THE RIGHT ADVICE AT THE RIGHT TIME: LEARNING PROACTIVE ASSISTANCE THROUGH NESTED CREDIT ASSIGNMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

In complex interactive tasks, users may spend considerable time pursuing ineffective strategies before pausing to seek external help. Proactive assistance can address this gap by offering contextual guidance without waiting for explicit requests or taking over task execution. Learning such assistance involves two coupled decisions: when to intervene and what advice to provide. Whether intervention is worthwhile depends on the advice provided, while evaluating that advice must account for the user's ability to progress without assistance. Downstream task outcomes therefore provide ambiguous feedback for learning intervention timing and advice content. In this work, we propose PACE (Proactive Assistance through nested Credit Estimation), a reinforcement learning framework that jointly trains a generative vision–language companion through nested credit assignment. By comparing estimated returns under silence and alternative advice from the same interaction state, PACE derives distinct learning signals for the intervention decision and conditional advice generation. These signals guide joint optimization of timing and content while balancing task progress against intervention cost. Experiments with multiple LLM-controlled players in *ALFWorld* and *Minecraft* show that PACE achieves higher overall task success rates with lower intervention rates than general-purpose LLMs and specialized game-playing models used as companions. A human study further demonstrates the effectiveness of PACE.

1 INTRODUCTION

AI assistants are increasingly applied across various domains to support users in complex interactive tasks by offering guidance on what to do next (Li et al., 2023; Wu et al., 2023). However, assistance that depends on explicit requests requires users to recognize when they need help and know what to ask (Ouyang et al., 2022). Users may instead spend considerable time and effort on an ineffective approach without realizing that they are not making progress toward their goal (Lu et al., 2025). Seeking external guidance may also interrupt the ongoing task. These limitations point to **proactive assistance**, in which an AI companion observes the interaction and offers contextual guidance without waiting for a request (Schilit & Theimer, 1994). The aim is to help users overcome difficulties earlier while leaving them in control of decisions and task execution. At the same time, proactive assistance should remain selective, since unnecessary interventions can interrupt the ongoing activity even when the guidance itself is useful (Altmann & Trafton, 2002; Adamczyk & Bailey, 2004). We study this problem in interactive game environments, which provide long-horizon, goal-directed interactions together with measurable task progress for evaluating the effect of assistance.

Effective proactive assistance therefore requires learning two coupled decisions, *when to intervene* and *what advice to provide* (Ouyang et al., 2022; Lu et al., 2025). The companion should intervene only when the expected benefit of assistance justifies the interruption, and this benefit depends on the advice delivered at that moment. At the same interaction state, a generic reminder may offer little benefit, whereas a specific suggestion addressing a missing prerequisite may help the user proceed. Conversely, even correct advice may be unnecessary when the user already knows the next step. Whether intervention is worthwhile therefore depends on the advice available, while evaluating that advice must account for the user's ability to progress without assistance. This naturally forms a **nested decision structure** in which the companion first chooses between intervention and

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.

silence, and generates advice only when intervention is selected. Learning effective proactive assistance therefore requires intervention timing and advice content to be considered jointly rather than optimized in isolation.

Existing proactive systems have begun to study when assistance should be offered and what form that assistance should take, but they are often trained with supervision that does not directly reflect the downstream effect of assistance on task progress. Proactive Agent (Lu et al., 2025) learns to anticipate assistance opportunities using human judgments on proposed tasks, such as whether they should be accepted or rejected, while CoLabScience (Wu et al., 2026) trains intervention detection and content generation using labeled intervention points and reference-based objectives. Related work (Nath et al., 2026; Arakawa et al., 2025) on collaborative dialogue further considers how interventions influence subsequent interactions through simulated collaborator responses. While these approaches demonstrate the value of proactive intervention, their training signals largely reflect labels, references, or judgments rather than whether assistance actually helped the user progress.

Constructing such distinct credit signals is challenging because the companion affects task outcomes only through the player's subsequent actions. After an intervention, successful progress may result from the advice, the player's own ability, or both. Conversely, a poor outcome does not reveal whether the intervention occurred at an ineffective moment, the advice itself was unhelpful, or the player was unable to make use of it. A single downstream return therefore evaluates the combined interaction but does not directly indicate whether the companion should have remained silent or provided different advice. This creates a **nested credit-assignment problem** because advice content is defined only when the companion intervenes, while the value of intervention depends on the advice being provided. Effective learning therefore requires distinct credit signals for the intervention decision and the conditional advice generation.

In this work, we propose PACE (*Proactive Assistance through nested Credit Estimation*), a reinforcement learning framework for jointly learning when to intervene and what advice to provide. PACE uses a single generative vision-language model to produce both the intervention decision and the corresponding advice when assistance is selected. The model is first initialized with assistance demonstrations to provide a reasonable starting policy. To address the nested credit-assignment problem, PACE then evaluates alternative continuations from the same interaction state during training, including remaining silent and providing different advice. These comparisons provide different learning signals for the two decisions. The intervention decision is guided by whether the realized advice improves over silence, while advice generation is guided by the relative performance of alternative messages. PACE further accounts for intervention cost so that the companion learns to improve task progress without relying on frequent assistance.

We evaluate PACE in *ALFWorld* (Shridhar et al., 2020) and *Minecraft* with multiple LLM-controlled players, together with a human study in *Minecraft*. Experimental results show that PACE consistently improves task performance over strong companion baselines and is able to effectively support real players.

Our main contributions are summarized as follows:

- We identify nested credit assignment as a key challenge in learning proactive assistance, where a companion must jointly decide *when to intervene* and *what advice to provide*. The objective is to improve user progress while keeping assistance minimal, with both decisions learned from downstream task outcomes mediated by the user's subsequent actions.

- We propose PACE, a reinforcement learning framework for proactive assistance that performs nested credit estimation from matched branch returns, deriving distinct learning signals for intervention timing and advice generation. PACE further incorporates a progress-aware intervention cost to encourage selective assistance and milestone-focused advice generation to explore task-relevant guidance across different planning horizons.

- We show that this learned assistance policy transfers beyond the training advisee. A single PACE companion generalizes to held-out LLM players across *ALFWorld* and *Minecraft*, while human evaluation provides complementary evidence that the learned assistance remains effective for real users.

2 Related Work

Large language models have enabled increasingly capable interactive agents with strong reasoning, planning, and task-solving abilities across complex environments (Zhang et al., 2024; Brohan et al., 2023; Yao et al., 2022; 2023; Schick et al., 2023). However, most LLM-based assistants remain reactive, providing support primarily in response to explicit user requests. Proactive assistance instead leverages ongoing activity and context to anticipate when support may be useful, building on the broader tradition of mixed-initiative interaction and interruption management (Horvitz, 1999; Adamczyk & Bailey, 2004). Recent LLM-based systems have begun to adopt this paradigm across general-purpose assistance, programming, planning, and conversational interaction, enabling agents to infer latent user needs without explicit requests (Lu et al., 2025; Chen et al., 2025; Liu et al., 2025). Most closely related to our work, several studies learn when or how to intervene during ongoing interaction using labeled intervention points, reference responses, or simulated collaborator feedback (Wu et al., 2026; Nath et al., 2026). These works highlight the importance of both intervention timing and assistance content, but do not explicitly decompose downstream task outcomes into distinct learning signals for these two coupled decisions.

