.66	44.80	63.22	67.83
Action-masked Fisher	85.11	80.13	61.70	67.38	56.32	56.01	60.87	76.52	63.27	62.99	94.01	95.21	94.05	26.59	41.57	57.90	67.21
AIM (Ours)	89.87	81.13	61.35	68.09	56.70	57.77	62.88	80.87	61.22	64.57	94.91	95.01	93.82	31.17	45.33	65.44	68.79
Additional direct-merge baselines (four-domain comparison)																	
Weighted averaging	86.63	80.85	61.88	67.91	54.79	55.43	60.54	76.52	63.27	61.81	94.31	94.41	93.14	29.04	43.16	60.27	67.68
Action-normalized Fisher (Ours)	86.46	79.58	60.11	68.79	54.02	56.60	61.87	77.39	66.33	62.99	95.21	95.61	94.05	27.40	41.87	59.38	67.52
Anchor refinements (four-domain comparison)																	
Mobile-balanced anchor (Ours)	86.12	76.31	57.27	65.60	52.11	53.96	56.19	72.17	56.63	59.45	94.01	92.42	92.22	22.81	36.11	51.70	64.21
Vision-repaired anchor (Ours)	85.39	75.37	56.91	65.25	51.34	51.91	56.52	73.04	59.18	58.27	94.31	93.41	92.68	22.36	35.94	54.36	64.02
AIM (balanced, Ours)	85.11	75.98	55.50	65.60	52.87	51.03	56.52	71.30	57.65	59.84	94.01	94.01	91.53	22.94	35.47	53.03	63.82

B.4 BINARY CONSENSUS GATE

The tool-use construction uses equal expert priors and a binary gate. Let $B_{i,p} = w_i F^{\mathrm{act}}_{i,p}$ and let $\varepsilon > 0$ be a numerical threshold. For tensor T, define

$$c_{i,T} = \max\left(\varepsilon, \frac{\sum_{p\in T} B_{i,p}}{\max(1, |\{p \in T : B_{i,p} > \varepsilon\}|)}\right), \qquad \widehat{B}_{i,p} = \frac{B_{i,p}}{c_{i,T(p)}}. \tag{11}$$

The gate combines action relevance, cross-expert support, and sign agreement:

$$\begin{aligned} E_p &= \mathbf{1}\left[\sum_i w_i F^{\mathrm{full}}_{i,p} > \varepsilon\right] \mathbf{1}\left[\frac{\sum_i w_i F^{\mathrm{act}}_{i,p}}{\max(\sum_i w_i F^{\mathrm{full}}_{i,p}, \varepsilon)} \geq \gamma\right], \\ U_p &= \mathbf{1}\left[\min_i \widehat{B}_{i,p} \geq \rho \max_i \widehat{B}_{i,p}\right], \qquad g_p = E_p U_p D_p. \end{aligned} \tag{12}$$

Here $D_p = 1$ when all experts with $B_{i,p} > \varepsilon$ have nonnegative updates or all have nonpositive updates. The implementation settings for γ and ρ are reported in Table 6. The consensus target is

$$\Delta^{\mathrm{agree}}_p = \mathrm{clip}\left(\sum_i \Delta_{i,p}, -\max_i |\Delta_{i,p}|, \max_i |\Delta_{i,p}|\right). \tag{13}$$

Clipping limits the combined update to the largest individual magnitude. The fallback is raw action-Fisher averaging, defined below.

