Decoupling Planning and Control for Instructable Agents

Zineng Tang
terran@berkeley.edu
UC Berkeley

Kelsey R. Allen*
krallen@cs.ubc.ca
UBC

Sjoerd van Steenkiste
svansteenkiste@google.com
Google DeepMind

Ishita Dasgupta
idg@google.com
Google DeepMind

Alane Suhr
suhr@berkeley.edu
UC Berkeley

Abstract

Recent work shows that pre-trained, instruction-tuned vision-language models (VLMs) perform well at mapping from instructions and observations to high-level plans, but struggle to realize such plans as reliable low-latency action sequences in unfamiliar environments. At the same time, world-model controllers excel at fast observation-to-action control, but lack open-ended task guidance. In this work, we combine these strengths into a single system, Instruct-to-Act, where we train a world-model controller to act autonomously at high frequency when conditioned on sparse, higher-latency, and high-level text instructions generated by a VLM planner. To train controllers to be language-instructable, we relabel segments of controller policy rollouts with synthetic instructions and jointly optimize a behavior-cloning objective along with existing reward-maximizing and world-modeling objectives. We evaluate our proposed approach across seven embodied environments, including three multi-agent environments where VLM planners coordinate through language while trained controllers serve as their actuators. Under matched observation and action spaces, our decoupled approach consistently outperforms controller-only and direct VLM action-generation variants, preserves fast control, and lets us swap in different pretrained VLM planners without fine-tuning, while remaining competitive with strong vision-language-action and multi-agent RL baselines on six of seven tasks.[1]

1 Introduction

Embodied agents increasingly operate in settings that demand both high-frequency, low-latency continuous control and intermittent, high-level reasoning and planning. For example, consider a long-horizon, multi-player game like Minecraft, where agents can communicate with one another to achieve shared goals. There have been two dominant approaches in building such agents. Learning world models through reinforcement learning (RL) has demonstrated that compact modeling of latent environment dynamics can support sample-efficient, low-level control from pixels (Hafner et al., 2020; 2025). In parallel, instruction-following large language and vision-language(-action) (LLM/VLM/VLA) models (Zitkovich et al., 2023; Black et al., 2025; Kim et al., 2024; Bjorck et al., 2025) can synthesize useful decompositions, subgoals, and plans from high-level task descriptions and visual context. Yet these two capabilities are typically available in isolation. VLM-only agents plan well, but struggle to realize plans as precise action streams under tight real-time budgets. Moreover, because VLM outputs are text tokens rather than time-critical action streams, evaluating them as embodied agents across diverse domains is challenging. Low-level controllers act smoothly and quickly but are difficult to steer with abstract, open-ended instructions.

*Work done at Google DeepMind.

[1]Code and a demo for our project is available at https://zinengtang.github.io/speak-to-act/.

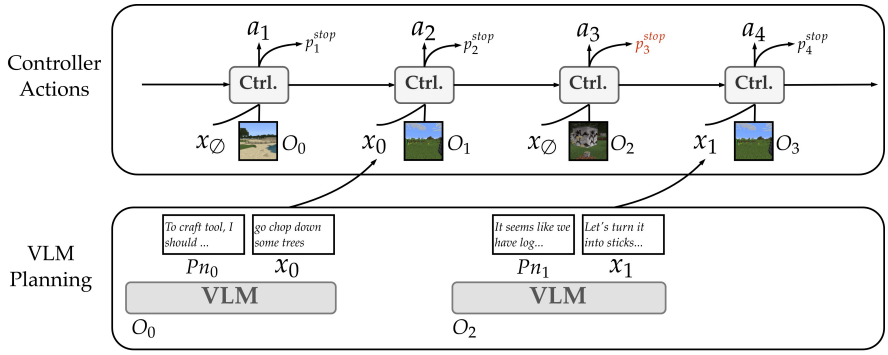


Figure 1: Illustration of inference with a VLM planner and trained controller.

In this work, we study the problem of grounding language into policy for real-time control coupled with high-level abstraction. We propose Instruct-to-Act, a plug-and-play paradigm that decomposes the problem of embodied decision-making into *planning* and *control*. In this paradigm, a pre-trained VLM *planner* maps environment observations to plans and instructions; these instructions are sent to a *controller*, which processes a stream of observations and produces low-level actions in real time. The VLM planner plans and issues instructions concurrently with low-level execution, without the need for finetuning. Our core contributions center around a framework for efficiently training environment-specific controllers that can be paired with any planner to support real-time planning and control. Each controller is built as a **language-aware world model and policy**, where both representation learning (on the world model) and control (actor and critic heads) are conditioned on a latent representation of an instruction sent by a planner. In contrast with single-task models like Dreamer (Hafner et al., 2020; 2025), where a policy is learned indirectly through rewards, our setting allows latent imagination and value learning to incorporate a wide range of abstract goals specified using a language instruction. To train controllers to be conditioned on instructions, we perform **post-hoc instruction supervision**, summarizing replay segments into language instructions with a VLM and using them to perform behavior cloning during training. Finally, during inference, we perform **asynchronous instruction**: the planner continuously reasons and plans in text space while the controller simultaneously executes low-level actions in the environment.

Because planning is decoupled from acting, inference scales well in both performance and efficiency: any number of language planners can each be coupled with identical controller instances, enabling efficient **multi-agent communication** in real time. This design also lets us evaluate both (a) how well individual VLMs act as embodied single agents and (b) how well they reason about, coordinate with, and communicate with other agents under real-time constraints.

We organize evaluation into two parts. First, we verify that without instructions, trained controllers match the performance of standard world-model RL baselines, and that they achieve an average instruction-following accuracy of 92.8% across instructions generated from a diverse set of VLM planners. Then, we use the controllers to evaluate VLM planners' ability to (a) act as agents across diverse single-player embodied environments by mapping high-level task descriptions to mid-level instructions, and (b) plan and coordinate with other VLM planners in multi-agent tasks. Pairing VLM planners with our trained controllers performs significantly better than mapping directly from high-level task to low-level actions, in both task score and inference efficiency, achieving competitive performance in 6 of the 7 environments; overall, decoupling planning and control supports strong task performance, low latency, and convenient modularity for evaluating VLMs as embodied agents in arbitrary heterogeneous environments.

In contrast to prior work (Section 5), no VLM is fine-tuned to an environment's low-level action space—only the lightweight controller is environment-specific; the controller, unlike pure world-model RL agents, is explicitly trained to execute open-ended language instructions at control frequency; and the stable language interface supports asynchronous planner swaps and direct multi-agent coordination with a shared controller architecture.

2 Overview and Experimental Setup

We study embodied agents that map observations to low-level actions, optionally conditioned on natural-language task specifications. Our goal is to adapt existing vision-language models (VLMs) into embodied, language-using agents out-of-the-box, without fine-tuning them to each environment's low-level action space.

At inference time, we decompose decision making into a *planner* and *controller*. The planner maps observations (and optional task context) to a natural-language instruction, and the controller maps the instruction and visual observations to low-level actions. This contrasts with approaches that require end-to-end low-level action generation by fine-tuned VLM/VLA models (Driess et al., 2023; Zitkovich et al., 2023; Black et al., 2025; Kim et al., 2024; Bjorck et al., 2025; Hafner et al., 2020; 2025), and follows language-conditioned control paradigms in robotics (Ahn et al., 2022); it allows us to plug in any instruction-tuned VLM as a planner with a lightweight environment-specific controller.

Task. We consider partially-observable environments $(\mathcal{S},\mathcal{A},\Omega,O,P,R,\gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, observation space Ω, observation kernel $O:\mathcal{S}\to\Delta^{\Omega}$, transition kernel $P:\mathcal{S}\times\mathcal{A}\to\Delta^{\mathcal{S}}$, reward function R, and discount factor γ. Pretrained VLMs struggle when $\mathcal{A}$ is high-frequency and environment-specific (e.g., repeated camera/control commands in Minecraft or Atari), making direct low-level action sequence emission impractical. In our target domains, the challenge is not only choosing the right subgoal, but sustaining a long, low-level action stream at control rate.