This credit-assignment challenge becomes more pronounced in multi-step agent settings, where the effect of an individual decision may only be reflected after several subsequent interactions. Recent RL methods for LLM agents primarily formulate credit assignment over the target agent's own decisions, estimating trajectory or step-level advantages for actions produced by the policy itself (Feng et al., 2026; He et al., 2026; Li et al., 2026; Wei et al., 2025). A proactive companion differs in that its outputs affect task outcomes only through an advisee, making the learning signal inherently mediated by the advisee's subsequent behavior. Nath et al. (2026) account for such mediation when aligning intervention agents, but do not learn when to intervene. These formulations therefore do not directly capture this mediated decision setting.

Group-relative policy optimization estimates advantages by comparing multiple sampled outcomes within a group (Shao et al., 2024), with recent extensions introducing finer-grained step-level and hierarchical credit assignment (Feng et al., 2026; He et al., 2026). PACE instead structures these comparisons around matched assistance branches, assigning separate credit to the nested intervention-timing and advice-content decisions.

3 Preliminaries

Problem setup. We consider a sequential interaction among a companion, an advisee, and an environment. The companion provides guidance, while the advisee selects and executes environment actions. To provide a consistent basis for evaluating alternative assistance decisions, the advisee policy π^{adv} is held fixed during training.

More specifically, at timestep t, the companion observes the interaction context s_t and samples an intervention gate $g_t \in \{\textsc{help}, \textsc{silence}\}$. When $g_t = \textsc{help}$, it generates an advice message a_t. Otherwise, it remains silent, with $a_t = \varnothing$. The two decisions are specified by:

$$g_t \sim \pi_\theta^g(\cdot \mid s_t), \qquad a_t \sim \begin{cases} \pi_\theta^a(\cdot \mid s_t), & g_t = \textsc{help}, \\ \varnothing, & g_t = \textsc{silence}. \end{cases}$$

where π_θ^g and π_θ^a denote the gate and conditional advice distributions of the same companion model, and $\varnothing$ is a point mass at the null advice $\varnothing$. This defines a nested decision structure in which advice generation is conditional on the decision to intervene.

Then the advisee selects an environment action, $u_t \sim \pi^{\mathrm{adv}}(\cdot \mid s_t, a_t)$. It may follow, partially follow, or ignore the advice, retaining control over the executed action. After this action is executed, the interaction yields reward r_t and an updated context s_{t+1}.

Learning Objective. Let $\tau = \{(s_t, g_t, a_t, u_t, r_t)\}_{t=1}^{T}$ denotes an interaction trajectory of length T. The companion aims to improve the advisee's task performance while avoiding unnecessary interventions. We therefore optimize $\max_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R_{\mathrm{task}}(\tau) - C_{\mathrm{int}}(\tau)\right]$, where R_{task} measures task performance and C_{int} denotes the cumulative discounted intervention cost. Here, $\tau \sim \pi_\theta$ denotes trajectories induced by the companion's interaction with the fixed advisee and the environment.

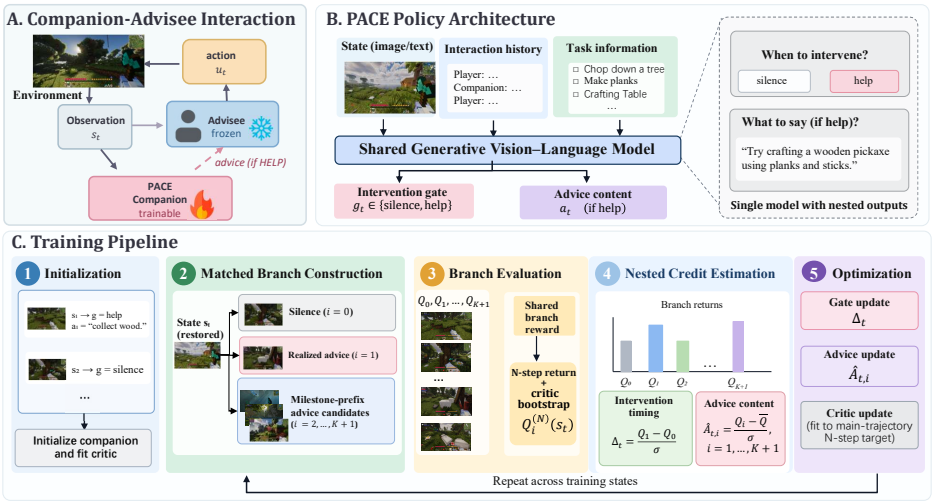


Figure 1: **Overview of PACE.** (A) The companion advises a frozen advisee that controls environment actions. (B) A shared vision–language model jointly predicts intervention timing and advice. (C) PACE learns through matched branch evaluation, nested credit estimation, and separate gate and advice updates.

4 Proactive Assistance through Nested Credit Estimation

As shown in Figure 1, we propose PACE to provide useful advice at appropriate moments while avoiding unnecessary interventions. To model the dependence between timing and content, it uses a single generative vision-language model that produces an intervention decision and generates advice when intervention is selected. Training consists of supervised initialization from assistance demonstrations, followed by reinforcement learning to refine both decisions based on task progress and intervention cost.

To obtain feedback for the two decisions, the reinforcement-learning stage begins with matched branch construction. From the same restored interaction state and with a fixed advisee, PACE re-simulates continuations under silence, under the realized advice, and under alternative messages. Milestone-focused generation encourages task-relevant advice alternatives at different planning horizons. Branch evaluation then estimates their returns using a reward that balances task progress against a progress-aware intervention cost, discouraging closely repeated assistance without new progress. These estimates support nested credit estimation, which compares the realized advice with silence to guide the intervention gate and uses returns relative to the evaluated group, including silence, to guide advice generation. Finally, the resulting credit signals guide updates to the corresponding outputs of the shared companion model. Note that state restoration and branch evaluation are used only during training and are not required at deployment.

4.1 Supervised Initialization

We initialize PACE using teacher-generated demonstrations of both intervention and silence, providing a starting assistance policy before optimizing its downstream usefulness through reinforcement learning.

We train separate models for *ALFWorld* and *Minecraft*. For both environments, assistance demonstrations pair interaction contexts sampled at individual trajectory steps with GPT-4o-generated intervention decisions and advice (Hurst et al., 2024). *ALFWorld* uses LLM-player trajectories. For *Minecraft*, we prepare additional visual supervision by sampling frames from human gameplay videos B.1 and obtaining GPT-4o captions describing the game scene, items, and on-screen status information. These captions support domain adaptation rather than directly supervising assistance behavior.

Both companions undergo supervised fine-tuning with next-token cross-entropy over the complete $\{gate, advice\}$ response conditioned on the interaction context, without milestone-prefix conditioning. For *Minecraft*, caption-based domain adaptation precedes this assistance fine-tuning. We then

initialize the critic using discounted returns-to-go from all visited states in complete episodes collected by the fine-tuned companion with the fixed advisee. These targets are computed without bootstrapping and use a simplified reward consisting of task success minus a fixed intervention cost. Milestone-prefix conditioning is introduced during reinforcement learning and retained at deployment. Further details of data preparation, domain adaptation, and critic initialization are provided in Appendix B.1.

