ANTICIPATING THE CONSEQUENCES OF CURRICULUM DECISIONS WITH LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic curriculum learning can improve the effectiveness of reinforcement learning by selecting the training experiences presented to the agent over time. Predicting the consequences of such decisions can, however, be difficult. We analyze automatic curriculum learning as a sequential decision-making problem, highlighting a gap between the quantities that determine the value of curriculum decisions and the information captured by local learning signals commonly used to guide them. We then investigate whether Large Language Models (LLMs) can exploit richer information about the learning problem to better anticipate the consequences of curriculum decisions. We introduce a method that combines online learning-progress estimates with LLM-informed estimates of (i) the potential downstream benefits of learning on each task and (ii) whether direct training on a task is currently likely to produce progress. We evaluate the approach on a custom benchmark of 256 textual goals in Craftax under different curriculum objectives. We observe the strongest gains when optimizing for individual target tasks. When optimizing across the full task set, the benefits vary across learners with different mechanisms for cross-task transfer, ranging from modest improvements in learning speed to larger gains that persist through the end of training.

1 INTRODUCTION

Reinforcement learning tasks can be difficult to learn through direct training, particularly when rewards are sparse or delayed. Curriculum learning addresses this by shaping the sequence of situations on which an agent trains, a principle that has played an important role in several influential deep reinforcement learning systems (Akkaya et al., 2019; Vinyals et al., 2019; Hubert et al., 2026). Choosing these training experiences, however, is itself a complex sequential decision-making problem (Narvekar et al., 2017). Evaluating the value of training on a task requires anticipating its immediate impact on the agent's performance across tasks, as well as its impact on the effectiveness of future training. These consequences are difficult to predict, and commonly used curriculum signals such as estimates of task competence or learning progress are limited for this purpose (Matiisen et al., 2019).

In this work, we analyze automatic curriculum learning explicitly from this sequential decision-making perspective. Using learning progress as a concrete lens, we relate local task-selection criteria to optimal curriculum decisions and make explicit how unaccounted downstream effects of training and limitations in estimating local progress can lead to suboptimal decisions. We then investigate whether Large Language Models (LLMs) can help account for these factors by exploiting richer information about the learning problem. We introduce a method in which an LLM, given descriptions of the tasks, learner, and training setup, provides judgments that help estimate quantities affecting the value of curriculum decisions. Concretely, we augment learning-progress-based prioritization with estimates of the potential downstream benefits of training on a task and of whether direct training on a low-performance task is likely to produce progress. We evaluate the approach on a custom benchmark of 256 textual goals in Craftax and find that it can materially improve learning outcomes. When optimizing for individual target tasks, the method makes substantial progress on targets where learning-progress-based prioritization makes little or none. When optimizing across the full task set, gains are modest with a learner restricted to primitive actions but larger for a learner that can invoke learned behaviors as subroutines.

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.

2 RELATED WORK

One approach to automatic curriculum learning is to directly optimize a curriculum-level objective defined over the overall learning process (Foglino et al., 2019b). For instance, Narvekar & Stone (2018) formulate curriculum selection as a higher-level MDP and use reinforcement learning to learn a curriculum policy from repeated executions of complete curriculum trajectories. Such approaches can in principle account for the full consequences of curriculum decisions, but evaluating candidate curricula entails repeatedly training the underlying learner, making direct optimization highly sample and computationally expensive (Narvekar & Stone, 2018; Foglino et al., 2019a;b).

In practice, most research on automatic curriculum learning has focused on methods that construct the curriculum within a single learner training run, using information available at the start of training or captured online to construct proxies for the utility of training on different tasks (Portelas et al., 2020b). One family of approaches uses measures of diversity or novelty to guide task selection, favoring tasks that are underrepresented in the learner's experience (Pong et al., 2019; Pitis et al., 2020). Related approaches use measures of surprise, such as prediction error or model disagreement, to prioritize situations that remain unfamiliar to the learner (Pathak et al., 2017; 2019). Another widely used principle is to favor tasks of intermediate difficulty relative to the learner's current capabilities, as tasks that are reliably solved or consistently failed are expected to provide less useful learning signal (Sukhbaatar et al., 2017; Florensa et al., 2018).

Other methods seek to estimate the learning progress (LP) expected from training on a task (Baranes & Oudeyer, 2013). Since future improvement cannot be observed in advance, most LP-based approaches estimate it from recent changes in competence and prioritize tasks or regions for which the estimated progress is greatest (Kanitscheider et al., 2021; Forestier et al., 2022). Several variants use absolute learning progress (ALP), prioritizing tasks on which performance is deteriorating as well as improving (Matiisen et al., 2019; Colas et al., 2019). These methods generally use recent changes in competence observed for a task as a proxy for the utility of further training on that task.

Recent work has explored combining such signals with LLMs to incorporate prior knowledge into curriculum construction. Zhang et al. (2024) combine an LP-based curriculum with foundation-model judgments of task interestingness, while other approaches condition LLM task generation on task-specific estimates of the learner's current competence or learning progress (Pourcel et al., 2024; Team, 2025). These methods use LLM knowledge to identify tasks that are individually valuable and well suited to the learner's current state, but they do not model how training on one task may affect learning on others. Another line of work uses LLMs to decompose a complex target task into a sequence of subtasks that structure training toward it (Ryu et al., 2025; Heravi et al., 2026). Such decompositions capture useful prerequisite structure, but are centered on a particular target and represent a more restricted class of relationship than general inter-task transfer.

A final line of work explicitly models inter-task transfer to guide curriculum construction. Some approaches approximate transfer using hand-designed proxies derived from available structure in the problem, including structural task descriptors and similarity or distance measures defined over a task representation (Svetlik et al., 2017; Silva & Costa, 2018; Tzannetos et al., 2024). Others estimate transfer from additional learner experience, including direct source-target transfer trials (Sinapov et al., 2015; Jain & Tulabandhula, 2017), cross-task evaluations during training (Huang et al., 2022), or meta-learned models that predict the short-term learning progress induced on a target by training on different source tasks (Xu et al., 2023). Our approach instead leverages knowledge encoded in LLMs and their inference capabilities to obtain a prior over inter-task transfer, avoiding both the need to design problem-specific transfer proxies and the additional interaction required to estimate these relationships empirically.

3 PROBLEM FORMULATION AND ANALYSIS

3.1 THE CURRICULUM PROBLEM

Automatic curriculum learning methods are concerned with the automatic selection of training data or situations so as to optimize learning. These selections can occur at different levels of granularity. Here, we focus on curricula that influence the tasks on which the agent collects new experience (Portelas et al., 2020b). We represent each task m as a partially observable Markov decision process

(POMDP), with tasks potentially differing in any of its components. Automatic curriculum learning can itself be viewed as a sequential decision-making problem (Narvekar et al., 2017; Matiisen et al., 2019). The consequences of a curriculum decision depend on all information relevant to how the learner will respond to subsequent training. This information includes the learning problem specification (e.g., the tasks and objective of interest), the learning setup (e.g., the learning algorithm and its hyperparameters), and the current state of the learning process (e.g., learner and optimizer state). In principle, an ideal curriculum decision maker could condition its decisions on all such information. In practice, however, most of this information may be unavailable or difficult to exploit directly. Curriculum methods therefore generally operate on more tractable subsets or summaries, such as estimates of task performance or other statistics derived from the training history.

