Dyad: EXTENDING LARGE LANGUAGE MODELS WITH NATIVE TYPED DECISION-MAKING

Anonymous authors
Paper under double-blind review

ABSTRACT

We study how to build more capable general-purpose agents by extending large language models (LLMs) with native typed decision-making. We introduce **Dyad**, an architecture that augments a pretrained LLM with an environment-conditioned action encoder that embeds each candidate action description in parallel, then scores these embeddings against the LLM's internal state to yield a distribution over typed actions. By factorizing decision-making into representations of the evolving interaction state and environment-specific action semantics, Dyad introduces an inductive bias for learning reusable representations while keeping action scoring efficient even as the action space grows. We investigate two complementary reinforcement learning settings driven by environment interaction. With the LLM frozen, training the action encoder alone achieves consistent gains across four unseen environments, enabling modular adaptation without modifying any LLM parameters. Jointly optimizing both components outperforms conventional RL post-training across diverse interactive tasks and model scales, including a 3.80% average absolute gain on ALFWorld with a 9B model, while improving general knowledge, reasoning, and coding.

1 INTRODUCTION

Solving tasks through interaction with an environment requires two fundamentally different forms of computation (Yao et al., 2023): deliberate reasoning about goals or plans, and typed decision-making over available actions, analogous to Kahneman's System 2 and System 1, respectively (Kahneman, 2011). For example, a shopping agent must reason about a customer's requirements and compare products across web pages (System 2), while repeatedly selecting from the available actions in the active page to search, inspect products, choose options, and complete a purchase (System 1).

Today, LLM-based agents typically handle both forms through autoregressive generation, *i.e.*, reasoning in language and generating text to specify actions, as illustrated in the middle of Figure 1. However, this overloads the language-generation interface: the LLM assigns probabilities to vocabulary tokens rather than directly to permissible typed actions (Yue et al., 2025). Recent dedicated System 1 models, such as Jev (Almeida, 2026) and its open-source alternatives (Nandakishor, 2026; Featherless AI, 2026), address this limitation by predicting typed decisions directly and in parallel, rather than generating their textual representations token by token, as shown on the left of Figure 1.

Yet language reasoning and typed decision-making need not be developed independently, as systems 1 and 2 are complementary modes of cognition rather than distinct physical systems (Kahneman, 2011): reasoning guides decisions, while decision outcomes inform subsequent reasoning. Recent works (Yue et al., 2025; Zhao et al., 2026) take a step toward this integration by extending LLMs with direct environment actions, enabling interleaved language reasoning and decision-making. However, they are limited to fixed, pre-defined action spaces, and do not generalize to dynamically provided actions in unseen environments (*e.g.*, unseen web pages). This raises a natural question: can pretrained LLMs support native typed decision-making over dynamic action spaces, and can jointly learning reasoning and decision-making yield more capable agents?

We introduce **Dyad**, an architecture that extends pretrained LLMs with *native typed decision-making* while retaining their autoregressive language reasoning capabilities, as illustrated on the right of Figure 1. Specifically, we augment the pretrained LLM with an action encoder that embeds available action descriptions independently and in parallel, conditioned on the environment. Because these

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

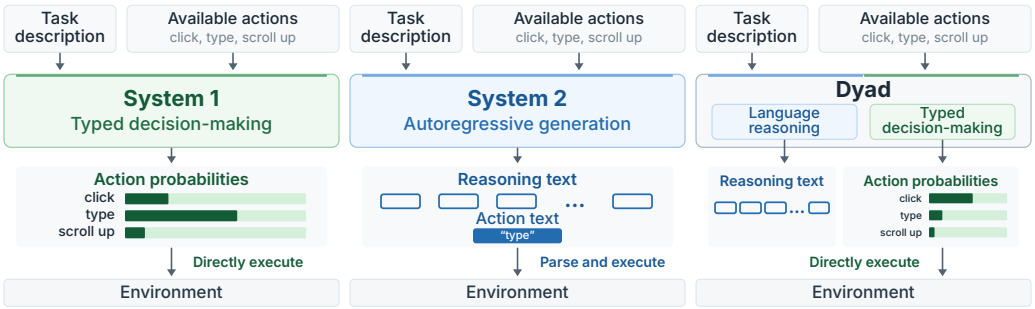

Figure 1: Conceptual comparison of System 1, System 2, and Dyad for reasoning and acting.

action representations depend on the environment and action semantics, but not on the evolving interaction state, they can be precomputed and reused across decision steps whenever the action definitions remain unchanged. When a decision is required, these embeddings are scored against the LLM's internal state representation to produce a distribution directly over the available typed actions. This factorization separates the evolving interaction state from environment-specific action semantics, providing an inductive bias for learning reusable representations rather than an entangled state–action mapping. It further amortizes action encoding across interactions, enabling efficient decision-time scoring and adaptation to previously unseen action spaces.

Dyad is trained in two complementary settings, both driven by end-to-end rewards from environment interaction. First, we freeze the pretrained LLM and train only the action encoder through reinforcement learning. The resulting agent consistently improves performance across four unseen environments, demonstrating that native typed decision-making can enhance existing LLMs without modifying their pretrained parameters. Second, we jointly optimize the LLM and action encoder on diverse interactive tasks. Dyad outperforms agentic RL baselines across model scales. On ALFWorld, it achieves a 3.80% average absolute gain with Qwen3.5-9B. Dyad also better preserves general knowledge, reasoning, and coding capabilities.

In summary, our contributions are threefold:

- We introduce Dyad, to our knowledge the *first* architecture to extend pretrained LLMs with native, parallel typed decision-making over dynamically provided action spaces.
- We develop an end-to-end inference and training pipeline integrated with vLLM (Kwon et al., 2023) and verl (Sheng et al., 2025), which we plan to open-source.
- We show that Dyad improves agent performance across both frozen-LLM adaptation and joint optimization, generalizes across environments and model scales, and yields broad gains in general knowledge, reasoning, and coding.

2 Dyad: Native Typed Decision-Making in Pretrained LLMs

2.1 Problem Formulation

Setting. We consider an LLM agent that interleaves language generation with direct interaction in an external environment e. At step t, the environment exposes a set of permissible typed actions $\mathcal{A}_t$, which may change over time. Each $a \in \mathcal{A}_t$ is an environment-defined operation, potentially with arguments, that can be executed directly. This setting encompasses both direct decisions, such as routing a support request to a team, and tasks combining language reasoning with action, such as navigating unfamiliar software.

Policy. The agent observes a language history h_t, comprising previously generated tokens and textual observations from its interactions. Its policy π_θ supports two modes: language generation, with a distribution $\pi_\theta^{\mathrm{LM}}(v \mid h_t, e)$ over the vocabulary $\mathcal{V}$, and typed decision-making, with a distribution $\pi_\theta^{\mathrm{act}}(a \mid h_t, e, \mathcal{A}_t)$ over the currently permissible actions $\mathcal{A}_t$.

Interaction. Generating a token $v \in \mathcal{V}$ extends the language history, whereas selecting a typed action $a \in \mathcal{A}_t$ invokes the corresponding environment operation. The resulting textual observation is appended to the history; for example, an `open(fridge)` action may return a description of the

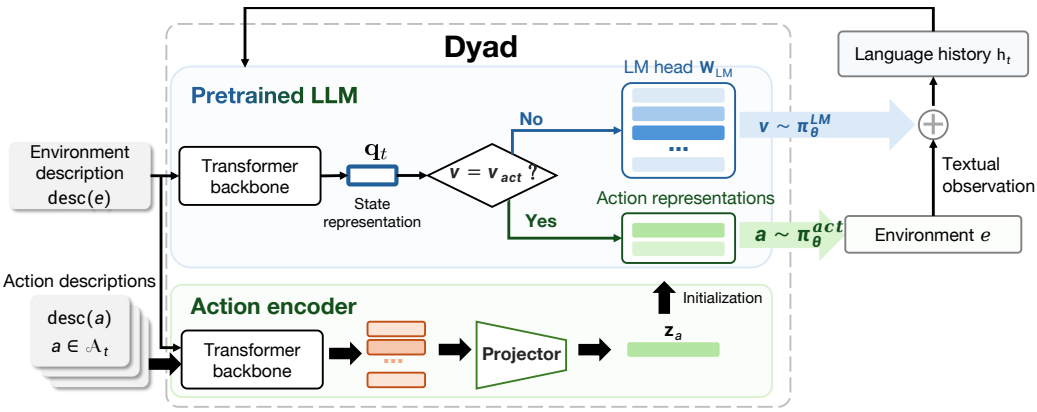

Figure 2: **Dyad architecture.** State and environment-conditioned action representations are computed independently. Generating the interaction token v_{act} triggers typed action selection; otherwise, the pretrained LM head generates language tokens.

fridge's contents. The environment provides rewards based on task outcomes, such as successfully retrieving a requested item from the fridge. The agent's objective is to maximize the expected cumulative reward from environment interaction.