4.2 Matched Branch Construction

PACE constructs $K + 2$ matched continuations at states s_t where the companion selects HELP, enabling comparisons with silence and alternative advice. All branches start from the same restored environment state and interaction history, with a frozen advisee and common random numbers. They comprise a silence branch ($i = 0$), a replayed-advice branch using $w_1 = a_t$ ($i = 1$), and K new advice branches ($i = 2, \ldots, K + 1$).

To encourage task-relevant exploration across different planning horizons, new messages are conditioned on prefixes of the ordered task milestones $\mathcal{M} = (M^{(1)}, \ldots, M^{(L)})$. Let $\mathcal{M}_t^{\text{rem}} = (M_t^{(1)}, \ldots, M_t^{(L_t)})$ denote the remaining milestones, provided to the companion but not directly to the advisee. For each fresh branch, we define

$$z_{t,i} = \left(M_t^{(1)}, \ldots, M_t^{(d_i)}\right), \qquad d_i = \min(i - 1, L_t), \tag{1}$$

$$w_i \sim \pi_\theta^a(\cdot \mid s_t, z_{t,i}), \quad i = 2, \ldots, K + 1. \tag{2}$$

The prefix lengths increase up to the number of remaining milestones, encouraging advice that ranges from immediate guidance to longer-term planning. When multiple candidates use the same prefix, they remain separate samples from the advice policy. Note that only the initial gate and advice are fixed. Subsequent decisions follow the current companion policy using each branch's evolving context. Milestone-prefix conditioning is retained at deployment, whereas state restoration and branch evaluation are used only during training.

4.3 Branch Evaluation

PACE evaluates all matched branches under a shared reward that balances task progress against intervention cost. Task-success rewards capture completion, while milestone rewards provide intermediate feedback,

$$r_t = r_t^{\text{succ}} + \sum_{m=1}^{L} r_m \, \mathbf{1}[M^{(m)} \text{ first completed at } t] - \mathbf{1}[g_t = \text{HELP}] \, c_t, \tag{3}$$

where r_t^{succ} denotes the task-success reward and r_m rewards the first completion of milestone $M^{(m)}$. Milestone completion is tracked over the full episode, including the history preceding branch construction, so previously completed milestones are not rewarded again.

To encourage spacing between interventions, we make the additional intervention penalty decrease with the number of steps since the previous HELP. Milestone completion also relaxes this penalty, as a new task stage may make further guidance relevant. We define

$$c_t = c_{\text{base}} + c_{\text{extra}} \exp\left(-\frac{\tilde{d}_t}{\tau_c}\right), \tag{4}$$

where c_{base} is the base cost, c_{extra} controls the additional penalty, and τ_c controls its decay. The effective recency $\tilde{d}_t$ measures the steps since the most recent HELP and uses only information available before the current intervention. It is set to $+\infty$ when no previous intervention exists or a new milestone has been completed since that intervention. Consequently, the first intervention and interventions following new progress incur only the base cost. Further assistance without progress remains possible when its expected benefit outweighs the penalty.

To limit simulation cost, each branch is evaluated for at most N steps. For branch i, we estimate its return as

$$Q_i^{(N)}(s_t) = \sum_{j=0}^{H_i - 1} \gamma^j r_{t+j}^{(i)} + \mathbf{1}[H_i < T_i] \, \gamma^{H_i} V_{\phi^-}\left(s_{t+H_i}^{(i)}\right), \qquad H_i = \min(N, T_i). \tag{5}$$

Here, γ is the discount factor and T_i is the remaining number of steps until task termination. The critic V_{ϕ^-}, held fixed during branch evaluation, supplies a learned estimate of the return beyond the rollout horizon. Combining this estimate with the observed rewards yields the approximate branch return $Q_i^{(N)}(s_t)$. The bootstrap term is omitted upon task termination but retained when evaluation stops solely because the N-step horizon is reached.

For simplicity, we write $Q_i = Q_i^{(N)}(s_t)$. Each branch return accounts for subsequent task progress and the costs of further interventions, providing a common basis for the credit comparisons in the next subsection. For subsequent critic training, the regression target $\hat{Q}_t$ is computed along the main trajectory using the same reward and bootstrap rule, rather than the simplified initialization reward.

4.4 Nested Credit Estimation

Using the matched branch returns, PACE constructs different credit signals for the two nested decisions. The intervention gate receives feedback on whether delivering the realized advice is preferable to remaining silent, while advice generation receives feedback on each message's performance relative to the evaluated group.

We first compute the mean and a shared normalization scale over all $K + 2$ branch returns:

$$\bar{Q} = \frac{1}{K+2} \sum_{i=0}^{K+1} Q_i, \qquad \sigma = \max\left(\mathrm{std}\left(\{Q_i\}_{i=0}^{K+1}\right), \sigma_{\min}\right), \tag{6}$$

where $\sigma_{\min} > 0$ limits amplification when the branch returns have little variation. The scale σ is shared by both credit signals, while $\bar{Q}$ serves as the baseline for advice credit.

For the intervention gate, we compare the replayed-advice return Q_1 with the silence return Q_0,

$$\Delta_t = \mathrm{clip}\left(\frac{Q_1 - Q_0}{\sigma}, -\kappa, \kappa\right). \tag{7}$$

A positive Δ_t supports the realized intervention, whereas a negative value indicates that silence has the higher estimated return. This comparison accounts for both task progress and intervention cost through the shared reward. Clipping bounds the magnitude of the gate credit.

For advice generation, each HELP branch receives credit relative to the mean return of the full group, including silence. An advice-only baseline distinguishes stronger and weaker messages but does not explicitly reference what happens when the current intervention is withheld. Including silence makes the baseline sensitive to the player's continuation without immediate guidance, rather than only to the outcomes of alternative messages. We compute

$$\hat{A}_{t,i} = \frac{Q_i - \bar{Q}}{\sigma}, \qquad i = 1, \ldots, K+1. \tag{8}$$

Messages with returns above the group mean receive positive credit, while those below it receive negative credit. The content signal therefore incorporates both differences among messages and the group's performance relative to silence. Positive content credit indicates performance above the group mean, not necessarily above silence. Advice credit is assigned only within the Help decision, preserving the nested structure.

4.5 Optimization

Given the timing and content credits, PACE optimizes the two outputs of the shared companion model using the PPO clipped objective (Schulman et al., 2017). Separate token masks and clipping ranges are applied to the intervention gate and advice content. Specifically, the gate update uses $A^{(g)} = \Delta_t$ on the intervention decision token, while the content update uses $A^{(c)} = \hat{A}_{t,i}$ on the advice tokens of branch i.

The critic provides the bootstrap estimate for the truncated branch returns and is trained toward the stop-gradient main-trajectory target $\hat{Q}_t$,

$$\mathcal{L}_V = \beta_v \, \mathbb{E}_{t \in \mathcal{H}} \left[\left(V_\phi(s_t) - \mathrm{sg}(\hat{Q}_t) \right)^2 \right], \tag{9}$$

where β_v controls the value-loss weight and $\mathcal{H}$ denotes the states used for critic regression.

Optimization is performed in two passes. The content pass updates the advice output together with the critic using the content credit $\hat{A}_{t,i}$. The gate pass then updates the intervention output using Δ_t, with the critic held fixed. We apply KL-based early stopping to the gate update to limit large policy changes. The complete optimization procedure and implementation details are provided in Appendix A.