At each environment step, an agent policy $\pi:\Omega\to\Delta^{\mathcal{A}}$ maps from observations to a distribution over actions. We study agents which are compositions of independent *planner* and *controller* agents, respectively $\pi_p:\Omega\to\Delta^{\mathcal{X}}$ and $\pi_c:\Omega\times\mathcal{X}\to\Delta^{\mathcal{A}}$, where $\mathcal{X}$ is the set of all possible natural language utterances. As illustrated in Fig. 1, this planner–controller split is closely related to hierarchical RL and the options framework, where higher levels issue temporally extended commands (Dayan & Hinton, 1992; Sutton et al., 1999; Bacon et al., 2017; Nachum et al., 2018; Levy et al., 2019). We sample actions from π given an observation o by first sampling an instruction $x\sim\pi_p(\cdot\,|\,o)$, then sampling a sequence of low-level actions $\overline{a}$ from π_c, where $a_i\sim\pi_c(\cdot\,|\,o_i,x)$. We augment the action space with an instruction-completion action; when selected, a new instruction is sampled from π_p. Controllers need not be conditioned on an instruction; when none is available, the controller can continue autonomously between planner updates. In experiments, π_p is an instruction-tuned VLM and π_c is a lightweight planner-agnostic environment-specific recurrent state space model (RSSM).

Controller. The controller π_c is an instruction-conditioned RSSM (Hafner et al., 2025) with an additional head predicting instruction completion (Sec. 3.2). It is trained by relabeling segments of its own on-policy rollouts with VLM-generated instructions and adding a behavior-cloning objective to the standard Dreamer world-model, actor, and value losses (Sec. 3.4). No expert demonstrations are used.

Evaluation. We evaluate on seven environments spanning classic single-agent RL benchmarks (e.g., Atari, Bellemare et al., 2013) and multi-agent tasks based on real cooperative video games (e.g., Pico Park); tasks, metrics, and analyses are detailed in Section 4.2.

3 Method

3.1 Decoupling Planning and Control in Inference

Our agent policy π composes a planner π_p and a controller π_c (Section 2). In practice, we run inference on π_p and π_c in independent parallel processes, which exploits π_p's capabilities beyond high-level decision-making—reasoning and communication with other agents—while low-level actions execute in the environment, and extends naturally to multi-agent settings by spawning planner and controller threads per agent.

Asynchronous Online Inference. Trading off inference between π_p and π_c, where π_p must wait for π_c to complete its current instruction before it starts to plan a new one, wastes wall-clock time. Instead, we allow planners π_p to run inference asynchronously in the background, given a live observation stream, continuously updating their reasoning, illustrated by Alg. 1 (pseudocode in Appendix C). Upon receiving a message that the controller has completed its current instruction, planners are prompted to immediately emit the next instruction from a draft plan. Given a draft plan, emitting an instruction requires generating only a few tokens. This simultaneous inference supports both the relatively slow language reasoning of VLMs beneficial to generating optimal instructions, and the fast low-level execution provided by the controller. In the offline variant (Alg 2), by contrast, the controller waits until the planner finishes producing a new instruction, so throughput directly inherits VLM latency at every instruction boundary.

Multi-agent Inference. We extend the framework to cooperative tasks with n agents $\left\{\pi^{(i)} = \left(\pi_p^{(i)}, \pi_c\right)\right\}_{i=1}^{n}$.[2] Each agent shares controller parameters, but at inference time, these controllers operate on different observation streams, maintain different latent states, and execute different actions. Planners need not be the same type of agent. Unlike classic centralized-training/decentralized-execution (CTDE, Kraemer & Banerjee, 2016), we do not train a separate centralized critic or impose structured role labels. Instead, coordination happens purely in language, with planners specifying and negotiating high-level goals. Agents communicate in discrete rounds aligned with the control step. At each step, the agents can (i) receive messages from the shared chatroom $\texttt{Chatroom}[i]_t$; (ii) instruct the controller to act; (iii) optionally speak. The chatroom $\texttt{Chatroom}[i]$ concatenates all peer messages at any given step. All agents share the same planner system prompt, which instructs them to process incoming messages and reply as if in a dialogue.[3] Qualitatively, agents use messages to claim subgoals, negotiate roles early in an episode, and report bottlenecks when progress stalls. Controllers do not process messages sent between planner agents.

3.2 Controller Architecture

Our controller is based on the recurrent state space model (RSSM; Hafner et al., 2019; 2020), which maintains a latent state s_t computed from previous observations $o_{<t}$ and actions $a_{<t}$, with decoder heads estimating the next observation, reward, and action. Following Hafner et al. (2025), we train the observation decoder with a KL-regularized reconstruction objective and optimize Dreamer-style actor and value heads on imagined rollouts.[4] We adapt this base architecture to optionally condition on a planner instruction x_t: a frozen, pretrained contrastive language encoder maps x_t to $e_t \in \mathbb{R}^{d_\ell}$, which is projected and concatenated with the RSSM features used by the policy and value heads, yielding

$$\pi(a \mid s_t, e_t) = \mathrm{Softmax}\big(\ell(s_t, e_t)/\tau\big)_a, \qquad \ell(s_t, e_t) \in \mathbb{R}^{|\mathcal{A}|}.$$

When no instruction is available, e_t is a learned null embedding, so the controller can act autonomously.

We additionally augment the output space of the controller to include an indicator of whether an instruction has been completed. At each step t, the actor emits $p_t^{\text{stop}} = \sigma\big(g(s_t, e_t)\big)$ where g is a small head on top of the latent state and σ is the logistic function.

3.3 Planner

Our framework supports any planner $\pi_p : \Omega \to \Delta^{\mathcal{X}}$ that maps from observations to instructions, where observations can include visual input and, for multi-task environments, an optional natural-language task specification. We implement planners using vision-language models (VLMs), prompted to additionally maintain, during inference, a

[2] We include more details on the multi-agent setup in Appendix C.

[3] Appendix A.11 contains all system prompts.

[4] Full equations and losses can be seen in Appendix A.1.

memory bank Mr and a partial plan Pn. Concretely, at inference step t', the VLM maps $(Sp, Mr_{t'-1}, Pn_{t'-1}, O_{<t'}, p^{\text{stop}}_{t'-1})$—system prompt, memory bank, partial plan, observation history, and the controller's most recent instruction-completion probability—to an updated $(Mr_{t'}, Pn_{t'})$ and, if $p^{\text{stop}}_{t'-1}$ indicates completion, an instruction x sent to the controller π_c. Crucially, as discussed in Sec. 3.1, the VLM steps t' need not be synchronous with the controller's timesteps t, supporting online planning of the next instruction while the agent executes its current one; in the synchronous variant, each instruction is generated only after the previous one completes, so t' aligns with instruction boundaries.

In multi-agent settings, planners additionally communicate in one of two modes: *decentralized*, where at each VLM inference step any planner may initiate at most one message to a peer, or *centralized*, where a fixed hub agent sends messages to all other agents (formal definitions in Appendix D.1). Relative to the single-agent setting, the planner input is augmented with the agent's inbox and recent outgoing messages, and the planner output can additionally include a short message for another agent.

3.4 Training

We train the controller parameters by alternating between collecting rollouts conditioned on the current policy, and optimizing the policy using both reward, world model, and instruction-following behavior-cloning objectives. The losses combine two objectives: (a) instruction following, via behavior cloning on on-policy action sequences post-hoc labeled with VLM-generated instructions; and (b) reward maximization and world modeling, which, following the Dreamer approach (Hafner et al., 2025), align the labeled behavior distribution with what the controller executes at inference time.

Behavior cloning with post-hoc instruction annotation. During learning, we use a FIFO replay buffer (Mnih et al., 2015) $\mathcal{D}$ (capacity $|\mathcal{D}|$=1024) that stores tuples $\tau_t \equiv (o_t, a_t, r_t, e_t, \text{complete}_t, \text{done}_t)$, where e_t is an optional instruction embedding, complete_t indicates whether a_t completed the instruction, and done_t marks episode termination. To acquire e_t, we annotate the buffer online: we sample a set of non-overlapping, variable-length intervals $\mathcal{I} = \{[u_k, v_k]\}_{k=1}^{K}$ and, for each interval, prompt a VLM summarizer with task-specific prompts, equally spaced frames, and the packed action-sequence text to produce a high-level instruction x. We encode x into e (Section 3.2), pair all tuples $t \in [u_k, v_k]$ with e, and set complete_t=1 only at t=v_k.[5]

During training, we treat $a_{u_k:v_k}$ as a label for the instruction x conditioned on observations $o_{u_k:v_k}$, and optimize a behavior cloning loss over labeled intervals:

$$\mathcal{L}_{\text{BC}} = -\lambda_{\text{BC}} \sum_{[u_k,v_k]\in\mathcal{I}} \sum_{t=u_k}^{v_k} \log \pi(a_t \mid s_t, e_t).$$

To learn to mark instructions as complete, we also apply a binary cross-entropy loss on p_t^{stop} using the label complete_t:

$$\mathcal{L}_{\text{stop}} = -\sum_t \Big(\text{complete}_t \log\Big(p_t^{\text{stop}}\Big) + (1-\text{complete}_t) \log\Big(1-p_t^{\text{stop}}\Big)\Big).$$

We annotate intervals covering 50% of the buffer, leaving the remaining unannotated so that the controller also learns to act autonomously from the environment reward R. Annotation reuses logged trajectories and runs in parallel with optimization, adding roughly 17% GPU-hours and a 12% training slowdown (Section 4). When no instruction is provided, we use the learned null embedding from Section 3.2. We (optionally) support a strict instruction-required mode that architecturally guarantees no actions are executed without a valid, non-completed instruction.