B.5 PAIRED NORMALIZATION AND CONTINUOUS CONSENSUS

The computer-use construction normalizes both sensitivity arrays by the same full-response tensor scale:

$$Z_{i,T} = \begin{cases} \max\left(\varepsilon, \dfrac{\sum_{p\in T} F^{\mathrm{full}}_{i,p}}{|\{p \in T : F^{\mathrm{full}}_{i,p} > \varepsilon\}|}\right), & \text{if the count is nonzero,} \\ 1, & \text{otherwise.} \end{cases} \tag{14}$$

The numerator includes all entries in the tensor. Nonfinite and negative importance values are sanitized before use. With $A_{i,p} = w_i a_{i,p}$ and $H_{i,p} = w_i h_{i,p}$, define

$$\begin{aligned} R_p &= \frac{\sum_i A_{i,p}}{\sum_i A_{i,p} + \beta \sum_i H_{i,p} + \varepsilon}, \\ Q_p &= \frac{(\sum_i A_{i,p})^2}{\max(N \sum_i A^2_{i,p}, \varepsilon)}, \\ C_p &= \frac{\left|\sum_{\Delta_{i,p}\geq 0} A_{i,p} - \sum_{\Delta_{i,p}<0} A_{i,p}\right|}{\max(\sum_i A_{i,p}, \varepsilon)}. \end{aligned} \tag{15}$$

The terms measure action relevance, breadth of expert support, and weighted sign agreement, respectively. We set $R_p = C_p = 0$ when total action evidence is at most ε, and $Q_p = 0$ when the sum of squared evidence is at most ε. Each term is clipped to $[0, 1]$.

Let s_p be the sign with greater action-evidence mass, with ties assigned to the nonnegative sign. Then

$$\Delta^{\mathrm{agree}}_p = \frac{\sum_{i:\Delta_{i,p}s_p\geq 0} A_{i,p}\Delta_{i,p}}{\sum_{i:\Delta_{i,p}s_p\geq 0} A_{i,p}}, \qquad g_p = \frac{R_p Q_p C_p}{R_p Q_p C_p + \lambda}. \tag{16}$$

Zero updates participate in the nonnegative sign vote. If the target denominator is below the numerical threshold, we use linear averaging. The values of β and λ are reported in Table 6. The proposal is applied to language-layer tensors, retaining linear averaging elsewhere.

B.6 FISHER-WEIGHTED REFERENCES

For nonnegative importance weights $I_{i,p}$, define

$$\theta_p^{\mathrm{Fisher}} = \theta_{0,p} + \frac{\sum_i I_{i,p}\Delta_{i,p}}{\sum_i I_{i,p}}. \tag{17}$$

Entries with a denominator below ε retain the base value. Full-response Fisher uses $I_{i,p} = w_i F_{i,p}^{\mathrm{full}}$; raw action-Fisher uses $I_{i,p} = w_i F_{i,p}^{\mathrm{act}}$. The computer-use action-normalized reference first forms $w_i F_{i,p}^{\mathrm{act}}$, then divides each expert tensor by the mean of its entries above ε. Tensors with no qualifying entries remain unnormalized. This normalization can cancel a scalar expert prior, apart from threshold effects. The full-response and action-normalized references therefore differ in both loss masking and normalization.

C SURROGATE BUDGETS AND INTERPOLATION

For a fixed anchor θ^A, the diagonal surrogate is

$$\widetilde{L}_i(\vartheta;\theta^A) = \frac{1}{2}\sum_p a_{i,p}(\vartheta_p - \theta_{i,p})^2 + \frac{\kappa}{2}\sum_p h_{i,p}(\vartheta_p - \theta_p^A)^2. \tag{18}$$

The first term weights distance to expert i by action sensitivity; the second weights distance to the anchor by full-response sensitivity. For disjoint blocks and binary selection variables z_b,

$$\widetilde{L}_i(\theta^A;\theta^A) - \widetilde{L}_i\left(\theta^A + \sum_b z_b u_b;\theta^A\right) = \sum_b z_b G_{i,b}. \tag{19}$$

This identity follows by expanding the quadratic and using $z_b^2 = z_b$. It establishes the additive gains used in block selection.