5 Experimental Setup

Environments. We evaluate PACE in *ALFWorld* (Shridhar et al., 2020) and *Minecraft*, covering text-based and vision-conditioned sequential decision making. In *ALFWorld*, we evaluate all six task categories, including Pick, Look, Clean, Heat, Cool, and Pick2. In *Minecraft*, we consider three multi-stage tasks, Craft Wooden Pickaxe, Make Furnace, and Obtain Diamond. During training, PACE evaluates alternative continuations from replayable interaction states. *ALFWorld* directly supports state restoration, while *Minecraft* states are reconstructed using environment commands.

Players and baseline companions. We evaluate a **player (advisee)** $\times$ **companion** grid to assess cross-player generalization. In *ALFWorld*, players include Qwen3.5-4B (**Qwen**) (Qwen Team, 2026), GPT-4o-mini (**ChatGPT**) (OpenAI, 2024), DeepSeek-V4-Flash (**DeepSeek**) (Xu et al., 2026), and GiGPO-7B (**GiGPO**) (Feng et al., 2026); in *Minecraft*, GiGPO-7B is replaced by Andy 4.2 (**Andy**) (Mindcraft-CE, 2026). PACE is trained against a frozen Qwen3.5-4B (Qwen Team, 2026) player. Companion baselines include prompt-only GPT-4o-mini, base Qwen3.5-9B (**Qwen**) companion, and GiGPO-7B in *ALFWorld*. All companion methods receive the same task-level milestone context, and Solo denotes no assistance.

Evaluation metrics. Across both environments, we report task success rate (SR, %), defined as the fraction of episodes in which the task goal is successfully completed, and intervention frequency (HELP, %), defined as the proportion of interaction steps at which the companion chooses to intervene. We consider both jointly, since an effective proactive companion should improve task performance without relying on excessive intervention. For the longer-horizon *Minecraft* tasks, we additionally report milestone progress (MS), capturing partial progress in episodes that do not reach the final goal.

Training details. We use Qwen3.5-VL-9B with a frozen backbone and vision encoder, updating only LoRA (Hu et al., 2021) adapters and a lightweight value head. PACE constructs $K+2$ matched branches per intervention state and applies separate content and gate PPO updates with $\gamma=0.95$. See Appendix B.2 for more details.

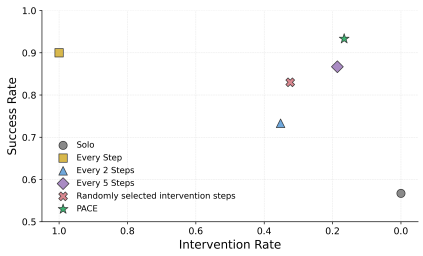
6 Results and Discussion

6.1 Main Results

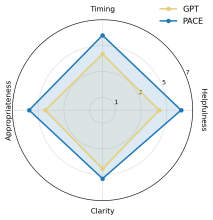
Tables 1 and 3 report the main results across *ALFWorld* and *Minecraft*. PACE achieves the highest overall success rate for all evaluated players in both environments, including the held-out advisees, while consistently requiring fewer interventions than the companion baselines. The gains are particularly clear in more challenging settings. For the ChatGPT player in ALFWorld, PACE improves overall SR from 25.0% with the Qwen companion to 58.6%, while reducing intervention frequency (HELP) from 62.9% to 32.6%. In *Minecraft*, PACE reaches 93.3% SR with 16.6% HELP for Andy and 100% SR with 21.3% HELP for DeepSeek. PACE also achieves the highest milestone progress on the long-horizon Obtain Diamond task across all evaluated players. For stronger players, PACE maintains high task performance with relatively few interventions. For example, with GiGPO in *ALFWorld*, it reaches 90.0% SR with 28.1% HELP. Overall, these results show that PACE improves task performance while maintaining substantially lower intervention rates across diverse players and task conditions.

Table 1: **Main results on *ALFWorld*.** Results across six task categories for different player-companion pairs. Dashes denote Solo play without companion. Best results are bold.

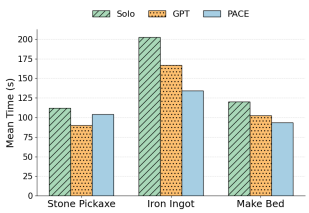
Player	Companion	Pick		Look		Clean		Heat		Cool		Pick2		Overall	
		SR	HELP	SR	HELP	SR	HELP	SR	HELP	SR	HELP	SR	HELP	SR	HELP
Qwen	–	61.9	–	30.8	–	17.3	–	17.0	–	14.9	–	15.3	–	28.9	–
	ChatGPT	51.4	99.7	46.2	96.7	17.3	98.1	**45.9**	96.7	6.7	99.4	12.5	99.9	26.4	98.3
	GiGPO	72.4	52.9	**51.3**	50.5	55.6	40.6	36.2	**47.9**	37.8	50.9	**48.6**	67.1	52.9	56.7
	Qwen	70.5	67.1	48.7	40.0	39.5	73.4	29.8	65.2	18.9	72.8	44.4	55.0	44.3	65.1
	PACE	**76.2**	**36.8**	48.7	**23.4**	**63.0**	**40.2**	36.1	51.2	**47.3**	**49.3**	**48.6**	**49.1**	**56.7**	**44.5**
ChatGPT	–	54.3	–	30.8	–	11.1	–	12.5	–	0.0	–	12.5	–	22.1	–
	ChatGPT	60.0	100.0	46.2	99.7	29.6	99.4	12.5	97.1	12.0	100.0	33.3	100.0	34.3	99.5
	GiGPO	71.4	54.3	53.8	47.1	29.6	43.6	25.0	42.4	24.0	36.5	58.3	45.0	45.7	46.9
	Qwen	60.0	56.8	30.8	35.1	18.5	56.6	6.2	87.1	4.0	70.6	12.5	55.8	25.0	62.9
	PACE	**74.3**	**36.9**	**61.5**	**32.4**	**51.9**	**21.5**	**43.8**	**35.8**	**44.0**	**27.2**	**66.7**	**32.7**	**58.6**	**32.6**
DeepSeek	–	85.7	–	76.9	–	51.9	–	18.8	–	24.0	–	16.7	–	47.9	–
	ChatGPT	85.7	100.0	76.9	94.5	33.3	99.6	37.5	99.3	32.0	99.6	33.3	100.0	50.7	99.4
	GiGPO	91.4	52.1	**84.6**	42.1	66.7	51.4	37.5	68.7	60.0	66.5	**62.5**	52.0	69.3	55.6
	Qwen	82.9	83.6	69.2	76.1	66.7	80.2	25.0	80.6	60.0	80.5	50.0	73.5	62.1	79.3
	PACE	**94.3**	**40.7**	**84.6**	**19.8**	**77.8**	**50.5**	**43.8**	**67.0**	**64.0**	**64.0**	54.2	**37.6**	**72.1**	**51.6**
GiGPO	–	**96.2**	–	89.5	–	**97.5**	–	87.5	–	73.3	–	83.3	–	88.5	–
	ChatGPT	90.5	99.8	64.1	96.9	70.4	97.9	83.4	93.9	56.0	95.5	62.5	99.8	72.1	96.7
	GiGPO	**96.2**	55.1	84.6	48.0	95.1	66.5	91.5	57.1	72.3	50.9	**89.0**	57.8	89.0	55.4
	Qwen	95.7	63.9	75.0	44.8	97.4	45.2	85.7	40.6	61.1	**32.5**	85.2	41.6	84.8	44.2
	PACE	**96.2**	**23.7**	**92.1**	**24.6**	96.2	**32.9**	**91.7**	**31.2**	**74.6**	36.0	87.5	**19.5**	**90.0**	**28.1**



(a) Intervention timing analysis.