[5]Further segment sampling details and per-timestep bookkeeping can be referred in Appendix D.1.

Final Training Objective. The overall objective augments the base objective with behavior cloning:

$$\mathcal{L} = \mathcal{L}_{\text{model}} + \lambda_V \mathcal{L}_{\text{value}} + \lambda_A \mathcal{L}_{\text{actor}} + \mathcal{L}_{\text{BC}} + \mathcal{L}_{\text{stop}}$$

where $\mathcal{L}_{\text{model}}$, $\mathcal{L}_{\text{value}}$, and $\mathcal{L}_{\text{actor}}$ are defined by RSSM training algorithm (Hafner et al., 2025).

4 Experiments

4.1 Architecture and Training

Our controller is a language-conditioned DreamerV3-style RSSM (Hafner et al., 2025) with CLIP-base-224 (Radford et al., 2021) and DINOv2-base (Oquab et al., 2024) visual features, and MiniLM-L6-H384-uncased (Wang et al., 2020) language embeddings. Main results use the 800M controller variant. During data collection, rollouts are generated with empty language ($e_t = \mathbf{0}$), then post-hoc instruction annotation labels random replay segments (length 1 to 20) while unlabeled gaps carry no behavior-cloning loss. GPT-4o is the annotator and 50% of replay segments are annotated.[6]

4.2 Evaluation

Each evaluation consists of 100 episodes (or the task's standard protocol), with no exploration noise (ε=0) and temperature τ=0.8 for stochastic policies. We follow each environment's official protocol and metric: Atari (Kaiser et al., 2020) (sticky actions, no-op starts), MineRL ObtainDiamond (Guss et al., 2019), Crafter (Hafner, 2022), DMLab `explore_goal_locations` (Beattie et al., 2016), Overcooked (Carroll et al., 2019) (scripted partners for single-agent tests, cross-play for multi-agent), Pico Park (cooperative puzzle completion with synchronized actions),[7] and MindCraft (White et al., 2025) (cooking/construction/crafting success).[8]

4.3 VLMs as Planners

Baselines Setup. We group baselines into three categories. Reproduced (DreamerV3, QMIX, MAPPO): trained and evaluated by us under our observation and action spaces, giving frameworks that require extensive language reasoning additional training and inference budget until convergence. Domain-adapted: RT-2, fully fine-tuned on demonstrations from a trained DreamerV3 policy with non-pixel observations tokenized alongside pixels; as this departs from its original real-robot setting, these numbers reflect a domain adaptation. Contextual (Voyager, DEPS, JARVIS-1, LS-Imagine): evaluated with released code on Minecraft Diamond and adapted code on MindCraft, as their prompting infrastructure is Minecraft-specific and new prompts on new environments would introduce confounds.

Results. Table 1 shows the main results, comparing different VLMs as planners with several baseline agents. We train one controller per environment, so training does not require planner-specific fine-tuning or VLM reasoning in the loop. The table includes: (i) our planner+controller agents using four VLM planners; (ii) ablations that remove the controller or language interface; (iii) prior environment-specific baselines; and (iv) recent VLA/MARL baselines when available. The main pattern is consistent across tasks: decomposition into *planning* and *control* improves over direct VLM action generation, and adding language guidance improves over the controller-only baseline. Nearly all tested planners significantly outperform the end-to-end GPT-4o agent. Using GPT-4o to generate sparse instructions executed by a trained controller, rather than directly producing low-level actions, improves performance on every task. The strongest planners (GPT-4o and Qwen-VL-2.5-72B) achieve the best overall results, while our method remains competitive with stronger domain-specialized systems such as JARVIS-1 in Minecraft.

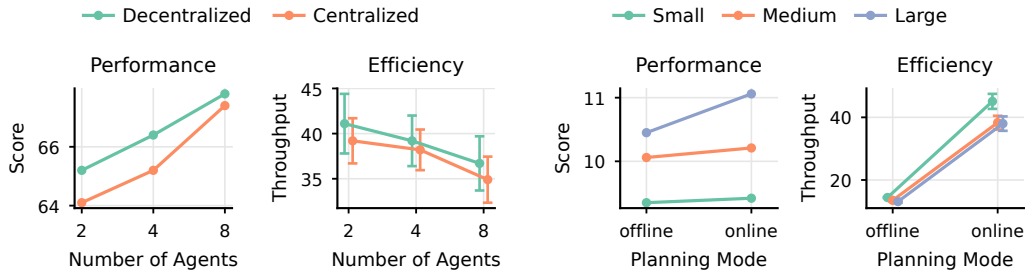
[6] Hyperparameters, model scales, and compute/runtime overhead are shown in Appendix A.3, Appendix A.5, and Appendix A.4 respectively.

[7] Pico Park (`https://picoparkgame.com/en/`) is a 2-8 player 2d cooperative platformer puzzle game.

[8] Detailed tasks descriptions are in Appendix B. We also include a lightweight human experiment in Appendix D, a multi-agent failure mode analysis in Appendix A.7, an instruction-length robustness study in Appendix A.9, and a taxonomy of generated instructions in Appendix A.8.

		Single-Agent Tasks				Multi-agent Tasks		
		Atari	MC-Dia	Craft	DML	OCook	Pico	MindC
Ours (planner swap)	Gemma-3-27B (Gemma Team et al., 2025)	880	9.7	13.8	71	187.4	68.9	53.0
	llava-v1.6-34b (Liu et al., 2024)	862	9.9	12.8	74	187.2	63.5	51.6
	Qwen-VL-2.5-72B (Bai et al., 2025)	878	11.1	13.4	77	192.3	68.4	58.5
	GPT-4o (Hurst et al., 2024)	891	11.7	14.1	76	193.2	70.1	70.2
Ablation	Qwen-VL-2.5-72B (Direct VLM Finetune)	581	10.1	7.6	45	150.1	30.6	38.2
	GPT-4o (w/o controller)	670	10.4	8.7	56	180.4	50.3	50.2
	Controller-Only	809	8.2	12.6	67	170.2	30.7	40.0
Reproduced Baselines	Dreamerv3 (Hafner et al., 2025)	811	8.6	10.5	65	-	-	-
	MAPPO (Yu et al., 2022)	-	-	-	-	182.5	50.8	44.9
	QMIX (Rashid et al., 2020)	-	-	-	-	187.2	58.5	53.2
Domain-Adapted (§4.3)	RT-2 (Zitkovich et al., 2023)	457	6.2	4.7	36	124.7	34.0	33.1
Contextual Comparisons	Voyager (Wang et al., 2024)	-	11.8	-	-	-	-	-
	DEPS (Wang et al., 2023)	-	9.4	-	-	-	-	45.6
	LS-Imagine (Li et al., 2025)	-	9.6	-	-	-	-	50.1
	JARVIS-1 (Wang et al., 2025)	-	12.3	-	-	-	-	54.1
	Mindcraft (White et al., 2025)	-	-	-	-	-	-	49.0

Table 1: Main results. Each cell reports the task's official metric where available. Baselines are grouped by comparability into **reproduced**, **domain-adapted**, and **contextual** categories, detailed in Section 4.3; our controlled claim rests on the matched comparison between our planner-controller variants and the ablation rows. Abbr: MC-Dia (Minecraft Diamond), Craft (Crafter), DML (DMLab), OCook (Overcooked), Pico (Pico Park), and MindC (MindCraft).



(a) Agent scaling (Pico Park)

(b) Planning modes (Minecraft Diamond)

Figure 2: Per-setting *performance* (left) and *efficiency* (right) results for (a) agent scaling and (b) planning modes. Model-size and reasoning scaling curves are shown in Figure 3 (Appendix A.6).