To realize such a policy, existing approaches often jointly encode the interaction state and available actions, entangling action semantics with the evolving state (Nandakishor, 2026; Lee, 2026). We introduce Dyad, which instead factorizes typed decision-making into pretrained LLM state representations within a single architecture (Section 2.2), followed by its learning procedure (Section 3).

2.2 Architecture

Dyad extends a pretrained LLM with an action encoder, enabling language generation and typed decision-making within a single policy. Let f denote the pretrained LLM's transformer backbone, $\mathbf{W}_{\text{LM}}$ its language modeling head, and g the action encoder. Given the interaction history h_t and environment description $\text{desc}(e)$, the two components produce a state representation $\mathbf{q}_t$ and an environment-conditioned representation $\mathbf{z}_a$ for each available action $a \in \mathcal{A}_t$:

$$\mathbf{q}_t = f(\text{desc}(e) \oplus h_t), \qquad \mathbf{z}_a = g(\text{desc}(e) \oplus \text{desc}(a)), \tag{1}$$

where $\text{desc}(\cdot)$ denotes textual description and $\oplus$ denotes concatenation. Dyad retains the pretrained language modeling head to generate tokens according to

$$\pi_\theta^{\text{LM}}(v \mid h_t, e) = \text{softmax}\big(\mathbf{W}_{\text{LM}}\mathbf{q}_t\big)_v. \tag{2}$$

Upon generating a dedicated interaction token $v_{\text{act}} \in \mathcal{V}$, Dyad selects a typed action by scoring each available action against the state representation:

$$\pi_\theta^{\text{act}}(a \mid h_t, e, \mathcal{A}_t) = \text{softmax}\big(\mathbf{Z}_t\mathbf{q}_t\big)_a, \qquad \mathbf{Z}_t = \begin{bmatrix}\mathbf{z}_{a_1} & \cdots & \mathbf{z}_{a_{|\mathcal{A}_t|}}\end{bmatrix}^\top. \tag{3}$$

The key design choice in Dyad is to factorize the computation of $\pi_\theta^{\text{act}}(a \mid h_t, e, \mathcal{A}_t)$ into a state representation $\mathbf{q}_t$ and an environment-conditioned action representation $\mathbf{z}_a$ with Eq. 1. This encodes the inductive bias that the evolving interaction state can be represented independently of the currently available actions, while an action's meaning depends on its environment rather than the current state. The action encoder therefore learns to express action semantics in the pretrained LLM's representation space, enabling their compatibility to be measured directly. Moreover, independent action encoding makes the decision policy permutation-equivariant: reordering the available actions simply reorders their probabilities. We detail the two representations below.

State Representation. We take the pretrained LLM's hidden state at the current decision position as $\mathbf{q}_t$. Pretrained for next-token prediction, the backbone provides contextual features that support both language generation and typed decision-making.

Action Representation. We implement g as a transformer that maps the concatenated environment and action descriptions to an embedding compatible with $\mathbf{q}_t$. Action embeddings are computed

independently and in parallel, and can be reused across decision steps as long as their descriptions and environment context remain unchanged.

Amortized Action Encoding. Action representations can be computed once and reused whenever their action descriptions and environment context remain unchanged. Newly introduced actions need only be encoded when first encountered. Consequently, steady-state decision-time inference requires only the pretrained LLM and cached action representations.

3 Learning Dyad from Environment Interaction

We first initialize the action encoder to align its representations $\mathbf{z}_a$ with the LLM's state representations $\mathbf{q}_t$, then optimize Dyad through reinforcement learning (RL) from environment interaction.

3.1 Action Encoder Initialization

The action encoder g consists of a pretrained transformer backbone, separate from f, and a projector that maps its outputs to action representations $\mathbf{z}_a$ with the same dimensionality as $\mathbf{q}_t$. To initialize the projector, we synthesize a small collection of action-selection examples $\mathcal{D} = \{(h_i, e_i, \mathcal{A}_i, a_i^\star)\}_{i=1}^N$. For each example, we sample a permissible action set $\mathcal{A}_i$ for environment e_i and select a target action $a_i^\star \in \mathcal{A}_i$. A teacher LLM then generates task context and reasoning h_i that support selecting $a_i^\star$. We initialize the projector by minimizing the cross-entropy loss, while keeping f, $\mathbf{W}_{\mathrm{LM}}$, and the action encoder backbone frozen:

$$\mathcal{L}_{\mathrm{align}} = -\mathbb{E}_{(h,e,\mathcal{A},a^\star)\sim\mathcal{D}} \left[\log \pi_\theta^{\mathrm{act}}\big(a^\star \mid h, e, \mathcal{A}\big) \right]. \tag{4}$$

This lightweight initialization aligns the two representation spaces before learning from environment rewards. Further details are provided in Appendix A.2.

3.2 Agentic Reinforcement Learning

Starting from the initialized model, we optimize Dyad using rewards from environment interaction. Trajectories interleave language generation and typed decision-making under the policy defined in Eqs. 2 and 3. We apply GRPO or GiGPO using task-outcome rewards, treating sampled vocabulary tokens and typed actions as decisions under their respective policy distributions. We consider two training settings. In joint optimization, we update both the pretrained LLM and the entire action encoder. In frozen-LLM adaptation, we keep f and $\mathbf{W}_{\mathrm{LM}}$ fixed and update only g, including its backbone and projector. Further optimization details are provided in Appendix A.3.

4 Related Work

Language-Based Action Interfaces. LLM agents commonly interact with external environments by generating textual action specifications, as in reasoning and tool-use agents (Yao et al., 2023; Schick et al., 2023). Program-guided methods use generated code to invoke external computation (Gao et al., 2023; Gou et al., 2024), while interactive agents extend this interface to web navigation and multi-step environment control (Zhou et al., 2024; Liu et al., 2024). These approaches offer a flexible interface, but express environment actions through vocabulary-token prediction rather than a policy directly over permissible typed actions.

Typed Decision-Making. Dedicated decision models such as Jev (Almeida, 2026) predict distributions directly over provided choices. As summarized in Table 1, such models admit several architectural parameterizations. Joint action-set encoders, as in Laya (Nandakishor, 2026), process the interaction state and available actions together, producing decision scores for all actions in parallel. AR encoders instead process each state–action pair to assign a scalar score, as exemplified by the reranker-based approach in SemIf (Lee, 2026). Single-token AR methods assign vocabulary identifiers to available choices and score them using the LLM's language modeling head, as in Simple Jev (Featherless AI, 2026). However, these approaches entangle action scoring with the evolving interaction state or a fixed vocabulary basis.

Native Actions in Pretrained LLMs. Beyond dedicated decision models, recent work integrates language reasoning and environment actions within pretrained LLMs. ToolkenGPT (Hao et al.,

Table 1: Conceptual comparison of methods for typed decision-making. F denotes a model jointly processing its specified inputs, and $\mathrm{id}(a)$ maps an available action to a vocabulary identifier.

Approach	Logit for $a \in \mathcal{A}_t$	Action interface
Joint action-set encoder	$F_a(e, h_t, \mathcal{A}_t)$	Joint scoring
AR encoder	$F(e, h_t, a)$	Independent scoring
AR Single-token	$\mathbf{w}_{\mathrm{id}(a)}^{\top} F(e, h_t, \mathcal{A}_t)$	Fixed token identifiers
Dyad	$f(e, h_t)^{\top} g(e, a)$	Compositional scoring

2023) introduces learned tool embeddings into the language modeling head, enabling tool invocation through autoregressive generation of special tokens. ExpA (Yue et al., 2025) goes beyond the vocabulary-token interface by extending LLMs with directly executable actions, enabling interleaved language reasoning and native decision-making. However, both rely on predefined action spaces. Dyad extends this direction by factorizing typed decision-making into pretrained LLM state representations and independently encoded, environment-conditioned action representations, constructing a native decision head over dynamically provided actions.

5 Experiments

5.1 Experimental Setup

Benchmarks. We perform agentic RL separately on DIVE (Chen et al., 2026) and CodeGym (Du et al., 2026), and evaluate the resulting agents on ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022), τ^2-bench (Barres et al., 2026), and SWE-bench Verified (SWE) (Jimenez et al., 2024) without further training on these environments. We assess general knowledge, mathematical reasoning, and coding with MMLU-Pro (Wang et al., 2024), HMMT February 2026, and LiveCodeBench v6 (Jain et al., 2025), respectively. We also train agents separately on ALFWorld and WebShop to evaluate Dyad's in-domain gains over the corresponding RL baselines.

Baselines. We compare Dyad-GRPO with GRPO (Shao et al., 2024) and Dyad-GiGPO with GiGPO (Feng et al., 2025), using the same LLM and RL training environment for each pair. We also evaluate each model without additional training as a reference for agent performance and general capabilities.