(b) Subjective evaluation.



(c) Task completion time in human study.

Figure 2: **Intervention timing and human evaluation in *Minecraft*.** (a) PACE is compared with fixed-frequency and random schedules using the same advice, showing success rate versus HELP rate. (b) Average participant ratings of GPT and PACE on a 7-point Likert scale. (c) Average completion time for Solo, GPT-assisted, and PACE-assisted conditions across three Minecraft tasks.

6.2 Ablation Study

Training-Stage Analysis. Table 2 studies how PACE evolves across training stages. Intermediate rows show the cumulative training progression from the raw companion to gate and advice optimization, while "PACE w/o Init." evaluates the effect of removing supervised initialization. The raw Qwen3.5-VL-9B companion already improves over Solo play but intervenes frequently. We find initialization provides a stronger starting policy, gate optimization reduces intervention frequency, and advice optimization further improves success. Full PACE achieves the best result with 93.3% SR and 16.6% HELP. Compared with "PACE w/o Init.", it also improves both success and selectivity, showing the benefit of supervised initialization.

Companion	SR	HELP
Solo	56.7	–
Base	73.3	58.8
+ Initialization	76.7	53.1
+ Gate Update	83.3	32.5
+ Advice Update	90.0	30.2
PACE w/o Init.	86.7	27.2
PACE	**93.3**	**16.6**

Table 2: Training-stage analysis of PACE in Minecraft with Andy as the player.

Intervention Timing Analysis Figure 2 evaluates intervention timing with fixed PACE-generated advice. PACE achieves the highest success rate with about 17% HELP, outperforming fixed-frequency, random, and Solo strategies, including "Every-5-Steps" at a similar HELP rate. This supports learning when to intervene rather than simply increasing assistance.

Table 3: **Main results on *Minecraft*.** Results across three tasks for different player-companion pairs.

Player	Companion	Craft Wooden Pickaxe			Make Furnace			Obtain Diamond			Overall	
		SR	MS	HELP	SR	MS	HELP	SR	MS	HELP	SR	HELP
Qwen	–	90.0	0.94	–	10.0	0.49	–	0.0	0.29	–	33.3	–
	ChatGPT	80.0	0.86	95.5	30.0	0.60	98.1	0.0	0.08	90.8	36.7	94.4
	Qwen	60.0	0.88	99.7	30.0	0.69	100.0	0.0	0.29	99.6	30.0	99.4
	PACE	**100.0**	**1.00**	**36.4**	**80.0**	**0.90**	**40.8**	**40.0**	**0.74**	**44.2**	**73.3**	**36.9**
ChatGPT	–	90.0	0.98	–	70.0	0.86	–	0.0	0.65	–	53.3	–
	ChatGPT	100.0	1.00	96.8	10.0	0.41	94.5	0.0	0.40	94.3	36.7	95.3
	Qwen	90.0	0.96	100.0	80.0	0.87	99.6	0.0	0.53	95.7	56.7	99.7
	PACE	**100.0**	**1.00**	**32.8**	**90.0**	**0.93**	**34.7**	**30.0**	**0.73**	**37.7**	**73.3**	**33.5**
Andy	–	70.0	0.70	–	70.0	0.70	–	30.0	0.62	–	56.7	–
	ChatGPT	80.0	0.80	100.0	90.0	0.90	90.3	60.0	0.76	92.3	76.7	92.6
	Qwen	70.0	0.96	77.5	100.0	1.00	52.1	50.0	0.94	52.9	73.3	58.8
	PACE	**100.0**	**1.00**	**24.8**	**100.0**	**1.00**	**24.1**	**80.0**	**0.97**	**12.0**	**93.3**	**16.6**
DeepSeek	–	100.0	1.00	–	90.0	0.96	–	80.0	0.89	–	90.0	–
	ChatGPT	100.0	1.00	98.4	100.0	1.00	94.1	90.0	0.96	87.0	96.7	92.5
	Qwen	100.0	1.00	94.9	90.0	0.93	97.7	80.0	0.95	86.6	90.0	95.1
	PACE	**100.0**	**1.00**	**23.8**	**100.0**	**1.00**	**25.6**	**100.0**	**1.00**	**18.5**	**100.0**	**21.3**

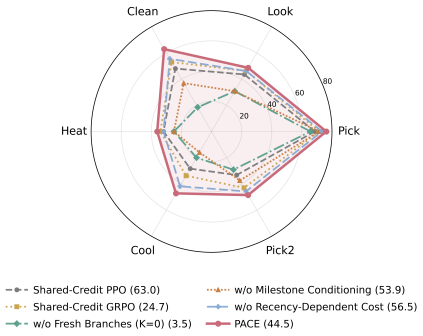


Figure 3: **Ablation study of PACE on ALFWorld with Qwen as the advisee**. The radar plot reports SR across six task categories, with overall HELP rates in parentheses. We compare PACE with shared-credit PPO/GRPO and variants without fresh branches, milestone conditioning, or the recency-dependent cost.

Credit and Component Analysis Figure 3 studies the main design choices of PACE with Qwen as the advisee. "Shared-Credit PPO" and "Shared-Credit GRPO" apply a common return-based signal to the gate and advice outputs instead of PACE's nested credit estimation. "w/o Fresh Branches ($K = 0$)" removes the additional advice branches, "w/o Milestone Conditioning" generates advice without milestone-focused context, and "w/o Extra Cost" removes the recency-dependent intervention penalty while retaining the base intervention cost. PACE achieves the strongest overall task performance while maintaining selective intervention. Shared-credit baselines perform worse across the six task categories, supporting the use of distinct credit signals for timing and advice. Removing fresh branches causes the largest degradation, while removing milestone conditioning also lowers success. Removing the extra intervention cost has a smaller effect on SR but increases HELP from 44.5% to 56.5%, indicating that it mainly improves intervention selectivity.

6.3 Human Study

We further evaluate PACE with 20 human participants in *Minecraft*. Participants rate PACE significantly higher than GPT-4o-mini on timing, helpfulness, clarity, and appropriateness ($p < 0.05$, paired Wilcoxon tests with Holm correction) in Figure 2. PACE also reduces completion time relative to Solo play across all three tasks and is fastest on two. Full study details and a qualitative case are provided in Appendices D and E.

7 Conclusion

We formulate proactive assistance as a nested decision problem over when to intervene and what advice to provide, and propose PACE, which compares matched branches under silence and alternative advice to assign distinct credit to each decision. Across ALFWorld, Minecraft, and a human study, PACE achieves higher task success with fewer interventions than LLM-based companions and generalizes to players unseen during training. PACE currently relies on replayable training states and a fixed advisee during training. Future work will extend it to more open-ended settings and to personalized assistance that adapts to individual users.

Ethics Statement

The human study was conducted with respect for participant privacy. Participation was voluntary, no personally identifying information was collected or disclosed, and the study data were used only for research purposes.