For the analysis that follows, we formalize the curriculum process as a higher-level MDP. To facilitate the discussion and its interpretation, we adopt a simplified representation in which the state at curriculum decision i corresponds to the learning agent's policy π_i, with initial state $\pi_0 \sim \rho_c$. The transition model $p_c(\pi_{i+1}|\pi_i, a_i)$ gives the probability that training policy π_i on the task distribution induced by curriculum action a_i results in policy π_{i+1}. We use π_c to denote the curriculum policy that acts in this higher-level MDP, leaving any dependence on the decision index implicit. We consider a class of curriculum objectives that evaluate the learner after N curriculum decisions:

$$J(\pi_c) = \mathbb{E}_{\rho_c, p_c, \pi_c}\left[\sum_m w(m) P(m; \pi_N)\right]. \tag{1}$$

Here, $P(m;\pi)$ is a performance measure of π on task m and the weight $w(m) \geq 0$ represents the importance assigned to task m. The expectation is taken over trajectories of the curriculum MDP induced by ρ_c, p_c, and π_c. Since ρ_c is independent of the curriculum policy, maximizing $J(\pi_c)$ is equivalent to maximizing the improvement in performance over training. We define $\mathrm{LP}(m, \pi_i, n) := P(m; \pi_{i+n}) - P(m; \pi_i)$ as the learning progress on task m over n curriculum decisions. We denote the equivalent objective by

$$J_{\mathrm{LP}}(\pi_c) := \mathbb{E}_{\rho_c, p_c, \pi_c}\left[\sum_m w(m)\,\mathrm{LP}(m, \pi_0, N)\right] = \sum_{i=0}^{N-1}\sum_m w(m)\mathbb{E}_{\rho_c, p_c, \pi_c}\left[\mathrm{LP}(m, \pi_i, 1)\right]$$

The final expression makes explicit how each task's expected one-step learning progress contributes to the curriculum objective. We next analyze how task-selection criteria relate to this objective. We do so through the lens of LP, but the underlying considerations are more general.

3.2 From local learning progress to optimal curriculum decisions

Local LP-guided curriculum methods generally estimate the recent rate of change in performance on each task and use this information to prioritize subsequent training. Let $\widehat{\mathrm{LP}}_i(m)$ denote an estimate of the recent learning progress on task m available before curriculum decision i. Methods differ in how these estimates are translated into task priority. We denote this transformation by h, such that task m is prioritized according to $h(\widehat{\mathrm{LP}}_i(m))$, e.g., $h(x) = \max(x, 0)$ for positive-LP. For the remainder of the analysis, we consider curriculum decisions that allocate the next unit of training to a single task. Given policy π_i and a task m selected for training, we write $\mathrm{LP}(m', \pi_i, 1; m) := P(m'; \pi_{i+1}) - P(m'; \pi_i)$, with $\pi_{i+1} \sim p_c(\cdot \mid \pi_i, m)$, for the one-step learning progress induced on task m'. We propose that local LP-based methods can be interpreted as using $h(\widehat{\mathrm{LP}}_i(m))$ as a proxy for the expected one-step progress on m resulting from allocating the next unit of training to m:

$$h\left(\widehat{\mathrm{LP}}_i(m)\right) \approx \mathbb{E}_{p_c(\pi_{i+1}|\pi_i, m)}\left[\mathrm{LP}\left(m, \pi_i, 1; m\right)\right]. \tag{2}$$

This interpretation allows us to analyze local LP-based selection in relation to the curriculum objective. If the proxy in Eq. (2) is sufficiently accurate to preserve the ordering across tasks, then the corresponding greedy rule selects the task whose performance is expected to improve the most if selected for training: $\arg\max_m h\left(\widehat{\mathrm{LP}}_i(m)\right) = \arg\max_m \mathbb{E}_{p_c(\pi_{i+1}|\pi_i, m)}\left[\mathrm{LP}\left(m, \pi_i, 1; m\right)\right]$. This criterion is not generally optimal. To understand its limitations, we consider three sufficient conditions under which it is: (1) *Equal task importance:* all tasks have the same objective weight $w(m)$; (2) *No immediate cross-task effects*: training on m does not immediately affect performance on any other task in expectation; and (3) *No effect on future learning*: training on m does not change the expected learning progress obtainable from subsequent training decisions.

Varying task importance. Suppose Conditions 2 and 3 hold, but tasks may have different objective weights. When these weights are known, this limitation can be addressed simply by weighting the expected local improvement: $m_i^* = \arg\max_m w(m)\mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}\left[\mathrm{LP}\left(m, \pi_i, 1; m\right)\right]$.

Immediate cross-task effects. A central source of curriculum learning's value is its ability to exploit cases where training on one task helps improve performance on others. Condition 2 excludes one important way in which such cross-task effects can arise, requiring $\mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}\left[\mathrm{LP}\left(m', \pi_i, 1; m\right)\right] = 0$ for all $m' \neq m$. This is a particularly strong assumption in many deep learning settings, where a shared parameterized agent is trained across tasks and an update induced by experience on one task can consequently alter its behavior on the others. If Condition 2 is removed while Conditions 1 and 3 are retained, the optimal decision must account for the expected immediate effects on performance across all tasks:

$$m_i^* = \arg\max_m \Big[\mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}[\mathrm{LP}(m, \pi_i, 1; m)] + \sum_{m' \neq m} \mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}[\mathrm{LP}(m', \pi_i, 1; m)]\Big].$$

Effects on future learning. The value of a curriculum decision can generally extend beyond the progress it produces immediately. Training on one task alters the learner's state, which can impact the effectiveness of future training. Condition 3 ignores such effects and the sequential nature of the curriculum problem. Under Conditions 1 and 2, the local criterion can be understood as greedily maximizing J_{LP} one iteration at a time. The optimal decision instead maximizes the expected progress accumulated over the full remaining horizon:

$$m_i^* = \arg\max_m \Big[\mathbb{E}_{p_c}\big[\mathrm{LP}(m, \pi_i, 1; m)\big] + \mathbb{E}_{p_c, \pi_c^*}\Big[\sum_{j=1}^{N-i-1} \mathrm{LP}(m_{i+j}, \pi_{i+j}, 1; m_{i+j}) \;\Big|\; \pi_i, m_i = m\Big]\Big].$$

Dropping all three conditions jointly yields the following general expression for the optimal curriculum action at iteration i:

$$\begin{aligned} m_i^* = \arg\max_m \Bigg[& w(m)\, \mathbb{E}_{p_c}\big[\mathrm{LP}(m, \pi_i, 1; m)\big] + \mathbb{E}_{p_c}\Big[\sum_{m' \neq m} w(m')\ \mathrm{LP}(m', \pi_i, 1; m)\Big] \\ & + \mathbb{E}_{p_c, \pi_c^*}\Big[\sum_{j=1}^{N-i-1} \sum_{m'} w(m')\ \mathrm{LP}(m', \pi_{i+j}, 1; m_{i+j}) \;\Big|\; \pi_i, m_i = m\Big]\Bigg]. \end{aligned} \tag{3}$$

Historical LP as a proxy for current progress. The preceding analysis assumed that the approximation in Eq. (2) is sufficiently accurate. Estimates of past LP, however, inherit statistical noise from performance measurements and from the finite and potentially stale observations available for estimating recent changes in performance. Moreover, even an accurate estimate of recent past LP summarizes progress observed at previous learner states and therefore need not reflect the progress that training on m would produce at the current policy π_i. In particular, for tasks on which no progress has yet been observed, past LP provides no basis for prioritization, even when their current expected learning progress differs.