Implementation Details. For joint training, we use Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-27B (Qwen Team, 2026). For frozen-LLM adaptation, we use the 4B and 9B Qwen models, Llama-3.2-3B-Instruct (Meta, 2024), and Gemma-4-E2B-it (Abd et al., 2026). Dyad uses attention pooling as the default projector and follows the alignment and agentic RL procedure in Section 3. We implement the training and inference pipeline with verl (Sheng et al., 2025) and vLLM (Kwon et al., 2023). Further implementation details are provided in Appendices A and B.

5.2 Agentic Post-training

In-Domain Performance.

When training and evaluation use the same environment, Dyad improves both GRPO and GiGPO on Qwen3.5-9B across ALFWorld and WebShop, outperforming the corresponding baseline in all four comparisons (Table 2). The largest absolute gain is 5.89% on ALFWorld with GRPO. GiGPO already outperforms GRPO in both environments, and Dyad provides further improvements over this stronger baseline.

Table 2: **In-domain performance of Qwen3.5-9B.** Evaluations use 3 seeds.

Method	ALFWorld	WebShop
Base model	45.66	33.43
+ GRPO	85.01	73.44
+ Dyad-GRPO	**90.90**	**75.17**
+ GiGPO	89.32	81.52
+ Dyad-GiGPO	**91.02**	**83.65**

Table 3: **Cross-environment performance.** Agents are evaluated without further training. Superscripts show absolute changes (%) from the corresponding RL baseline. ALF denotes ALFWorld. Evaluations use 3 seeds.

	DIVE				CodeGym			
Method	ALF	WebShop	τ^2	SWE	ALF	WebShop	τ^2	SWE
Qwen3.5-4B	23.45	28.54	61.05	43.21	23.45	28.54	61.05	43.21
+ GRPO	29.71	30.42	60.52	**45.67**	32.18	33.65	64.10	**47.98**
+ Dyad-GRPO	**31.52**$^{+1.81}$	**31.22**$^{+0.80}$	**64.54**$^{+4.02}$	44.28$^{-1.39}$	**33.65**$^{+1.47}$	**35.71**$^{+2.06}$	**65.26**$^{+1.16}$	45.02$^{-2.96}$
+ GiGPO	28.22	29.54	61.84	45.02	31.09	27.97	63.48	46.64
+ Dyad-GiGPO	**33.07**$^{+4.85}$	**32.45**$^{+2.91}$	**61.86**$^{+0.02}$	**47.10**$^{+2.08}$	**31.72**$^{+0.63}$	**31.21**$^{+3.24}$	**64.78**$^{+1.30}$	**49.61**$^{+2.97}$
Qwen3.5-9B	45.66	33.43	62.17	44.56	45.66	33.43	62.17	44.56
+ GRPO	52.67	34.52	62.06	**47.22**	51.55	36.23	61.31	49.40
+ Dyad-GRPO	**53.82**$^{+1.15}$	**36.31**$^{+1.79}$	**64.88**$^{+2.82}$	46.20$^{-1.02}$	**55.13**$^{+3.58}$	**38.07**$^{+1.84}$	**63.02**$^{+1.71}$	**50.62**$^{+1.22}$
+ GiGPO	50.98	36.23	**63.52**	48.83	52.08	**36.58**	64.21	49.78
+ Dyad-GiGPO	**52.06**$^{+1.08}$	**37.02**$^{+0.79}$	61.90$^{-1.62}$	**50.42**$^{+1.59}$	**55.01**$^{+2.93}$	35.10$^{-1.48}$	**65.85**$^{+1.64}$	**51.25**$^{+1.47}$
Qwen3.5-27B	68.45	44.56	68.21	66.78	68.45	44.56	68.21	66.78
+ GRPO	69.25	46.68	69.82	68.92	**71.89**	42.13	69.29	67.91
+ Dyad-GRPO	**71.42**$^{+2.17}$	**48.81**$^{+2.13}$	**71.46**$^{+1.64}$	**70.24**$^{+1.32}$	71.11$^{-0.78}$	**48.29**$^{+6.16}$	**70.10**$^{+0.81}$	**68.34**$^{+0.43}$
+ GiGPO	69.84	**47.20**	69.56	68.02	70.79	45.91	69.51	67.90
+ Dyad-GiGPO	**72.30**$^{+2.46}$	45.44$^{-1.76}$	**71.87**$^{+2.31}$	**71.53**$^{+3.51}$	**73.21**$^{+2.42}$	**47.72**$^{+1.81}$	**72.30**$^{+2.79}$	**68.98**$^{+1.08}$

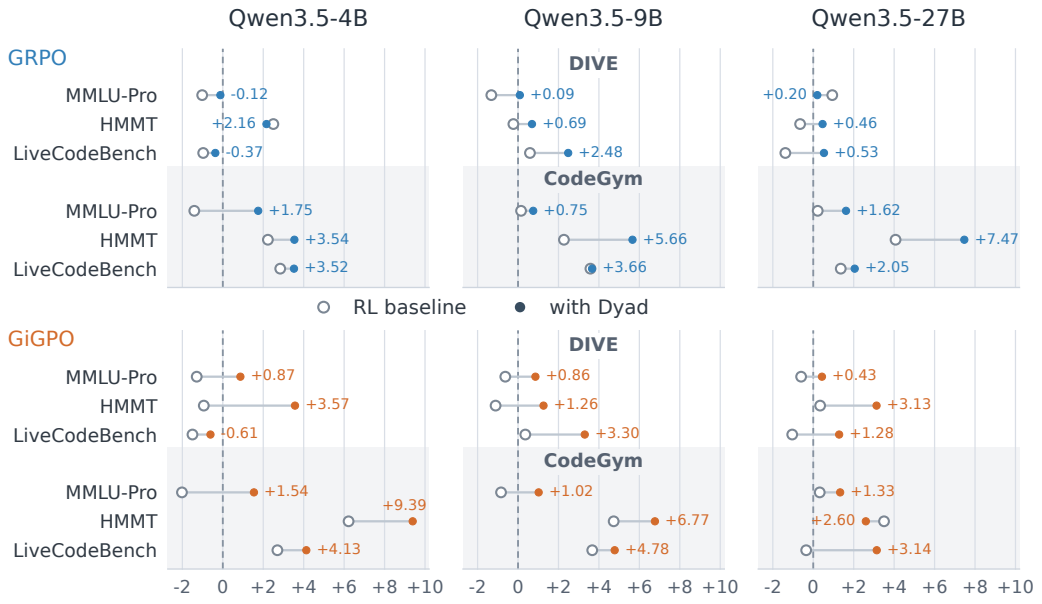

Figure 3: **General capabilities.** Dashed lines indicate zero change relative to each model without additional training. Evaluations are on 3 seeds.

Cross-Environment Performance. When trained on DIVE or CodeGym and evaluated without further adaptation, Dyad outperforms the corresponding RL baseline in 41 of 48 model–algorithm–environment comparisons (Table 3). It also improves over the original pretrained model in 47 of 48 evaluations, with gains spanning both training environments and all three model scales. These results show that the benefits of Dyad are not confined to the environment used for agentic RL.

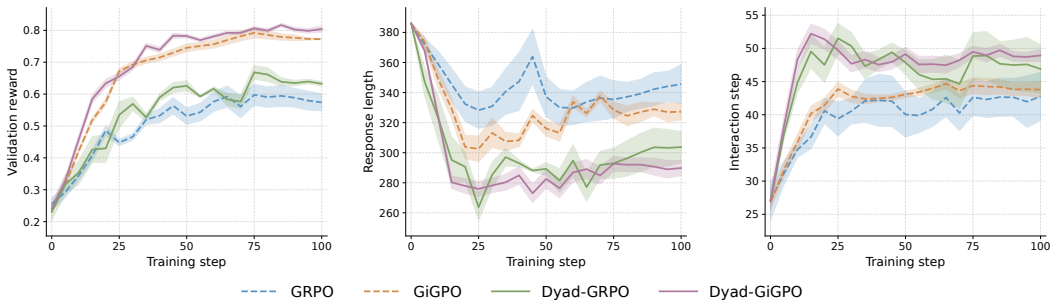

Figure 4: **Training dynamics.** Curves show validation means for Qwen3.5-9B on CodeGym; shading indicates ± 1 standard deviation.

Table 4: **Frozen-LLM adaptation.** Dyad is trained with GRPO on DIVE. Evaluations use 3 seeds.