AI Use Disclosure

We used generative AI tools to assist with writing and debugging code, refining experimental implementations, polishing parts of the manuscript, and creating figures. We also used GPT-4o to generate teacher demonstrations for intervention decisions and advice, as well as captions for Minecraft visual adaptation, as described in the paper. We did not use generative AI tools to prove mathematical claims or to produce the experimental results reported in this paper. All AI-assisted work was reviewed, tested, and verified by the authors, who take full responsibility for the final content.

References

Piotr D Adamczyk and Brian P Bailey. If not now, when? the effects of interruption at different moments within task execution. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 271–278, 2004.

Erik M Altmann and J Gregory Trafton. Memory for goals: An activation-based model. *Cognitive science*, 26(1):39–83, 2002.

Riku Arakawa, Prasoon Patidar, Will Page, Jill Lehman, and Mayank Goel. Scaling context-aware task assistants that learn from demonstration and adapt through mixed-initiative dialogue. In *Proceedings of the 38th annual acm symposium on user interface software and technology*, pp. 1–19, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. Need help? designing proactive ai assistants for programming. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–18, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.

Shuo He, Lang Feng, Xin Cheng, Lei Feng, Bo An, et al. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. In *International Conference on Learning Representations*, volume 2026, pp. 27572–27593, 2026.

Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 159–166, 1999.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. SALT: Step-level advantage assignment for long-horizon agents via trajectory graph. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4709–4725, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-386-9. doi: 10.18653/v1/2026.findings-eacl.247. URL https://aclanthology.org/2026.findings-eacl.247/.

Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and Xiang'Anthony' Chen. Proactive conversational agents with inner thoughts. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–19, 2025.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, et al. Proactive agent: Shifting llm agents from reactive responses to active assistance. In *International Conference on Learning Representations*, volume 2025, pp. 47431–47457, 2025.

Markov AI. Gaming-500-hours. https://huggingface.co/datasets/markov-ai/gaming-500-hours, 2026. Hugging Face dataset.

Mindcraft-CE. Andy-4.2. https://huggingface.co/Mindcraft-CE/Andy-4.2, 2026. Hugging Face model.

Abhijnan Nath, Carine Graff, and Nikhil Krishnaswamy. Collaborate, deliberate, evaluate: How llm alignment affects coordinated multi-agent outcomes, 2026. URL https://arxiv.org/abs/2509.05882.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, July 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Qwen Team. Qwen3.5: Towards native multimodal agents, 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. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32, 1994.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Quan Wei, Siliang Zeng, Chenliang Li, William Brown, Oana Frunza, Wei Deng, Anderson Schneider, Yuriy Nevmyvaka, Yang Katie Zhao, Alfredo Garcia, et al. Reinforcing multi-turn reasoning in llm agents via turn-level reward design. *arXiv preprint arXiv:2505.11821*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Yang Wu, Jinhong Yu, Jingwei Xiong, Zhimin Tao, and Xiaozhong Liu. "excuse me, may I say something..." CoLabScience, a proactive AI assistant for biomedical discovery and LLM-expert collaborations. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36109–36129, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1671. URL https://aclanthology.org/2026.acl-long.1671/.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13643–13658, 2024.

Table 4: Notation for interaction, policies, and branch construction.

Symbol	Meaning
Interaction and policies	
s_t	Interaction state/context at timestep t
g_t	Intervention gate, $g_t \in \{\text{HELP}, \text{SILENCE}\}$
a_t	Advice generated by the companion; $a_t = \varnothing$ under SILENCE
u_t	Environment action selected by the advisee
π_θ^g	Companion gate policy
π_θ^a	Companion advice policy
π^{adv}	Frozen advisee policy
τ	Interaction trajectory
$R_{\text{task}}(\tau)$	Task-performance return of trajectory τ
$C_{\text{int}}(\tau)$	Cumulative intervention cost of trajectory τ
Task structure and branch construction	
$\mathcal{M}$	Ordered task milestone sequence $(M^{(1)}, \ldots, M^{(L)})$
$M^{(m)}$	m-th task milestone
L	Total number of milestones in a task
$\mathcal{M}_t^{\text{rem}}$	Remaining milestone sequence at timestep t
L_t	Number of remaining milestones at timestep t
$z_{t,i}$	Milestone-prefix context used to generate fresh advice branch i
d_i	Number of remaining milestones included in $z_{t,i}$
w_i	Advice associated with branch i; $w_1 = a_t$ for the replay branch
K	Number of fresh advice branches
χ_t	Replayable environment state recorded at intervention timestep t

A Notation and Joint-Stage Training Algorithm

Tables 4 and 5 summarize the main notation used throughout the paper.

Algorithm 1 summarizes the joint-stage optimization procedure used after initialization.

B Train

B.1 Initialization Details

Parameter-efficient setup. All initialization stages update only LoRA adapters and, when training the critic, the value head; the pretrained backbone and vision encoder remain frozen. LoRA uses rank r=16, α=32, and dropout 0.05, and is applied to the attention projections used by the trainable language layers. Feed-forward blocks, token embeddings, and the language-model head are not adapted. The critic uses a small MLP value head over the final-layer hidden representation. We retain the same LoRA configuration during subsequent reinforcement learning.

***Minecraft* visual adaptation.** We construct the visual initialization data from a public gameplay-video corpus (Markov AI, 2026). After filtering for *Minecraft* clips, we extract frames with `ffmpeg` at 0.5 fps, discard the first 20 s of each clip, and resize frames to 640 px width. GPT-4o captions each frame using a fixed prompt describing the player's current action, visible scene, and HUD/hotbar contents. We fine-tune the companion to reproduce these captions from the corresponding frames, adapting the language-side LoRA parameters to *Minecraft*-specific visual features. This stage is omitted for the text-only *ALFWorld* environment.

Decision-policy supervised fine-tuning. We initialize the companion policy using per-step teacher demonstrations of $(\text{state}, \text{gate}, \text{advice})$. A portion of these demonstrations is collected from

Table 5: Notation for reward, branch evaluation, and nested credit assignment.

Symbol	Meaning
Reward and branch evaluation	
r_t	Learning reward at timestep t
r_t^{succ}	Task-success reward
r_m	Progress reward associated with milestone $M^{(m)}$
c_t	Intervention cost at timestep t
c_{base}	Base intervention cost
c_{extra}	Additional recency-dependent intervention cost
$\tilde{d}_t$	Effective number of steps since the previous HELP
τ_c	Decay temperature for the recency-dependent intervention cost
γ	Discount factor
N	Branch rollout horizon
T_i	Remaining episode length of branch i until termination
H_i	Effective rollout horizon, $H_i = \min(N, T_i)$
$Q_i^{(N)}(s_t)$	N-step return of branch i from state s_t
$\hat{Q}_t$	N-step critic target along the realized main trajectory
V_ϕ	Learned value function / critic
V_{ϕ^-}	Critic frozen at rollout time for bootstrapping
Nested credit assignment	
$\bar{Q}$	Mean return over the $K+2$ matched branches
σ	Floored state-local standard deviation of branch returns
$\sigma_{\min}$	Minimum standard-deviation floor
Δ_t	Timing credit from realized HELP versus SILENCE
κ	Clipping bound for timing credit
$\hat{A}_{t,i}$	Relative content credit assigned to advice branch i
$A^{(g)}$	Advantage used for the gate update
$A^{(c)}$	Advantage used for the advice content update
$\mathcal{H}$	Set of timesteps at which the companion selects HELP

trajectories generated by a separately trained lightweight intervention scheduler. The scheduler determines when assistance should be triggered, while the corresponding guidance is produced by an LLM instructor through prompting based on the current task and interaction context. This teacher pipeline is used only for data collection and is not part of PACE.