Interpolation of a feasible update. Let $v = \sum_b z_b u_b$ be a fixed selected update and define $D_i(t) = \widetilde{L}_i(\theta^A;\theta^A) - \widetilde{L}_i(\theta^A + tv;\theta^A)$. If $D_i(1) \geq -\epsilon_i$, then for $0 \leq t \leq 1$,

$$D_i(t) = tD_i(1) + \frac{t(1-t)}{2}\sum_p (a_{i,p} + \kappa h_{i,p})v_p^2 \geq -t\epsilon_i. \tag{20}$$

Thus, reducing the update magnitude preserves each expert's budget when the weights are nonnegative and the anchor and update direction remain fixed. The result holds before numerical rounding. Changing the selected blocks or interpolation target requires a new budget check. These bounds concern the surrogate, not task success.

D CHECKPOINT CONSTRUCTION

For clarity, write the update operator as

$$\mathcal{R}(\theta^A, \theta^P; S, \alpha) = \mathcal{Q}\left[\theta^A + \alpha P_S(\theta^P - \theta^A)\right], \tag{21}$$

where P_S retains entries in S and $\mathcal{Q}$ stores the result in BF16. Each interpolation is stored before the next construction step.

D.1 TOOL-USE CONFIGURATION

The tool-use checkpoint is

$$\theta_{\mathrm{tool}} = \mathcal{R}(\theta_{\mathrm{lin}}, \theta^P; S_{4:7}, \alpha_{\mathrm{tool}}), \tag{22}$$

where θ_{lin} is the equal-weight linear merge and $S_{4:7}$ contains all tensors in language layers 4–7, indexed from zero. All other parameters retain their linear-merge values. We select the layer group and interpolation coefficient on the three BFCL development categories, subject to a per-category tolerance δ_{tool} relative to linear averaging; implementation values are summarized in Table 6. This uses a development-performance constraint rather than a surrogate budget. Missing auxiliary prediction tensors contribute zero task updates; unmapped extra tensors are excluded.

D.2 COMPUTER-USE CONSENSUS CHECKPOINT

The continuous proposal is interpolated into selected feed-forward groups. These groups comprise the feed-forward modules within language layers. For group g, the interpolation strength is denoted by α_g; groups not selected by the block criterion have $\alpha_g = 0$. Block selection is followed by corroboration with a second calibration view or reference-anchor calibration:

$$S_{\text{keep}} = S_{\text{primary}} \cap (S_{\text{second}} \cup S_{\text{reference}}) . \tag{23}$$

This operation filters support without changing candidate target values. The second view is not assumed statistically independent. We denote the resulting checkpoint by $\theta_{\text{consensus}}$. Support filtering can remove compensating gains, so it does not automatically preserve the original selection budgets.

D.3 BALANCED CHECKPOINT

The balanced construction separates two decisions: forming an anchor that combines complementary capabilities, and refining that anchor through module-specific updates. This separates the choice of transferred knowledge from the location and strength of its application.

Capability-preserving anchor. Starting from a source specialist, we select blocks of the consensus update using the expert-wise gains $G_{i,b}$. For each protected expert, we scale the admissible surrogate loss increase by the cost of applying the complete candidate update:

$$\epsilon_i = \rho_i \max\left(0, -\sum_b G_{i,b}\right), \tag{24}$$

where $\rho_i \in [0, 1]$ controls the loss tolerance for expert i. A smaller value imposes a tighter preservation constraint. The selection objective rewards gains on complementary domains, while the constraints control deviation from the source specialist. We then construct the anchor by convex interpolation among the source, the consensus checkpoint, and the block-selected checkpoint.

Module-specific refinement. Given an anchor θ^A and a reference merge θ^R, visual and language refinements take the form

$$\begin{aligned} \theta^V &= \mathcal{R}(\theta^A, \theta^R; S_V, \alpha_V), \\ \theta^M &= \mathcal{R}(\theta^V, \theta^R; S_L, \alpha_L). \end{aligned} \tag{25}$$

The sets S_V and S_L specify where the update is applied, while α_V and α_L control its magnitude. Parameters outside each selected module remain unchanged at that step. These interpolation strengths are selected through development performance after constructing the anchor.

