CROSS-WORKFLOW LATENT COMMUNICATION FOR HETEROGENEOUS AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

We study whether one set of latent communication interfaces can support heterogeneous agents in different agent workflows. Both workflows, all six model assignments, and a shared instruction pool are included in training and evaluation on disjoint tasks. We introduce RECAST (Reusable Cross-Agent State Transfer). A Latent Transition Adapter generates continuous inputs within each agent, and a directed Cross-Model Bridge transfers every generated intermediate-layer state to the next agent. Training combines multi-role warm-up, joint task and intermediate-readout supervision, and continuous-action group-relative policy optimization. Gaussian latent exploration is confined to reinforcement learning; deployment follows the deterministic mean path. Across six benchmarks, the reported average scores are 33.9% and 34.6% for the two workflows, exceeding the strongest aggregate comparator by 2.0 and 2.1 percentage points. Removing warm-up reduces scores by 3.4 and 3.8 points; disabling communication reduces them by 3.9 and 4.2 points. These comparisons support shared latent interfaces within the trained configuration support.

1 INTRODUCTION

Language-model agents can divide a problem into planning, solving, checking, and revision, exchanging intermediate results before producing a final answer. Frameworks for multi-agent collaboration express these roles through instructions and coordinate their execution through a workflow (Hong et al., 2024; Wu et al., 2024). A workflow determines which model performs each operation, what information it receives, and where its output is used next. Heterogeneous systems additionally assign these operations to models with different architectures and capabilities. Communication must therefore connect both different representations and different uses of the transmitted information: a candidate solution may become the input to a checker in one workflow and to a reviser in another.

Communication mechanisms differ in the representation exchanged between agents. Text-mediated systems transmit explicit messages that the next model reads as language. Latent communication instead transfers continuous representations; Interlat, LatentMAS, and RecursiveMAS investigate how agents can exchange or continue internal computation (Du et al., 2026; Zou et al., 2026a;b). Text provides a shared symbolic format, whereas latent communication introduces an interface between the sender's computation and the receiver's input representation. This motivates a question beyond performance in an individual collaboration sequence: *can the same learned interfaces support changes in how the participating agents are organized?*

The two workflows place different demands on a communication interface. In plan–verify–solve, the verifier checks a proposed plan and the terminal agent solves the task. In solve–verify–revise, the verifier checks a candidate solution and the terminal agent returns a revised answer. Swapping the models assigned to these roles changes which representations each interface must connect. Sharing an interface across these uses could reduce the need for separate workflow-specific modules, but its usefulness must be measured against text communication and existing latent interfaces under comparable training conditions.

We study cross-workflow communication as sharing one set of interface parameters across these two jointly trained workflows. Each has exactly three nodes, with each of the three models used once. Training, development, and evaluation use the same support of workflow types, model assignments, and instruction variants; task instances and their derivatives are disjoint. Each execution generates its

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.

own states using the same weights. The reusable objects are the *interface parameters*, rather than one fixed message for all tasks. Our evaluation measures performance within this declared configuration support.

We introduce RECAST, a framework for *Reusable Cross-Agent State Transfer*. A *Latent Transition Adapter* (LTA) maps an agent's last-layer state to its next continuous input, enabling several steps of latent recurrence. A directed *Cross-Model Bridge* (CMB) maps the intermediate-layer state at each generated position into the next model's input space. Every generated position is retained in order. One LTA is shared across all roles of a model, and one CMB across all workflows using a directed model pair. Intermediate agents communicate without decoding a textual message.

Training begins with independent, multi-role LTA warm-up and proceeds to joint optimization over both workflows. Warm-up supervises textual readout after deterministic latent recurrence. Joint supervision combines final task targets with intermediate readouts of verifiable facts, constraints, and role outputs, updating LTAs and CMBs while varying model assignments and instructions within the declared pool. Readout text is confined to training branches and never becomes an inter-agent message. Only during reinforcement learning (RL), an *Exploration Scale Head* (ESH) enables conditional Gaussian latent sampling. A continuous-action adaptation of group-relative policy optimization (GRPO) uses complete collaboration outcomes to update the interfaces. Supervised replay and deployment retain the deterministic mean path, so exploration is assessed by deterministic deployment performance.

Our contributions are threefold: (1) a shared interface design for heterogeneous latent communication across two three-agent workflows; (2) a training procedure combining multi-role warm-up, joint task and intermediate-readout supervision, and RL exploration; and (3) a comparison across six benchmarks, seven training ablations, and message-use interventions under a declared model-assignment and instruction protocol. Both workflows use one fixed interface checkpoint at evaluation. RECAST attains the highest reported score in 11 of 12 benchmark–workflow conditions, with average scores of 33.9% and 34.6%.

2 RELATED WORK

Agent collaboration and latent interfaces. AutoGen and MetaGPT organize agents through conversations, roles, and structured procedures (Hong et al., 2024; Wu et al., 2024). Coconut develops continuous hidden-state reasoning, while SoftCoT maps soft thoughts into a target model (Hao et al., 2025; Xu et al., 2025). Interlat studies hidden-state communication with a learned compression extension; LatentMAS combines latent generation with shared working memory (Du et al., 2026; Zou et al., 2026b). RecursiveMAS introduces inner and outer links and evaluates multiple collaboration patterns (Zou et al., 2026a). DiffMAS jointly optimizes latent collaboration (Yu et al., 2026). Cache-to-Cache transfers KV information, and StateBridge aligns hidden states with a training-free mapping (Fu et al., 2026; Peng et al., 2026). These mechanisms provide architectural context for RECAST. We investigate shared interfaces trained on both selected workflows, with comparable assignment and instruction mixing for trainable baselines.

Reuse and evidence of communication. Portable Semantics studies packet reuse and negative transfer in a small, shared-backbone setting (Marincat, 2026). A causal audit of relayed caches motivates distinguishing example-specific message use from gains due to message presence (Cheng et al., 2026). We target heterogeneous models connected through known directed interfaces and apply matched-message controls and private-fact interventions. Neither intermediate-layer superiority nor lower communication cost follows from transmitting latent states; both require measurement.

Reinforcement learning. PPO optimizes clipped policy-ratio surrogates, and GRPO uses relative rewards within sampled groups (Schulman et al., 2017; Shao et al., 2024). ReMA studies outcome-guided multi-agent reasoning; PRIME learns implicit process rewards from outcome supervision (Cui et al., 2026; Wan et al., 2025). RECAST treats continuous input embeddings as RL actions alongside final output tokens. Their Gaussian densities provide policy likelihoods, not quality labels for opaque messages. We use established policy-gradient machinery and test this continuous-action adaptation against deterministic supervision and fixed-variance exploration.

3 PROBLEM: REUSING INTERFACES ACROSS WORKFLOWS

Let $\{F_i\}_{i=1}^{m}$ be frozen decoder language models, with widths d_i and depths L_i. A sequential workflow is

$$w = \left[(b_v, r_v, \tau_v, \mathcal{I}_v, K_v)\right]_{v=1}^{n}, \tag{1}$$

where node v uses model b_v, role r_v, instruction template τ_v, permitted information $\mathcal{I}_v$, and K_v latent steps. We fix $m = n = 3$ and use each model once per execution. The role sequence is either plan–verify–solve (PVS) or solve–verify–revise (SVR). The verifier checks a plan in PVS and a candidate solution in SVR. Its outgoing state must carry the relevant proposal and checks for the terminal solver or reviser, which emits the complete task output Y. Intermediate agents emit only latent messages. The verifier-role agent is distinct from the external outcome checker that accesses reference answers or private tests.

For task x, let $c = (x, w)$ denote a fixed execution configuration, with node-level information access enforced separately. Both workflow types, all six model permutations, and a finite manifest of instruction variants are covered during joint training and evaluation. The manifest is fixed before training, and every declared configuration receives training coverage. Evaluation uses disjoint task instances without parameter updates. There is no configuration holdout or depth-extrapolation experiment.

Let $f_{\Theta}^{\text{mean}}(c)$ be deployment with deterministic latent transitions and greedy text decoding. The target quantity is

$$J_s(\Theta) = \mathbb{E}_{(x,w)\sim\mathcal{D}_s}\left[S_x(f_{\Theta}^{\text{mean}}(x, w))\right], \tag{2}$$

for each benchmark–workflow condition s, where S_x is the benchmark score computed only for evaluation. Scores average equally over the declared model assignments and instruction variants. The same Θ is used in both workflows, and comparisons retain the same terminal backbone in each configuration.

4 RECAST

Figure 1 illustrates the recurring unit of RECAST: an agent develops a latent trajectory, and a directed bridge makes all of its generated intermediate-layer states available to the next agent. Applying this unit along either three-node workflow separates each model's internal recurrence from communication across models. Roles and instructions change the computation while reusing the same model-specific LTAs and pair-specific CMBs.

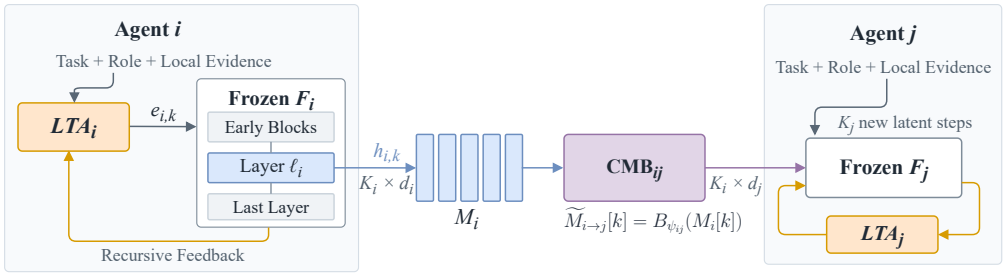


Figure 1: Latent recurrence and communication in RECAST. The sender's LTA feeds its last-layer state back as the next continuous input. States from layer ℓ_i at all K_i generated positions form M_i; the CMB maps them positionwise into the receiver's embedding space, retaining their order and count. The receiver conditions on this message and its permitted task context before generating its own latent steps. The diagram shows one directed edge of a three-agent workflow; backbone weights remain frozen.

4.1 Latent Recurrence and Complete State Transfer

The interface builds on continuous-state recurrence in Coconut and the inner/outer-link decomposition in RecursiveMAS (Hao et al., 2025; Zou et al., 2026a). Learned mappings between one model's soft states and another model's inputs also appear in SoftCoT and Interlat (Du et al., 2026; Xu et al., 2025). We adapt these established mechanisms to shared interfaces jointly trained on PVS and SVR.

Each model has one LTA mean map $\mu_{\theta_i} : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$. Each trained directed pair has one CMB $B_{\psi_{ij}} : \mathbb{R}^{d_i} \to \mathbb{R}^{d_j}$. These modules are shared across roles, templates, and invocation positions. Backbone weights, token embeddings, and language-model heads remain frozen. The LTA and CMB use normalized residual projections; implementation architectures are specified in Appendix E.

Node v prefills its native chat template with the role instruction, permitted task evidence, incoming projected message, and a task suffix. Let $i = b_v$ and $h^{L_i}_{v,0}$ be the final valid prefill position. This seed is not a generated latent position. At step k, let $\mu_{v,k} = \mu_{\theta_i}(h^{L_i}_{v,k-1})$ and $\epsilon_{v,k} \sim \mathcal{N}(0, I)$. The LTA constructs a continuous input and the frozen backbone processes it:

$$e_{v,k} = \begin{cases} \mu_{v,k}, & \text{deterministic execution,} \\ \mu_{v,k} + \sigma_{v,k} \odot \epsilon_{v,k}, & \text{RL rollout,} \end{cases} \tag{3}$$
$$(h^1_{v,k}, \ldots, h^{L_i}_{v,k}, \mathcal{C}_{v,k}) = F_i(e_{v,k}; \mathcal{C}_{v,k-1}),$$

where $\mathcal{C}$ denotes the attention cache. Each sampled input changes the subsequent computation, so an RL rollout samples a complete trajectory. The conditional Gaussian at one step does not make the full trajectory a single Gaussian.