To increase state and task coverage, we sample a small number of checkpoints at different stages of each task trajectory. At each sampled checkpoint, GPT-4o is provided with the current interaction context and task reference plan, and is asked to generate the gate decision and, when HELP is selected, staged advice. The resulting dataset therefore covers different stages of task progress while keeping the number of demonstrations per task controlled. The dataset contains both HELP and SILENCE decisions. We do not filter demonstrations according to downstream task success or return; only malformed or invalid generations are discarded. During GPT-4o generation, the prompt encourages a mixed HELP/SILENCE distribution to avoid a strongly biased initialization policy.

Each target is represented as the complete JSON decision $\{\texttt{gate}, \texttt{advice}\}$, with empty advice for SILENCE, and we apply completion-only cross-entropy loss over the assistant tokens. This supervision is used only to provide a reasonable initialization; intervention timing and advice generation are subsequently refined through reinforcement learning.

Critic Initialization. Using rollouts from the SFT companion, we compute the realized discounted return-to-go $G(s_t) = \sum_{j \geq 0} \gamma^j r_{t+j}$ with γ=0.95 by backward accumulation from the end of each rollout, without value bootstrapping. For initialization, we use a simplified shaped reward, task success minus a fixed per-intervention cost, without the milestone and recency-weighted terms used

Algorithm 1 Joint-Stage Update

Require: companion policy π_θ; frozen advisee π^{adv}; critic V_ϕ; branch horizon N; discount γ; number of fresh advice branches K; PPO clips ϵ_g, ϵ_c; credit clip κ; std floor $\sigma_{\min}$; gate KL budget δ; epochs E; value weight β_v.

$\mathcal{S}_{\mathrm{gate}}, \mathcal{S}_{\mathrm{cont}} \leftarrow \varnothing$
for each episode **do**
  Roll out the main trajectory under $(\pi_\theta, \pi^{\mathrm{adv}})$ with Eq. 3; at each $g_t = \textsc{help}$, snapshot χ_t and compute the main-trajectory N-step target $\hat{Q}_t$ using return rule in Eq. 5.
  for all $t : g_t = \textsc{help}$ **do**
    Set $w_1 \leftarrow a_t$; for $i = 2, \ldots, K+1$, set $d_i = \min(i-1, L_t)$, $z_{t,i} = (M_t^{(1)}, \ldots, M_t^{(d_i)})$, and sample $w_i \sim \pi_\theta^a(\cdot \mid s_t, z_{t,i})$.
    From χ_t under shared CRN, roll $K+2$ branches for at most N steps: Q_0 for SILENCE, and Q_i for $\textsc{help}(w_i)$, $i = 1, \ldots, K+1$.
    Compute each Q_i using Eq. 5, with zero bootstrap at terminal states.
    $\bar{Q} \leftarrow \mathrm{mean}(\{Q_i\}_{i=0}^{K+1})$, $\sigma \leftarrow \max\big(\mathrm{std}(\{Q_i\}_{i=0}^{K+1}), \sigma_{\min}\big)$.
    Timing credit: $\Delta_t \leftarrow \mathrm{clip}((Q_1 - Q_0)/\sigma, -\kappa, \kappa)$; store $(s_t, g_t, \Delta_t) \rightarrow \mathcal{S}_{\mathrm{gate}}$.
    Content credit: $\hat{A}_{t,i} \leftarrow (Q_i - \bar{Q})/\sigma$; store $(s_t, w_i, \hat{A}_{t,i}) \rightarrow \mathcal{S}_{\mathrm{cont}}$, $i = 1, \ldots, K+1$.
    Attach $\hat{Q}_t$ only to replay entry $i = 1$ as its critic target.
  end for
end for
for E epochs **do** ▷ content update
  PPO-clip update on advice tokens using $\hat{A}_{t,i}$; regress $V_\phi(s_t)$ to $\hat{Q}_t$ on replay entries $i = 1$.
end for
for E epochs **do** ▷ gate update; critic frozen
  PPO-clip update on gate tokens using Δ_t; early stop if $\widehat{\mathrm{KL}} > \delta$.
end for

during reinforcement learning. We regress V_ϕ toward G with mean-squared error while updating the LoRA adapters and value head.

B.2 Details of Training

***ALFWorld* environment and training.** At each step, the advisee observes a textual description of the environment together with the task goal and a set of admissible commands, and selects one command as its environment action. The companion additionally conditions on a sliding history of the most recent 10 interaction steps. Episodes are capped at 40 environment steps. Task success provides $r_t^{\mathrm{succ}} = 1$, while all intermediate milestones use a uniform progress reward of $r_m = 0.3$. Intervention cost is incorporated into the branch-evaluation reward at each HELP step, with c_{base}=0.02, c_{extra}=0.08, and τ_c=5. We use three random seeds: 2024, 2026, and 1234. When task progress occurs after the previous intervention, the recency-dependent extra cost is cleared as described in Eq. 4, leaving only the base intervention cost. Training runs for three stages of 30 updates each (gate, content, then joint; 90 updates total). Each update collects 8 episodes; we expand at most 6 HELP states per trajectory with an N-step branch horizon of 8.

***Minecraft* environment and training.** The companion receives first-person visual observations together with the advisee's inventory state, while the frozen advisee interacts with the environment through high-level skill calls. Episodes use per-task step budgets ranging from 20 to 40 steps according to task depth, with each interaction step paced to at least 5 seconds. Intermediate progress is defined by item-based milestones along the task's technology tree, where a milestone is triggered the first time its target item is acquired. Unlike the uniform *ALFWorld* setting, *Minecraft* uses item-specific milestone rewards r_m, with intermediate resources receiving smaller rewards and the terminal target item receiving the largest reward and ending the episode on success. Intervention cost follows the same recency-dependent formulation as in *ALFWorld*, with milestone progress clearing the additional recency penalty after the previous intervention. We train for three stages of 20 updates each. Each update collects 8 episodes; we expand at most 4 HELP states per trajectory with an N-step branch horizon of 6.

Interaction and replay in *Minecraft*. *Minecraft* runs as a live server controlled through a Mineflayer-based bot. At each interaction step, the environment provides the current first-person frame and inventory state to the companion. The companion first decides whether to intervene and, when HELP is selected, generates an advice message that is appended to the frozen advisee's context. The advisee then produces a high-level skill call, which is executed by the bot before the next observation is collected. Because the live *Minecraft* environment does not provide exact native snapshotting, we emulate replayable states through command-line environment resets that restore the relevant recorded world and agent state. Alternative branches are then rolled out from the restored interaction state under matched conditions.