In practice, LP-based curriculum methods use several mechanisms to address these limitations. The transformation h applied to past LP can adjust how observed progress is translated into task priority. Absolute-LP criteria, for example, account for the discrepancy between past and future progress by treating recent deterioration as evidence that renewed training may be useful (Matiisen et al., 2019; Colas et al., 2019). Methods also improve the LP estimate itself, for example, by refreshing performance estimates across the task set on every iteration at additional evaluation cost, or by aggregating performance observations over recent windows using moving averages or regression slopes (Matiisen et al., 2019). Finally, rather than selecting tasks greedily, methods commonly incorporate exploration through bandit-style selection, probabilistic sampling, or mixtures of LP-based and uniform sampling (Matiisen et al., 2019; Portelas et al., 2020a).

4 METHOD

Section 3 highlights that the quality of a curriculum decision depends on consequences that are generally difficult to estimate. We seek to account for their influence without relying on additional

learner experience for their estimation (see Section 2). Commonly used curriculum signals, such as past performance estimates, are limited and may be insufficient for this purpose. We propose incorporating LLMs into curriculum decision making both as a flexible mechanism for reasoning about hard-to-estimate consequences affecting the value of curriculum actions and as a means of exploiting richer sources of information that may help infer their influence.

The proposal is intentionally general. LLMs can draw both on knowledge encoded in the model and on any supplied information about the curriculum problem or training history that admits a textual representation. They can support curriculum decision making in different ways and be used in both selection- and generation-based curricula. In this work, we develop a specific instantiation for task selection over a fixed and manageable task set $\mathcal{T}$. We further restrict the modeled downstream consequences of training to beneficial effects, omitting negative cross-task effects, and query the LLM only before training. Adaptation as learning proceeds relies on online task-performance estimates $\widehat{P}_i(m) \in [0,1]$ and learning-progress estimates $\widehat{\mathrm{LP}}_i(m)$ (Appendix A.2). These choices provide a controlled setting in which to study the proposed use of LLMs to enhance curriculum decisions, while limiting both the range of modeled effects and the computational cost of online decision making. In our experiments, we supply the LLM with textual descriptions of the tasks, the learner and training setup, and the intended use of its judgments in curriculum selection.

4.1 Estimating downstream effects

Equation (3) shows that, beyond the direct contribution $w(m)\,\mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}[\mathrm{LP}(m,\pi_i,1;m)]$, the value of training on m depends on immediate effects on other tasks and effects on future learning. Rather than attempting to estimate all these downstream terms explicitly, we construct a tractable approximation to their aggregate positive contribution. For downstream effects to arise, training on m must impact the learner state. This motivates us to approximate their aggregate contribution through two factors: the extent to which next-step training on m is expected to induce relevant change, and the downstream value that such change has the potential to unlock. We consider $\mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}[\mathrm{LP}(m,\pi_i,1;m)]$ as a proxy for the former, and represent the latter through the **downstream potential** $D_i(m)$.

To estimate $D_i(m)$, we use an LLM before training to construct a directed **learning-connectivity graph** $\mathcal{G}_{\mathrm{LC}}$ (Appendix B). An edge $s \to t$, with LLM-derived weight $C(s,t) \in [0,1]$, represents evidence that useful learning on s would either improve performance on t directly or make subsequent training on t more productive. We compute $D_i(m)$ by aggregating the contributions of tasks that useful learning on m is predicted to benefit. Each task t contributes according to its importance and the strength of the strongest path from m. Let $\mathcal{P}_{i,L}(m,t)$ denote the paths from m to t of length at most L in the adapted graph defined below. We define:

$$D_i(m) = \sum_{\mathcal{P}_{i,L}(m,t)\neq\emptyset,\, t\neq m} w(t) \max_{p\in\mathcal{P}_{i,L}(m,t)} \left[\kappa^{|p|} \prod_{(s,u)\in p} C_i(s,u)\right],$$

where $\kappa \in (0,1]$ downweights longer paths. Here, we adapt the graph to the current learner state by setting $C_i(s,t) = C(s,t)\mathbf{1}\left[\widehat{P}_i(t) < \tau_{\mathrm{solved}}\right]$, so that $D_i(m)$ captures only potential downstream benefits associated with tasks that remain unsolved.

Finally, using $h(\widehat{\mathrm{LP}}_i(m))$ as a nonnegative proxy for the unavailable expected next-step progress, following Eq. (2), yields the LP-based prioritization score

$$S_i^{\mathrm{LP}}(m) = h\left(\widehat{\mathrm{LP}}_i(m)\right)\left[\widetilde{w}_i^{\mathrm{LP}}(m) + \alpha \widetilde{D}_i^{\mathrm{LP}}(m)\right].$$

The first component recovers importance-weighted local LP prioritization, while the second additionally values progress according to the task's estimated downstream potential. Here, $\widetilde{w}_i^{\mathrm{LP}}(m)$ and $\widetilde{D}_i^{\mathrm{LP}}(m)$ are normalized versions of $w(m)$ and $D_i(m)$ over the current support of $h(\widehat{\mathrm{LP}}_i(m))$. The normalization places the two quantities on comparable scales so that their relative influence can be controlled explicitly through $\alpha \geq 0$ (Appendix A.3).

4.2 Estimating feasibility of progress

Historical LP provides no evidence for distinguishing among tasks on which no success has yet been observed. We introduce $f_i(m) \in [0,1]$ to complement historical LP in estimating the ex-

pected progress from training on a low-performance task. Specifically, $f_i(m)$ estimates the current feasibility of making measurable progress by training directly on m.

To compute $f_i(m)$, we use an LLM before training to construct a directed **feasibility-evidence graph** $\mathcal{G}_{\mathrm{FE}}$ (Appendix B). For each target m, the LLM considers tasks s on which it would ordinarily expect the learner to exhibit competence before direct training on m begins producing measurable improvement. From these, it selects a small number of the most informative tasks, such that an edge $s \to m$, with nonnegative LLM-derived weight $E(s, m)$, represents competence on s as evidence that m is ready for productive training. The LLM may also judge m feasible from scratch, which we represent by an auxiliary source $\perp$. At curriculum decision i, we combine these evidence weights with the learner's current performance estimates so that $f_i(m)$ increases as the learner becomes competent on tasks identified as evidence:

$$f_i(m) = E(\perp, m) + \sum_{s \in \mathcal{T} \setminus \{m\}} E(s, m) \widehat{P}_i(s).$$

The incoming evidence weights are normalized to sum to one, ensuring $f_i(m) \in [0, 1]$.

Following the construction of S_i^{LP} in Section 4.1, we define

$$S_i^f(m) = \mathbf{1}\left[\widehat{P}_i(m) \le \tau_f\right] f_i(m) \left[\widetilde{w}_i^f(m) + \alpha\, \widetilde{D}_i^f(m)\right].$$

Here, $f_i(m)$ serves in place of $h(\widehat{\mathrm{LP}}_i(m))$ as a proxy for $\mathbb{E}_{p_c(\pi_{i+1}|\pi_i,m)}[\mathrm{LP}(m, \pi_i, 1; m)]$, while the bracket values that progress according to its direct importance and estimated downstream potential. The performance threshold restricts this contribution to low-performance tasks. $\widetilde{w}_i^f(m)$ and $\widetilde{D}_i^f(m)$ are normalized versions of $w(m)$ and $D_i(m)$ over the current support of the thresholded feasibility signal (Appendix A.3).