Implementation. We use the MobileWorld specialist as the source and select blocks of $m = 262{,}144$ entries toward $\theta_{\text{consensus}}$. The protected experts are MobileWorld, desktop, grounding, and Web; the utility weights desktop, grounding, and Web equally and assigns MobileWorld zero weight, retaining it through its constraint. For anchor construction, we first interpolate the consensus checkpoint toward the source with coefficient λ_1, interpolate the block-selected checkpoint toward this mixture with λ_2, and interpolate the result toward consensus with λ_3. We use the full-response Fisher merge as θ^R. The visual subset covers the vision module; the language subset covers the language backbone, including its embeddings and final normalization but excluding the separately stored output head. The final strengths are $\alpha_V = 10.00\%$ and $\alpha_L = 2.50\%$; Table 6 summarizes the settings. The refinement ablation in Table 4 shows how these two module-specific updates contribute to cross-domain performance.

E BASELINE CONFIGURATIONS

Uniform averaging assigns equal weights to all specialists. Weighted averaging uses the priors defined in Section A. Full-response and action-normalized Fisher references follow Section B.6. The fixed representative configurations use task-arithmetic scale $\lambda = 0.30$, TIES-Merging density 0.20, and DARE drop probability 0.90 applied to the prior-weighted merge. Magnitude trimming without sign election serves as a separate diagnostic control. These are fixed recipes rather than configurations obtained through equal-budget hyperparameter optimization.

Table 6: Symbols used for the implementation settings of the reported checkpoints. The definitions are shared across configurations; the numerical values are those used in our experiments, rounded to two decimal places.

Stage	Symbol	Experimental setting
Tool-use update	$\alpha_{\text{tool}}, \delta_{\text{tool}}$	0.66; 0.50 percentage points
Tool-use consensus gate	γ, ρ	$0.25, 0.20$
Paired sensitivity gate	β, λ	$1.00, 1.00$
Consensus proposal	α_g, m	Layers 0–3, 8–11, 12–15, 16–19, 24–27, 28–31; $\alpha_g = (0.27, 0.29, 0.30, 0.28, 0.26, 0.32)$; $m = 262{,}144$
Protected anchor	$\rho_{\text{MW}}, \lambda_{1:3}$	0.05; $(0.60, 0.50, 0.30)$
Final refinement	α_V, α_L	$10.00\%, 2.50\%$

Orthogonal-residual control. For tensor T, define $r_T = \theta_{\text{action},T} - \theta_{\text{vision},T}$ and $d_T = \theta_{\text{vision},T} - \theta_{\text{lin},T}$. The control removes the projection of r_T onto d_T:

$$r_T^{\perp} = r_T - \frac{\langle r_T, d_T \rangle}{\|d_T\|_2^2} d_T, \qquad \theta'_T = \mathcal{Q}\big[\theta_{\text{vision},T} + 0.03 r_T^{\perp}\big] . \tag{26}$$

If $\|d_T\|_2^2 \leq 10^{-12}$, we set $r_T^{\perp} = r_T$. Orthogonality holds before rounding and does not guarantee preservation of task performance.

F QUALITATIVE ACTION-ATTENTION ANALYSIS

We visualize how AIM (balanced) aligns visual attention and action prediction with the requested interface element, comparing it with weighted averaging. The mobile example in Figure 4 shows this distinction: AIM (balanced) selects *Devices*, while weighted averaging selects the adjacent *Me* control. The examples below examine desktop interfaces and resolve the attention changes across language layers.

Visualization protocol. Both checkpoints receive identical tokenized prompts, image tensors, and an action prefix, using the balanced model's processor and chat template. We extract attention at the query predicting the first action-coordinate token, before any reference coordinate is visible, and average the 16 heads in each of the eight full-attention layers (indices 3, 7, 11, 15, 19, 23, 27, and 31). Linear-attention layers are excluded from these softmax maps. Clicks are generated independently from the user prompt using greedy decoding, with thinking disabled and a 160-token limit; the image budget is 2,097,152 pixels.