Method	ALFWorld	WebShop	τ^2	SWE
Qwen3.5-4B	23.45	28.54	61.05	43.21
+ Dyad w/ freezing	**27.75** +4.30	**31.47** +2.93	**63.88** +2.83	**43.78** +0.57
+ Dyad w/o freezing	31.52	31.22	64.54	44.28
Qwen3.5-9B	45.66	33.43	62.17	44.56
+ Dyad w/ freezing	**48.82** +3.16	**35.00** +1.57	**64.23** +2.06	**45.50** +0.94
+ Dyad w/o freezing	53.82	36.31	64.88	46.20
Llama-3.2-3B-Instruct	16.98	8.41	**18.72**	24.32
+ Dyad w/ freezing	**18.32** +1.34	**10.09** +1.68	18.44 −0.28	**24.80** +0.48
+ Dyad w/o freezing	21.03	12.94	20.10	25.12
Gemma-4-E2B-it	24.24	17.11	31.23	**28.66**
+ Dyad w/ freezing	**27.73** +3.49	**19.30** +2.19	**33.94** +2.71	28.15 −0.51
+ Dyad w/o freezing	29.95	21.48	34.01	30.56

General-Capability Retention. Learning both reasoning and environment control through vocabulary generation can create interference between the two objectives during agentic RL (Yue et al., 2025). We therefore measure changes in general knowledge, mathematical reasoning, and coding relative to the original pretrained models (Figure 3). Dyad generally outperforms the corresponding GRPO and GiGPO baselines, both by recovering capability losses after DIVE training and by amplifying gains after CodeGym training. Notably, both Dyad variants exceed the original models on every CodeGym capability evaluation, while the RL baselines still regress on general knowledge or coding in some settings. Together with the agent-performance gains, these results suggest that the action encoder provides an additional route for task adaptation, reducing interference with pretrained language capabilities.

Training Dynamics. Both Dyad variants attain higher final validation rewards than their corresponding RL baselines (Figure 4). As training progresses, Dyad produces substantially shorter responses, suggesting more concise and efficient reasoning, while increasing the number of environment interactions per trajectory. This shift toward less generation and more interaction is especially beneficial for multi-step tasks, where progress depends on repeatedly acting on and incorporating feedback from the environment.

5.3 Modular Adaptation with a Frozen LLM

Frozen-LLM Adaptation. With the LLM frozen, training the action encoder improves agent performance over the original model in most evaluated settings (Table 4). Both Qwen models improve on all four environments, and the gains extend to other model families, with improvements on WebShop for every model. Joint training generally achieves higher scores, but the gap varies by model and

Table 5: **JevBench results.** Accuracy (%) on the public subset. Best accuracy and latency (s) values are bold. † Jev latency comes from published API measurements. Benchmark and measurement details are provided in Appendix B.3.

Method	Easy	Standard	Hard	Overall	Latency	Output tokens
Jev 1.13	**100.00**	96.79	70.88	85.01	0.67†	—
Qwen3.5-4B (w/o reasoning)	**100.00**	96.41	56.32	77.89	0.32	—
Qwen3.5-4B (w/ reasoning)	98.76	95.62	59.88	79.10	1.02	102.82
Qwen3.5-4B + Dyad (w/o reasoning)	**100.00**	97.88	67.07	83.52	**0.31**	—
Qwen3.5-4B + Dyad (w/ reasoning)	**100.00**	**98.72**	**72.08**	**86.18**	1.35	120.44

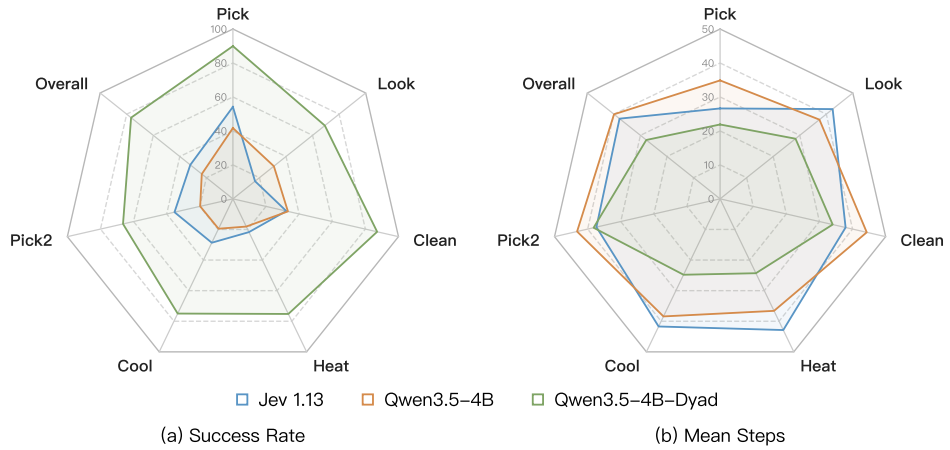

Figure 5: **Comparison with Jev.** Qwen3.5-4B serves as the zero-shot reference on ALFWorld.

environment: for both Qwen models and Gemma, frozen adaptation is close on τ^2-bench, whereas larger gaps remain on ALFWorld. For both Qwen models, the gap on SWE is also small. These results show that training only the action encoder can improve agent performance beyond the RL training environment without updating the LLM.

5.4 System 1 Evaluation

Comparison with Dedicated System 1 Models. We evaluate Qwen3.5-4B + Dyad, trained on CodeGym using GRPO, against Jev on both direct decision-making and agentic interaction. On JevBench, Dyad with reasoning reaches 86.18% overall accuracy, slightly exceeding Jev's 85.01%, with the largest advantage on the hard subset (Table 5). Without reasoning, Dyad remains competitive while reducing latency to 0.31 s, compared with 0.67 s reported for Jev. On the more agentic ALFWorld benchmark, Dyad achieves higher success rates than both Jev and zero-shot Qwen3.5-4B across all six task types, while requiring fewer interaction steps overall and on most task types (Figure 5). Together, these results show that Dyad can serve as an effective typed decision model while retaining autoregressive reasoning within the same policy.

5.5 Ablation Studies

Trainable Components. Updating the full action encoder yields the highest final validation reward, with the LLM trainable in all variants (Figure 6a). Training the projector while freezing the encoder backbone is competitive early on, but reward declines later and ends below the variant with the entire encoder frozen. Even with fixed action representations, LLM updates improve reward over the shared aligned initialization.

Alignment and Reinforcement Learning. Alignment improves initial performance, and the full model outperforms RL-only training throughout the evaluated budget (Figure 6b). Although RL without alignment improves substantially, it does not close the gap. Continuing RL from the aligned

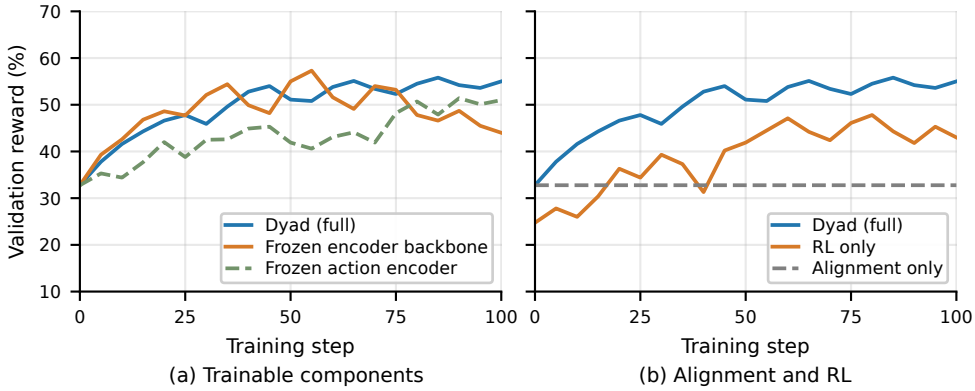

Figure 6: **Ablation studies.** CodeGym validation reward with Qwen3.5-4B, GRPO, and attention pooling. Both panels reuse the Dyad (full) curve; the horizontal dashed line denotes the alignment-only reference.

checkpoint also raises reward above the alignment-only reference. These comparisons support complementary roles for the two stages: supervised alignment provides a useful initialization, while agentic RL further improves task performance.

Projector Architecture.

Table 6: **Action pooling ablation.**

Projector	Validation reward
Mean pooling	41.23
MLP-weighted pooling	48.20
Attention pooling	55.92

Validation reward is highest with attention pooling, followed by MLP-weighted pooling and mean pooling (Table 6). In Dyad, each pooled action vector forms a row of the action-scoring matrix $\mathbf{Z}_t$ and is compared directly with the LLM state through a dot product (Equation 3). The projector therefore needs to produce representations suitable for this comparison from a separate encoder's token features. MLP-weighted pooling learns token weights, but its output remains a weighted average of the encoder states. Attention pooling additionally learns value and output projections, allowing the projector to transform the features used for action scoring (Appendix A.1). This gives it a direct way to adapt the pooled representation to the LLM's state representation space, in addition to learning token weights. Its advantage over MLP-weighted pooling is consistent with a benefit from combining token aggregation and feature transformation.