4.4 Scalability and Efficiency

We evaluate the framework's performance and efficiency (throughput as environment steps per second) under multiple conditions.[9] In Figure 2a, we compare decentralized with centralized control in the multi-agent Pico Park environment. Each agent is implemented as a Qwen-VL-2.5-72B planner paired with an 800M-parameter controller. We measure task performance and throughput as we vary the number of agents from 2 to 8. We find that decentralized communication results in improved task performance while also maintaining a throughput equivalent to centralized control. This is consistent with the motivation for centralized-training/decentralized-execution schemes in MARL, where routing all decisions through a central entity scales poorly with the number of agents (Kraemer & Banerjee, 2016; Rashid et al., 2020); in our framework the analogous bottleneck appears at the hub planner in centralized mode, whose failure modes we analyze in Appendix A.7. Unlike CTDE, however, our agents coordinate through language at the planner level rather than through a centralized critic.

[9] Additional conditions are described in Appendix A.6 and Figure 3.

Train-time instruction type	Diamond ↑
VLM-generated	11.1
Template-based instructions	9.9
Clustered action labels	8.9
Random strings	8.3
No instructions	8.2
Environment-derived labels	11.4

Table 2: Impact of post-hoc instruction source on Minecraft Diamond, using a Qwen-VL-2.5-72B planner and an 800M-parameter controller.

	Eval Planner ↑		
Annotator	Gemma	Qwen	GPT
Gemma	9.6	9.2	10.2
Qwen	9.7	9.9	9.9
GPT	9.8	9.6	10.0

Table 3: Cross-planner train × eval matrix (Minecraft Diamond). Each cell is score of a controller trained on instructions annotated by the row planner and evaluated by the column planner.

In Figure 2b, we compare online with offline planning modes on Minecraft Diamond. Here we use a Qwen-VL-2.5 (Bai et al., 2025) planner with three sizes: small (50M controller with 2B planner), medium (200M controller with 7B planner), and large (800M controller with 72B planner). As model sizes increase, online planning offers increased task performance compared to offline planning, while also remaining an order of magnitude faster in terms of throughput.

4.5 Generalization

We also study how far the framework generalizes beyond the exact configuration it was trained with: does a controller trained on one planner's annotations transfer to other planners and to different instruction-timing policies, do the gains depend on our specific controller architecture, and how reliably does the controller execute held-out planner instructions? Across these studies,[10] planners that decide instruction timing outperform fixed cadence on Minecraft Diamond (e.g., GPT-4o: 10.75 for VLM-decided proactiveness vs. 9.60 fixed cadence), and train × eval cross-planner transfer remains stable (Table 3). The framework is also not tied to RSSM specifically: a transformer world model is comparable to ours (11.2 vs. 11.0), while weaker controller variants degrade performance (RNN policy: 9.7; value-only world model: 10.2). Controller reliability remains high across domains and planners, with instruction-following accuracy between 172/200 and 194/200 (86% to 97%).

Annotation quality. Table 2 shows the influence of annotation quality during controller training on controller quality. We compare VLM-generated instructions to template-generated and cluster-based instructions, as well as environment-derived symbolic labels. Environment-derived symbolic labels slightly outperform VLM-generated instructions (11.4 vs. 11.1), but these require access to environment metadata, whereas VLM-based annotation needs only raw observations and thus generalizes to domains without such metadata. The results show that controllers benefit from semantic content even if instructions are of lower quality; using random strings as instructions, or no instructions at all, significantly underperform template- or cluster-based instructions.

Cross-planner generalization. Train × eval transfer remains stable across all planner pairs in Table 3 (9.2 to 10.2), indicating the controller does not overfit to the phrasing of its annotating planner.

Controller cost and scope. Our main results use one 800M-parameter controller trained per environment, a one-time cost of ∼23 GPU-hours on 4 RTX A6000 GPUs (∼5.5M environment steps; full compute and annotation overhead in App. A.4). In early experiments, a single controller trained jointly on all environments lost 1.12 points on Minecraft Diamond (≈10% relative), so lightweight domain-general instruction-conditioned controllers remain future work.

[10] Full tables are included in Appendix A.5.

5 Related Work

Our controller builds on RSSMs and imagination-based control: PlaNet introduced stochastic–deterministic latent transitions for planning from pixels (Hafner et al., 2019), and Dreamer learns policies by backpropagating value gradients through imagined latent trajectories (Hafner et al., 2020; 2025); we instantiate this line of work with language conditioning and post-hoc instruction supervision.

Decoupling slow, deliberate reasoning from fast, reactive control is a dominant paradigm across embodied domains, including autonomous driving (Tian et al., 2024; Qian et al., 2024; Liu et al., 2025), robot manipulation (Zhang et al., 2024; Shi et al., 2025; Bjorck et al., 2025; Figure AI, 2025; Song et al., 2025; Chen et al., 2025), and vision-and-language navigation (Wei et al., 2025); HiRT (Zhang et al., 2024) in particular runs an asynchronous planner-controller scheme communicating through latent representations. We do not claim the slow/fast decomposition as novel; we differ in that (i) the planner-controller interface is natural language rather than latents or waypoints, making the planner plug-and-play and enabling language-based multi-agent coordination; (ii) the fast system is a world-model controller rather than a domain-specific stack or waypoint follower, so it can generalize across heterogeneous domains; and (iii) our controller requires no expert data—all behavior-cloning supervision comes from post-hoc VLM relabeling of its own on-policy rollouts.

Language-conditioned robot-control systems motivate our planner-controller decomposition: SayCan, Code as Policies, and Inner Monologue compose language-model reasoning with low-level skills (Ahn et al., 2022; Liang et al., 2023; Huang et al., 2022), while end-to-end VLA approaches such as RT-1, RT-2, RT-H, OpenVLA, π_0, GR00T, and Octo map observation-language inputs directly to actions (Brohan et al., 2023; Zitkovich et al., 2023; Belkhale et al., 2024; Kim et al., 2024; Black et al., 2025; Bjorck et al., 2025; Octo Model Team et al., 2024). Unlike monolithic VLA adaptation, we train only the smaller environment-specific controller and keep the planner plug-and-play, enabling planner swaps without running full VLM inference at control frequency. For multi-agent control, we replicate one shared-parameter controller across agents with separate recurrent states, similar to prior MARL work (Terry et al., 2020; Chu & Ye, 2017; Rashid et al., 2020); unlike CTDE-style methods with centralized critics or value factorization, our low-level controllers stay decentralized and coordination happens in language at the planner level.

Embodied planner–controller agents in open-ended domains (Voyager, DEPS, LS-Imagine, JARVIS-1 (Wang et al., 2024; 2023; Li et al., 2025; Wang et al., 2025)) and iterative-reasoning LM agents (ReAct, Reflexion (Yao et al., 2023; Shinn et al., 2023)) demonstrate the promise of language-guided planning, but are typically tightly coupled to particular domains, planner backbones, or action-generation pipelines; we instead emphasize a reusable language interface between planner and controller, asynchronous inference, and evaluation across both single-agent and multi-agent domains.

Language is not the only possible conditioning interface: latent goal or skill abstractions such as LISA (Garg et al., 2022) provide an alternative, but language lets humans and off-the-shelf VLMs act as planners without retraining the controller-side interface, which is central to our plug-and-play evaluation setup.

6 Conclusion

We introduced Instruct-to-Act, a framework that combines the low latency of world-model controllers with the high-level reasoning ability of VLMs. Across diverse embodied tasks, the resulting planner+controller agents achieve strong performance while preserving high control-rate throughput, and the same interface extends naturally to multi-agent coordination without requiring specialized MARL training. Important next steps include studying human planners directly in the loop, extending the framework to asymmetric multi-agent roles and multi-task controller training, and improving planner–controller alignment through prompting or lightweight planner adaptation. More broadly, the framework suggests a practical path for deploying general-purpose language models in domains where abstract guidance must be translated into long, low-level action sequences.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, pp. 1726–1734, 2017.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801*, 2016.

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quan Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. RT-H: Action hierarchies using language. In *Robotics: Science and Systems*, 2024.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems*, 2023.

Micah Carroll, Rohin Shah, Mark K. Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-AI coordination. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. TransDreamer: Reinforcement learning with transformer world models. *arXiv preprint arXiv:2202.09481*, 2022.

Hao Chen, Jiaming Liu, Chenyang Gu, Zhuoyang Liu, Renrui Zhang, Xiaoqi Li, Xiao He, et al. Fast-in-slow: A dual-system VLA model unifying fast manipulation within slow reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Xiangxiang Chu and Hangjun Ye. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*, 2017.

Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5, 1992.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. PaLM-E: An embodied multimodal language model. In *International Conference on Machine Learning*, pp. 8469–8488, 2023.

Figure AI. Helix: A vision-language-action model for generalist humanoid control. https://www.figure.ai/news/helix, 2025.

Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. LISA: Learning interpretable skill abstractions from language. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 2442–2448, 2019.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2022.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, A. J. Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for Atari. In *International Conference on Learning Representations*, 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning*, 2024.

Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2019.

Jiajian Li, Qi Wang, Yunbo Wang, Xin Jin, Yang Li, Wenjun Zeng, and Xiaokang Yang. Open-world reinforcement learning over long short-term imagination. In *International Conference on Learning Representations*, 2025.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation*, pp. 9493–9500, 2023.

Haichao Liu, Haoren Guo, Pei Liu, Benshan Ma, Yuxiang Zhang, Jun Ma, and Tong Heng Lee. VLM-UDMC: VLM-enhanced unified decision-making and motion control for urban autonomous driving. *arXiv preprint arXiv:2507.15266*, 2025.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. LLaVA-NeXT: Improved reasoning, OCR, and world knowledge. https://llava-vl.github.io/blog/2024-01-30-llava-next/, 2024.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems*, 2024.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

Kangan Qian, Zhikun Ma, Yangfan He, Ziang Luo, Tianyu Shi, Tianze Zhu, Jiayin Li, et al. FASIONAD: FAst and slow FusION thinking systems for human-like autonomous driving with adaptive feedback. *arXiv preprint arXiv:2411.18013*, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763, 2021.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. In *International Conference on Machine Learning*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023.

Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, et al. Hume: Introducing System-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112 (1-2):181–211, 1999.

J. K. Terry, Nathaniel Grammel, Sanghyun Son, Benjamin Black, and Aakriti Agrawal. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020.

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. DriveVLM: The convergence of autonomous driving and large vision-language models. In *Conference on Robot Learning*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788, 2020.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34153–34189, 2023.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. JARVIS-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(3):1894–1907, 2025.

Meng Wei, Chenyang Wan, Jiaqi Peng, Xiqian Yu, Yuqiang Yang, Delin Feng, Wenzhe Cai, et al. Ground slow, move fast: A dual-system foundation model for generalizable vision-and-language navigation. *arXiv preprint arXiv:2512.08186*, 2025.

Isadora White, Kolby Nottingham, Ayush Maniar, Max Robinson, Hansen Lillemark, Mehul Maheshwari, Lianhui Qin, and Prithviraj Ammanabrolu. Collaborating action by action: A multi-agent LLM framework for embodied reasoning. *arXiv preprint arXiv:2504.17950*, 2025.

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 *International Conference on Learning Representations*, 2023.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. HiRT: Enhancing robotic control with hierarchical robot transformers. In *Conference on Robot Learning*, 2024.

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

A Model Details

A.1 World-model and control details

RSSM transition and inference. Conditioning on language, the deterministic transition and stochastic prior/posterior are

$$h_t = f_\theta(h_{t-1}, z_{t-1}, a_{t-1}, e_{t-1}),$$
$$p_\theta(z_t \mid h_t) = \mathcal{N}\big(\mu_\theta(h_t), \mathrm{diag}\sigma^2_\theta(h_t)\big),$$
$$q_\phi(z_t \mid h_t, o_t, e_t) = \mathcal{N}\big(\mu_\phi(h_t, o_t, e_t), \mathrm{diag}\sigma^2_\phi(h_t, o_t, e_t)\big).$$

We denote the model state as $s_t = (h_t, z_t)$ with $z_t \sim q_\phi(\cdot)$ during training and $z_t \sim p_\theta(\cdot)$ during imagination.

Decoders and predictors. From s_t we decode/predict:

$$\begin{aligned} p_\theta(o_t \mid s_t) &\quad \text{(observation decoder)},\\ p_\theta(r_t \mid s_t) &\quad \text{(reward model)},\\ p_\theta(c_t \mid s_t) &\quad \text{(continuation/discount model with } c_t \in (0,1)),\\ p_\theta(x_{t+1} \mid s_t) &\quad \text{(next-instruction predictor; optional teacher forcing on } x_t). \end{aligned}$$

The instruction head is trained only when $m_{t+1} = 1$; otherwise its loss is masked out.

Model learning objective. Over a sequence $t = 1{:}T$, the model loss combines reconstruction/prediction terms with a KL regularizer:

$$\mathcal{L}_{\text{model}} = \sum_{t=1}^{T} \mathbb{E}_{q_\phi}\Big[-\log p_\theta(o_t \mid s_t) - \lambda_r \log p_\theta(r_t \mid s_t) - \lambda_c \log p_\theta(c_t \mid s_t) - \lambda_x m_{t+1} \log p_\theta(x_{t+1} \mid s_t)\Big]$$
$$+ \beta \sum_{t=1}^{T} \mathrm{KL}\big(q_\phi(z_t \mid h_t, o_t, e_t) \,\big\|\, p_\theta(z_t \mid h_t)\big),$$

with weights $\lambda_r, \lambda_c, \lambda_x \geq 0$ and KL scale β.

A.2 Dreamer-style Control in Latent Space

Given a learned world model, we learn a policy and value on latent states conditioned on language:

$$\pi_c(a_t \mid s_t, e_t), \qquad V_\psi(s_t, e_t).$$

Imagined rollouts. Starting from posterior states s_t on real trajectories, we 'imagine' H-step futures using the prior dynamics and current policy:

$$\tilde{s}_{t+1} \sim p_\theta(\cdot \mid \tilde{h}_{t+1}), \tilde{h}_{t+1} = f_\theta(\tilde{h}_t, \tilde{z}_t, \tilde{a}_t, \tilde{e}_t), \tilde{a}_t \sim \pi_\eta(\cdot \mid \tilde{s}_t, \tilde{e}_t), \tilde{r}_t \sim p_\theta(\cdot \mid \tilde{s}_t), \tilde{c}_t \sim p_\theta(\cdot \mid \tilde{s}_t),$$

where $\tilde{e}_t$ is the instruction embedding available to the agent during imagination (e.g., last known instruction embedding, or an imagined instruction from $p_\theta(x_{t+1} \mid s_t)$).

Value targets via λ-returns with continuation. Define the continuation as the learned discount $\gamma \tilde{c}_\tau \in [0,1]$. The λ-return is

$$\hat{G}^\lambda_\tau = \tilde{r}_\tau + \gamma \tilde{c}_\tau \Big((1-\lambda) V_\psi(\tilde{s}_{\tau+1}, \tilde{e}_{\tau+1}) + \lambda \hat{G}^\lambda_{\tau+1}\Big), \quad \hat{G}^\lambda_{t+H} = V_\psi(\tilde{s}_{t+H}, \tilde{e}_{t+H}).$$

Actor and value losses. We optimize the value to regress to the return and the actor to maximize it through imagined trajectories:

$$\mathcal{L}_{\text{value}} = \sum_{\tau=t}^{t+H-1} \big\| V_\psi(\tilde{s}_\tau, \tilde{e}_\tau) - \mathrm{stopgrad}(\hat{G}^\lambda_\tau) \big\|_2^2,$$

$$\mathcal{L}_{\text{actor}} = -\sum_{\tau=t}^{t+H-1} \mathbb{E}_{\tilde{a}_\tau \sim \pi_\eta}\Big[\mathrm{stopgrad}(\hat{G}^\lambda_\tau)\Big] - \alpha \sum_{\tau=t}^{t+H-1} \mathcal{H}\big(\pi_\eta(\cdot \mid \tilde{s}_\tau, \tilde{e}_\tau)\big),$$

Table 4: Model-scale configurations.

Scale (tag)	rssm.deter	rssm.hidden	rssm.classes	depth	units
50M (size50m)	4096	512	32	32	512
200M (size200m)	8192	1024	64	64	1024
800M (size800m)	24576	3072	192	192	3072

Method	**Throughput**
RT-2 (Zitkovich et al., 2023)	16.2
DEPS (Wang et al., 2023)	20.5
LS-Imagine (Li et al., 2025)	23.7
JARVIS-1 (Wang et al., 2025)	26.0
Instruct-to-Act (Qwen-2.5-VL-72B)	31.7

Table 5: Inference throughput (environment steps/s) for representative VLA baselines and our framework with a Qwen-based planner.

with entropy scale $\alpha \geq 0$. Gradients backpropagate through the imagined dynamics (world model) as in Dreamer.