Layerwise localization. In the Quartus example (Figure 5), both models allocate little attention to the target in the earliest full-attention layers. AIM (balanced) develops substantially stronger target localization in layers 19–27: target-region attention at layer 19 is 15.13% for AIM (balanced) versus 2.02% for weighted averaging; at layer 23, it is 23.39% versus 0.93%. This concentrated mid-to-late-layer attention accompanies the correct File-menu click, while weighted averaging selects a neighboring control.

Additional action corrections. Figure 6 shows three further tasks on which weighted averaging clicks outside the annotated target while AIM (balanced) clicks inside it. The examples cover opening a menu in Quartus, closing a sequence in Premiere, and choosing a file-view control in an OSWorld file dialog. In Quartus, target attention increases from 1.7% to 9.5% when averaged across layers. In Premiere, AIM (balanced) selects the sequence's close control rather than the neighboring tab region; in the file dialog, it selects the requested grid-view control rather than the adjacent view button. Together, these examples demonstrate more precise action selection across distinct desktop interfaces, complementing the layerwise localization pattern in Quartus.

Map interpretation. The detail maps are normalized over image tokens relative to uniform allocation, with a shared scale for each model pair. Difference maps show AIM (balanced) minus weighted averaging using symmetric red/blue limits. Display limits use the 99.5th percentile; target mass is computed from unclipped maps with fractional patch overlap. Target mass denotes the share of image-token attention inside the target, and visual mass the share of all attention assigned to image tokens. Cropping does not change either quantity. Green boxes mark targets and magenta crosses mark clicks.

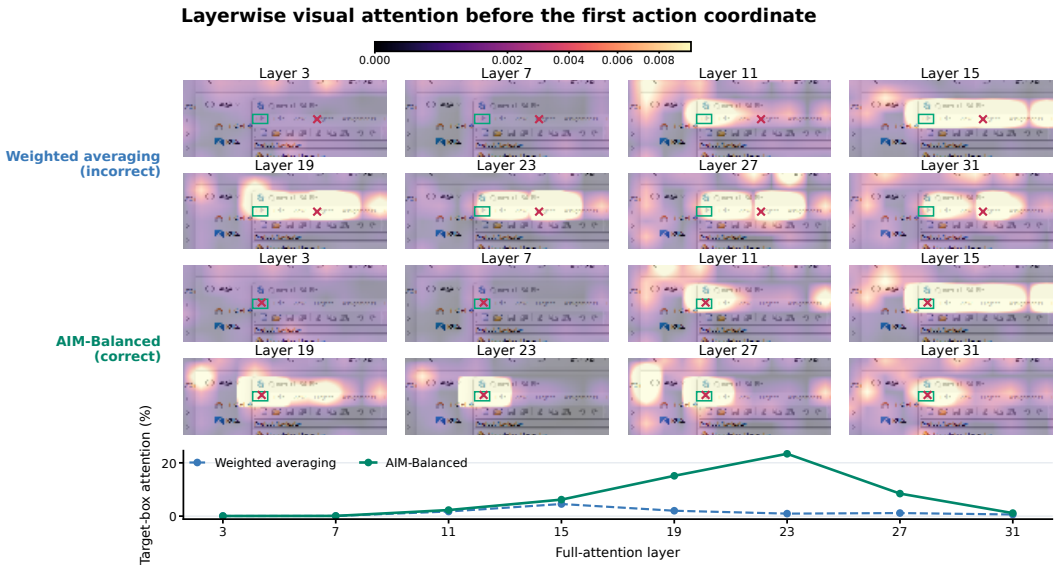

Figure 5: **Layerwise attention in a Quartus grounding example.** Tiles show head-averaged image-token attention under matched inputs; the lower plot reports target-region attention at each full-attention layer. All tiles share a color scale. Green boxes mark the target and magenta crosses mark the independently generated clicks. AIM (balanced) places more attention on the target in layers 19–27 and clicks inside it, whereas weighted averaging clicks outside.