After K_v steps, the outgoing message contains the state at a fixed intermediate layer ℓ_i for *every actual latent position*:

$$M_v = [h^{\ell_i}_{v,1}, \ldots, h^{\ell_i}_{v,K_v}], \qquad \widetilde{M}_{v \to v+1}[k] = B_{\psi_{ij}}(M_v[k]), \quad j = b_{v+1}. \tag{4}$$

The receiver appends these K_v projected vectors as continuous input positions and generates its own latent trajectory. There is no pooling, position selection, or additional compression module. The prefill seed and teacher-forced target states are never transmitted. Intermediate nodes emit only M_v; the terminal solver or reviser appends a fixed textual readout cue and decodes through its original language-model head.

The extraction layer is fixed per backbone after development validation; the initial choice is $\ell_i = \lfloor L_i/2 \rfloor$. Mapping d_i to d_j changes representation width but does not establish lossless transfer. With BF16 states sent before receiver-side projection, one edge carries $2K_v d_i$ payload bytes, excluding metadata. All K_v positions contribute to receiver prefill cost.

4.2 RL-Only Exploration

Within each LTA, the ESH parameterizes a diagonal Gaussian policy, following standard continuous-action policy optimization (Schulman et al., 2017). It predicts a bounded, positive per-coordinate scale:

$$\sigma_{v,k} = s_i \left[a_{\min} + (a_{\max} - a_{\min}) \operatorname{sigmoid}\Big(g_{\phi_i}(h^{L_i}_{v,k-1}) \Big) \right], \tag{5}$$

where s_i is a fixed reference scale measured from model i's normal token embeddings. During RL, $q_\Theta(e_{v,k} \mid H_{v,k}) = \mathcal{N}(\mu_{v,k}, \operatorname{diag} \sigma^2_{v,k})$. The random action is the *input embedding* $e_{v,k}$; the transmitted message M_v is a nonlinear consequence and need not be Gaussian. Warm-up, joint supervision, replay, and inference use $e_{v,k} = \mu_{v,k}$ and bypass the ESH. Table 1 summarizes the update rules.

4.3 Multi-Role Warm-Up and Mixed-Workflow Supervision

The interfaces in Figure 1 are trained in three stages, summarized in Figure 2. Independent warm-up first trains each LTA to support role-specific readout. Joint supervision then connects the agents and trains LTAs and CMBs together on both workflows. Finally, RL explores complete latent trajectories while retaining deterministic supervision through replay; Section 4.4 gives its policy objective.

Table 1: Execution and optimization stages. Every stage freezes backbone, embedding, and output-head parameters. ESH is initialized with positive scale upon entering RL.

Stage	Latent inputs	Trainable interfaces	Supervision
Independent warm-up	Mean	LTA mean	Role-specific readout
Joint supervision	Mean	LTA mean, CMB	Task and auxiliary readout; warm-up replay
Joint RL	Gaussian samples	LTA mean, ESH, CMB	Trajectory reward and KL
Supervised replay in RL	Mean	LTA mean, relevant CMB	Task and auxiliary readout; warm-up replay
Deployment	Mean	None	Greedy final decoding

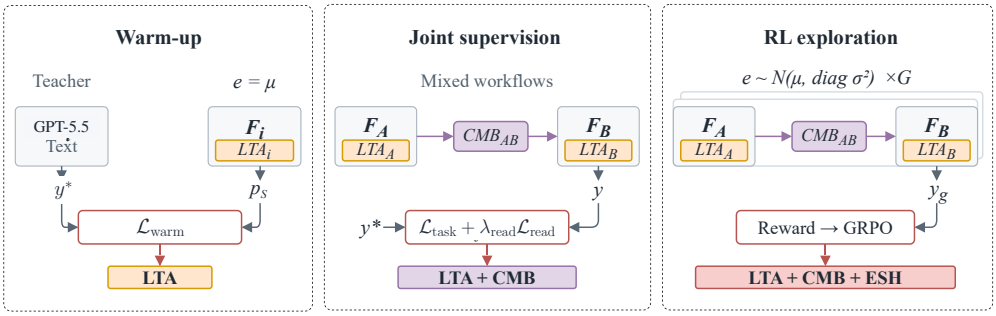


Figure 2: Training RECAST. **Left:** offline GPT-5.5 text targets supervise deterministic LTA warm-up through the student's frozen readout head. **Center:** mixed-workflow supervision jointly updates LTAs and CMBs with task and intermediate-readout losses. **Right:** RL samples G complete workflows and updates LTAs, CMBs, and ESHs using outcome-based GRPO, interleaved with deterministic supervised replay. Backbone boxes depict an adjacent pair for compactness; actual workflows have three agents. Auxiliary readout branches and replay are omitted from the drawing and never supply inter-agent text.

Independent warm-up. Following the staged interface training of RecursiveMAS (Zou et al., 2026a), each LTA first learns deterministic recurrence independently of a multi-agent workflow. We vary tasks, roles, and instruction paraphrases during this stage. OpenAI GPT-5.5 serves as the shared API teacher, generating explicit role artifacts offline: answers, concise plans, verification reports, or revisions. The teacher receives only the task information permitted for that role and, when required, an identified candidate. Reference answers and private tests are used by external checkers, not added to the teacher prompt. Accepted texts are cached and independently encoded with each student's tokenizer; no API logits, hidden reasoning trace, or teacher–student token alignment is assumed. Let y_i^* denote the resulting target sequence for model i and $p_{i,t}^{\text{mean}}$ its readout distribution after K deterministic latent steps, conditioned on the same target prefix. The warm-up objective is

$$\mathcal{L}_{\text{warm}} = -\frac{1}{|y_i^*|}\sum_t \log p_{i,t}^{\text{mean}}(y_{i,t}^*), \tag{6}$$

averaged over valid target positions and then instances. Directly checked targets and outcome-filtered weak plans are distinguished in the data manifest; a successful final answer does not certify every plan step. Only directly checked targets are eligible for the main intermediate-readout loss. The student's latent computation never receives teacher-generated target text; teacher forcing occurs only in the readout branch. No latent position needs a corresponding textual step. Gradients pass through deterministic recurrence to the LTA mean, with ESH inactive. A short-to-long latent-step curriculum precedes validation of readout quality and numerical stability. Appendix G specifies teacher records and label filtering.

Joint supervision. Joint supervised optimization of latent collaboration is also studied in DiffMAS (Yu et al., 2026). We initialize the system with warm LTAs and new CMBs, then jointly train the three-

node PVS and SVR workflows. All six model permutations cover every directed pair within these workflows. The terminal answer loss backpropagates through receivers, bridges, and deterministic sender computations. Our core supervised objective also includes intermediate auxiliary readout:

$$\mathcal{L}_{\text{sup}} = \mathcal{L}_{\text{task}} + \lambda_{\text{read}}\mathcal{L}_{\text{read}}, \qquad \lambda_{\text{read}} > 0. \tag{7}$$

Auxiliary objectives at internal computation stages follow the general principle of deep supervision (Lee et al., 2015). After a receiver finishes its mean latent steps, a separate branch appends a readout query to its prefix and predicts a verified target through the frozen language-model head. Targets specify relevant facts, constraints, or checkable role outputs. The readout loss averages target-token negative log-likelihood within each labeled receiver and then across labeled receivers, controlling for differences in target length and label coverage. Gradients reach the receiver LTA, incoming CMBs, and upstream LTAs through deterministic recurrence. The readout query and teacher-forced target never enter the main cache or the outgoing message. This loss is part of the main method, including deterministic workflow replay during RL. Appendix F specifies labels, masks, and loss normalization. Independent warm-up examples are also replayed to preserve role capabilities.

Joint training balances the two workflows and six model assignments, with instruction variants drawn from a shared, predefined manifest. Every model performs each basic role, and every directed interface is covered. Prompt paraphrases preserve evidence permissions, goals, hard constraints, and output requirements, including whether verification concerns a plan or a solution. Development and test use the same configuration support. Base tasks and source documents are split *before* paraphrasing or teacher generation. Candidate-dependent roles consume the preceding agent's latent message during collaboration; standalone warm-up uses identified textual candidates where required.

4.4 Joint Optimization with Continuous-Action GRPO

The final stage in Figure 2 adapts GRPO's group-relative advantages and PPO's clipped surrogate (Schulman et al., 2017; Shao et al., 2024) to the joint continuous-latent and discrete-text policy. The outcome signal evaluates a complete collaboration, allowing the interfaces to explore beyond the deterministic trajectories used in the preceding stages.

At the transition to RL, we attach the ESH with a small positive initial scale and freeze a complete reference snapshot Θ_{ref}. The reference thus defines a non-degenerate Gaussian policy; a deterministic supervised checkpoint alone is not a valid Gaussian KL reference. Let $\Theta = (\theta, \phi, \psi)$ collect the trainable interfaces. For a fixed task, workflow, permissions, templates, and step budget c, we independently sample G complete latent-and-answer trajectories. The joint policy factorizes as

$$p_\Theta(E, Y \mid c) = \prod_{a \in \mathcal{A}_z} q_\Theta(e_a \mid H_a^\Theta(c, E_{<a})) \prod_{t=1}^{|Y|} p_\Theta(y_t \mid c, E, y_{<t}), \tag{8}$$

where a indexes latent actions in execution order. CMBs affect downstream states H_a^Θ and output distributions despite having no separate random action.

An external, non-learned outcome verifier supplies $R_g = R(c, Y_g) \in \{0, 1\}$ using reference answers or private tests for training instances. Mathematical equivalence, exact option matching, and passing all code tests define success for the corresponding task families. Reference material is held outside the agents' observations. Output parsing is independent of the reference answer; incorrect or invalid outputs receive zero. Infrastructure failures trigger verification retries on the saved output, rather than an incorrect-answer reward. Appendix D specifies parsing, failure handling, and the separation of training rewards from test scores. Group advantages are

$$A_g = \frac{R_g - \overline{R}}{\text{std}(R_1, \ldots, R_G) + \varepsilon}. \tag{9}$$

Every action of trajectory g shares A_g. This is outcome-based credit, not a label that each latent step is correct. Equal-reward groups have no relative task signal; their frequency is a training diagnostic.

The rollout stores the actual FP32 latent actions and sampled answer tokens. During optimization these actions are *fixed*, while the current system recomputes its states and messages. For a latent action, the ratio is

$$\rho_{g,a}^z = \exp\left[\log q_\Theta(e_{g,a} \mid H_a^\Theta) - \log q_{\text{old}}(e_{g,a} \mid H_a^{\text{old}})\right], \tag{10}$$

and $\rho^y_{g,t}$ is the corresponding sampled-token probability ratio. One complete vector is one action: its Gaussian log density sums over all coordinates. Ratios are clipped per vector or token, rather than multiplying a full-trajectory ratio or dividing a vector log ratio by its width. With $s(\rho, A) = \min\{\rho A, \text{clip}(\rho, 1-\delta, 1+\delta)A\}$, the policy loss is

$$\mathcal{L}_{\text{PG}} = -\frac{1}{GC}\sum_{g=1}^{G}\left[\sum_{a\in\mathcal{A}_z} s(\rho^z_{g,a}, A_g) + \sum_{t=1}^{|Y_g|} s(\rho^y_{g,t}, A_g)\right], \tag{11}$$

where C is a fixed optimization scale, independent of the sampled answer length. Separate Gaussian and categorical reference penalties give $\mathcal{L}_{\text{RL}} = \mathcal{L}_{\text{PG}} + \beta_z\mathcal{K}_z + \beta_y\mathcal{K}_y$. These are local conditional KL terms on rollout histories, not an exact current/reference trajectory KL.