4.3 Task-selection distribution

The scores introduced in Sections 4.1 and 4.2 provide complementary criteria for task prioritization. Both rely on imperfect estimates of the consequences of training on each task. As discussed in Section 3, this motivates probabilistic rather than fully greedy selection. A sampling distribution also naturally extends the selection mechanism to settings where multiple tasks are selected at each decision. We convert each score separately into a sampling distribution,[1] $q_i^{\mathrm{LP}}(m) = S_i^{\mathrm{LP}}(m) / \sum_{m'} S_i^{\mathrm{LP}}(m')$ and $q_i^f(m) = S_i^f(m) / \sum_{m'} S_i^f(m')$. We also retain a uniform distribution q^{U} over the task set for score-independent exploration. After an initial period of uniform sampling, the curriculum task-selection distribution is

$$q_i = \eta_{\mathrm{U}} q^{\mathrm{U}} + \eta_f q_i^f + (1 - \eta_{\mathrm{U}} - \eta_f)\, q_i^{\mathrm{LP}},$$

where $\eta_{\mathrm{U}}, \eta_f \ge 0$ and $\eta_{\mathrm{U}} + \eta_f \le 1$. The coefficients therefore directly control the proportion of training allocated through each component.

5 Experiments

Our experiments evaluate whether incorporating LLM-informed downstream potential and feasibility guidance improves learning. We consider two settings: learning a designated target with access to auxiliary training tasks, and learning across an equally weighted task set.

5.1 Benchmark and tasks

We conduct experiments in Craftax (Matthews et al., 2024), a partially observable, procedurally generated environment designed for open-ended learning. In Craftax, the agent can traverse distinct biomes, gather and combine resources, craft and upgrade equipment, manage survival-related state, and interact with a diverse set of creatures. In our experiments, tasks differ only in their goal specification. Each task replaces the native Craftax rewards with a reward of one upon satisfying

[1] If the scores of a component sum to zero, its sampling distribution is defined to be uniform.

its goal and zero otherwise, with success terminating the episode. For this benchmark, $P(m; \pi)$ is the probability that policy π achieves goal m within an episode. We introduce a custom set of 256 goals, each paired with a corresponding textual description. The set covers a broad space of objectives, with goals ranging from immediately accessible to those requiring long sequences of intermediate accomplishments. The complete goal set is provided in Appendix C.1. We use this task set under two kinds of curriculum objectives. In the first, one task is designated as the target, while the remaining tasks are available for auxiliary training. We consider four target tasks: reaching the Dungeon, defeating a skeleton, having at least 64 wood, and placing a furnace. We refer to them as Dungeon, Skeleton, Wood, and Furnace, respectively. In the second, all tasks receive equal weight, and performance is evaluated across the complete set.

5.2 Learner and compared methods

The compared curriculum methods differ only in how they allocate experience across tasks. In all experiments, the trained agent is a goal-conditioned policy optimized with proximal policy optimization (Schulman et al., 2017), consisting of a Gated Transformer-XL backbone (Dai et al., 2019; Parisotto et al., 2020) with separate MLP heads for the actor and critic. The policy processes Craftax observations together with a fixed embedding of the textual goal produced by a frozen Qwen3-Embedding-4B model (Zhang et al., 2025). Unless stated otherwise, the policy's action space consists of Craftax's primitive actions. For learning across the task set, we also evaluate a variant that can invoke its learned behavior for other goals as temporally bounded subroutines. Further details on the learner and training setup are provided in Appendices C.2–C.4.

For the main method, we use $h(x) = |x|^{\rho}$, with $\rho > 0$, and denote the resulting curriculum by **ALP+D+F**, where **D** denotes the inclusion of downstream potential when valuing progress on a goal and **F** the inclusion of the feasibility-guided distribution. The exponent ρ controls the sensitivity of the sampling distribution to differences in estimated absolute learning progress. We compare against variants that remove each of the two proposed components (**ALP+D** and **ALP+F**), and against **ALP**, an LLM-free baseline that adapts conventional ALP prioritization to the weighted objective and removes both components. We additionally compare against uniform goal sampling (**UNIF**) and an intermediate-difficulty curriculum (**I-DIFF**), which prioritizes goals for which current performance lies within a fixed competence range. In the single-target experiments, we also include **TARGET**, which allocates the entire training budget to the target goal, and **PERM**, an ablation of ALP+D+F that randomly permutes the goal labels of the learning-connectivity graph while leaving the feasibility-evidence graph unchanged. Full definitions are provided in Appendix D.

All adaptive curricula allocate 80% of their sampling mass according to their primary prioritization score. For ALP, ALP+D, and I-DIFF, the remaining 20% is assigned to uniform sampling; for ALP+F, ALP+D+F, and PERM, it is split equally between uniform and feasibility-guided sampling. Because feasibility-guided sampling can prioritize low-performance tasks before measurable progress is observed, we assign it part of the uniform exploration budget while keeping the LP-based allocation fixed. At each curriculum decision, we sample 64 goals with replacement from the resulting distribution and assign each to eight parallel environments, for a total of 512 environments. In the single-target experiments, for every method other than **TARGET**, four of the 64 goal selections are fixed to the target, with the remaining 60 sampled from the corresponding curriculum distribution. This guarantees all curriculum strategies a common minimum amount of direct training on the target. Further evaluation details are provided in Appendix C.5.

5.3 Results

5.3.1 Learning target tasks

With all objective weight assigned to a single target, ALP+D+F substantially improves learning over ALP (Figure 1). After approximately 1.5 billion environment steps, it reaches mean success rates of 44%, 17%, and 79% on Dungeon, Wood, and Furnace respectively, compared with 2%, 0%, and 0% for ALP. On Skeleton, the advantage is primarily in learning speed: ALP+D+F reaches 50% evaluation success earlier in every paired seed, while both methods finish with similar performance. TARGET has zero success at the final evaluation on all four targets. The onset of learning and final performance vary substantially across seeds.

The ablations indicate a stronger and more consistent benefit from downstream potential than from feasibility guidance. ALP+D improves over ALP on all four targets, and methods without D achieve no success by the end of training on Wood and Furnace. PERM retains some of the benefits of adding D, suggesting some tolerance to inaccuracies in the learning-connectivity graph. However, in its successful Dungeon and Furnace runs, we find that the permuted graphs still assign substantial downstream potential to prerequisite goals (approaching the ladder and obtaining a wooden pickaxe, respectively). Adding F to ALP and ALP+D leads to earlier improvements on Dungeon and Skeleton. For ALP+D, it also brings forward improvements on Wood but delays them on Furnace. These earlier gains do not consistently translate into higher final performance.

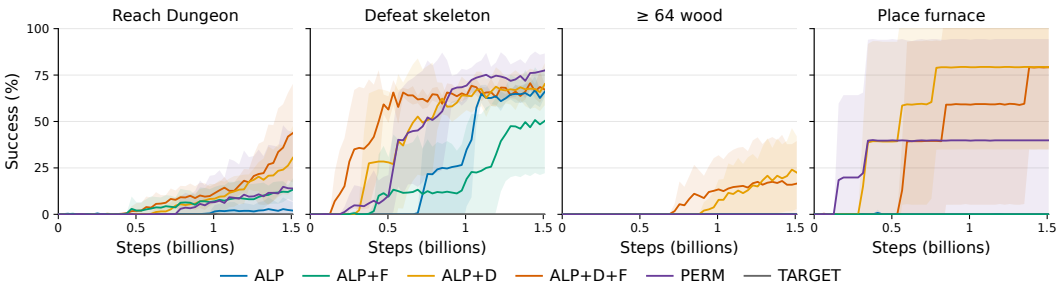

Figure 1: Evaluation success on selected target tasks during training, with all objective weight assigned to the corresponding target. Lines show means across five seeds; shaded regions indicate one standard deviation.