6 Conclusion

We introduced **Dyad**, an architecture for extending pretrained LLMs with native typed decision-making over dynamically provided action spaces. Dyad factorizes action selection into a representation of the evolving interaction state and independently encoded, environment-conditioned action representations, allowing the same pretrained LLM state to support both autoregressive language reasoning and direct typed decisions. This design provides a reusable decision interface without requiring actions to be represented through the language-model vocabulary. Across agentic post-training experiments, Dyad improves both in-domain and cross-environment performance over conventional RL baselines across model scales. These gains are accompanied by shorter reasoning traces, more frequent environment interaction, and generally stronger performance on the evaluated knowledge, reasoning, and coding benchmarks. Dyad also supports modular adaptation: training only the action encoder improves agent performance without modifying the pretrained LLM. Finally, on dedicated System 1 evaluations, Dyad is competitive with specialized decision models while retaining language reasoning within the same policy. Together, these results suggest that explicitly separating language reasoning from typed decision-making is a promising foundation for general-purpose agents that must reason and act across diverse, previously unseen environments.

AI USE STATEMENT

We used generative AI to assist with synthetic data generation and literature search. The authors take responsibility for the final content.

ETHICS STATEMENT

This work studies decision-making in language model agents using public benchmarks. We encourage responsible use of the proposed method.

REPRODUCIBILITY STATEMENT

Implementation details, data construction, experimental settings, hyperparameters, and prompt templates are provided in Appendices A and B to support reproducibility.

REFERENCES

Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Carbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Ju-yeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya O. Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adámek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean-Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle K. Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, and Arthur Brazinskas. Gemma 4 technical report. *CoRR*, abs/2607.02770, 2026. doi: 10.48550/ARXIV.2607.02770. URL https://doi.org/10.48550/arXiv.2607.02770.

Diogo Almeida. Introducing system one models & Jev. TypeSafe AI blog, September 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev. Published September 15, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R. Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OC2z7iSQKa.

Aili Chen, Chi Zhang, Junteng Liu, Jiangjie Chen, Chengyu Du, Yunji Li, Ming Zhong, Qin Wang, Zhengmao Zhu, Jiayuan Song, Ke Ji, Junxian He, Pengyu Zhao, and Yanghua Xiao. DIVE: Scaling diversity in agentic task synthesis for generalizable tool use. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4uM4P5NaLU.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with LLMs. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026. URL https://openreview.net/forum?id=DmPE4byHuN. ICML 2026 Workshop.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use RL with synthetic Code-

Gym. In *International Conference on Learning Representations*, pp. 17733–17756, 2026. URL https://openreview.net/forum?id=QRSeFZfu8E.

Featherless AI. Simple Jev: Structured classification and scoring from next-token logits. GitHub repository, 2026. URL https://github.com/featherless-ai/simple-jev. Revision c077d5df, accessed September 26, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *International Conference on Learning Representations*, pp. 57734–57811, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/fef126561bbf9d4467dbb8d27334b8fe-Abstract-Conference.html.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/8fd1a81c882cd45f64958da6284f4a3f-Abstract-Conference.html.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, pp. 58791–58831, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, pp. 54107–54157, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Theodore Lee. SemIf: Semantic ifs from open models. GitHub repository, 2026. URL https://github.com/TheoLeeCJ/SemIf-OpenJev. Revision 23cf1f39, accessed September 26, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui

Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Meta. Llama-3.2-3B-Instruct. Model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct.

Nandakishor. Laya: Non-autoregressive system 1 decision engine. GitHub repository, 2026. URL https://github.com/NandhaKishorM/laya. Revision 4066d5d5, accessed September 26, 2026.

OpenAI. GPT-5.5 model. OpenAI API documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.5. Accessed September 25, 2026.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhongqi Yue, Weishi Wang, Yundaichuan Zhan, Juncheng Li, Daniel Dahlmeier, and Fredrik D. Johansson. Expanding the Action Space of LLMs to Reason Beyond Language. arXiv preprint arXiv:2510.07581, 2025. URL https://arxiv.org/abs/2510.07581.

Kesen Zhao, Beier Zhu, Junbao Zhou, Xingyu Zhu, Zhongqi Yue, and Hanwang Zhang. Thinking with Images as Continuous Actions: Numerical Visual Chain-of-Thought. arXiv preprint arXiv:2602.23959, 2026. URL https://arxiv.org/abs/2602.23959.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

Table 7: **Action encoder input formats.** Qwen3.5-4B with Dyad-GRPO is trained on CodeGym and evaluated on ALFWorld and WebShop. Length denotes the mean number of description tokens.

	ALFWorld		WebShop	
Input format	**Length**	**Success rate**	**Length**	**Success rate**
MCP-style (default)	2665.57	**33.88**	396.50	**34.26**
Natural language	132.64	31.01	151.50	29.87

A Implementation Details

A.1 Action Encoding

The action encoder g uses a pretrained transformer backbone separate from f, followed by a shared projector. Alignment trains only the projector. Agentic RL updates both the encoder backbone and projector, including when the LLM is frozen. The projector pools the encoder's token representations into an action representation with the same dimensionality as $\mathbf{q}_t$.

Each candidate is encoded with the environment's action descriptions and its own definition. These descriptions supply the action context in $\mathrm{desc}(e)$. The permissible set $\mathcal{A}_t$ determines which candidates participate in the action distribution at a decision step. The encoder input excludes the current task context and reasoning in h_t. Action representations can be reused while their input descriptions and encoder parameters remain unchanged, and are recomputed when either changes.

With these textual inputs fixed, permuting the rows of $\mathbf{Z}_t$ permutes the corresponding action probabilities. This property concerns the order of candidates in the scoring matrix. Reordering the action descriptions within the encoder or LLM input can change their representations and is not covered by this property.

Tool Calls. Each environment interaction involves a single typed decision over $\mathcal{A}_t$. The language modeling head generates reasoning and any required arguments. The selected action's fixed textual form is appended to h_t, even when it spans multiple tokens. Once complete, the call is executed and the resulting observation is appended to the history.

Projector Architectures. Mean pooling uniformly averages encoder token representations and adds no trainable pooling parameters; this variant skips alignment. MLP-weighted pooling uses an MLP to score each token, then computes a softmax-weighted average of the representations. Attention pooling uses a learnable query to attend to projected keys and values, followed by an output projection. Padding tokens are masked in all three variants.

Action Encoder Input Formats. We use structured MCP-style action descriptions as the default input to the action encoder. Table 7 compares this format with natural-language descriptions for Qwen3.5-4B with Dyad-GRPO trained on CodeGym. MCP-style inputs yield absolute gains of 2.87% on ALFWorld and 4.39% on WebShop. The consistent advantage across these two evaluation environments supports structured descriptions as the encoder's default input format beyond the environment used for agentic RL.

Natural-language descriptions are substantially shorter in both environments, with a particularly large length difference on ALFWorld. The performance advantage of MCP-style inputs therefore comes with additional description tokens. This comparison favors the structured input in terms of task success, but does not isolate formatting from differences in description length and content.

A.2 Action Encoder Initialization

Data Construction. We construct synthetic examples for action selection across five domains: spatial manipulation, state change, information, workflow and communication, and transformation. Each domain contains ten training actions and four additional actions excluded from training, giving 50 training actions and 20 held-out actions. For each action, we obtain a structured definition

Table 8: **Alignment dataset.** Each case has paired MCP-style and natural-language records.

		Records		
Split	**Cases**	**MCP-style**	**Natural language**	**Total**
Train	1,050	1,050	1,050	2,100
Validation	140	140	140	280
Test	140	140	140	280
Total	1,330	1,330	1,330	2,660

specifying its name, purpose, and arguments, and a corresponding natural-language description. We construct synthetic action-selection examples across five domains, with 50 training actions and 20 held-out actions. Each example contains 4–10 permissible actions $\mathcal{A}_i$ from one domain and a target $a_i^\star \in \mathcal{A}_i$. GPT-5.5 (OpenAI, 2026) generates task context and reasoning h_i conditioned on these preselected actions and the target. Each case has paired MCP-style and natural-language action descriptions, sharing its context, reasoning, action order, and target. Table 8 reports the dataset sizes.

For each example, we first fix the permissible actions $\mathcal{A}_t$ and target action $a \in \mathcal{A}_t$. Each set contains 4–10 actions from one domain. We balance set sizes and target frequencies within each domain, approximately balance the use of other candidates, and shuffle their order. GPT-5.5 (OpenAI, 2026) then generates task context and reasoning conditioned on these actions and the target. The action descriptions provide $\mathrm{desc}(e)$, while the generated context and reasoning enter h_t. These examples supervise action selection without environment rollouts.

Inputs and Supervision. The LLM input combines the action descriptions, task context, and reasoning with format instructions, an example call, and a prefix marking the action-selection position. The example call uses an action sampled uniformly from $\mathcal{A}_t$, without conditioning on the target. Each input to g contains the same action descriptions, the definition of the action being encoded, and an instruction identifying that action. It excludes the task context and reasoning and does not identify the supervised target. All candidates are encoded as $\mathbf{z}_a$ and scored against $\mathbf{q}_t$ at the action-selection position. We minimize Equation 4, averaged over training examples, with a softmax temperature of one. Only the projector is updated, and no next-token prediction loss is applied to the supplied text.