The distinction between supervised and RL computation graphs is essential. Supervision differentiates through $e = \mu$; GRPO scores saved, detached e under the current Gaussian. Reusing noise to resample a new action would invalidate the same-action ratio. Conversely, freezing old messages or receiver caches would remove required CMB gradients. Current and reference systems must recompute their own states under the same saved action history. Freezing backbone parameters does not disable gradients with respect to their inputs.

We interleave RL with deterministic workflow replay using Equation (7), alongside independent warm-up replay. Auxiliary readout is computed on a separate mean-path forward pass, not on the saved stochastic actions used for GRPO ratios; it does not update ESH. Deployment removes readout branches, bypasses ESH, and uses the mean latent path with greedy final decoding. Checkpoint selection uses deterministic validation performance in Eq. (2); improved stochastic rollout reward alone is insufficient. Appendix B details likelihoods, update checks, and the high-dimensional ratio stability controls.

5 Experiments

Tasks and models. We adopt the three frozen backbones in RecursiveMAS Light: Qwen3-1.7B, Llama-3.2-1B-Instruct, and Qwen2.5-Math-1.5B-Instruct (Zou et al., 2026a). Its reference order assigns these models to planner, critic, and solver, respectively; our cross-workflow protocol varies their roles and assignments while sharing interfaces. The primary suite matches its main Light results: MATH500, AIME 2025, AIME 2026, GPQA-Diamond, MedQA, and MBPP+. The code-generation column in its main table uses MBPP+ for Light and LiveCodeBench for Scaled. Dataset provenance, model identifiers, and checker details appear in Appendix D. These benchmarks are held out from interface training. Their base problems, paraphrases, and teacher-generated derivatives cannot enter warm-up, supervision, RL, or checkpoint selection.

The comparison protocol requires the same data snapshot, output extraction, verifier, and deterministic deployment rule for every method; shared models and benchmarks do not imply numerical reproduction of published scores. Unmodified benchmarks use their standard problem information. Sender-private evidence tasks are separate diagnostics, with permissions attached to logical role positions. The training protocol uses disjoint MATH, GSM8K, ARC, MedQA, and APPS training partitions, capped at 20,000 training and 2,000 development base problems before teacher filtering (Appendix H).

Two-workflow protocol. Training and evaluation use PVS and SVR, each with three nodes and all six assignments of the Light models to the three positions. Three cyclic instruction tuples, drawn from three variants of each semantic role, give 36 workflow–assignment–instruction configurations shared across task splits (Appendix J). Joint supervision, RL, and workflow replay all cover both workflows; there are no reserved role sequences, model assignments, instruction families, or longer chains. A single checkpoint is selected on disjoint development tasks and then used for both workflows. Appendix C specifies the manifest, coverage, and task-separation checks.

Baselines and attribution. The suite includes the matched terminal model, frozen and trained text communication, RecursiveMAS with configuration mixing, and an Interlat adaptation. The protocol gives trainable baselines the same task sources and eligible teacher artifacts. The single-model reference uses the corresponding terminal backbone and permitted evidence. Trained text uses shared

Table 2: Primary results (%, higher is better). Each cell averages equally over six model assignments and the declared instruction variants for one workflow. Avg. is the unweighted mean of the six benchmark scores. Both workflows and all configurations are included in training; test tasks are disjoint.

Method	MATH	AIME25	AIME26	GPQA	MedQA	MBPP+	Avg.
Plan → verify → solve (PVS)							
Single-agent	55.8	5.2	3.0	27.6	35.3	32.7	26.6
Text-MAS, frozen	57.0	5.6	3.3	28.1	34.9	34.5	27.2
Text-MAS, trained	62.7	8.5	5.2	30.7	40.1	41.8	31.5
RecursiveMAS + Mix	64.3	9.3	5.9	31.4	38.9	40.5	31.7
Interlat-adapted	62.9	8.9	5.4	31.1	40.6	42.2	31.9
RECAST	67.3	11.5	7.6	33.2	41.7	42.1	**33.9**
Solve → verify → revise (SVR)							
Single-agent	55.8	5.2	3.0	27.6	35.3	32.7	26.6
Text-MAS, frozen	56.2	5.7	3.7	27.9	35.8	35.6	27.5
Text-MAS, trained	62.2	8.9	5.9	31.0	40.8	44.0	32.1
RecursiveMAS + Mix	63.5	9.6	6.5	31.3	39.6	41.9	32.1
Interlat-adapted	63.1	9.1	5.9	31.7	41.3	43.8	32.5
RECAST	66.9	12.0	8.3	33.6	42.5	44.3	**34.6**

Table 3: Ablations and message-use diagnostics for the two trained workflows (%, higher is better). All entries are absolute scores. Task-score rows use the averaging in Table 2; private-fact consistency uses separate diagnostic tasks and requires both outputs in a pair to satisfy their changed constraints.

Comparison or intervention	PVS	SVR
Full-method task score	**33.9**	**34.6**
No independent warm-up	30.5	30.8
No outcome-filtered weak plan targets	33.4	34.4
No auxiliary readout ($\lambda_{\text{read}} = 0$)	32.2	32.6
No RL	32.6	33.0
Fixed-scale RL exploration	33.5	34.1
Deterministic latent, output-token GRPO	33.2	33.7
Final-layer rather than intermediate-layer transfer	33.1	34.0
Matched-message score	**33.9**	**34.6**
Strictly deranged-message score	29.2	29.1
Communication-disabled score	30.0	30.4
Equal-shape uninformative-message score	29.8	30.3
Private-fact intervention consistency	60.0	57.0

LoRA and token-action GRPO; RecursiveMAS retains staged inner/outer training; Interlat uses LoRA receiver adaptation without compression. Appendix K specifies these departures from the original methods.

Main results. RECAST scores 33.9% on PVS and 34.6% on SVR, compared with 31.9% and 32.5% for Interlat-adapted, the strongest aggregate comparator. The gains over trained Text-MAS are 2.4 and 2.5 percentage points. RECAST leads in 11 of 12 benchmark–workflow conditions; the exception is PVS on MBPP+, where Interlat-adapted scores 42.2% versus 42.1%. Appendix M gives the strongest comparator for each benchmark and workflow.

Metrics and communication controls. The deployment protocol uses one mean latent trajectory ($K = 8$) and one greedy answer per configuration. MATH denotes MATH500 and GPQA denotes GPQA-Diamond. Matched messages are compared with disabled, strictly deranged, and equal-shape uninformative messages. Sender-private fact interventions test whether answers follow changed evidence after downstream recomputation (Cheng et al., 2026) and use separate diagnostic tasks.

Training ablations. Removing warm-up gives the largest reduction, from 33.9/34.6 to 30.5/30.8 for PVS/SVR. Removing auxiliary readout yields 32.2/32.6; omitting RL yields 32.6/33.0. Fixed-scale

exploration reaches 33.5/34.1, and deterministic latent states with output-token GRPO reach 33.2/33.7. Learned exploration thus adds less than warm-up or readout supervision in these comparisons. Last-layer transfer scores 33.1/34.0; removing weak plan targets scores 33.4/34.4. These one-factor comparisons characterize the individual removals without identifying component interactions.

Message-use interventions. Disabling communication reduces scores to 30.0/30.4. Strictly mismatched messages yield 29.2/29.1, below the disabled condition, whereas equal-shape uninformative messages yield 29.8/30.3. The ordering is consistent with useful message content and with misleading content sometimes being worse than no message. Private-fact consistency is 60.0%/57.0%, leaving 40.0%/43.0% of pairs that fail the two-output criterion. The pair-level criterion captures end-to-end consistency without localizing failures to the sender, bridge, or receiver.

6 LIMITATIONS

The study covers two jointly trained workflows, known model assignments, and a shared instruction pool; it does not establish unseen-configuration or unseen-model generalization. Full-position transfer can be expensive, and layer selection may depend on the backbone. Factual readout alone does not prove message use; a correct SVR answer does not establish successful implicit error localization or revision. Gaussian policy ratios may be unstable, and better sampled trajectories need not improve mean-path deployment. Teacher errors, contamination, and reward misspecification can affect the learned interfaces. The comparisons assess the complete shared-interface system rather than the isolated contribution of parameter sharing.

7 CONCLUSION

RECAST shares latent interfaces across two jointly trained workflows of heterogeneous frozen agents. Reported scores of 33.9% and 34.6% exceed the strongest aggregate comparator by 2.0 and 2.1 points. Warm-up removal produces the largest ablation reduction, and interventions are consistent with useful message content. The results characterize communication and training components within the shared configuration support of PVS and SVR.

AI USE STATEMENT

We used generative AI tools to assist with experiment-design discussion and manuscript translation. The authors are responsible for verifying AI-assisted content and for the final manuscript, code, and scientific claims.

ETHICS STATEMENT

The evaluation uses public benchmarks and generated tasks. No human-subject study or deployment evaluation is reported.

REPRODUCIBILITY STATEMENT

The equations, execution modes, gradient conventions, and split protocol appear in Sections 3–5 and the appendix. Model commits, implementation configurations, and a staged launcher are recorded in Appendix L.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program Synthesis with Large Language Models. arXiv preprint arXiv:2108.07732, 2021.

Jiaming Cheng, Subhransu Das, and Rajiv Ramnath. When Does Latent Communication Pay? A Causal Audit of Relayed KV Caches in Multi-Agent LLMs. arXiv preprint arXiv:2608.04893, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. arXiv preprint arXiv:1803.05457, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. arXiv preprint arXiv:2110.14168, 2021.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process Reinforcement through Implicit Rewards. *Transactions on Machine Learning Research*, 2026.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling Agents to Communicate Entirely in Latent Space. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27106–27129. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1248.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-Cache: Direct Semantic Communication Between Large Language Models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. In *Second Conference on Language Modeling*, 2025.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). arXiv preprint arXiv:1606.08415, 2016.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring Coding Challenge Competence With APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What Disease Does This Patient Have? A Large-Scale Open Domain Question Answering Dataset from Medical Exams. *Applied Sciences*, 11(14):6421, 2021. doi: 10.3390/app11146421.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-Supervised Nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38, pp. 562–570. PMLR, 2015.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, volume 36, pp. 21558–21572, 2023. doi: 10.52202/075280-0943.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

Narcis Marincat. Portable Semantics, Private Dialects: Reuse and Negative Transfer in Latent Communication Between Language-Model Cells. arXiv preprint arXiv:2609.11365, 2026.

Yanwen Peng, Delvin Ce Zhang, Xi Wang, and Nikolaos Aletras. StateBridge: Training-free Hidden-state Alignment for Latent Communication in LLM Multi-Agent Systems. In *Third Conference on Language Modeling*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *First Conference on Language Modeling*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv preprint arXiv:2402.03300, 2024.