Figure 6: **Examples where AIM corrects actions missed by weighted averaging.** From left to right, each row shows the target crop, weighted-averaging attention and click, AIM (balanced) attention and click, and the difference (AIM (balanced) minus weighted averaging). In all three tasks, the weighted-averaging click falls outside the target, whereas the AIM (balanced) click falls inside it. Green boxes mark targets, magenta crosses mark generated clicks, and red/blue denote increased/decreased attention. Heatmap scaling and mass definitions follow Figure 4.

G EVALUATION PROMPTS

We present the evaluation prompts, including their action interfaces and output instructions.

G.1 BFCL

The system message supplies the function definitions and the function-call format. User requests and subsequent tool responses follow in conversation order.

```
# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML
tags:
<tools>
{{ function_definitions }}
</tools>

For each function call, return a json object with function name and
arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
```

Figure 7: **BFCL system prompt.** Function definitions and the required tool-call format.

G.2 TERMINAL-BENCH 2.1

The Terminus-2 agent places the following instructions in the initial user message; it does not supply a separate system message. Subsequent messages contain command outputs and terminal observations.

```
You are an AI assistant tasked with solving command-line tasks in a
Linux environment. You will be given a task description and the output
from previously executed commands. Your goal is to solve the task by
providing batches of shell commands.

Format your response as JSON with the following structure:

{
  "analysis": "Analyze the current state based on the terminal output
provided. What do you see? What has been accomplished? What still needs
 to be done?",
  "plan": "Describe your plan for the next steps. What commands will
you run and why? Be specific about what you expect each command to
accomplish.",
  "commands": [
    {
      "keystrokes": "ls -la\n",
      "duration": 0.1
    },
    {
      "keystrokes": "cd project\n",
      "duration": 0.1
    }
  ],
  "task_complete": true
}
```

```
Required fields:
- "analysis": Your analysis of the current situation
- "plan": Your plan for the next steps
- "commands": Array of command objects to execute

Optional fields:
- "task_complete": Boolean indicating if the task is complete (defaults
 to false if not present)

Command object structure:
- "keystrokes": String containing the exact keystrokes to send to the
terminal (required)
- "duration": Number of seconds to wait for the command to complete
before the next command will be executed (defaults to 1.0 if not
present)

IMPORTANT: The text inside "keystrokes" will be used completely
verbatim as keystrokes. Write commands exactly as you want them sent to
 the terminal:
- You must end every command with a newline (\n) or it will not execute
.
- For special key sequences, use tmux-style escape sequences:
  - C-c for Ctrl+C
  - C-d for Ctrl+D

The "duration" attribute specifies the number of seconds to wait for
the command to complete (default: 1.0) before the next command will be
executed. On immediate tasks (e.g., cd, ls, echo, cat) set a duration
of 0.1 seconds. On commands (e.g., gcc, find, rustc) set a duration of
1.0 seconds. On slow commands (e.g., make, python3 [long running script
], wget [file]) set an appropriate duration as you determine necessary.

It is better to set a smaller duration than a longer duration. It is
always possible to wait again if the prior output has not finished, by
running {"keystrokes": "", "duration": 10.0} on subsequent requests to
wait longer. Never wait longer than 60 seconds; prefer to poll to see
intermediate result status.

Important notes:
- Each command's keystrokes are sent exactly as written to the terminal
- Do not include extra whitespace before or after the keystrokes unless
 it's part of the intended command
- Extra text before or after the JSON will generate warnings but be
tolerated
- The JSON must be valid - use proper escaping for quotes and special
characters within strings
- Commands array can be empty if you want to wait without taking action

Task Description:
{instruction}

Current terminal state:
{terminal_state}
```

Figure 8: **Terminal-Bench 2.1 initial user prompt.** Instructions for command execution and task completion.