Curriculum dynamics Figure 2 illustrates how the proposed components shape curriculum behavior on Wood using each method's highest-performing run. Appendix E.1 contains illustrations for the other targets and all ALP+D+F seeds. Because only the target has nonzero objective weight, ALP and ALP+F can only adapt its sampling probability while all auxiliary goals remain equally likely. ALP distributes its adaptive mass uniformly in the absence of estimated target progress, and assigns it entirely to the target when progress is detected. Appendix E.2 examines the effect of relaxing this restriction by assigning only half of the objective weight to the target. ALP+F can increase target probability earlier, once competence is acquired on the target's feasibility-evidence goals. Here, learning on the 32-wood goal activates feasibility-guided sampling of the 64-wood target while target success remains zero.

Downstream potential changes the curriculum treatment of auxiliary training by explicitly valuing each task's predicted contribution to the target. This allows learning progress on related goals to

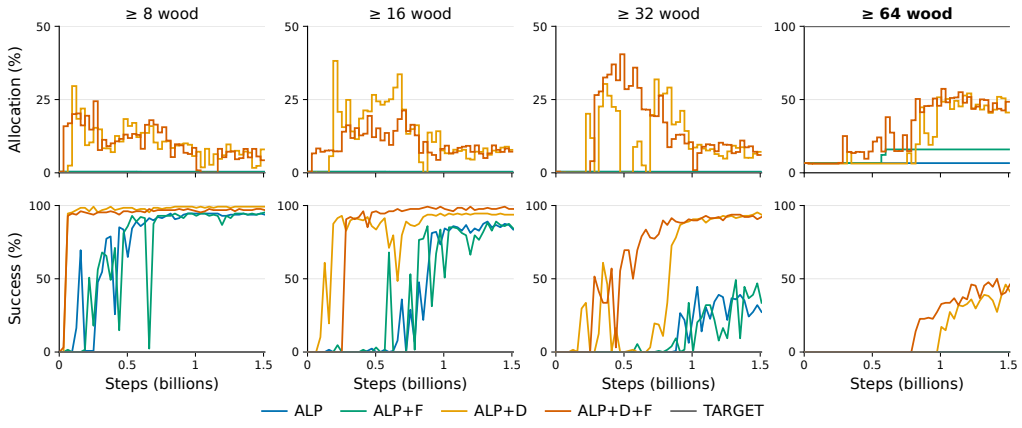

Figure 2: Illustrative curriculum dynamics with all objective weight assigned to having at least 64 wood. The top row shows the probability mass allocated to the target and selected related auxiliary goals, averaged over non-overlapping blocks of 15 curriculum decisions. The bottom row shows the policy's success rate when evaluated on each goal. For each method, we show the run with the highest final target success among the five seeds in Figure 1.

guide task selection even before progress is observed on the target itself. In Figure 2, ALP+D and ALP+D+F devote substantial probability to lower wood thresholds while target success remains zero. The emphasis subsequently shifts toward direct target practice as performance on the lower thresholds becomes reliable and target success begins to increase. These trajectories illustrate how the task relationships supplied before training combine with online learning signals to prioritize potentially useful auxiliary goals and adapt their share of training as learning opportunities develop.

5.3.2 Learning across the task set

Under uniform task weights, ALP and ALP+D+F outperform both uniform sampling and intermediate-difficulty prioritization for both learner variants (Figure 3), supporting the use of learning progress to allocate training in this setting. For the learner restricted to primitive actions, adding downstream potential and feasibility guidance provides a modest advantage over ALP during the earlier stages of learning, but the methods reach similar performance by the end of training. Their final task coverage also overlaps strongly: about 94% of the goals achieved by each method are also achieved by the other in paired runs, averaged across seeds. In this setting, the added components primarily bring improvements forward in training without materially extending final task coverage.

We additionally evaluate the curricula with a learner that can explicitly reuse behavior learned on other goals through temporally bounded subroutine calls. With this additional mechanism for inter-task transfer, the advantage of ALP+D+F over ALP is more pronounced and persists through the end of training. At the final evaluation, ALP+D+F reaches a mean success rate of 31.3% across the task set, compared with 25.5% for ALP, and achieves nonzero success on 41.3% of goals, compared with 31.6%. The results suggest that the value of accounting for the additional considerations captured by the LLM-informed components can depend on how learning transfers across tasks.

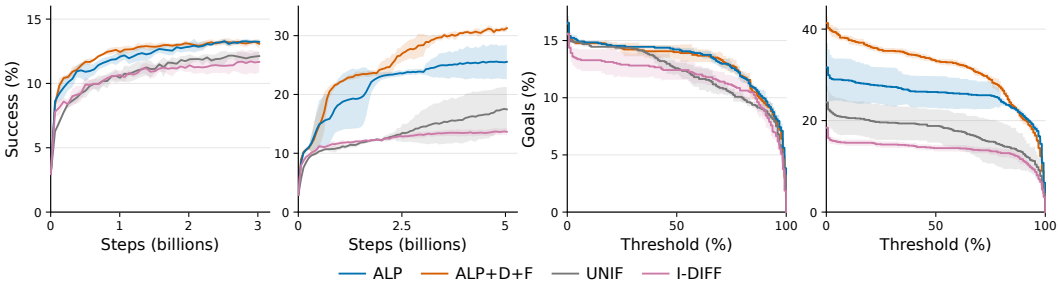

Figure 3: Evaluation under uniform task weights. The two left panels show mean success rate across all 256 tasks during training; the two right panels show the percentage of tasks whose success rate strictly exceeds each threshold after training. Within each pair, the policy restricted to primitive actions is shown first and the policy with subroutine calls second. Lines show means across five seeds; shaded regions indicate one standard deviation.

6 Conclusion

We analyzed automatic curriculum learning as a sequential decision making problem and examined several challenges involved in determining the value of curriculum actions. Motivated by this analysis, we investigated the use of LLMs to draw on richer information about the learning problem and better anticipate the consequences of curriculum decisions. We developed a concrete instantiation that augments learning-progress-based prioritization with LLM-informed estimates of the potential downstream benefits of training on a task and of whether direct training on a low-performance task is likely to produce progress. Our experiments in Craftax show that these components can improve learning and produce curriculum behavior consistent with their intended roles. The gains are most pronounced when optimizing for a single target and when learning across the full task set with a learner that can reuse goal-directed behaviors through subroutine calls. Our empirical evaluation is limited to a fixed set of goals, while the proposed instantiation uses offline LLM judgments and models only beneficial downstream effects. Future work could evaluate the approach on tasks that differ in aspects other than their goal specification, extend it to account for negative cross-task effects, and explore alternative ways of using LLMs to address the challenges identified by our analysis.