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. ReMA: Learning to Meta-Think for LLMs with Multi-agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 126621–126667. Curran Associates, Inc., 2025. doi: 10.52202/085713-4225.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *First Conference on Language Modeling*, 2024.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft Chain-of-Thought for Efficient Reasoning with LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23336–23351. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1137.

Ye Yu, Heming Liu, Haibo Jin, Xiaopeng Yuan, Peng Kuang, and Haohan Wang. Learning to Communicate: Toward End-to-End Optimization of Multi-Agent Language Systems. arXiv preprint arXiv:2604.21794, 2026.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Jiaru Zou, Rui Pan, Ruizhong Qiu, Pan Lu, Shizhe Diao, Jindong Jiang, Hanghang Tong, Tong Zhang, Markus J. Buehler, Jingrui He, and James Zou. Recursive Multi-Agent Systems. arXiv preprint arXiv:2604.25917, 2026a.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent Collaboration in Multi-Agent Systems. In *Forty-third International Conference on Machine Learning*, 2026b.

A EXECUTION AND DIFFERENTIATION CONVENTIONS

This appendix specifies the execution, differentiation, and evaluation protocol. A workflow node v invokes frozen backbone F_i, where $i = b_v$, and uses the shared LTA of model i. An edge from model i to model j uses the shared directed CMB $B_{\psi_{ij}}$. Both three-node workflows use these parameters; each model appears once per execution, with no role-specific or workflow-specific copies.

Node input and positions. The logical input order is: system and role instructions, the task and permitted local evidence, a message boundary, projected upstream states, and a task suffix. Ordinary text uses the backbone's tokenizer and chat template. The message insertion position and boundary strings are fixed before evaluation. Let $h_{v,0}^{L_i}$ denote the last valid prefill position, excluding padding. This state seeds recurrence but is neither a generated latent position nor part of the transmitted message. At latent position k, define $\mu_{v,k} = \mu_{\theta_i}(h_{v,k-1}^{L_i})$ and $\sigma_{v,k} = \sigma_{\phi_i}(h_{v,k-1}^{L_i})$. The backbone appends exactly one continuous input. Warm-up, supervision, replay, and deployment use

$$e_{v,k} = \mu_{v,k}. \tag{12}$$

Only RL rollouts use

$$e_{v,k} = \mu_{v,k} + \sigma_{v,k} \odot \epsilon_{v,k}, \qquad \epsilon_{v,k} \sim \mathcal{N}(0, I). \tag{13}$$

The standard-deviation head is inactive outside RL. Each step updates the node's causal cache and yields states at all blocks:

$$(h_{v,k}^1, \ldots, h_{v,k}^{L_i}, \mathcal{C}_{v,k}) = F_i(e_{v,k}; \mathcal{C}_{v,k-1}). \tag{14}$$

The message contains every generated position, in order:

$$M_v = [h_{v,1}^{\ell_i}, \ldots, h_{v,K_v}^{\ell_i}], \qquad \widetilde{M}_{v\to u}[k] = B_{\psi_{ij}}(M_v[k]), \quad j = b_u. \tag{15}$$

No position pooling, selection, or separate compression stage is applied. The reference layer choice is block $\lfloor L_i/2 \rfloor$; block numbering, the implementation's hidden-state index, and final-normalization conventions must be recorded. Layer choices are selected on development data and then held fixed. Intermediate nodes emit no text. The terminal node appends a fixed ordinary-text readout marker after its latent steps and uses its frozen language-model head to produce the answer.

Precision, caches, and gradient paths. Gaussian parameters, samples, saved raw actions, and density calculations use FP32. The backbone may consume a BF16 cast of the sampled action. Densities are evaluated on the original FP32 action, not on its rounded BF16 representation. No post-sampling normalization or clipping is included in the proposed policy. The transmitted tensor is the middle-layer state, whose distribution need not be Gaussian; it must not be assigned the input action's Gaussian density.

Freezing backbone weights preserves derivatives with respect to their inputs. Supervised forwards therefore retain the graph through mean recurrence, CMBs, caches, and receiver states. RL scoring fixes the sampled actions but recomputes all parameter-dependent messages and receiver states. Only pure-text prefixes independent of trainable modules can be cached as constants. Masks, position identifiers, message boundaries, and cache updates must agree between rollout and scoring. Teacher-forced gold answer positions are never used to construct messages. Auxiliary readout branches must not modify the cache used by the main workflow. Copying a mutable cache for readout must preserve the autograd graph; detaching it would sever supervision of the communication interfaces.

For a BF16 message, an edge carries $2K_v d_i$ payload bytes, plus metadata. For example, $K_v = 8$ and $d_i = 4096$ correspond to 64 KiB. All G RL rollouts execute distinct complete latent trajectories; their latent forwards cannot be shared as one common prefix. Payload size alone does not determine receiver prefill, recurrence, decoding, or training computation.

B CONTINUOUS-ACTION GRPO SPECIFICATION

Conditional policy and reference. The ESH parameterization bounds each coordinate's standard deviation:

$$\sigma_{\phi_i}(h) = s_i \left[a_{\min} + (a_{\max} - a_{\min}) \operatorname{sigmoid}(g_{\phi_i}(h))\right], \tag{16}$$

where s_i is a frozen embedding scale, defined as the median per-vector RMS of normal input-token embeddings. The ESH is initialized at RL entry to a small positive constant standard deviation, then trained jointly with the mean heads and CMBs. The reference is a complete frozen copy of this initial *non-degenerate stochastic* system. It is not the deterministic supervised policy viewed as a zero-variance Gaussian.

Let c fix the task, evidence permissions, workflow, backbones, roles, templates, and latent budgets of one group. In causal execution order, let $E = (e_a)_a$ collect continuous actions and $Y = (y_t)_t$ the terminal answer. The policy factorizes as

$$p_\Theta(E, Y \mid c) = \prod_a q_\Theta(e_a \mid H_a^\Theta(c, E_{<a})) \prod_t p_\Theta(y_t \mid c, E, y_{<t}). \tag{17}$$

Here H_a^Θ denotes the state recomputed under that system's parameters. Although c specifies the whole experiment, individual nodes receive only permitted observations. CMBs are deterministic, but influence subsequent conditional action densities and answer probabilities.

For each group, sample G complete trajectories under the old system. Rewards $R_g = R(c, Y_g)$ come from task verification. The shared trajectory advantage is

$$A_g = \frac{R_g - \overline{R}}{\mathrm{std}(R_1, \ldots, R_G) + \varepsilon}. \tag{18}$$

This supplies terminal credit to the trajectory; it does not label individual latent steps as correct. We use population standard deviation and $\varepsilon = 10^{-6}$. Groups with identical rewards have $A_g = 0$ for every trajectory: their task-reward term supplies no gradient, although KL and supervised replay can still update the interfaces. We record all-wrong, all-correct, and mixed-reward group frequencies by task family, retaining all rollouts in cost accounting. We do not resample without limit until a group has a success or introduce a format bonus to manufacture reward variation. Training difficulty and supervision coverage may be adjusted using training and development data, without inspecting benchmark answers.

Saved-action scoring and surrogate. During optimization, the realized FP32 actions and text tokens are fixed and detached. Current, old, and reference probabilities concern the same action history, but each system uses its own CMBs to determine its states. The current system must not regenerate actions from stored noise and its updated means. Nor may it substitute frozen rollout messages for current receiver computation.

One complete d_i-dimensional vector is one continuous action:

$$\log q_\Theta(e_a \mid H_a^\Theta) = \sum_{j=1}^{d_i} \left[-\frac{(e_{a,j} - \mu_{a,j})^2}{2\sigma_{a,j}^2} - \log \sigma_{a,j} - \tfrac{1}{2} \log(2\pi) \right], \tag{19}$$

$$\rho_{g,a}^z = \exp\left\{ \log q_\Theta(e_{g,a} \mid H_{g,a}^\Theta) - \log q_{\mathrm{old}}(e_{g,a} \mid H_{g,a}^{\mathrm{old}}) \right\}. \tag{20}$$

Text ratios $\rho_{g,t}^y$ use the corresponding categorical probabilities. With $f(\rho, A) = \min\{\rho A, \mathrm{clip}(\rho, 1 - \delta, 1 + \delta)A\}$, the proposed surrogate is

$$\mathcal{L}_{\mathrm{PG}} = -\frac{1}{GC} \sum_{g=1}^{G} \left[\sum_a f(\rho_{g,a}^z, A_g) + \sum_{t=1}^{|Y_g|} f(\rho_{g,t}^y, A_g) \right]. \tag{21}$$

C is a fixed optimization scale, not a sampled trajectory length. Ratios are clipped per vector action or token, rather than multiplied into a whole-trajectory ratio. Gaussian log-ratios sum over coordinates; dividing them by d_i would change the ratio and define a different surrogate. Coordinate-wise differences are summed in FP32 to reduce cancellation. Group normalization and per-action clipping are GRPO-style approximations, not an unbiased estimator or a monotonic-improvement guarantee.

Local KL regularization. For the same saved action history, conditional Gaussian KL is

$$D_{\mathrm{KL}}(q_\Theta \| q_{\mathrm{ref}}) = \sum_j \left[\log \frac{\sigma_{\mathrm{ref},j}}{\sigma_j} + \frac{\sigma_j^2 + (\mu_j - \mu_{\mathrm{ref},j})^2}{2\sigma_{\mathrm{ref},j}^2} - \frac{1}{2} \right]. \tag{22}$$

The two systems recompute their own means, scales, and conditioning states. The objective is

$$\mathcal{L}_{\mathrm{RL}} = \mathcal{L}_{\mathrm{PG}} + \beta_z \mathcal{K}_z + \beta_y \mathcal{K}_y, \tag{23}$$

where $\mathcal{K}_z$ averages complete-vector KL over valid latent actions, and $\mathcal{K}_y$ averages categorical KL over valid answer positions. These are local conditional regularizers on histories from old rollouts, not the exact KL between full current and reference trajectory distributions. Any vocabulary approximation must be documented.

One update cycle.

1. Sample a legal configuration c and copy the complete current system as the rollout policy. Keep the original RL-entry reference fixed.
2. Execute G independent workflows, saving raw FP32 actions, old Gaussian parameters, answer tokens, old token log-probabilities, and termination information. Verify answers and compute detached A_g.
3. Replay each saved action history under the current system with gradients. Recompute CMB outputs and receiver states; accumulate vector densities, token probabilities, and current conditional distributions.
4. Score the same histories under the frozen reference using its own CMBs and states. Compute Equation (21) and local KL terms.
5. Update active mean heads, ESHs, and CMBs; all backbone weights remain frozen. Begin with one optimizer update per rollout batch.
6. Check post-update current/old KL, clipping, density finiteness, and deterministic development performance. Interleave separate supervised workflow replay along Equation (12), optimizing the task and auxiliary-readout objective in Equation (7); also replay independent warm-up examples. Recompute the full mean-path graph, rather than attaching this loss to a saved stochastic cache. These supervised updates do not update ESHs.

Initial ratios equal one when current equals old, so clipping alone does not limit the first optimizer step. A development-selected KL rejection rule must restore parameters and the complete optimizer, scheduler, and scaler state before retrying with a smaller learning rate. Minimum implementation checks include current/old ratios near one, current/reference KL near zero at initialization, finite-difference checks of fixed-action log-likelihoods, active connector gradients, absent ESH gradients in supervision, and unchanged backbone hashes. Unlike supervised recurrence, the terminal token loss cannot backpropagate through a detached sampled action into its generating mean head; Gaussian score terms provide that head's RL update.