Input Formats and Data Splits. Table 8 summarizes the dataset sizes. Each case has two versions, using structured MCP-style tool definitions or natural-language action descriptions. They share the task context, reasoning, candidate order, and target action, and remain in the same split.

Validation and test are split by case, stratified by domain, target action, and number of permissible actions, using seed 42. Their targets come from the same 20 actions excluded from all training candidate sets; other candidates may include both training and held-out actions. Thus, validation and test contain distinct cases within a shared action pool. We select the projector checkpoint by validation negative log-likelihood. Because this evaluation pool had been used for model selection before subdivision, the test split provides a diagnostic rather than a historically independent evaluation.

Supervision and Data Splits. Only the projector is trained using Equation 4; the generated text is supplied as input, with supervision applied to the target action. Validation and test targets come from actions absent from all training candidate sets. The two splits contain distinct cases from the same held-out action pool, with paired formats kept together. We select checkpoints by validation cross-entropy. The evaluation pool was used in earlier model selection, so its test split is not a historically independent holdout.

Data Checks. We check action-definition structure, candidate coverage, paired-format consistency, and split isolation. Near-duplicate contexts are identified by word-set overlap within each domain and generation pool and regenerated. These checks do not establish semantic equivalence of every description pair or disjointness from downstream benchmark actions.

A.3 Agentic Reinforcement Learning

Sampled Decisions and Loss Aggregation. Policy-gradient terms are computed for sampled vocabulary tokens and typed actions. The fixed text appended after an action selection records that decision and contributes no additional policy-gradient terms. Environment observations are also excluded. Let i index a sampled decision and $p_i(\theta)$ denote its probability under π_θ^{LM} or $\pi_\theta^{\mathrm{act}}$, evaluated at the recorded history and, for a typed action, its permissible action set. The rollout parameters θ_{old} and the advantage $\widehat{A}_i$ computed from environment rewards are held fixed during each update. The standard clipped surrogate for either decision type is

$$\begin{aligned} \rho_i(\theta) &= \frac{p_i(\theta)}{p_i(\theta_{\mathrm{old}})}, \\ \ell_i(\theta) &= -\min\Big(\rho_i(\theta)\widehat{A}_i,\ \mathrm{clip}(\rho_i(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}})\,\widehat{A}_i\Big). \end{aligned} \tag{5}$$

The clipping bounds follow Table 10. Rollout and training probabilities use the same sampling temperature and permissible actions. GRPO and GiGPO differ in how environment rewards are used to estimate advantages (Shao et al., 2024; Feng et al., 2025).

For aggregation group b, let $\mathcal{I}_b^{\mathrm{LM}}$ and $\mathcal{I}_b^{\mathrm{act}}$ index sampled vocabulary tokens and typed actions, respectively. Let N_b be the number of response tokens in that group and c_b the weight assigned to that group. We aggregate the sampled-decision losses into language and action contributions:

$$\begin{aligned} \mathcal{L}^{\mathrm{LM}} &= \sum_b \frac{c_b}{N_b} \sum_{i \in \mathcal{I}_b^{\mathrm{LM}}} \ell_i(\theta), \\ \mathcal{L}^{\mathrm{act}} &= \sum_b \frac{c_b}{N_b} \sum_{i \in \mathcal{I}_b^{\mathrm{act}}} \ell_i(\theta). \end{aligned} \tag{6}$$

Both contributions share the same denominator and group weights. The response-token count N_b includes fixed action text but excludes padding and environment observations. The action contribution is not separately normalized by the number of actions.

In joint optimization, the policy-gradient update to f and $\mathbf{W}_{\mathrm{LM}}$ uses $\mathcal{L}^{\mathrm{LM}} + \mathcal{L}^{\mathrm{act}}$, with action representations $\mathbf{z}_a$ treated as constants. The update to g uses $\mathcal{L}^{\mathrm{act}}$, with state representations $\mathbf{q}_t$ treated as constants, and trains both its backbone and projector. The action contribution has the same numerical value in both updates, but its gradients are taken with respect to different components. In frozen-LLM adaptation, only the update to g is applied.

B Experimental Details

B.1 Agentic Environments

Table 9 summarizes task counts and the average number of available tool or operation types per task. For ALFWorld and WebShop, these counts exclude state-dependent arguments and therefore do not measure the number of permissible choices at each decision step. The substantially larger toolsets in DIVE contrast with CodeGym's smaller sets of atomic functions. SWE exposes only one Bash interface, but its command arguments support repository inspection, editing, and testing. These statistics characterize the breadth of exposed interfaces; the number of interfaces alone does not determine the complexity of choosing and executing an action.

DIVE (Chen et al., 2026). The dataset contains diverse tool-use tasks synthesized from execution traces of real-world tools. Each task pairs a query with an available toolset and a verifiable reference answer, requiring agents to gather and process information through multiple tool calls.

CodeGym (Du et al., 2026). Coding problems are converted into interactive tool-use environments by exposing atomic functions from their solutions as callable tools. Agents solve tasks through sequential tool interactions, while the underlying code and test cases support automatic verification. The counts in Table 9 use the English-task reproduction released as VanishD/CodeGym.

Table 9: **Environment statistics.** Environments cover training and transfer evaluation.

Environment	# Tasks	Avg. tools / actions
DIVE	1,499	40.87
CodeGym	80,130	6.48
ALFWorld	3,827	14
WebShop	11,687	2
τ^2-bench	278	15.41
SWE	500	1

ALFWorld (Shridhar et al., 2021). This benchmark provides text-based environments for household tasks, such as finding, placing, cleaning, heating, and cooling objects. Agents navigate and manipulate objects through textual commands, using observations from the environment to track progress toward the task goal.

WebShop (Yao et al., 2022). In this simulated e-commerce environment, agents follow natural-language shopping instructions. They search for products, inspect product details, select options, and make a purchase that satisfies the requested attributes and constraints.

τ^2-bench (Barres et al., 2026). This benchmark evaluates conversational agents that use tools and communicate with users to complete tasks. Its dual-control telecom environment allows both the agent and the user to act on a shared environment state, testing coordination alongside tool use and reasoning. The statistics use the Retail, Airline, and Telecom base splits at repository revision `a2c0247`, which includes subsequent τ^3 updates.

SWE-bench Verified. This human-filtered subset of SWE-bench (Jimenez et al., 2024) evaluates the resolution of real-world GitHub issues.[1] Given an issue description and a code repository, agents inspect and modify the code to produce a patch, whose correctness is assessed using the benchmark's tests. We evaluate a 100-task subset of the 500 verified instances. We use the Bash interface of mini-swe-agent.[2] The evaluated model directly generates shell commands for repository inspection, editing, and testing. A GiGPO-style prompt provides recent observation–action pairs and the current execution feedback at each step. The same Bash interface submits the patch and terminates the episode.

The cross-environment evaluations use agents trained with RL on DIVE or CodeGym without further RL on ALFWorld, WebShop, τ^2-bench, or SWE-bench Verified. This separation refers to the RL training environments; the alignment-data split is described in Appendix A.2.

B.2 General Capability Benchmarks

MMLU-Pro (Wang et al., 2024). The benchmark extends MMLU with more challenging, reasoning-focused multiple-choice questions across academic subjects and up to ten answer choices per question. We evaluate on the full test set of 12,032 questions to assess general knowledge and reasoning after agentic post-training.

HMMT February 2026. We use the MathArena release (Dekoninck et al., 2026) of problems from the February 2026 Harvard–MIT Mathematics Tournament. These competition problems assess mathematical reasoning across algebra, geometry, number theory, and combinatorics. We evaluate on all 33 problems and report avg@4, averaging correctness over four generated answers per problem and then across problems.

LiveCodeBench v6 (Jain et al., 2025). The benchmark evaluates coding capabilities using programming contest problems collected over time from LeetCode, AtCoder, and Codeforces. We

[1] `https://www.swebench.com/`

[2] `https://github.com/SWE-agent/mini-swe-agent`

evaluate on the full version 6 test set of 1,055 problems, released from May 2023 to April 2025, with generated programs evaluated against the provided test cases.

B.3 JEVBENCH

Tasks and Evaluation Subset. JevBench evaluates typed decision-making given a state, instructions, decision criteria, and a finite set of candidate answers.[3] Its tasks include categorical choices, yes/no judgments, and ordinal ratings, covering problems such as routing, policy compliance, information extraction, and numerical reasoning. Each item evaluates a standalone decision without an environment rollout. We use the public subset of 231 items: 48 Easy, 72 Standard, and 111 Hard.