A.3 Model Hyperparameters

We use MiniLM-L6-H384-uncased (Wang et al., 2020) as the language encoder. For small controller as an example, we instantiate a language-conditioned DreamerV3-style (Hafner et al., 2025) agent with a CNN encoder (4 convolutional blocks; channels [32,64,128,256], stride 2, SILU), an RSSM with a deterministic GRU core of size 512 and a diagonal-Gaussian stochastic latent $z_t \in \mathbb{R}^{32}$, and decoders for observation/reward/continuation as MLPs (two layers, 512 units, SILU). The actor and value heads operate in latent space and use three-layer MLPs with 512 units (SILU). The policy head produces action logits for discrete control or mean/scale for continuous control, depending on the environment action space. We follow dreamer to use KL balancing with scale β=1.0 and free-nats 1.5, reconstruction/prediction weights $(\lambda_r,\lambda_c,\lambda_x)=(1,1,1)$, imagined horizon H=15 (as above), and continuation-based λ-returns. The language pathway pools a frozen text encoder into an embedding e_t that conditions both the world model and control; when absent, a learned null embedding is used. Optimizer and update schedule follow the hyperparameters above (Adam, 3e$-$4, clip 40, 1:1 world-model/control updates).

Table. 4 show controller scales from 50M, 200M, to 800M. These configs are adapted from Dreamerv3 (Hafner et al., 2025) and re-used the 50M and 200M configuration.

A.4 Training Compute and Annotation Overhead

Unless otherwise stated, GPT-4o is used as the replay annotator and 50% of replay segments are instruction-annotated. Training uses 4$\times$ NVIDIA RTX A6000 GPUs and an Intel Xeon Gold 6338 CPU. Across all tasks, the maximum training time for one environment is approximately 36 wall-clock hours (23 hours on average). Relative to controller-only training, post-hoc annotation adds roughly 17% GPU hours and a 12% training slowdown.

A.5 Additional Results

Efficiency comparison with other works: Table 5 complements the throughput plots with a direct comparison against representative end-to-end or tightly coupled baselines. Because our controller executes at every step while the planner is queried only sparsely, Instruct-to-Act achieves the best throughput among these methods.

	VLM Planner		
Instruction Mode	**Gemma**	**Qwen**	**GPT**
Fixed Cadence	8.86	9.32	9.60
Fully controlled by VLM	9.89	10.12	10.52
Proactiveness decided by VLM	10.12	10.23	10.75

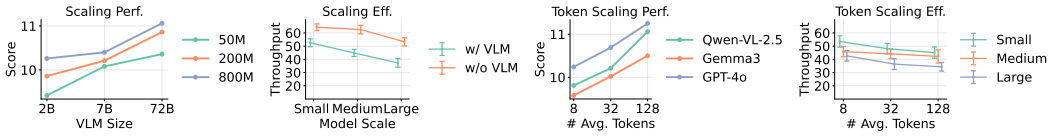
Table 6: Single-agent tests under different instruction modes. (Minecraft Diamond)

Table 7: Controller architecture ablation on Minecraft Diamond. The gains are not specific to an RSSM controller, though world-model controllers perform best.

Controller architecture	**Diamond**
Dreamer-V3-style RSSM (ours)	11.0
Transformer world model (Chen et al., 2022)	11.2
RNN policy (model-free)	9.7
Value-only world model	10.2

Table 8: Instruction-following accuracy measured on 200 held-out planner instructions per domain.

Task	**Gemma**	**Qwen**	**GPT-4o**
Minecraft Diamond	193/200	190/200	194/200
Pico Park	186/200	180/200	189/200
Atari	181/200	172/200	185/200



(a) Model size scaling (Minecraft Diamond) (b) Reasoning scaling (Minecraft Diamond)

Figure 3: Per-setting *performance* (left) and *efficiency* (right) results for the model-size and reasoning scaling analyses in Sec. 4.4. Agent scaling and planning modes are shown in Figure 2 in the main paper. Model scales for controller and VLM are small (50M, 2B), medium (200M, 7B), and large (800M, 72B).

Instruction modes (Minecraft): We contrast three instruction regimes (Table 6): (1) *Fixed cadence*: planners issue instructions every K=16 steps; (2) *VLM-decided proactiveness* (the VLM emits instructions only when its uncertainty exceeds a threshold, with a minimum gap of 32 steps); (3) *Fully controlled by VLM*. We report achievement scores for three planners.

Controller architecture: Table 7 shows that the broader framework is not tied to Dreamer specifically. A transformer world-model controller yields similar trends, whereas removing the world-model structure degrades performance.

Human Evaluated Instruction Following Success: We perform human eval in Table 8 that quantifies controller reliability under planner instructions. Across domains and planners, instruction-following accuracy stays around 86–97%. The most common failures arise from vague or underspecified instructions such as "explore a bit more" or "try something different", which can lead to locally suboptimal behavior until a clearer instruction arrives.

	Minecraft Diamond		Overcooked	
	Online	Offline	Online	Offline
VLM latency mean (ms)	1037.4	1221.9	1106.3	1010.4
VLM latency p95 (ms)	1701.1	2035.7	1872.5	1523.7
Staleness mean (steps)	51.2	0.0	165.1	0.0
Blocking steps mean	0.0	35.4	0.0	0.0
Throughput (env steps/sec)	37.5	13.4	169.9	92.7

Table 9: Latency, instruction staleness, and throughput for online (asynchronous) versus offline (blocking) planning. Overcooked staleness and latency are averaged over both agents.

A.6 Scalability and Efficiency Curves

Figure 2 (main paper) and Figure 3 report paired curves for each scaling axis, with performance on the left and throughput (environment steps/s) on the right. Across all four settings, we observe a favorable frontier: configurations that improve task performance usually retain most of the efficiency gains from decoupled control rather than collapsing to the latency profile of end-to-end VLM action generation. Regarding relative planner performance, Figure 3(a) provides a controlled scaling study in which every planner-controller combination shares matched settings; there, performance on Minecraft Diamond improves monotonically as either planner or controller size grows. Deviations from a pure parameter-count ordering in Table 1 reflect differences in planner pre-training rather than the framework. We also note that Voyager is engineered specifically for Minecraft Diamond, which explains its strong performance on that single task, whereas our framework targets robustness across heterogeneous environments with a single recipe.

Asynchronous planning efficiency. Table 9 quantifies the benefit of asynchrony beyond the two-bar comparison in Figure 2(b). Online planning achieves 2.8× higher throughput on Minecraft Diamond and 1.8× on Overcooked (whose shorter episode horizon limits the gain). Asynchrony introduces instruction staleness of roughly 51 steps (Minecraft) and 165 steps (Overcooked) between a stop signal and the arriving instruction, but because the controller continues executing the previous instruction during this window, staleness affects only the next high-level instruction and never low-level control.

A.7 Multi-agent Failure Mode

Centralized settings tend to bottleneck on the hub planner, while decentralized settings most often fail through conflicting or duplicated subgoal allocation. To quantify this, we manually examined 100 failure cases in Overcooked, defining controller-level failures as instruction-following errors or stale loops, and planner-level failures as duplicated task assignment, unmet preconditions (e.g., serving a soup that is not yet made), or pipeline stalls. Planner-level coordination failure dominates: 96 of 100 failure windows show near-duplicate sub-task assignments across agents (instruction-embedding cosine similarity above 0.7), while only 3 show controller instruction-following failures.

A.8 Taxonomy of Generated Instructions

Across our Minecraft experiments we collected 47,547 instruction instances (3,477 unique). The set is dominated by local action descriptions (82.8%, n=39,363), short imperatives describing a single immediate action such as "Mine stone blocks with a wooden pickaxe" or "Swim across the river towards the trees." The remainder (17.2%, n=8,184) are high-level subgoals that chain a primary action to a downstream task, e.g., "Mine stone to craft a stone pickaxe." We observed no retrospective summaries of behavior.

A.9 Instruction-length Robustness

We generated 100 Minecraft Diamond instructions across 25 length checkpoints (1 to 100 words, 4 per length) spanning navigation, collection, crafting, and survival categories, and measured whether the controller terminates at an appropriate time (judged by an API model), averaged over 5 seeds. The stop head is robust from L=2 through L=50, with no cliff at the training-distribution boundary (L=11), indicating generalization beyond training lengths. Degradation begins gradually above L=50, reaching $\sim$0.55 success at L=100; the only sharp drop is at L=1, where a single word carries insufficient semantic content to condition behavior.

A.10 Training Dynamics of the Composite Loss

Figure 4 shows how each sub-objective evolves over training. Early in training, the composite loss is dominated by the world-model reward term, as the controller has not yet learned to predict environment returns. As training progresses and instruction-relabeled segments accumulate in the replay buffer, the behavior-cloning loss becomes the dominant component. The dynamics loss exhibits an initial spike followed by a steady decrease, reflecting that the policy first encounters many unexpected observations during early exploration and subsequently learns the environment dynamics. Notably, the policy and behavior-cloning objectives continue to improve even as the expanding behavior distribution exposes the world model to new surprises, and we observed no single component collapsing or diverging. Training was stable across this run without loss re-weighting or scheduling beyond the fixed linear combination described in Section 3.2.