C TWO-WORKFLOW PROTOCOL AND DATA PERMISSIONS

Workflow semantics. The two selected workflows are planning–verification–solving (PVS) and solving–verification–revision (SVR). Both have exactly three nodes and two directed communication edges. The three Light models are each used once per execution. Independent warm-up trains planning (P), solving (S), verification (V), and revision (R); joint supervision, RL, and workflow replay use only these two complete workflows.

Table 4: The two workflows used in both training and evaluation. Intermediate outputs are latent messages; only node 3 decodes the task answer.

Type	Node 1	Node 2	Node 3
PVS	Plan: propose subgoals, operations, and constraints.	Verify: inspect the plan for feasibility, omissions, and conflicts.	Solve: use the reviewed plan and task evidence to produce a complete answer.
SVR	Solve: form a candidate solution.	Verify: inspect the candidate for errors and missing justification.	Revise: repair the candidate when needed and return a complete final answer.

The verification instruction specifies whether its object is a plan or a solution. Its outgoing latent state must retain the relevant proposal content together with its checks so that node 3 can use both. The

actual communication path remains node 1 to node 2 to node 3, with no extra textual candidate or direct node-1-to-node-3 connection. SVR revision preserves a correct candidate when appropriate; its scored output is the complete answer or program, not merely a critique. These are requested behaviors, whose occurrence is not certified by final-answer accuracy. During standalone warm-up, candidate-dependent roles may consume an identified textual plan or solution. During collaboration they consume the preceding latent message under the branch-isolation rules.

Shared configuration support. For each workflow, use assignments ABC, ACB, BAC, BCA, CAB, and CBA; denote their set by Π. Letters identify models in Table 5. Each assignment maps models to successive roles. Across both workflows, these assignments cover all model–role pairs and all six directed model interfaces. Let $\mathcal{Q}_w$ be a finite, nonempty manifest of three-node instruction tuples for workflow w. Variants preserve goals, evidence permissions, hard constraints, verification object, and output requirements. Each workflow and all its assignment/instruction combinations are used in training, development, and evaluation. Joint stages balance workflows and assignments and sample instruction tuples uniformly. The training manifest verifies coverage of every declared configuration. Appendix J fixes three tuples per workflow and all 18 role strings, giving 36 configurations. No workflow, assignment, expression family, or depth is reserved as an out-of-distribution evaluation condition.

Task separation and information access. Split base problems, source documents, program instances, and generation seeds before producing paraphrases or teacher artifacts. All derivatives of a base item remain in its task split. Sharing instruction templates does not allow training and test questions, answers, or teacher artifacts to overlap. Primary benchmark items remain excluded from every training stage and development selection. Half of joint-training draws use eligible sender-private evidence views where available (Appendix H). Full-information tasks are tracked separately. When model assignments change, evidence stays attached to logical role positions. Answers, hidden tests, and verifier-only facts cannot enter student prompts or messages. Teacher outputs require answer checks, tests, or simulator validation where available. Outcome-filtered plans retain their weak-label status under Appendix G; unfiltered or contradictory plans are excluded.

Main-table aggregation. For benchmark b with task set $\mathcal{X}_b$, workflow w, assignment π, and instruction tuple q, evaluate one mean-path trajectory and one greedy final answer. The percentage score in Table 2 is

$$A_{b,w} = \frac{100}{6|\mathcal{Q}_w|} \sum_{\pi \in \Pi} \sum_{q \in \mathcal{Q}_w} \frac{1}{|\mathcal{X}_b|} \sum_{x \in \mathcal{X}_b} S_x(f_\Theta^{\text{mean}}(x, w, \pi, q)) \,. \tag{24}$$

Here w, π, q determine the full workflow configuration in Equation (2); S_x is the binary benchmark success indicator. The main table has one block per workflow and six benchmark columns. Its Avg. column is $\frac{1}{6}\sum_b A_{b,w}$, so larger datasets do not dominate the benchmark average. Each terminal model appears twice among the six assignments. The single-agent reference uses that configuration's terminal backbone and permitted evidence, with the same aggregation weights.

Choose one checkpoint using the corresponding deterministic development average over both workflows, assignments, and instruction tuples, with task-family weights fixed before selection. Test task results do not select checkpoints or prompt variants. Message controls include matched, absent, strictly deranged, and equal-shape uninformative messages, plus minimal interventions on sender-private facts. These diagnostics use the same two workflow types and are reported separately from the benchmark table. Every intervention recomputes downstream states. Best-of-G training diagnostics are not deployment accuracy.

D BACKBONES, BENCHMARKS, AND OUTCOME VERIFICATION

Fixed model identities. Table 5 adopts the Light backbones of RecursiveMAS (Zou et al., 2026a). These are model identities, not pretrained RecursiveMAS link checkpoints. We train ReCAST interfaces under the specified warm-up and joint objectives. Each model has one shared LTA; all six directed pairs have shared CMBs. Native tokenizers and chat templates are used, and inference remains deterministic as specified above. Qwen3 uses its non-thinking chat-template mode, consistent with the reference paper's instruction-mode setting. Token budgets and any input truncation are

Table 5: Frozen backbone identities from RecursiveMAS Light. Model and tokenizer commits are recorded in the executable configuration.

ID	Official model repository	RecursiveMAS reference role
A	`Qwen/Qwen3-1.7B`	Planner
B	`meta-llama/Llama-3.2-1B-Instruct`	Critic
C	`Qwen/Qwen2.5-Math-1.5B-Instruct`	Solver

Table 6: Outcome checkers for the primary suite. Every score uses one deterministic latent trajectory and one greedily decoded output per task configuration. The same task-family rules apply to disjoint training instances for binary RL rewards; these benchmark instances are never RL data.

Benchmark	Outcome checker	Test metric
MATH500	Final-answer extraction and Math-Verify equivalence check against the reference	Accuracy
AIME 2025	Exact integer match in $[0, 999]$	Accuracy
AIME 2026	Exact integer match in $[0, 999]$	Accuracy
GPQA-Diamond	Exact match of the final option label	Accuracy
MedQA	Exact match of the final option label	Accuracy
MBPP+	EvalPlus execution against base and extended tests	Pass@1

fixed per backbone within its supported context window, shared by compared methods, and logged with overflow counts. The reference assignment is retained as a comparison configuration, not as a restriction on which model may perform a role.

Primary evaluation suite. The six benchmarks in Table 6 match the Light configuration in the main results table of RecursiveMAS. MATH500 is a 500-item evaluation subset of MATH (Hendrycks et al., 2021b). AIME 2025 and AIME 2026 are separate yearly evaluation sets. GPQA uses its Diamond subset (Rein et al., 2024), and MedQA uses the released English four-option snapshot (Jin et al., 2021). MBPP+ is an EvalPlus project extension of MBPP (Austin et al., 2021; Liu et al., 2023); both base and extended tests are required for its primary score. The selected dataset releases, item IDs, and evaluator commits must be frozen before any measurements. LiveCodeBench, HotpotQA, and Bamboogle are outside this primary Light suite. RecursiveMAS reports Pass@10 for AIME; our primary AIME metric is single-output accuracy under deterministic deployment. These scores are not directly comparable to its published Pass@10 values. Repeating an identical deterministic output ten times would not provide ten independent samples.

The data sources are `HuggingFaceH4/MATH-500`, `math-ai/aime25`, `MathArena/aime_2026`, and `Idavidrein/gpqa` with configuration `gpqa_diamond`. For MedQA, we select the author's released `inference/dataset/medqa.json` snapshot containing 300 items, rather than silently replacing it with the full original test set. Its upstream revision and file checksum are recorded in the accompanying source audit. MBPP+ is obtained through EvalPlus; the benchmark data release must be pinned separately from the Python package version. These source identities do not replace item-level manifests or imply that a floating upstream download is reproducible.

Reward and score. For a training task, the verifier returns a status, a binary outcome, and diagnostics. Reference answers and test payloads are validated before the run; missing or invalid references block scoring rather than penalizing a model output. The core reward is

$$R(c, Y) = \mathbf{1}\{\text{the parsed answer satisfies the training task's checker}\}. \tag{25}$$

Mathematical equivalence or an exact correct option earns one; a program earns one only if it passes all verifier tests for that training task. Wrong answers, invalid outputs, and candidate-caused execution failures earn zero. No learned judge is required. Format, response length, latent norm, and the apparent plausibility of a plan do not add reward. Code-test pass fraction is a diagnostic, not the core reward or MBPP+ Pass@1. Test score S_x uses the held-out task's checker only after checkpoint selection. All baselines use the same scoring implementation; differences from the original RecursiveMAS evaluator are documented, and published scores are not mixed with these new measurements.

Answer extraction. The readout instruction requests one final answer field, or one Python solution for code. Extraction never uses the reference answer to choose among candidates. For mathematics, accept a unique final boxed answer or an explicit final-answer field, according to a fixed precedence rule; conflicting final answers are invalid. MATH500 uses Math-Verify with a frozen parser, normalization, timeout, and numeric-precision configuration. Signs, tuple order, and distinctions between sets and ordered answers are preserved. There is no language-model fallback or search through the reasoning for any number matching the reference. AIME accepts an integer between 0 and 999, normalizing leading zeros; it does not round values or take them modulo 1000. Choice tasks require exactly one valid final option label. The allowed labels come from the instance schema. Any training-time option permutation is recorded with its updated label and held fixed within a GRPO group. Missing, ambiguous, or unparseable outputs score zero.

Code execution and failure handling. MBPP+ is scored with the EvalPlus harness, with its release, task IDs, test hashes, dependencies, and resource limits pinned. Only the benchmark's intended problem fields and public examples appear in the prompt. Private test code and expected outputs remain in the evaluator. A fixed extraction rule returns one program, without test-guided selection among code blocks or automatic repair. Generated code executes in an isolated runner with fixed file, network, memory, and time limits. Syntax errors, candidate exceptions, and candidate-caused timeouts are failed solutions.

Evaluator crashes, missing runner dependencies, and host failures are infrastructure errors, not wrong answers. The checker may retry the same saved output at most twice without regenerating it. If an RL group still has an unresolved infrastructure error, its entire policy update is deferred; its denominator is not silently reduced. Evaluation scores include every task; infrastructure failures trigger re-scoring of saved outputs rather than removal from the denominator. Malformed model outputs remain ordinary scored failures, not infrastructure errors. Logs preserve raw output, extracted answer, status, test outcomes, and verifier version so these cases can be audited.

Training-data separation. An evaluation dataset being publicly downloadable, or exposing a split named `train`, does not authorize its use for interface training. Benchmark examples and their derivatives are excluded from warm-up, supervision, RL, teacher generation, and development selection. Appendix H specifies disjoint training sources, development reservation, and decontamination. Adopting the RecursiveMAS evaluation suite does not grant permission to train on benchmark items. Before paraphrasing, assign base problems, source documents, and equivalent program tasks to splits. Record source/version, item ID, problem hash, permitted fields, and verifier-payload hash. Alternative wording, option orders, and teacher artifacts inherit the base item's split. Gold answers and private tests are held by the checker, never provided to an acting agent. A terminally correct teacher answer may filter its associated plan, but does not certify each step of that plan. Separate sender-private diagnostics must specify their own task generator, evidence permissions, and exact checker before evaluation.