AI USE STATEMENT

Generative AI tools assisted with implementing the methods and experimental infrastructure, constructing the custom goal benchmark, preparing figures, and suggesting alternative phrasings and editorial revisions to author-written text. They also supported discussion of methodology, experimental design, and analysis of the curriculum problem and experimental results. These discussions had limited influence on the work, primarily affecting secondary aspects of the method and setup. The authors developed the core research ideas, arguments, methodology, experimental design, analyses, and manuscript. The code was developed iteratively with guidance and review from the authors. The authors take responsibility for the final content of this work.

REFERENCES

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International conference on machine learning*, pp. 1331–1340. PMLR, 2019.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

Francesco Foglino, Christiano Coletto Christakou, Ricardo Luna Gutierrez, and Matteo Leonetti. Curriculum learning for cumulative return maximization. *arXiv preprint arXiv:1906.06178*, 2019a.

Francesco Foglino, Christiano Coletto Christakou, and Matteo Leonetti. An optimization framework for task sequencing in curriculum learning. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 207–214. IEEE, 2019b.

Sébastien Forestier, Rémy Portelas, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *Journal of Machine Learning Research*, 23(152):1–41, 2022.

Faraz Heravi, James Ouyang, Zifan Xu, Arjun Kumar, Yoonchang Sung, and Peter Stone. Leacl: Llm-enhanced automatic curriculum learning for reinforcement learning in long-horizon manipulation tasks. *arXiv preprint arXiv:2607.23515*, 2026.

Hanchi Huang, Deheng Ye, Li Shen, and Wei Liu. Curriculum-based asymmetric multi-task reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 45(6): 7258–7269, 2022.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026.

Vikas Jain and Theja Tulabandhula. Faster reinforcement learning using active simulators. *arXiv preprint arXiv:1703.07853*, 2017.

Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Houghton, Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint arXiv:2106.14876*, 2021.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

Sanmit Narvekar and Peter Stone. Learning curriculum policies for reinforcement learning. *arXiv preprint arXiv:1812.00285*, 2018.

Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *IJCAI*, pp. 2536–2542, 2017.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pp. 5062–5071. PMLR, 2019.

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International conference on machine learning*, pp. 7750–7761. PMLR, 2020.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pp. 835–853. PMLR, 2020a.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020b.

Guillaume Pourcel, Thomas Carta, Grgur Kovač, and Pierre-Yves Oudeyer. Autotelic llm-based exploration for goal-conditioned rl. In *Intrinsically-Motivated and Open-Ended Learning Workshop@ NeurIPS2024*, 2024.

Kanghyun Ryu, Qiayuan Liao, Zhongyu Li, Payam Delgosha, Koushil Sreenath, and Negar Mehr. Curricullm: Automatic task curricula design for learning complex robot skills using large language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4470–4477. IEEE, 2025.

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

Felipe Leno Da Silva and Anna Helena Reali Costa. Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems*, pp. 1026–1034, 2018.

Jivko Sinapov, Sanmit Narvekar, Matteo Leonetti, and Peter Stone. Learning inter-task transferability in the absence of target task samples. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pp. 725–733, 2015.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. Automatic curriculum graph generation for reinforcement learning agents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

SIMA Team. Sima 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

Georgios Tzannetos, Parameswaran Kamalaruban, and Adish Singla. Proximal curriculum with task correlations for deep reinforcement learning. *arXiv preprint arXiv:2405.02481*, 2024.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

Zifan Xu, Yulin Zhang, Shahaf S Shperberg, Reuth Mirsky, Yuqian Jiang, Bo Liu, and Peter Stone. Model-based meta automatic curriculum learning. In *Conference on Lifelong Learning Agents*, pp. 846–860. PMLR, 2023.

Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. Omni: Open-endedness via models of human notions of interestingness. In *International Conference on Learning Representations*, volume 2024, pp. 6074–6120, 2024.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

APPENDIX CONTENTS

A Method details **14**

A.1 Pseudocode 14

A.2 Online performance and learning-progress estimation 14

A.3 Score normalization 15

B Offline LLM graph generation **15**

B.1 Construction procedure 15

B.2 Prompts 16

C Experimental setup **23**

C.1 Benchmark goal set 23

C.2 Learner architecture 27

C.3 Training protocol 27

C.4 Policy with subroutine calls 28

C.5 Evaluation procedure 29

C.6 Hyperparameter selection 29

C.7 Sensitivity analysis 30

D Baselines and ablations **32**

D.1 Baselines 32

D.2 Ablations 32

E Additional results and analyses **33**

E.1 Curriculum dynamics 33

E.2 Smoothing the target-centered objectives 36

E.3 Discussion of downstream potential 36

A METHOD DETAILS

For the main method, we use a learning-progress exponent $\rho = \frac{1}{2}$, downstream coefficient $\alpha = 1$, maximum path length $L = 2$, path discount $\kappa = 0.3$, solved-task threshold $\tau_{\text{solved}} = 0.9$, and feasibility threshold $\tau_f = 0.05$. Ablations retain these settings for the components they preserve. The downstream-potential and feasibility hyperparameters were fixed without tuning; tuning of shared hyperparameters is described in Appendix C.6, and sensitivity to L and α is examined in Appendix C.7. Performance and learning-progress estimation settings are specified in Appendix A.2, while task-allocation settings are given in Section 5.2 and Appendix D.

A.1 PSEUDOCODE

Algorithm 1: Curriculum procedure

Input: Task set $\mathcal{T}$; importance weights w; fixed graphs $\mathcal{G}_{\text{LC}}$ and $\mathcal{G}_{\text{FE}}$; initial learner π_0

Initialize the performance and learning-progress estimators;
for *each curriculum decision* i **do**
  Update $\widehat{P}_i$ and $\widehat{\text{LP}}_i$ using observations from the previous curriculum iteration, if available;
  Adapt $\mathcal{G}_{\text{LC}}$ to $\widehat{P}_i$ and compute the downstream potential D_i;
  Compute the feasibility estimate f_i from $\mathcal{G}_{\text{FE}}$ and $\widehat{P}_i$;
  Compute the scores S_i^{LP} and S_i^f;
  Construct the task-selection distribution q_i, using uniform sampling during warm-up;
  Sample tasks from q_i and assign them to the training environments;
  Train the learner on the assigned tasks and record the resulting performance observations;
end

A.2 ONLINE PERFORMANCE AND LEARNING-PROGRESS ESTIMATION

At each curriculum decision, the performance and learning-progress estimates are updated using training outcomes from the preceding curriculum iteration. After iteration i, let $\bar{P}_i(m)$ denote the empirical success rate across completed episodes in environments assigned task m. For each environment, we include only its $K = 5$ most recent completed episodes.

Performance estimates are initialized as $\widehat{P}_0(m) = 0$. The first observation of a task replaces this initial value, while subsequent observations update the estimate using an exponential moving average:

$$\widehat{P}_{i+1}(m) = \begin{cases} \bar{P}_i(m), & \text{if } m \text{ is observed for the first time,} \\ (1-\beta)\widehat{P}_i(m) + \beta\bar{P}_i(m), & \text{if } m \text{ is observed again during iteration } i, \\ \widehat{P}_i(m), & \text{if } m \text{ is not observed during iteration } i, \end{cases}$$

where $\beta = 0.3$.

Learning progress is estimated from the unsmoothed observations $\bar{P}_i(m)$, rather than from $\widehat{P}_i(m)$. For each task, we retain at most $N_{\text{obs}} = 7$ observations, each paired with the curriculum iteration in which it was collected. When a new observation is available, $\widehat{\text{LP}}_{i+1}(m)$ is the slope of an ordinary least-squares line fitted to the current observation and the retained observations collected within the preceding $H_{\text{LP}} = 12$ curriculum iterations. Thus, N_{obs} limits the number of observations in the fit, whereas H_{LP} limits the interval over which they may have been collected.