A.11 Prompt Details

In this section, we list the prompts we used for training and inference including multi-agents. The summarizer prompt for GPT-4o as shown in Fig. 5 is used for post-hoc training language annotation in Sec. 3.4. The inference prompt as shown in Fig. 6 is used for online reasoning and issuing instructions as mentioned in Sec. 3.1. The multi-agent prompt as shown in Fig. 7 is used for prompting multi-agents communication. The full prompt is simply appending this to the single-agent inference prompt. Note that the math notations are all replaced by actual inputs or in-context examples.

B Tasks Details

We list and describe the evaluation tasks in details here totaling 7 environments. Every environment can be discussed from its goal, the observation space, evaluation protocol, and the metric.

Arcade Learning Environment (ALE)

- **Goal:** Achieve high scores across Atari 2600 titles using standard ALE evaluation.
- **Obs/Act:** Raw visual frames with common preprocessing (frame skip/stack); full primitive action set.
- **Protocol:** Follow the evaluation practices popularized in recent Atari work, e.g., Kaiser et al. (2020); Machado et al. (2018), including fixed evaluation episodes and capped frames per episode; when using sticky actions or ALE "game flavours," also follow the ALE protocol paper.
- **Metric:** Game score per episode; report means (and variance) over seeds.

MineRL `ObtainDiamond`

- **Goal:** Obtain a diamond in survival Minecraft starting from scratch.
- **Obs/Act:** 64×64 first-person RGB plus discrete inventory observations; actions combine continuous camera control with discrete navigation, mining/crafting/smelting interactions.

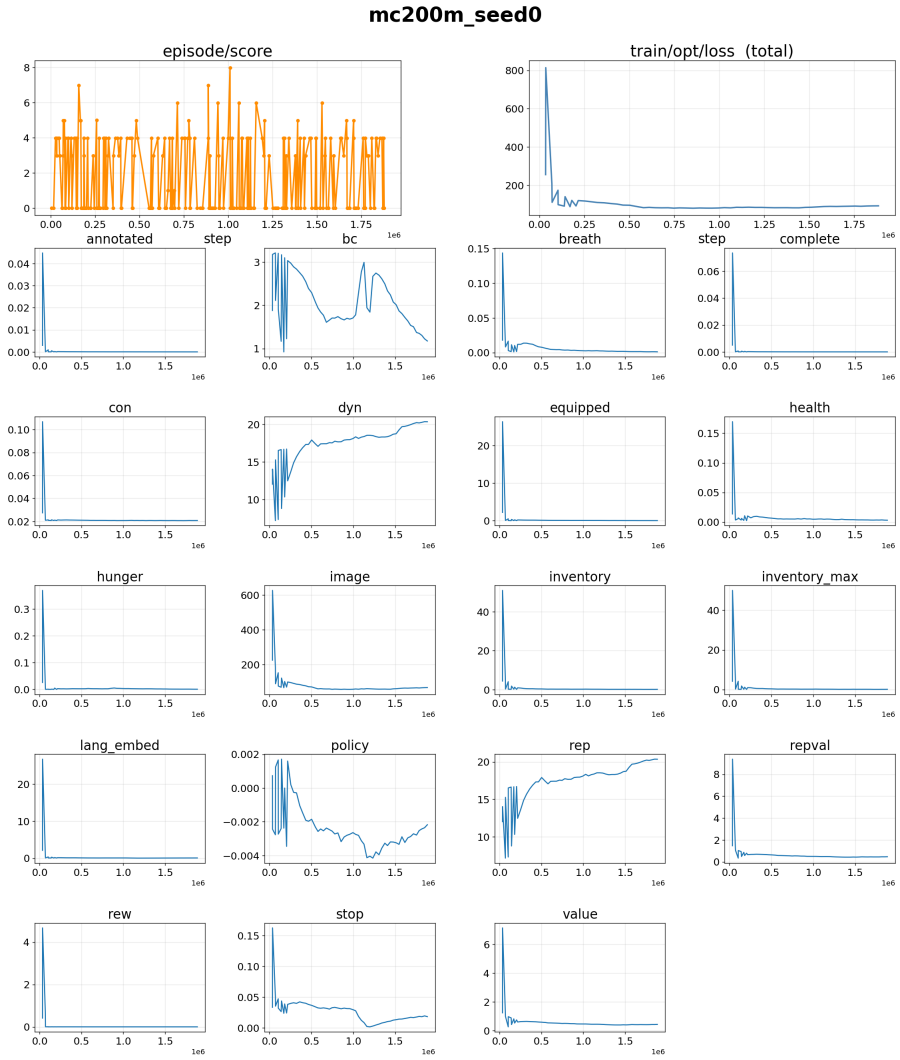


Figure 4: Training curves of the individual loss components for a 200M-parameter controller on Minecraft Diamond (seed 0). The overall objective is a linear combination of the world-model losses (reconstruction, dynamics, and reward prediction), the actor and value objectives, the behavior-cloning (BC) loss on instruction-relabeled segments, and the stop-token loss.

- **Protocol:** Episode terminates on death, diamond obtained, or 18,000-frame (15 min) limit. Competition evaluation averages performance over 500 episodes on fixed but unseen seeds; strict train-from-scratch compliance.
- **Metric:** Shaped milestone reward (e.g., logs, planks, pickaxes, iron, diamond), summed per episode; tie-break by fewest episodes to last milestone.

Crafter

- **Goal:** Survive and progress in a procedurally generated 2D open world by unlocking semantically meaningful achievements (e.g., find water, craft tools, defeat enemies).

You are a concise action summarizer for training labels. Read inputs IN THIS ORDER:

1. TaskGuidance G (what succeeds in this environment) and a short Message M (any extra note),
2. sampled visual Frames F[t : t+L-1] with timestamps,
3. the executed Actions A[t : t+L-1] with timestamps.

Goal

produce ONE short imperative instruction x that, if given before t, would make a competent controller reproduce A[t : t+L-1]. Keep it atomic and task-grounded.

Rules

- Focus on intent, not low-level joystick/button spam.
- Refer only to what is visible/achieved in Frames and consistent with Actions.
- No hallucinations; if ambiguous, pick the MINIMAL instruction that explains A.
- 6–18 tokens; avoid conjunction chains; one clause; present tense verb first word.
- If the segment ends at completion, align x to that subgoal's completion boundary.

Output JSON ONLY

```
{
 "instruction": "<single-line imperative>",
 "justification": "<8-20 token rationale tied to frames/actions>",
 "segment": {"t_start": t, "t_end": t+L-1}
}
```

[USER]

TaskGuidance (G): {G}
Message (M): {M}
Frames F[t:t+L-1]: {FRAME_LIST_OR_CAPTIONS}
Actions A[t:t+L-1]: {ACTION_LIST}

Figure 5: Training summarization prompt.

- **Obs/Act:** Local top-down pixel observations showing surroundings and inventory status; discrete actions for movement, interaction, crafting, sleeping, placing.
- **Protocol:** Two tracks: with extrinsic rewards and reward-free. Agents get a fixed budget of 1M environment steps (commonly also reported at 5M in baselines). Success rates computed across the entire training run to emphasize sample efficiency.
- **Metric:** *Crafter score* is the geometric mean of the 22 achievement success rates, emphasizing breadth/depth of capabilities.

DeepMind Lab: `explore_goal_locations`

- **Goal:** First-person 3D navigation in maze-like levels to discover and reach goal locations under partial observability.
- **Obs/Act:** RGB first-person frames; continuous look and movement controls.
- **Protocol:** Standard DM-Lab evaluation with fixed episode caps; commonly evaluated on held-out maps (unseen layouts), sometimes with environment-provided debug info *only* for visualization/analysis (not agent inputs).
- **Metric:** Episode return (goals found/reached), success rate, and path efficiency; report means over seeds/maps.

Overcooked-AI

- **Goal:** Two-player cooperative cooking (deliver soups quickly) with strong *zero-shot coordination* (ZSC) to unseen partners and layouts.

[SYSTEM = Sp]
You are the planner module π_p running ASYNCHRONOUSLY with a low-latency controller π_c.

Maintain

- Reasoning memory M_r (brief scratch notes).
- Partial plan P_n (bullet list of next subgoals → micro-steps).

You receive a live observation stream O_{<t'} and a stop signal probability p_stop_{t'-1} from π_c.