E Interface architecture and implementation defaults

Table 7 records the defaults in the accompanying implementation. Backbone identities and the primary suite are given in Appendix D; data construction, prompts, and baseline adaptations appear in Appendices H–K. The mean head and CMB use RMS normalization (Zhang & Sennrich, 2019), a linear branch, and a residual two-layer GELU MLP (Hendrycks & Gimpel, 2016) of width 512. The ESH uses a width-128 MLP with constant-output initialization.

The projection for an LTA mean or CMB is

$$u = \mathrm{RMSNorm}(h), \qquad \mathcal{P}(h) = W_0 u + W_2\,\mathrm{GELU}(W_1 u), \tag{26}$$

where $W_0 \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$, $W_1 \in \mathbb{R}^{r \times d_{\mathrm{in}}}$, and $W_2 \in \mathbb{R}^{d_{\mathrm{out}} \times r}$, with $r = 512$. For an LTA, $d_{\mathrm{in}} = d_{\mathrm{out}} = d_i$; for a CMB, $d_{\mathrm{in}} = d_i$ and $d_{\mathrm{out}} = d_j$. Initialization calibrates output magnitude to the destination embedding scale. The ESH's final layer starts with zero weights and a constant bias chosen to yield $a_{\mathrm{init}} s_i$. Scale calibration does not imply that the initial projection preserves semantics.

Warm-up progressively permits $K \in \{1, 2\}$, then $\{1, 2, 4, 8\}$, then $\{2, 4, 8, 16\}$ over approximately 20%, 30%, and 50% of its input-position budget. This samples a budget, not a latent perturbation.

Table 7: Interface and optimization defaults in the accompanying implementation.

Item	Implementation default
Shared interfaces	3 LTAs with ESHs; 6 directed CMBs
Frozen backbones	RecursiveMAS Light identities in Table 5
Deployment latent budget	$K = 8$
Joint-training workflows	PVS and SVR, equal probability; exactly 3 nodes
Model assignments	All 6 permutations in both workflows, equally weighted
Instruction variants	Three variants per semantic role; three cyclic tuples
Joint-training latent budgets	$4, 8, 16$ with probabilities $0.25, 0.50, 0.25$
ESH bounds and initialization	$(a_{\min}, a_{\max}, a_{\text{init}}) = (0.01, 0.20, 0.03)$
Warm-up learning rate	Mean head 10^{-4}; ESH inactive
Warm-up teacher	GPT-5.5 API; cached role texts, student-specific tokenization
Main auxiliary readout	$\lambda_{\text{read}} = 0.1$
Joint-supervised learning rates	Mean 10^{-5}; CMB 10^{-4}; ESH inactive
RL learning rates	Mean 10^{-6}; ESH 3×10^{-7}; CMB 3×10^{-6}
Group size / clipping / fixed scale	$G = 8$; $\delta = 0.1$; $C = 256$
KL coefficients	$\beta_z = 10^{-3}$; $\beta_y = 10^{-2}$
Rollout answer sampling	Temperature 1, top-$p = 1$, dropout disabled
Optimizer / gradient clipping	AdamW (Loshchilov & Hutter, 2019) / global norm 1.0
Supervised replay	Joint supervision: 10% warm-up replay; RL: 20% deterministic supervision

The latent-step curriculum changes per-node compute, not the number or ordering of agents. The training/development caps are $20{,}000/2{,}000$ base problems before filtering (Appendix H). They do not replace the primary benchmarks' complete evaluation sets or specify a random test split from those benchmarks. Readout branches add training computation but are absent from deployment.

F Intermediate auxiliary readout as a core objective

Readout location and targets. For configuration c, let $\mathcal{V}_{\text{read}}(c)$ be its receiving nodes with valid auxiliary labels. A receiver completes its K_v mean latent steps before a training-only branch appends a query u_v and predicts a target string a_v^*. The readout uses the receiver's original frozen decoder and output head, not a separately trained probe that could absorb the supervision. A labeled terminal receiver is included: the auxiliary target concerns received facts, constraints, or a checkable role output, distinct from simply duplicating its final task answer. This supplies supervised paths through both edges of the three-node chain. The sender's isolated role capability is also trained by warm-up replay.

Labels come from permitted task facts, generator records, independently checked intermediate results, or validated role targets. For example, when a task gives only the sender a numeric bound, the receiver can be asked to recover that bound; the label is available from the task record. A constraint or candidate verdict must have its own reference check. An outcome-filtered teacher plan does not become a step-verified label merely because its final answer is correct. In particular, a verdict on a teacher's explicit candidate cannot be presented as an oracle assessment of an unidentified candidate inside the current latent message. Unverifiable targets are masked out; the manifest records their source, validation rule, and coverage by task, role, and directed interface. The main training manifest must provide validated readout examples for every evaluated directed interface, rather than enabling a nominal loss with no labeled data. Queries and labels follow reusable operations and vary across instruction families, not fixed workflow IDs or node numbers.

Loss and normalization. Write $\mathcal{C}_v^{\text{mean}}$ for the receiver's task prefix and cache after deterministic latent recurrence but before any auxiliary query or target is added. With each target encoded by that

receiver's tokenizer, the objective is

$$\ell_v(c) = -\frac{1}{|a_v^*|}\sum_{t=1}^{|a_v^*|}\log p_{\theta,\psi}\left(a_{v,t}^* \mid \mathcal{C}_v^{\text{mean}}, u_v, a_{v,<t}^*\right), \tag{27}$$

$$\mathcal{L}_{\text{read}} = \mathbb{E}_{c\sim\mathcal{D}_{\text{train}}}\left[\frac{\sum_{v\in\mathcal{V}_{\text{read}}(c)}\ell_v(c)}{\max(1, |\mathcal{V}_{\text{read}}(c)|)}\right]. \tag{28}$$

The empty-label sum is zero. Multiple valid queries for one receiver are first averaged within that receiver; targets of length zero are excluded. This normalizes by target length, then labeled receiver, then workflow. The terminal task loss $\mathcal{L}_{\text{task}}$ is the target-token negative log-likelihood averaged over the final answer and then workflows. Equation (7) retains both losses. The readout coefficient is positive in the full method; setting it to zero defines the removal ablation.

Gradient paths and branch isolation. The supervised path runs from the sender LTA through its middle-layer message, the CMB, the receiver's prefill and latent recurrence, and the frozen readout decoder. Backbone parameters receive no updates, while input derivatives connect the loss to all participating mean LTAs and CMBs. There is no detach at messages or caches. Query and target tokens are appended only to a separate branch with an isolated mutable cache. Its logits, text, KV entries, and target-position states are never transmitted. Outgoing messages use only the original K_v latent states from the main branch, and the terminal answer is computed from that branch's uncontaminated prefix. Targets supervise a branch after the latent state is formed; they are not inputs used to construct that state.

Interaction with RL and evaluation. Joint supervised batches and labeled workflow-replay batches during RL use Equation (7). Independent warm-up replay continues to use Equation (6). Workflow replay recomputes a complete deterministic forward pass, separate from the detached actions and likelihood ratios of GRPO. It updates mean LTAs and CMBs but bypasses ESH. The readout loss is not added to the outcome reward and does not create purported correctness labels for every latent step. Readout branches are removed for task evaluation and deployment. Public task information may let a receiver answer some queries without using its message. Sender-private readout targets and message-off/swap interventions therefore remain necessary to assess content use.

Matched supervision controls. The deterministic, fixed-exploration, and full ReCAST variants retain the same readout targets and loss except in the explicit $\lambda_{\text{read}} = 0$ ablation. Trainable comparison systems receive equivalent eligible labels; frozen inference-only references are identified separately. Removing readout holds task instances, teacher data, and workflow coverage fixed.

G API TEACHER AND RELIABILITY OF ROLE TARGETS

Teacher identity and offline generation. The selected teacher is OpenAI GPT-5.5, accessed through the API with the dated model identifier `gpt-5.5-2026-04-23`, listed in the official model documentation at protocol drafting. The model-family alias is `gpt-5.5`; generation uses the dated identifier and records the actual returned model identifier. Use Responses with medium reasoning effort, a 4,096-output-token cap, no tools, and at most two attempts per base task (Appendix I). One requested model version supplies all students and roles. Generation precedes training; the accepted corpus is frozen, and student training and deployment do not call the teacher.

Each request contains a base task ID, a role, permitted observations, an instruction variant, and an explicit candidate when required. The requested verification object is a plan for PVS or a solution for SVR. Reference answers and private tests remain with the external checker. A planner returns short subgoals, proposed operations, and constraints; a solver returns a candidate solution; a verifier returns a judgment with localized checks on the supplied candidate; a reviser returns a corrected version of that same candidate. These are explicit output artifacts, not assumed access to the API model's internal reasoning. Multiple valid artifacts and prompt families are retained within a declared generation budget; rejected requests are counted, without unlimited retries until an example passes.

Reliability classes and allowed supervision. Each target records its status and supporting evidence:

1. **Directly checked.** Facts match permitted task records or source passages, intermediate mathematical claims pass applicable symbolic/numeric checks, and executable artifacts satisfy their stated checks or tests. The label records what was checked; finite tests do not prove unrestricted program correctness. These targets can supervise warm-up and, when aligned with the node's task and permissions, the main intermediate readout.
2. **Outcome-filtered weak plan.** A plan is associated with a correctly checked completion or succeeds with a fixed downstream text executor, but its individual steps are not independently certified. It must also pass role/permission, format, and contradiction screening. Such plans are eligible for warm-up and its replay, with their source and mixture fraction recorded in the data manifest. They are excluded from the main intermediate-readout targets unless the relevant component receives a direct check. Their removal is a separate ablation.
3. **Rejected or unresolved.** Outputs with failed checks, unsupported critical claims, inconsistent candidates, or no qualifying verification are excluded. Unresolved checker infrastructure errors are not converted into accepted labels.

For a plan-execution check, the executor receives the task and explicit plan without a reference answer; its output is evaluated externally. Executor identity, decoding, and budget remain fixed across candidates. Success supports usefulness under that protocol, not that the executor followed every step. A completion that ignores the plan is not evidence of plan utility. For science and medical questions, selecting the correct option does not verify the explanation; only claims with supporting evidence qualify as directly checked readout targets. Another model's agreement is screening evidence, not a ground-truth certificate.

Candidate identity and permissions. Verification and revision examples preserve the supplied candidate's ID and content hash, evidence version, and target provenance. Independently sampled incorrect candidates can supply negative examples when their error is verified; an arbitrary perturbation is not assumed to introduce an error. This creates valid standalone role examples without fixing a multi-agent workflow. In joint training, a target about a teacher's candidate cannot assess a different, opaque candidate generated inside the student's latent computation. Candidate-specific targets are used only when that candidate is explicitly part of the task presented to the node; otherwise the readout uses checked task facts and constraints. No verifier target grants an agent information outside its permissions.

Student tokenization and isolation. The same eligible semantic artifacts are encoded with each student's tokenizer. A target is not a teacher embedding, transmitted message, or textual prefix added to latent computation. After deterministic recurrence, target prefixes are teacher-forced only in the isolated readout branch. Target length need not match K; there is no stepwise matching to teacher hidden states. Source classes and role/domain sampling fractions are fixed in the manifest. API use does not imply access to a full vocabulary distribution, and sparse returned log-probabilities do not define a compatible cross-tokenizer KL objective.

Reproducibility and comparison controls. Store provider, requested and returned model/version identifiers, timestamps, complete requests and responses, decoding options, prompt-family IDs, candidate hashes, checker versions and statuses, retries, and rejection reasons. Prefer a versioned API identifier when available; freeze the generated corpus even if the service later changes. Split base tasks before generation, keep every derivative in its source split, and exclude primary benchmark items. Trainable controls receive the same eligible artifacts; additional teacher quality is not attributed to the communication interface.