Metrics and Latency Source. Table 5 reports accuracy within each difficulty tier; Overall is the average weighted by the number of items per tier. Mean output tokens include reasoning and final answers, and mean latency measures time until the complete response is returned. The Jev latency entry is recomputed from the benchmark's published per-item API timings for the same public subset. These timings come from a separate evaluation and include network overhead. They provide an API latency reference; the different deployments do not establish a matched-hardware speed comparison. A corresponding Jev output-token measurement for this subset is unavailable.

B.4 HYPERPARAMETERS

Table 10 summarizes the hyperparameters for agentic RL in the joint optimization setting. The response limit bounds each turn, while the interaction limit bounds the trajectory. Shorter individual responses can therefore coexist with longer interaction sequences, as observed on CodeGym in Appendix C.2. Since these budgets and rollout batch sizes differ across environments, we interpret training curves through comparisons within each environment and model scale.

B.5 PROMPT TEMPLATES

Figures 7–12 summarize the environment prompt templates. Named placeholders represent the task, available actions, observations, and interaction history. The templates request explicit reasoning; disabling the model's native thinking mode does not remove these instructions.

For templates with explicit history fields, these contain recent observation–action pairs, while the current observation supplies the latest feedback. When no history is included, the corresponding history text is omitted; DIVE, ALFWorld, and WebShop also omit step-count text. The response instructions remain unchanged. WebShop additionally falls back to its no-history template when a prompt with history exceeds 13,000 characters. SWE-bench Verified follows the same GiGPO-style per-step structure, with fixed repository and submission instructions in the system message. The user message contains the task, recent history, current observation, and available tool. We retain our `<think>` and `<action>` response format while using mini-swe-agent's Bash interface.

[3] https://github.com/fstandhartinger/jevbench

Table 10: **Agentic RL hyperparameters.** Settings correspond to joint post-training.

Environment	ALFWorld	WebShop	DIVE	CodeGym
Interaction budgets				
Max prompt length (tokens)	2,048	4,096	32,768	5,120
Max interaction steps	50	15	30	80
Memory context window (steps)	2	2	2	2
Max response length (tokens/turn)	512	512	512	386
Sampling and batching				
Task groups per rollout	16	32	32	512
Trajectories per task group	8	8	8	8
Update mini-batch size (steps)	256	64	32	64
Training budget	2 epochs	2 epochs	2 epochs	3 epochs
Temperature (training)	1.0	1.0	0.9	1.0
Temperature (evaluation)	0.4	0.4	0.4	0.4
Top-p (evaluation)	1.0	1.0	1.0	1.0
Optimization				
Policy learning rate	10^{-6}	10^{-6}	10^{-6}	10^{-6}
PPO clip range $(\epsilon_{\text{low}}, \epsilon_{\text{high}})$	$(0.2, 0.2)$	$(0.2, 0.2)$	$(0.2, 0.2)$	$(0.2, 0.2)$
KL loss coefficient	0.01	0.01	0.001	0
Method-specific settings				
GiGPO discount factor γ	0.95	0.95	0.95	0.95
GiGPO step advantage weight ω	1.0	1.0	1.0	1.0

Notes. Prompt and response limits apply per interaction; responses exclude tool outputs. Mini-batches count interaction steps. Thinking mode is disabled.

DIVE prompt template

USER

You are an expert agent solving a task using the available tools.
Your task is to: `{task_description}`
Prior to this step, you have already taken `{step_count}` step(s). Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`
You are now at step `{current_step}` and your current observation is:
`{current_observation}`
Your available tools are: `{available_actions}`
Use these tool schemas to construct calls; the interface specifies the native tool-call syntax inside `<action>`.

Reason about the next step within `<think> </think>` tags.
Then enclose your action within `<action> </action>` tags.
Inside `<action>`, call the appropriate tools using the native tool-call format and their declared arguments, then close `</action>` and wait for their results.
When ready, put your final answer inside `<action> </action>` instead of tool calls, in the format requested by the query.
An action without tool calls ends the task; do not output reasoning alone or text outside these two blocks.

Figure 7: **Prompt template for DIVE.** It specifies how the agent calls tools and submits its final answer.

CodeGym prompt template

SYSTEM

`{original_system_messages}`

USER

You are an expert agent operating in an interactive function-calling environment.
Your task is to: `{task_description}`
Prior to this step, you have already taken `{step_count}` step(s).
Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`
You are now at step `{current_step}` and your current observation is:
`{current_observation}`
Your available functions are: [`{available_actions}`].

Reason about the next step within `<think> </think>` tags.
Then output exactly one function call using its declared arguments:
`<|FunctionCallBegin|>`[{"name": "<function name>", "parameters": {"<argument name>": "<value>"}}]`<|FunctionCallEnd|>`
Use {} for no arguments; do not write executable code.
Wait for feedback after each call and follow the task's submission rules.

Figure 8: **Prompt template for CodeGym.** The agent issues one function call at a time and waits for feedback before continuing.

ALFWorld prompt template

USER

You are an expert agent operating in an interactive household environment. Your task is to:
`{task_description}`
Prior to this step, you have already taken `{step_count}` step(s). Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`
You are now at step `{current_step}` and your current observation is:
`{current_observation}`
Your admissible actions of the current situation are: [`{admissible_actions}`].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation. This reasoning process MUST be enclosed within `<think> </think>` tags.
Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

Figure 9: **Prompt template for ALFWorld.** The agent selects an admissible action based on the current observation and interaction history.

WebShop prompt template

USER

You are an expert autonomous agent operating in an online shopping environment.
Your task is to: `{task_description}`.
Prior to this step, you have already taken `{step_count}` step(s). Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`
You are now at step `{current_step}` and your current observation is:
`{current_observation}`.
Your admissible actions of the current situation are:
[
`{available_actions}`
].

Now it's your turn to take one action for the current step.
You should first reason step-by-step about the current situation, then think carefully which admissible action best advances the shopping goal. This reasoning process MUST be enclosed within `<think> </think>` tags.
Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

Figure 10: **Prompt template for WebShop.** The agent uses the shopping instruction and current observation to select its next action.

τ^2-bench prompt template

SYSTEM

`{domain_policy}`
#Available tools
`{tool_schemas}`
Instruction
Follow the policy above and the declared tool schemas.
Ask the customer for missing information or required confirmation; do not invent facts or tool results.

USER

You are an expert agent helping a customer in an interactive service environment.
Your task is to: `{task_description}`
Prior to this step, you have already taken `{step_count}` step(s).
Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`
You are now at step `{current_step}` and your current observation is:
`{current_observation}`
Your available actions are: [`{available_actions}`].

Reason about the next step within `<think> </think>` tags.
Then output exactly one action as valid JSON enclosed within `<action> </action>` tags:
`<action>`
`{"name": "<action name>", "arguments": {"<argument name>": "<value>"}}`
`</action>`
To message the customer, use name="respond" and arguments={"content": "<message>"}; this does not end the conversation.
Stop after `</action>` and wait for feedback; output no text outside the `<think>` and `<action>` blocks or Markdown fences.

Figure 11: **Prompt template for τ^2-bench.** The agent follows the domain policy when using tools and communicating with the customer.

SWE-bench Verified prompt template

SYSTEM

Fix the issue in /testbed and verify the fix. Each command runs in a fresh shell. Do not modify tests or configuration files, or commit changes.

Save a source-only `git diff` to `patch.txt`, using `git add -N` for new files. Exclude reproduction scripts, helpers, and binaries. Inspect the patch in a separate call, then submit in a final bash call:
`echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT && cat patch.txt`
Exit status 0 completes submission.

USER

Task: **{task_description}**
Recent observation–action history: **{action_history}**
Current step: **{current_step}**
Current observation: **{current_observation}**
Available tools: [**{available_actions}**].

Put reasoning inside **<think> </think>**, followed by one **<action> </action>** block with native bash calls. Include at least one call; group only independent commands. Output nothing outside these blocks; then stop and wait for results.

Figure 12: **Prompt template for SWE-bench Verified.** GiGPO-style context with mini-swe-agent's Bash interface.

C Additional Results and Analyses

C.1 Task-Specific Post-training on ALFWorld and WebShop

We train separate agents on ALFWorld and WebShop, jointly updating the LLM and action encoder for Dyad. Table 11 reports general capabilities for the same agents evaluated in Table 2. These runs are separate from the DIVE- and CodeGym-trained agents used for cross-environment evaluation.

Comparison with RL Baselines. Both Dyad variants outperform their corresponding RL baselines in all 24 comparisons across the two model scales, two training environments, and three capability benchmarks. The advantage extends to general knowledge, mathematical reasoning, and coding. Together with the consistently higher in-domain scores in Table 2, these results show that stronger task adaptation with Dyad can accompany better general capabilities than conventional RL post-training.