Optimization. Across both environments, we train only LoRA adapters (r=16, α=32, dropout 0.05) on the $q/k/v/o$ attention projections together with a lightweight value head, while keeping the pretrained backbone and vision encoder frozen. Reinforcement learning proceeds in three stages of equal length: gate-only, content-only, and joint optimization. We optimize the gate first because the initialized policy initially intervenes too frequently, and a more selective intervention policy provides a cleaner distribution for subsequent content optimization. During the joint stage, each update performs a content PPO pass followed by a gate PPO pass. We use K=2 milestone-focused alternative advice branches in addition to the silence and replay branches, with common random numbers (CRN) shared across matched rollouts where applicable. PPO uses a clipping coefficient of 0.2, two optimization epochs, a mini-batch size of 4, a value-loss coefficient of 0.5, and a gate entropy coefficient of 0.01. The learning rates are 1×10^{-5} for the LoRA adapters and 5×10^{-5} for the value head, with discount factor $\gamma = 0.95$. Branch expansion requires a realized HELP decision. To avoid insufficient branch coverage when the gate becomes overly conservative, training rollouts use an ϵ-mixed gate policy. With probability $p_{\text{mix}} = 0.1$, we execute the alternative gate decision. This allows states that would otherwise remain SILENT to occasionally be evaluated through matched HELP–SILENCE branches. Exploration is disabled during evaluation.

Evaluation. We report task success rate (SR) and intervention frequency (HELP), where HELP is defined as the fraction of companion decision steps at which HELP is selected. For *ALFWorld*, each evaluation run consists of 140 episodes from the validation split. For *Minecraft*, we additionally report milestone progress (MS), defined as the fraction of predefined task milestones completed. During evaluation, the learned companion weights are held fixed across all advisees, and only the advisee backend is changed. Solo play is evaluated without a companion and serves as the no-assistance baseline. For *ALFWorld*, results are averaged over three independent runs, and uncertainty across runs is assessed using Student's t-distribution. For *Minecraft*, each task is evaluated over 10 episodes, and variability is assessed from the episode-level outcomes.

C Ablation

Table 6 reports the stage-wise ablation on *Minecraft* with Andy 4.2 as the player. Visual pretraining provides a modest improvement over the base companion, while subsequent gate and content optimization progressively increase success and reduce intervention frequency. The full model achieves the best overall performance, reaching 93.3% SR with only 16.6% HELP, compared with 73.3% SR and 58.8% HELP for the base model.

D Human Study

Study design. We conduct a human study in *Minecraft* with 20 participants using a within-subject design. Among the 20 participants, 5 self-identified as experienced Minecraft players, 13 had some prior experience with the game, and 2 had never played Minecraft before. Each participant completes three trials under three assistance conditions: SOLO, GPT-4o-mini, and PACE. To reduce ordering and task-specific effects, we use a 3×3 counterbalancing scheme in which each participant encounters each assistance condition exactly once and completes each task exactly once. Partici-

Table 6: **Ablation results on *Minecraft* with Andy 4.2 as the player.** We report task success rate (SR, %), milestone progress (MS), and intervention frequency (HELP, %) for each task. Overall SR and HELP are aggregated across all three tasks.

Companion	Craft Wooden Pickaxe			Make Furnace			Obtain Diamond			Overall	
	SR	MS	HELP	SR	MS	HELP	SR	MS	HELP	SR	HELP
Solo	70.0	0.70	–	70.0	0.70	–	30.0	0.62	–	56.7	–
Base	70.0	0.96	77.5	100.0	1.00	52.1	50.0	0.94	52.9	73.3	58.8
Initialization	80.0	0.98	68.6	100.0	1.00	47.5	50.0	0.92	47.4	76.7	53.1
After timing stage	90.0	0.98	37.5	100.0	1.00	37.4	60.0	0.88	23.7	83.3	32.5
After content stage	100.0	1.00	29.4	100.0	1.00	34.0	70.0	0.96	20.7	90.0	30.2
w/o Initialization	90.0	0.96	35.2	100.0	1.00	28.7	70.0	0.96	26.3	86.7	27.2
PACE	**100.0**	**1.00**	**24.8**	**100.0**	**1.00**	**24.1**	**80.0**	**0.97**	**12.0**	**93.3**	**16.6**

pants are assigned to one of three groups:

Group	Trial 1	Trial 2	Trial 3
$G1$	$A +$ SOLO	$B +$ GPT	$C +$ PACE
$G2$	$A +$ GPT	$B +$ PACE	$C +$ SOLO
$G3$	$A +$ PACE	$B +$ SOLO	$C +$ GPT

where A, B, and C denote the three *Minecraft* tasks described below. Participants were distributed as evenly as possible across the three groups. This design exposes every participant to all three assistance conditions while distributing task–condition combinations across participants. For subjective ratings, we compare PACE and GPT-4o-mini using paired two-sided Wilcoxon signed-rank tests and apply Holm correction across the four questionnaire dimensions. PACE receives significantly higher ratings 6.2 than GPT-4o-mini across all four subjective dimensions (paired Wilcoxon signed-rank tests with Holm correction, all $p < 0.05$), with clear gains in intervention timing and appropriateness.

Tasks. We evaluate three *Minecraft* tasks with different levels of interaction complexity: *Craft a Stone Pickaxe*, *Obtain an Iron Ingot*, and *Make Bed*. The first task requires a relatively short crafting sequence, whereas the latter tasks involve longer prerequisite chains and additional resource collection.

Assistance conditions. Under SOLO, participants complete the task without companion assistance. Under the GPT condition, GPT-4o-mini observes the ongoing interaction and independently decides whether to intervene and what guidance to provide. Under PACE, the trained companion similarly observes the interaction and produces either HELP with an advice message or SILENCE. The participant remains in control of all environment actions in both assisted conditions; companion outputs are advisory only. In both assisted conditions, participants could freely interact with the companion through chat, either waiting for proactive guidance or initiating interaction themselves by asking questions or sending messages. Participant-initiated queries accounted for 11.2% of companion responses under PACE and 3.7% under GPT-4o-mini. This suggests a more bidirectional interaction pattern under PACE, where assistance was not solely delivered as unsolicited proactive guidance. We retain these interactions in the human-study results to reflect naturalistic companion use.

Measures. We record task completion time for all trials. For assisted trials, participants additionally rate the companion on a 7-point Likert scale along four dimensions: *Helpfulness*, *Timing*, *Intervention Appropriateness*, and *Clarity*. Higher scores indicate more favorable evaluations on all four dimensions. We use *Intervention Appropriateness* to capture whether assistance is perceived as occurring when it is needed rather than being unnecessary or disruptive.

Procedure. Each participant completes the three assigned trials sequentially according to the counterbalancing schedule. During an assisted trial, the companion can provide guidance while the participant continues controlling the game. After each assisted trial, the participant completes



Figure 4: **Case study in *Minecraft*.** PACE intervenes when the player is blocked, remains silent during productive progress, and intervenes again when a new bottleneck arises.

the subjective questionnaire. Completion time is recorded automatically from the beginning of the trial until successful task completion.

E Case Study

Figure 4 shows a representative interaction in *Minecraft*. The player first attempts to mine iron using a wooden pickaxe and fails to make progress. PACE intervenes by identifying the missing prerequisite and advising the player to obtain stone and upgrade to a stone pickaxe. After the player successfully collects stone, PACE provides brief encouragement, but remains silent once the player resumes productive behavior and proceeds to mine iron independently. A new intervention is triggered only when the player later becomes uncertain about how to convert raw iron into an iron ingot. At this point, PACE provides the next actionable step, instructing the player to use a furnace for smelting. This example illustrates how PACE adapts its behavior to the evolving interaction state, providing targeted guidance at bottlenecks while avoiding unnecessary intervention during effective progress.