Each task history is initialized with the pseudo-observation $(-1, 0)$. If no previous observation lies within the lookback interval, the most recent earlier observation is included and treated as occurring at the beginning of the interval. If no new observation is available for a task, its previous learning-progress estimate is retained.

A.3 SCORE NORMALIZATION

Because the LP- and feasibility-based scores have different active supports, we normalize task importance and downstream potential separately for each component. Define

$$\mathcal{A}_i^{\mathrm{LP}} = \left\{ m \in \mathcal{T} : h\Big(\widehat{\mathrm{LP}}_i(m)\Big) > 0 \right\},$$

and

$$\mathcal{A}_i^{f} = \left\{ m \in \mathcal{T} : \mathbf{1}\Big[\widehat{P}_i(m) \le \tau_f\Big]\, f_i(m) > 0 \right\}.$$

For each component $c \in \{\mathrm{LP}, f\}$ with $\mathcal{A}_i^c \neq \emptyset$, let

$$\bar{w}_i^c = \frac{1}{|\mathcal{A}_i^c|} \sum_{m \in \mathcal{A}_i^c} w(m), \qquad \bar{D}_i^c = \frac{1}{|\mathcal{A}_i^c|} \sum_{m \in \mathcal{A}_i^c} D_i(m).$$

The normalized quantities used in the corresponding score are

$$\widetilde{w}_i^c(m) = \frac{w(m)}{\bar{w}_i^c}, \qquad \widetilde{D}_i^c(m) = \frac{D_i(m)}{\max\{\bar{D}_i^c, \bar{w}_i^c\}}.$$

When $\bar{D}_i^c \ge \bar{w}_i^c > 0$, each quantity is normalized by its own active-set mean, giving the direct and downstream terms the same mean scale before weighting by α. When $\bar{D}_i^c < \bar{w}_i^c$, the downstream term is instead normalized by $\bar{w}_i^c$, preventing a small estimated downstream signal from being amplified to the scale of the direct contribution solely through normalization.[2]

B OFFLINE LLM GRAPH GENERATION

B.1 CONSTRUCTION PROCEDURE

Both graphs are constructed offline over the 256 benchmark goals from repeated LLM judgments about which directed edges should be included. Each query provides the LLM with descriptions of the Craftax environment, goal suite, learner, and training setting, together with graph-specific instructions and a structured response schema. For each graph type, we generate $R = 5$ versions using different goal orderings and aggregate their judgments into the edge weights used during training. We use Gemini 3.7 Flash with thinking level set to high. Temperature and top-p are left at their API defaults.

Learning-connectivity graph. Every goal is considered both as a potential source and as a potential target. In each repetition, we independently shuffle two copies of the complete goal set, one for each role. The source goals are divided into blocks of at most 64 and the target goals into blocks of at most 16, and we query every pairing of a source block with a target block. Consequently, every ordered pair of distinct goals is considered exactly once per repetition.

Let $C^{(r)}(s,t) \in \{0,1\}$ indicate whether the LLM includes the edge $s \to t$ in repetition r, with self-connections excluded. The final connectivity weight is

$$C(s,t) = \frac{1}{R} \sum_{r=1}^{R} C^{(r)}(s,t). \tag{4}$$

Thus, $C(s,t)$ is the fraction of repetitions in which the LLM includes the edge.

[2] If $\mathcal{A}_i^c = \emptyset$, S_i^c is defined to be identically zero. If a denominator in the normalization above is zero, the corresponding normalized quantity is defined to be zero.

Feasibility-evidence graph. For each goal m, we evaluate all other goals as potential evidence sources. In each repetition, we shuffle the order of the potential evidence goals and independently shuffle the target goals. The targets are divided into blocks of at most eight, and each query evaluates one target block while providing the complete goal list as evidence candidates; the evaluated goal itself cannot be selected. Thus, every goal is evaluated exactly once per repetition with all other goals available as potential evidence.

For each target m, the LLM either judges direct training feasible from initialization or selects between one and three other goals as evidence. Let $A^{(r)}(m)$ contain $\perp$ in the former case and the selected evidence goals otherwise. We convert each judgment into edge weights by assigning total weight one across the members of $A^{(r)}(m)$: each selected source receives weight $1/|A^{(r)}(m)|$, and all other sources receive zero. The final feasibility-evidence weight averages these values across repetitions:

$$E(s,m) = \frac{1}{R}\sum_{r=1}^{R} \frac{\mathbf{1}\big[s \in A^{(r)}(m)\big]}{|A^{(r)}(m)|}, \qquad s \in (\mathcal{T} \setminus \{m\}) \cup \{\perp\}. \tag{5}$$

Dividing the weight equally prevents a repetition from contributing more total evidence merely because it selects more sources. It also ensures

$$E(\perp, m) + \sum_{s \in \mathcal{T}\setminus\{m\}} E(s,m) = 1 \tag{6}$$

for every target m.

B.2 Prompts

The natural-language prompt templates used to construct the two graphs are reproduced below. Each query instantiates the relevant template with three context blocks describing the environment, goal suite, and learner, together with the query-specific goal lists determined by the batching procedure in Appendix B.1. In our experiments, we use these templates to generate one pair of graphs for each learner variant, changing only the learner context. Each pair is then reused across runs with the corresponding learner.

Angle-bracketed terms denote these inserted fields. For learning-connectivity queries, `<SOURCE GOALS>` and `<TARGET GOALS>` contain the corresponding source and target blocks. For feasibility-evidence queries, `<EVIDENCE GOALS>` contains the complete goal set and `<TARGET GOALS>` the current target block. Each query was additionally supplied with a graph-specific JSON response schema used to constrain the model output to the required structured format; the schema definitions are omitted here.

Learning-connectivity prompt. The template used for each learning-connectivity query is:

```
Task: infer directed learning-connectivity edges between the supplied Craftax goals. Your
predictions will be used to construct a learning-connectivity graph for an automatic
curriculum-learning algorithm.

## Environment context

<ENVIRONMENT CONTEXT>

<GOAL-SUITE CONTEXT>

## Learner context

<LEARNER CONTEXT>

## Meaning of an edge

Assume that training on the source goal meaningfully improves the learner's ability to
achieve it. Include an edge `source -> target` only when the resulting learning is
reasonably expected to improve the policy's ability to achieve the target or make
subsequent training on the target more productive.

That is, include an edge when you assess that either of the following holds:
```

```
1. **Immediate transfer:** source learning directly improves the policy's ability to
achieve the target in a new episode where the policy is conditioned on the target, before
any target-specific training.
2. **Enabling transfer:** source learning makes later training on the target more likely to
produce meaningful policy improvement. It may make future progress on the target possible
or allow it to occur faster or more reliably.

Include only transfer that is expected to be meaningful and attributable to what is learned
while improving on the source; benefits expected from getting any generic additional
experience on the environment are not sufficient.

During training, the curriculum algorithm decides which goals to train. It uses these edges
to value progress on a source partly according to whether the resulting learning could also
benefit other goals later.

Textual similarity, thematic relatedness, or a game-mechanical prerequisite relationship
may provide evidence of transfer when judging an edge, but they do not determine the
judgment by themselves. Learning transfer need not follow relative goal difficulty or
prerequisite order; a harder source may also benefit an easier target. Within-episode state
changes associated with completing a source are temporary and should be distinguished from
learning retained in the shared policy. Include an edge only when progress on the source is
expected to produce retained learning that benefits the target under the definition above.
Include every qualifying edge for which progress on the source is reasonably expected to
improve the policy's ability to achieve the target or make subsequent training on the
target more productive.

## Candidate goals

Potential sources:
<SOURCE GOALS>

Targets:
<TARGET GOALS>

## Response requirements

Populate the supplied structured response with exactly one record for every prompted
target.

* Reproduce every `target_id` exactly as it appears in the prompted target list.
* Use only exact IDs from the prompted source list in `transfer_source_ids`.
* Use an empty `transfer_source_ids` list when no source qualifies.
* Do not duplicate target IDs or source IDs.
* Do not include a goal as a transfer source for itself.
* Return only the supplied structured response.
```