Emit a NEW instruction x ONLY when either (a) p_stop_{t'-1} is true/high (≥ 0.5), or (b) you proactively decide the current plan needs revision. Otherwise ADVANCE your background plan silently.

Constraints

- Instructions are 6–18 tokens, single-clause, imperative, specific and verifiable from observations.
- Never block control; if unsure, issue the safest non-destructive next step.
- Keep M_r and P_n ultra-compact; prefer deltas over full rewrites.

Output JSON ONLY

```
{
 "emit": <true|false>,
 "instruction": "<if emit=true, the instruction; else empty>",
 "reasoning_update": "<<=40 tokens delta for M_r>",
 "plan_update": ["<<=12 tokens step 1>", "<step 2>", "..."],
 "confidence": <0.0-1.0>
}
```

[USER]

Previous state

- M_r(t'-1): {M_r_prev}
- P_n(t'-1): {P_n_prev}
- p_stop_{t'-1}: {p_stop_prev}

New context

- O_{<t'} (latest frames/summaries): {OBS_STREAM_SNIPPET}
- Task spec (if any): {TASK_SPEC}

Figure 6: Single-agent prompt.

- **Obs/Act:** Gridworld state or egocentric features; discrete actions (move, interact, pick/place).
- **Protocol:** Evaluate on canonical layouts (*Cramped Room*, *Asymmetric Advantages*, *Coordination Ring*, etc.) and out-of-distribution layouts; cross-play with behavior-cloned human models and held-out agents; human studies where applicable.
- **Metric:** Team return (deliveries/time), success rate, and ZSC scores (cross-play averages across unseen partners/layouts).

Pico Park (Co-op Puzzles)

- **Goal:** Complete short, cooperative puzzle-platforming levels that demand synchronized actions (e.g., stacking, shared switches, tethered movement).
- **Obs/Act:** Platformer state with simple discrete controls for 2–8 players (levels scale with player count).

[SYSTEM = Sp_multi_agent_addon]
You are planner i in a cooperative team of n agents. You may send at MOST ONE short message per step.

Inbox processing

first read Inbox[i] (messages addressed to you or broadcast). Update M_r and P_n accordingly.

Communication modes

- Decentralized: you decide when to message.
- Centralized (hub=h): only agent h broadcasts role/assignment messages; others reply sparsely.

Messaging rules

- Keep messages ≤ 25 tokens.
- Prefer structured intents: {claim:<subgoal>}, {status:<brief>}, {block:<issue>}, {request: <help on X>}, {handoff:<asset/role>}.
- Addressing: use @all for broadcast or @j for a specific agent j.
- Do not restate observations verbatim; send actionable deltas that change teammates' plans.

Augmented OUTPUT (append to (2)'s JSON)

```
"comm": {
 "send": <true|false>,
 "to": "@all" | "@j" | "@hub",
 "message": "<one line following the intent tags above>"
}
```

Figure 7: Multi-agent prompt (to be appended to single-agent prompt).

- **Protocol:** Curate a fixed subset of multiplayer levels (e.g., 48 classic levels) and evaluate multiple seeds/player configurations; require that all agents reach the goal to clear a level.
- **Metric:** Level completion rate and median completion time across the suite; optionally, coordination error counts (drops, desyncs).

MindCraft (Minecraft Multi-Agent Collaboration)

- **Goal:** Multi-agent embodied collaboration on *Cooking*, *Crafting*, and *Construction* tasks via language-enabled coordination.
- **Obs/Act:** First-person Minecraft control with inventories; agents exchange natural-language messages; tasks provide recipes/blueprints and split resources/knowledge across teammates.
- **Protocol:** Procedurally generated tasks per category; for construction, initialize agents with blueprints and disjoint materials/skills; for cooking/crafting, vary recipe complexity and information asymmetry ("Hell's Kitchen" variants).
- **Metric:** Average success rate for Cooking & Crafting; Construction uses an edit-distance alignment between the built structure and the target blueprint; overall score averages category scores.

C Algorithm

Shared-state semantics. Inbox, Stop, and Plan are synchronized through step-indexed buffers rather than arbitrary cross-thread writes. The controller writes a completion event when p_{stop} crosses its threshold. If this happens while the planner is mid-generation, the planner finishes its current ADVANCE call using the latest available observation context and

Algorithm 1 Online planning at timestep t

Input: environment P, controller π_c, VLM planner π_p, `Stop` $\in \{0,1\}$, `Inbox` $\in \mathcal{X}$, `Plan` $\in \mathcal{X}$, latest observation o_t.

thread Controller():
  while running **do**
    $(a_t, s_t) \leftarrow \pi_c.\text{step}(o_t, x_t)$
    $P.\text{step}(a_t)$
    if s_t **then** `Stop` $\leftarrow 1$; $x_{t+1} \leftarrow \varnothing$
    if `Inbox` **then** $\mathbf{x}_{t+1} \leftarrow$ `Inbox`

thread VLM():
  while running **do**
    if `Stop` **then**
      `Inbox` $\leftarrow \pi_p.\text{emit}(o_t, \texttt{Plan})$
      `Stop` $\leftarrow 0$; `Plan` $\leftarrow \varnothing$
    else
      `Plan` $\leftarrow \pi_p.\text{advance}(o_t, \texttt{Plan})$

Algorithm 2 Offline planning at timestep t

Input: environment P, controller π_c, VLM planner π_p, `Stop` $\in \{0,1\}$, latest observation o_t, plan state Pn_t.

while running **do**
  ▷*Add to the current plan, potentially emitting an instruction.*
  $(x_t, Pn_{t+1}) \leftarrow \pi_p.\text{step}(o_t, Pn_t, \texttt{Stop})$
  if x_t **then** `Stop` $\leftarrow 0$

  ▷*Execute the existing instruction.*
  $(a_t, s_t) \leftarrow \pi_c.\text{step}(o_t, x_t)$
  $P.\text{step}(a_t)$
  if s_t **then** `Stop` $\leftarrow 1$

stores the resulting draft plan in Plan. At the planner's next scheduling point, the completion event triggers a short EMIT call that converts the draft into the next executable instruction and writes it atomically to Inbox; the controller takes in the latest instruction at the next control step. Observation staleness is therefore bounded by the duration of the planner's current background reasoning call plus one control-step boundary, and the controller continues executing its previous instruction throughout, so staleness affects only the next high-level instruction and never low-level control. In terms of token cost, ADVANCE carries most of the reasoning tokens, as it updates the planner's memory and partial plan while the controller acts, whereas EMIT only decodes a short instruction from the existing draft. This asymmetry is why asynchronous planning reduces instruction-boundary latency relative to blocking inference (quantified in Table 9).

Multi-agent chatroom synchronization. Each agent runs its own asynchronous planner thread and controller thread. Planner outputs, including chat messages and instructions, are never injected into the environment at arbitrary wall-clock times. Instead, they are written to a per-agent mailbox and consumed at the next synchronized communication/control boundary, at which point each agent reads the latest incoming messages and updates its planner context. Communication thus proceeds in discrete rounds aligned with control steps while planner reasoning proceeds asynchronously in the background.

Planner observability. In all environments, the planner receives only the observations available to the agent itself. No privileged simulator state, full-state access, or hidden information is provided, ensuring the framework applies to settings where privileged information is unavailable.

D Instruction-following evaluation protocol.

Instruction-following accuracy was judged by one of the authors on 200 held-out planner-generated instructions per domain. The evaluator was shown the instruction, the corresponding observation/trajectory segment, and a real-time video of the controller's execution, and judged whether the behavior completed the intended instruction. As a sanity check, we repeated the same annotation task with an API model (GPT-5.5), which agreed with the human annotator on 93.2% of cases across tasks. We emphasize that single-annotator judgment combined with model agreement serves as a sanity check rather than a substitute for independent multi-annotator human evaluation, and we present these accuracies with that caveat.

D.1 Additional Training and Communication Details

When collecting a rollout of length T, different timesteps t may be associated with different instruction embeddings e_t and completion indicators complete_t, whereas done_t is true if and only if $t = T$. To sample annotation segments, a segment length L is first randomized within an integer interval, then a start index is drawn as $t \sim \text{Uniform}\{1,\ldots,|\mathcal{D}|-L\}$; sampled segments are constrained to be non-overlapping.

Formally, for multi-agent communication with n planner agents, at each VLM inference step any planner i may initiate at most one message m_{ij} to planner j in the decentralized mode. In the centralized mode, the sender is a fixed hub agent h that sends messages m_{hi} to all $i \in [n] \setminus h$.