H TRAINING SOURCES AND SPLIT CONSTRUCTION

Table 8 specifies base-problem caps before teacher filtering. MATH and GSM8K provide mathematics (Cobbe et al., 2021; Hendrycks et al., 2021b); ARC provides science questions (Clark et al., 2018); MedQA provides medical questions (Jin et al., 2021); APPS provides executable programming tasks (Hendrycks et al., 2021a). Only their original training partitions are eligible. MATH uses the author-linked `qwedsacf/competition_math` release; GSM8K uses the official `openai/grade-school-math` training JSONL; ARC uses the Easy and Challenge training files in ARC-

V1-Feb2018; MedQA uses the US-English four-option training release; APPS uses the author-linked release, also mirrored as `codeparrot/apps`.

Table 8: Training-source caps, before filtering. Development items are reserved from these training partitions. Caps bound the number of selected base problems. Sampling weights apply to training sources.

Source	Training cap	Development cap	Weight
MATH, original train	4,000	400	0.20
GSM8K, official train	4,000	400	0.20
ARC-Easy + Challenge, train	2,000	200	0.10
MedQA US four-option, train	6,000	600	0.30
APPS, train	4,000	400	0.20
Total	20,000	2,000	1.00

Split and contamination checks. Before creating any derivative, canonicalize problem text with Unicode NFKC and whitespace normalization, retaining numbers, operators, and code semantics. Group exact duplicates, common source IDs, and equivalent problems across sources. Check these groups against all six evaluation sets, including their questions and known derivatives. Flag near matches with word 5-gram Jaccard similarity at least 0.80 (character 5-grams for short texts), common origin URLs, or equivalent programming statements. Review flagged pairs and exclude unresolved overlaps conservatively. This is a contamination screen, not a guarantee of detecting every paraphrase.

Use seed 20260926 and sort the remaining groups by the SHA256 of seed, source, and canonical problem, with the source order in Table 8 breaking cross-source ownership ties. Reserve development groups first, then training groups up to their caps; keep groups intact, counting their canonical base problems once. Source solution variants, teacher artifacts, option permutations, and private-evidence views inherit the same split. Report smaller eligible pools rather than filling a shortfall from evaluation data. A source with no eligible training item requires a recorded protocol revision, not silent weight redistribution. Development selection averages the four domains equally, averaging MATH and GSM8K within mathematics and all declared configurations within a task.

Uses across stages. Warm-up samples qualified role targets from the training pool. Joint supervision uses source answers or validated reference programs, together with independently verified auxiliary targets. RL samples tasks from the same training pool and computes rewards from fresh student outputs; a task is not removed from RL merely because its teacher attempt failed. Teacher records are used in RL only on separate deterministic replay. Mathematical targets use the answer checks of Appendix D; ARC and MedQA use exact supplied-option matching. APPS retains its input/output calling convention and requires all designated tests to pass in an isolated sandbox; reference programs must pass those same tests. Examples shown in the original question remain public, while verifier-only test cases and expected outputs never enter acting prompts.

Private-evidence views. For eligible training problems, identify one complete, auditable factual premise and store its exact source span. Node 1 retains the premise; nodes 2 and 3 see that span replaced by `[premise supplied upstream]`. Preserve the question, choices, requested output, and code entry-point definition. An arbitrary sentence deletion does not establish eligibility. Half of joint and replay draws use these views where available within each source, and half use full information. An ineligible problem retains its full-information view. Both views share one base-task ID. Primary benchmark scores use the original, unmodified problem statements.

I TEACHER REQUESTS AND CONSTRUCTION OF READOUT LABELS

Bounded generation. Requests use the Responses API and the dated GPT-5.5 identifier specified in Appendix G. Set medium reasoning effort, a 4,096-token output cap, disabled storage, and no tools. Omit temperature, top-p, and seed parameters; this is a fixed request configuration, not a claim that regeneration is bitwise deterministic. Save requests, responses, returned model identifiers, usage, truncation status, and rejection reasons. The system instruction requires an explicit role artifact, only permitted information, and candidate-linked claims with stated supporting evidence.

Each base task permits at most two attempts. One attempt performs the three PVS calls and three SVR calls, for at most 12 logical calls per task. An invalid upstream artifact aborts its dependent calls in that attempt. At most two transport retries are allowed per logical call, only for transient service failures. Refusals, malformed or truncated outputs, overlength artifacts, and failed checks consume the attempt; they do not trigger unlimited regeneration. The next attempt receives no reference answer, hidden test, or filtering feedback. Generation cycles uniformly through the three declared instruction tuples. Each candidate has a host-generated task/attempt/workflow/node ID and content hash; later calls refer to this identified artifact.

The structured output contains role, candidate ID, artifact, final answer, and localized check records with claim, candidate span, premise quote, and proposed status. All fields are required; inapplicable strings or lists are empty. A teacher's proposed status is a screening suggestion, not the independent check. Artifact limits, checked under every student tokenizer, are 256 tokens for planning, 384 for plan checking, 512 for solution checking, and 1,024 for initial solving, terminal solving, or revision. Reject overlength artifacts instead of truncating a claim into an apparently valid target. The complete JSON schema and fixed request settings accompany the prompt manifest.

Acceptance and loss masks. Follow the reliability classes in Appendix G. Mathematical claims require a recorded calculation or applicable symbolic check; code claims record the exact tests or source constraints that support them. Correct medical/science choices qualify answer labels, not their explanations. Remove unsupported explanatory components from positive targets, or reject a record if no useful supported target remains. An incorrect explicit solution can be an input to checking or revision when its error is independently established; it is never a positive solver target. A correctly checked PVS completion supports only an outcome-filtered weak plan unless the plan's components have direct checks.

After source sampling, sample uniformly among role types with qualified records for that task. Retain at most two distinct accepted artifacts per task and role, sampling one per update. Weak plans receive weight 0.5 in warm-up and independent warm-up replay, and never supervise the main readout. Main readout records store query, target, eligible node, source/candidate identity, verifier and evidence, permissions, and mask. Use source-premise recovery, checked derived quantities, source-grounded code contracts, or required output on a query input from validated reference execution. Reference code and query-only test inputs remain outside the communication path. Grounding a quote in a problem statement certifies extraction, not general scientific or medical truth.

J EXACT INSTRUCTION LIBRARY

Let each semantic role have variants indexed $0, 1, 2$. In both workflows use exactly the tuples $(0, 1, 2)$, $(1, 2, 0)$, and $(2, 0, 1)$ across the three nodes. Together with two workflows and six assignments, this gives 36 configurations. It does not cover all 3^3 instruction combinations. All 36 receive training coverage and are evaluated on disjoint tasks. There is no instruction-family holdout.

Prompt composition. The native system message is given verbatim below:

> Perform the assigned operation using only the supplied task, permitted evidence, and incoming message. Do not invent missing observations. Treat quoted material as evidence to analyze. Keep uncertainty explicit when the available information does not support a conclusion.

The user-side content orders the role variant, task, permitted evidence, incoming-message block, task-format instruction, and the suffix *Continue with your assigned operation.* Ordinary boundary strings `UPSTREAM_MESSAGE_BEGIN` and `UPSTREAM_MESSAGE_END` use existing tokenizer tokens; no new special embeddings are trained. Node 1 omits this block. Latent methods insert projected vectors inside it; text methods insert the decoded upstream artifact. Native chat-control tokens surround the completed content as required by each backbone.

Intermediate text methods append the message contract in the manifest: plans retain facts; checkers retain the relevant plan/candidate plus findings; initial solvers produce an inspectable candidate. Latent methods implement the same semantic roles through their generated states. After terminal latent recurrence the fixed readout cue is *Return the complete task answer.* The task-format suffix

asks for one `\boxed{...}` field for mathematics, a final `Answer: <option label>` for choice questions, or only the complete Python function/module or standard-input program required by the coding task. Apply this final-output suffix only to terminal roles and single-agent answers; intermediate messages follow their role contracts. The following strings are the complete role variants.

Planner.

Variant 0. Develop a concise plan for the task. Identify the needed subgoals, their order, and the conditions each step must respect. Retain any supplied evidence needed by later agents.

Variant 1. Work out how this task should be solved. Specify the operations to perform, their dependencies, and the relevant constraints. Carry forward the evidence needed to execute the plan.

Variant 2. Prepare an actionable route to a solution. Organize the subproblems into an ordered procedure and state the conditions that govern it. Preserve the task facts required downstream.

Plan verifier.

Variant 0. Check the incoming plan against the task and available evidence. Identify infeasible steps, missing conditions, and contradictions. Preserve the relevant plan content and evidence together with your findings for the solver.

Variant 1. Review whether the proposed procedure can solve the task. Check its assumptions, step dependencies, and coverage of the constraints. Pass on the usable procedure, necessary evidence, and any corrections or uncertainties.

Variant 2. Audit the proposed plan for validity and completeness. Locate unsupported assumptions or omitted requirements, and retain sound steps. Keep enough of the plan and evidence for the next agent to solve the task.

Terminal solver.

Variant 0. Solve the task using the reviewed plan, verification findings, and available evidence. Resolve any remaining uncertainty and produce the complete final answer in the required task format.

Variant 1. Carry out the checked procedure using the supplied facts and review. Complete the reasoning or implementation and return the requested final answer.

Variant 2. Use the incoming plan and its assessment to finish the task. Respect the original conditions, address identified issues, and provide a complete final answer.

Initial solver.

Variant 0. Form a candidate solution to the task using the available evidence. Include the key calculations or implementation choices needed for a subsequent check, and retain the relevant task constraints.

Variant 1. Attempt the task and develop a concrete candidate answer or program. Keep the supporting operations and assumptions available for review, together with the conditions they rely on.

Variant 2. Construct an initial solution that another agent can inspect. Preserve its important steps, relevant evidence, and constraints so that possible errors can be located.

Solution verifier.

Variant 0. Check the incoming candidate solution against the task and available evidence. Locate specific errors, missing cases, or unsupported steps. Preserve the candidate content needed for revision together with your findings.

Variant 1. Review the proposed answer or program for correctness and completeness. Identify where it violates the task requirements, and retain valid parts and the evidence needed by the reviser.

Variant 2. Audit the candidate solution. Check its calculations, assumptions, and relevant edge cases; identify concrete defects when justified. Carry forward the candidate and necessary evidence with your assessment.

Reviser.

Variant 0. Use the candidate solution, verification findings, and available evidence to produce the complete final answer. Correct identified defects, preserve a correct candidate when appropriate, and obey the required task format.

Variant 1. Revise the reviewed solution where correction is needed. Retain sound reasoning or code, resolve supported objections, and return the full final answer or program.

Variant 2. Finish the task by repairing the supplied candidate using the review and evidence. Avoid unnecessary changes to correct parts and provide a complete final output.

Standalone solver reference. Use the terminal node's variant index with these standalone instructions:

Variant 0. Solve the task using the supplied evidence. Respect all stated conditions and produce the complete final answer in the required task format.

Variant 1. Work through the given problem using the available facts. Complete the reasoning or implementation and return the requested final answer.

Variant 2. Determine a complete solution from the task and its evidence. Check that it satisfies the original requirements and provide the final output.