G.3 WEB-L2

The web environment is operated through the `mobile_use` interface. The user message supplies the task goal and the current screenshot.

```
You are a multi-purpose intelligent assistant. Based on my requests,
you can use tools to help me complete various tasks.

# Tools

You have access to the following functions:

<tools>
{"type": "function", "function": {"name": "mobile_use", "description":
"Use a touchscreen to interact with a mobile device, and take
screenshots.\n* This is an interface to a mobile device with
touchscreen. You can perform actions like clicking, typing, swiping,
etc.\n* Some applications may take time to start or process actions, so
 you may need to wait and take successive screenshots to see the
results of your actions.\n* The screen's resolution is 1000x1000.\n*
Make sure to click any buttons, links, icons, etc with the cursor tip
in the center of the element. Don't click boxes on their edges unless
asked.", "parameters": {"properties": {"action": {"description": "The
action to perform. The available actions are:\n* `key`: Perform a key
event on the mobile device.\n    - This supports adb's `keyevent`
syntax.\n    - Examples: \"volume_up\", \"volume_down\", \"power\", \"
camera\", \"clear\".\n* `click`: Click the point on the screen with
coordinate (x, y).\n* `long_press`: Press the point on the screen with
coordinate (x, y) for specified seconds.\n* `swipe`: Swipe from the
starting point with coordinate (x, y) to the end point with
coordinates2 (x2, y2).\n* `type`: Input the specified text into the
activated input box.\n* `system_button`: Press the system button.\n* `
open`: Open an app on the device.\n* `wait`: Wait specified seconds for
 the change to happen.\n* `terminate`: Terminate the current task and
report its completion status.", "enum": ["key", "click", "long_press",
"swipe", "type", "system_button", "open", "wait", "terminate"], "type":
 "string"}, "coordinate": {"description": "(x, y): The x (pixels from
the left edge) and y (pixels from the top edge) coordinates to move the
 mouse to. Required only by `action=click`, `action=long_press`, and `
action=swipe`.", "type": "array"}, "coordinate2": {"description": "(x,
y): The x (pixels from the left edge) and y (pixels from the top edge)
coordinates to move the mouse to. Required only by `action=swipe`.", "
type": "array"}, "text": {"description": "Required only by `action=key
`, `action=type`, and `action=open`.", "type": "string"}, "time": {"
description": "The seconds to wait. Required only by `action=long_press
` and `action=wait`.", "type": "integer"}, "button": {"description": "
Back means returning to the previous interface, Home means returning to
 the desktop, Menu means opening the application background menu, and
Enter means pressing the enter. Required only by `action=system_button
`", "enum": ["Back", "Home", "Menu", "Enter"], "type": "string"}, "
status": {"description": "The status of the task. Required only by `
action=terminate`.", "type": "string", "enum": ["success", "failure
"]}}, "required": ["action"], "type": "object"}}}
</tools>

If you choose to call a function ONLY reply in the following format:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>
```

```
<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format: an inner <function
=...></function> block must be nested within <tool_call></tool_call>
XML tags
- Required parameters MUST be specified
- You may provide optional reasoning for your function call in natural
language BEFORE the function call, but NOT after
- If there is no function call available, answer the question like
normal with your current knowledge and do not tell the user about
function calls
</IMPORTANT>

# Response format

Response format for every step:
1) Action: a short imperative describing what to do in the UI.
2) A single <tool_call>...</tool_call> block.

Rules:
- Output exactly in the order: Action, <tool_call>.
- Be brief: one for Action.
- Do not output anything else outside those three parts.
- If finishing, use action=terminate in the tool call.
```

Figure 9: **Web-L2 system prompt.** The interaction interface and action-response format.