Retention Relative to the Original Models. GRPO and GiGPO score below the original models in every reported capability evaluation. Dyad reduces these losses, but recovery varies with the training environment and capability. At 9B, both Dyad variants exceed the original model's coding score after training on either environment. Mathematical reasoning is more dependent on the training source: both 9B Dyad variants exceed the original HMMT score after ALFWorld training, whereas both remain below it after WebShop training. At 4B, several scores also remain below their original levels despite improving over the RL baselines. Thus, the paired advantage supports better capability retention during task-specific joint training, although it does not eliminate degradation in every setting.

C.2 Additional Training Dynamics

CodeGym. Figure 13 extends the 9B results in Section 5.2 to Qwen3.5-4B and Qwen3.5-27B. At both scales, Dyad-GRPO and Dyad-GiGPO achieve higher final validation rewards than their corresponding baselines, with generally shorter responses and more environment interactions. Despite greater fluctuations at 4B, the paired advantage remains visible late in training.

DIVE. Figure 14 reports training dynamics for Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-27B. Both Dyad variants attain higher final validation rewards and shorter responses than their corresponding baselines. Their interaction counts rise initially but decline later, finishing below the baselines. The reward advantage is therefore consistent across the two environments, while the interaction pattern differs: more interactions on CodeGym and fewer on DIVE.

Table 11: **General capability retention.** Models are post-trained on ALFWorld or WebShop.

	ALFWorld			WebShop		
Model / Method	MMLU-Pro	HMMT	LiveCodeBench	MMLU-Pro	HMMT	LiveCodeBench
Qwen3.5-4B	**69.34**	36.26	47.39	69.34	**36.26**	47.39
+ GRPO	68.08	30.33	46.21	67.93	33.30	46.62
+ Dyad-GRPO	68.96	33.36	47.02	69.02	34.76	47.11
+ GiGPO	67.62	32.46	46.78	67.51	33.05	46.06
+ Dyad-GiGPO	68.10	**36.33**	**47.52**	**69.53**	35.16	**47.89**
Qwen3.5-9B	71.88	39.75	59.62	**71.88**	**39.75**	59.62
+ GRPO	70.71	35.13	58.32	70.39	36.66	57.81
+ Dyad-GRPO	**71.92**	**42.90**	**60.89**	71.84	36.98	59.67
+ GiGPO	70.02	37.06	58.06	70.89	37.34	58.42
+ Dyad-GiGPO	71.21	40.29	59.99	71.42	38.09	**60.10**

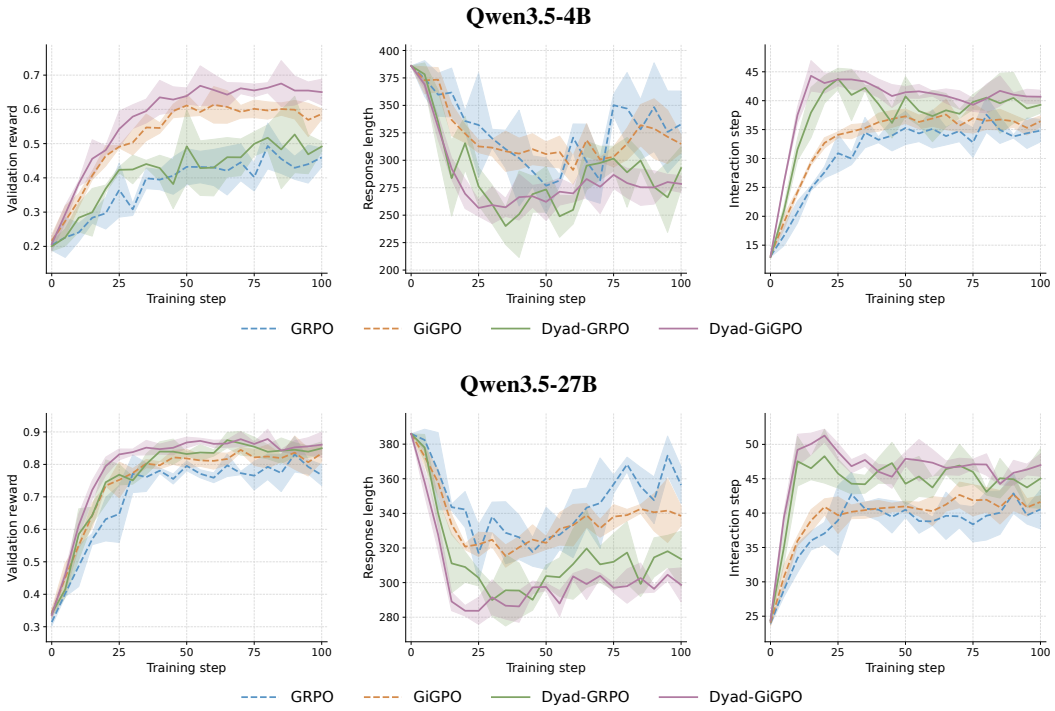

Figure 13: **Additional training dynamics on CodeGym.** Validation reward, response length, and environment interactions for Qwen3.5-4B (top) and Qwen3.5-27B (bottom).

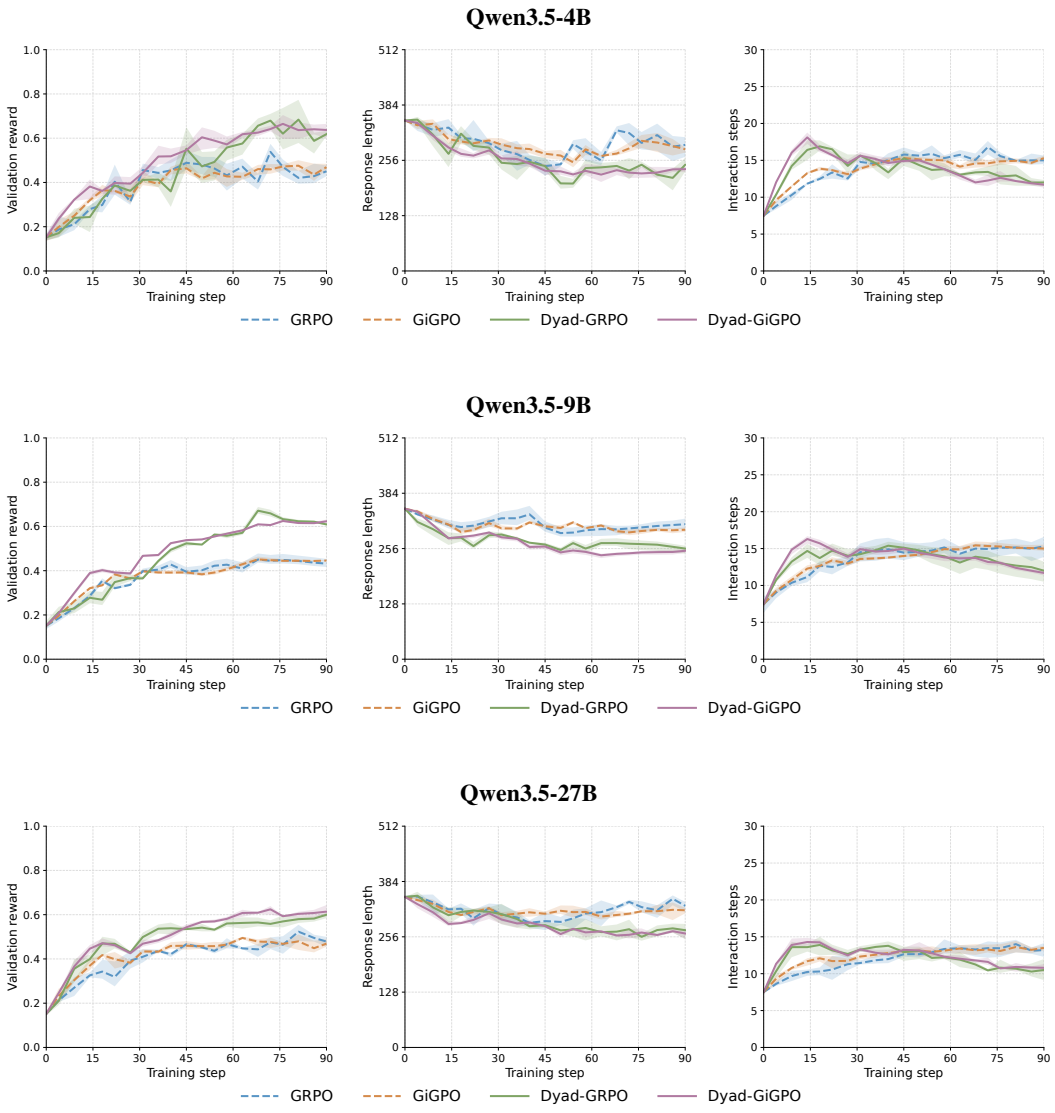

Figure 14: **Training dynamics on DIVE.** Models are jointly post-trained on DIVE.