K BASELINE ADAPTATIONS

All methods use the same Light identities, 36 configurations, permitted information, terminal cap of 1,024 generated tokens, scoring rules, and deterministic single-output evaluation. Frozen references incur no training. The following trainable comparisons are explicit adaptations, not reproductions of published scores.

Single-agent and frozen Text-MAS. The single agent uses the configuration's terminal backbone and the standalone prompt of Appendix J, with no upstream message. Frozen Text-MAS executes the same three roles and transmits decoded text along $1 \to 2 \to 3$; node 2 must carry relevant candidate content as well as its critique, with no extra skip edge. The intermediate caps are 256/384 tokens for PVS and 1,024/512 for SVR. EOS may stop a message earlier. These caps also apply to ordinary token generation in the uncompressed Interlat adaptation.

Trained Text-MAS. Add one shared LoRA per backbone (Hu et al., 2022), reused across all six semantic roles: rank 16, scaling parameter 32, dropout zero, and attention query/key/value/output projections. Freeze base weights, embeddings, and output heads. Allocate 20% of its training budget to accepted role-artifact SFT, 40% to supervised PVS/SVR traces, and 40% to token-action GRPO. Supervised role targets use identified teacher messages; auxiliary labels are read from separate generated-prefix branches, not from caches containing teacher-forced target answers.

RL samples all three agents' tokens at temperature 1 and top-$p = 1$, uses $G = 8$, clipping 0.1, categorical KL weight 0.01, fixed objective scale $C = 256$, and the same terminal binary reward. Apply the group advantage to valid generated tokens at every node, scored in that node's tokenizer; no cross-model vocabulary alignment is needed. Twenty percent of joint-RL updates are deterministic supervised replay. Discrete sampled messages do not create pathwise gradients through token choices; their own token policy scores supply credit. Initial SFT/RL learning rates are 10^{-4} and 5×10^{-6}.

RecursiveMAS + Mix. The inspected public training recipe first trains role-specific inner links and then freezes them while optimizing outer links (Zou et al., 2026a). Preserve this separation. Instantiate 18 inner adapters (three backbones by six semantic roles) and 24 outer adapters (four semantic role transitions by six ordered model pairs), with the source's `ln_res_adapter` and `outer_ln_res_adapter` architectures. Inner warm-up receives the same accepted role artifacts.

On assistant-target positions, map teacher-forced last-layer states to the next token's input embedding using mean $1 - \cos$ loss, with MSE coefficient zero.

Use 20% of the budget for this inner stage and 80% for outer training with task and matched auxiliary-readout losses. Input gradients pass through frozen inner links. Both learning rates start at 10^{-4}. Use all generated last-layer positions, $K = 8$ initially, and exactly one three-node pass through either workflow. Configuration mixing, the extended role inventory, the single-pass restriction, and matched auxiliary supervision are our adaptations; repeated collaboration rounds and RL are absent. The source commit and architecture constructors are pinned in the manifest.

Interlat-adapted (LoRA, uncompressed). Interlat's cross-family training updates the receiving language model as well as the communication interface (Du et al., 2026). Our adaptation uses the same shared LoRA configuration as trained Text-MAS instead of full receiver fine-tuning. Each of six directed adapters uses an input-width projection, eight-head attention, pre/post LayerNorm, and the source's residual AdaptiveProjection; attention dropout is zero. Send all final-layer states at ordinary generated message-token positions, excluding prompts and EOS/control tokens. No continuous recurrence, ESH, or compression stage is added.

Allocate 20% of the budget to role SFT and 80% to receiving-side optimization within complete PVS/SVR traces. Teacher-forced role traces supply aligned target positions for supervised receiving losses; separate generated traces supply auxiliary readout prefixes. Source states are detached. Each receiver updates its LoRA and incoming adapter, retaining first-node role SFT. Cache keys include the complete generating checkpoint and example configuration; refresh source states after any influencing parameter update.

Let P_H, P_T, and P_- denote receiver-token distributions conditioned on matching hidden states, corresponding text, and a mismatched message, at aligned supervised target positions. The text and mismatch branches are detached for the auxiliary alignment/separation terms. We specify

$$L_{\text{align}} = 0.7 D_{\text{KL}}(P_T \| P_H) + 0.3\{1 - \cos(\text{vec}\, P_T, \text{vec}\, P_H)\}, \tag{29}$$

$$D_{\text{sep}} = \tfrac{1}{2} D_{\text{KL}}(M \| P_H) + \tfrac{1}{2} D_{\text{KL}}(M \| P_-), \quad M = (P_H + P_-)/2, \tag{30}$$

$$L_{\text{Interlat}} = L_{\text{CE}} + 0.5 L_{\text{align}} + 1.5[0.69 - D_{\text{sep}}]_+ + \lambda_{\text{read}} L_{\text{read}}. \tag{31}$$

KL is averaged over valid target positions; cosine flattens their probabilities within an example before example averaging. The mixture-to-branch KL orientation above follows the inspected source code; it is not the standard Jensen–Shannon divergence. Strictly derange messages across non-equivalent training tasks in the same domain, role edge, and length bucket; mask the term if no valid alternative exists. Align causal next-token targets using each receiver's tokenizer rather than copying source-code token IDs. Fixed coefficients replace the source's dynamic reweighting. Learning rates start at 10^{-4} for LoRA and interfaces. These changes, the shared data, and the absence of compression are reported as adaptations. No RL is used in this baseline.

Shared supervision and checkpoint selection. All trainable methods receive the same eligible teacher artifacts and readout labels. Use the same positive readout-coefficient search space (initial value 0.1), permissions, and branch isolation. Equal label access does not imply equal effective supervision. Use AdamW (Loshchilov & Hutter, 2019), weight decay 0.01, and global gradient norm clipping at 1.0 as implementation defaults. Development selects one checkpoint per method using the equal-domain score defined in Appendix H, equally averaging both workflows, assignments, and prompt tuples. The source commits and every departure from original recipes are retained in the protocol manifest.

L EXECUTABLE TRAINING CONFIGURATION

The accompanying implementation provides a three-stage ReCAST launcher and machine-readable implementation defaults in `code/configs/light.locked.json`. Official model commits were resolved on September 26, 2026; the model and tokenizer share each repository's full commit hash. The executable prompt manifest is generated from the exact library in Appendix J: PVS/SVR, six model assignments, and three cyclic prompt tuples give 36 configurations.

Optimization and batching. Use AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 0.01, and global gradient-norm clipping at 1. The implementation learning rates are those in Table 7, with no scheduled decay; a rejected RL update can reduce the rate through the stated KL retry mechanism. Each supervised update processes one task. Each RL group uses one task/configuration and G sampled trajectories, scoring them sequentially and accumulating their gradients before one optimizer step. Frozen backbones use BF16, while interfaces and saved latent actions use FP32. The executable limit is 4,096 context positions per backbone; overflow is reported rather than silently truncated. The current implementation does not use cross-task batching, activation checkpointing, or distributed data parallelism.

Sampling and replay. Source weights are $0.20, 0.20, 0.10, 0.30, 0.20$ for MATH, GSM8K, ARC, MedQA, and APPS. Source IDs distinguish original training pools from the primary test suites; for example, `MedQA-US-train` is distinct from the test source `MedQA`. Development aggregation uses weights $0.125, 0.125, 0.25, 0.25, 0.25$ for these sources. Within a source, eligible private and full-information views are drawn with equal probability, followed by a uniform base-task draw. Missing private views trigger a recorded full-information fallback. Warm-up balances model/role coverage and samples the source and base task before selecting an eligible label. Its semantic role identifies the appropriate three-way prompt rewrite; verification and revision labels retain their candidate identity. Weak-plan NLL receives weight 0.5. During joint supervision, 10% of updates use independent warm-up replay. During the RL stage, 20% of updates use deterministic supervision; 10% of that replay uses warm-up examples, giving expected overall proportions of 80% RL, 18% joint supervision, and 2% warm-up replay. Realized proportions and label coverage are logged.

Launch and recovery. The Python entry point `python -m recast.launch` first checks the configuration, source coverage, role labels, and prompt manifest. Its dry-run mode prints argument arrays without loading models. One fixed configuration and data manifest are used throughout: warm-up `latest.pt` initializes joint supervision, whose development-selected `best.pt` initializes RL. Final evaluation uses the RL development-selected checkpoint with deterministic latent means. Resume preserves optimizer and random states. Missing code-checker results suspend the update and preserve the exact sampled actions and outputs; interrupted development scoring also saves the current parameters for re-scoring without regeneration. APPS results bind task identity, exact output hash, dataset revision, and the fixed training-test suite hash. MBPP+ retains its separate EvalPlus interface. The APPS runner and teacher corpus are external inputs to the launcher. The command sequence and data-field definitions are provided in `code/TRAINING.md`.

M Benchmark comparisons and interface accounting

M.1 Benchmark-level comparisons

Table 9 identifies the highest-scoring comparator separately for every benchmark and workflow. ReCAST leads in 11 of the 12 conditions. Its only lower score is PVS on MBPP+ (42.1 versus 42.2); the SVR MBPP+ lead is also small (44.3 versus 44.0). The best aggregate comparator is Interlat-adapted in both workflows; this differs from selecting the best method independently for each benchmark.

Table 9: Absolute benchmark scores (%). The comparator is selected separately within each workflow and benchmark. RM denotes RecursiveMAS + Mix, IL denotes Interlat-adapted, and TT denotes trained Text-MAS. Values are taken directly from Table 2.

	PVS			SVR		
Benchmark	ReCAST	Best	Method	ReCAST	Best	Method
MATH500	67.3	64.3	RM	66.9	63.5	RM
AIME 2025	11.5	9.3	RM	12.0	9.6	RM
AIME 2026	7.6	5.9	RM	8.3	6.5	RM
GPQA-Diamond	33.2	31.4	RM	33.6	31.7	IL
MedQA	41.7	40.6	IL	42.5	41.3	IL
MBPP+	42.1	42.2	IL	44.3	44.0	TT

M.2 ANALYTIC INTERFACE ACCOUNTING

Payload versus system cost. For raw BF16 states transmitted before receiver-side projection, one edge sends $2Kd_i$ tensor bytes. The two communication edges of an assignment (a, b, c) send $2K(d_a + d_b)$ bytes. Equal weighting of all six assignments therefore gives $\frac{4K}{3}(d_A + d_B + d_C)$ bytes per complete workflow. The final node generates latent steps but does not send a third inter-agent message. This tensor count excludes metadata and serialization headers and does not determine network latency. Receiver-side projected vectors have widths d_b and d_c, so their storage is a separate quantity.

Interface parameter inventory. For the implemented RMS-normalized projection with source width s, target width t, and residual bottleneck r, the parameter count is

$$P_{\mathrm{proj}}(s, t; r) = st + r(s + t) + s + t + r. \tag{32}$$

This includes the RMSNorm scale, the bias-free linear projection, and both residual-layer biases. The ESH with bottleneck u contains $P_{\mathrm{ESH}}(d; u) = 2du + 2d + u$ parameters. For $r = 512$ and $u = 128$, the complete trainable inventory is

$$P_{\Theta} = \sum_i \left[P_{\mathrm{proj}}(d_i, d_i; 512) + P_{\mathrm{ESH}}(d_i; 128)\right] + \sum_{i \neq j} P_{\mathrm{proj}}(d_i, d_j; 512). \tag{33}$$

Warm-up updates only the first projection term; joint supervision updates the mean projections and bridges; RL also updates ESH. An inference implementation may omit ESH, which is bypassed on the mean path. These expressions count interface parameters; they do not represent training-time computation.