G.4 MOBILEWORLD

The agent receives the task instruction, screenshots, and retained action history, and returns the next action using the following interface.

```
You are a GUI agent. Given a task, action history, and screenshots,
perform the next action to complete the task.

## Output Format
<think>...</think>
<tool_call>
{"name": "mobile_use", "arguments": <args-json-object>}
</tool_call>

## Action Space (arguments)
- click / double_click / long_press: {"action_type": "...", "coordinate
": [x, y]}
- type / open / ask_user: {"action_type": "...", "text": "..."}
- drag: {"action_type": "drag", "start_coordinate": [x1, y1], "
end_coordinate": [x2, y2]}
- system_button: {"action_type": "system_button", "button": "
button_name"}
- wait: {"action_type": "wait", "waiting_time": "xxx"}
- terminate: {"action_type": "terminate", "answer": "xxx", "status": "
success or failed"}

Available apps: {{ apps }}. Prefer the open action to launch apps.
Follow the action schema and return valid JSON inside <tool_call> tags.
```

Figure 10: **MobileWorld system prompt.** Mobile actions and the structured response format.

G.5 OSWORLD

The GUI instructions specify screenshot-based interaction, normalized coordinates, and action formatting.

```
You are a computer agent. Given a task instruction and the current
screenshot, perform actions to complete the task.

## GUI Actions
- pyautogui.click(x, y), rightClick(x, y), doubleClick(x, y),
tripleClick(x, y): click at (x, y).
- pyautogui.moveTo(x, y): move the cursor.
- pyautogui.dragTo(x, y, duration=0.5): drag to (x, y).
- pyautogui.typewrite(text): type ASCII text.
- pyautogui.press(key): press a key.
- pyautogui.keyDown(key), keyUp(key): hold or release a key.
- pyautogui.hotkey(key1, key2, ...): press a key combination.
- pyautogui.scroll(clicks, x, y): scroll up (positive) or down (
negative).

## Coordinates and Observations
Use normalized coordinates in [0, 1000] on both axes, not original
image pixels. (500, 500) is the screen center. After each action,
observe the updated screenshot.

## Response Format
Wrap each action in its own <tool_call> block. Actions execute in order
; batch actions only when they do not depend on a new observation.

<tool_call>pyautogui.click(835, 420)</tool_call>

## Completion
Use computer.wait() to wait for 20 seconds. Verify the requested result
 before ending with computer.terminate(status="success") or computer.
terminate(status="failure").
```

Figure 11: **OSWorld system prompt.** GUI actions, coordinate conventions, and task completion.

G.6 VISUAL GROUNDING

MMBench grounding, OS-G, OS-G Refine, ScreenSpot-Pro, ScreenSpot-V2, and UI-Vision share the following system prompt. Each user message contains the screenshot and the dataset's localization instruction.

```
You are a helpful assistant.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML
tags:
<tools>
```

```
{"type": "function", "function": {"name": "grounding", "description": "
Localize the UI element followed by the user's instruction\n* The
screen's resolution is 1000x1000.\n* Whenever you intend to move the
cursor to click on an element like an icon, you should consult a
screenshot to determine the coordinates of the element before moving
the cursor.\n* Make sure to click any buttons, links, icons, etc with
the cursor tip in the center of the element. Don't click boxes on their
 edges.", "parameters": {"properties": {"action": {"description": "The
action to perform. The available actions are:\n* `click`: Click the UI
element at a specified (x, y) pixel coordinate on the screen.", "enum":
 ["click"], "type": "string"}, "coordinate": {"description": "(x, y):
The x (pixels from the left edge) and y (pixels from the top edge)
coordinates to move the mouse to.", "type": "array"}}, "required": ["
action"], "type": "object"}}}
</tools>

For each function call, return a json object with function name and
arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

# Image
```

Figure 12: **Visual-grounding system prompt.** The shared localization interface used for all six grounding benchmarks.