Feasibility-evidence prompt. The template used for each feasibility-evidence query is:

```
Task: assess when direct training on each supplied Craftax goal is likely to produce
measurable learning progress. Your judgments will be used to construct a feasibility graph
for an automatic curriculum-learning algorithm.

## Environment context

<ENVIRONMENT CONTEXT>

## Goal-suite context

<GOAL-SUITE CONTEXT>

## Learner context

<LEARNER CONTEXT>

## What to decide for each goal

For each goal in `Goals to evaluate`, imagine that the learner does not yet know how to
achieve it. You must help decide when training directly on that goal (using it to condition
the policy and rewarding only success on it over repeated fresh episodes) is likely to
begin improving the policy's success on it.

To make this judgment, identify the goals from `Goals available as readiness evidence` on
which you would ordinarily expect the learner to exhibit competence before direct training
on the evaluated goal is likely to begin producing improvement. The learner is competent on
a goal when it can achieve that goal with nontrivial, reasonably reliable success while
conditioned on it.

This judgment requires careful reasoning about the characteristics of the learner and
learning setting, the goals' success conditions and relationships, and the relevant Craftax
mechanics and environment structure.
```

```
During training, the algorithm will use these judgments only while the evaluated goal's own
success rate remains zero or low. When `feasible_from_scratch` is false, it will average the
learner's current success rates on the selected readiness-evidence goals. A higher average
will be interpreted as stronger evidence of readiness and make the evaluated goal more
likely to be selected for direct training. A `feasible_from_scratch` judgment instead
provides readiness evidence that does not depend on the learner's success on another goal.

Set `feasible_from_scratch` to true only when a learner whose trainable policy parameters
have just been initialized is likely to encounter rewarded success often enough for direct
training on the evaluated goal to begin improving the policy. The fact that the goal can be
achieved in principle is not sufficient.

Otherwise, set `feasible_from_scratch` to false and select between 1 and 3 goals from `Goals
available as readiness evidence`. Select the most informative goals that satisfy the
criterion above. Do not select a goal that the learner would ordinarily be expected to
master only after the evaluated goal. Among the remaining qualifying goals, omit earlier or
broader goals when competence on a more advanced or specific goal already provides the same
evidence. The upper bound is only a cap, not a target. Include only goals whose competence
would provide meaningful readiness evidence on its own, and do not include goals whose
evidence is subsumed by that of other selected goals.

Before returning the response, review every judgment for consistency with the relevant
Craftax mechanics, the supplied goal definitions, and the learner and readiness-evidence
definitions above.

## Goals

Goals available as readiness evidence:
<EVIDENCE GOALS>

Goals to evaluate:
<TARGET GOALS>

## Response requirements

Return exactly one entry in `judgments` for every goal in `Goals to evaluate`.

* In each entry, copy the corresponding goal's `id` exactly into `evaluated_goal_id`.
* If `feasible_from_scratch` is true, return an empty `readiness_evidence_goal_ids` list.
* If `feasible_from_scratch` is false, return between 1 and 3 exact IDs from `Goals
available as readiness evidence` in `readiness_evidence_goal_ids`.
* Do not repeat an `evaluated_goal_id` or any ID within a `readiness_evidence_goal_ids`
list.
* Do not include the evaluated goal's own ID in its `readiness_evidence_goal_ids` list.
* Return only the supplied structured response.
```

Shared context. The following environment, goal-suite, and learner descriptions are inserted into the corresponding placeholders in both prompt templates. The environment and goal-suite descriptions are shared across learner variants, while the learner description depends on the variant.

Environment context.

```
Craftax is a survival, exploration, crafting, and combat environment set in a procedurally
generated world. The player begins empty-handed in the Overworld and can explore a sequence
of regions below it. Play can involve navigating the world, maintaining the player's basic
needs, gathering materials, making and improving equipment, fighting creatures, and using
systems such as ranged combat, magic, and enchanting.

### The player and the world

Each episode creates a new world containing nine regions, or floors. Every floor is a
48-by-48 grid, but its terrain, resources, ladders, rooms, chests, and creature activity
vary from one world to the next. The player begins at the centre of the Overworld, facing
north, with no resources or equipment. Health, food, drink, energy, and mana all begin at
9; dexterity, strength, and intelligence begin at 1.

The player explores from a local viewpoint rather than receiving a complete map. It can
perceive a 9-by-11 area centred on its current position, together with its inventory,
equipment, survival values, experience and attributes, facing direction, current floor,
learned spells, and relevant floor or encounter state. Unexplored parts of the world are
not revealed through a global map. Several underground regions are dark away from their lit
arrival areas, lava, and player-placed torches, further limiting what can be seen there.

Moving between floors does not reset the player or the world. Inventory, equipment,
survival state, attributes, spells, opened chests, planted crops, creatures, and progress
through each floor remain part of the same episode. Death, completion of the Necromancer
encounter, or reaching the configured time limit ends the episode. A new episode replaces
the entire world and returns the player to the initial state.

The Overworld has a day-night cycle. Night makes hostile melee creatures appear more
frequently there. The underground regions have their own fixed patterns of light and
darkness rather than following the surface cycle.
```

```
### Movement, facing, and actions

The player can move one tile north, south, east, or west. A movement action also changes
the direction it faces, even if an obstacle prevents it from entering the destination tile.
Grass, sand, paths, gravel, fire grass, and ice grass are traversable. Water, lava, rock,
trees, ore deposits, walls, crops, workstations, chests, fountains, graves, and the
Necromancer are not. Creatures also block the tiles they occupy.

Most interaction with the world is local. Direct interaction, placement, shooting, and
spell casting operate in the direction the player is facing. Crafting is the main
exception: it can use a required crafting table or furnace on any of the eight surrounding
tiles, including diagonals.

The action set contains 43 actions. Choosing an action does not guarantee that its named
effect occurs: an action succeeds only when all of its mechanical conditions are satisfied.
The main action groups are:

| Action | Conditions and effect |
|---|---|
| No-op | Takes no deliberate action while time and the rest of the world continue to
advance. |
| Move | Enters the adjacent tile only if that tile is traversable and unoccupied. The
player faces the chosen direction whether or not movement succeeds. |
| Direct interaction | Acts on the adjacent tile in front of the player. A creature is
attacked first if one occupies the tile. Otherwise the same action may gather or mine a
block, open a chest, drink, eat a ripe crop, remove a workstation, or interact with a
vulnerable Necromancer, depending on what is present and whether any tool requirement is
met. |
| Place | Attempts to place a crafting table, furnace, stone block, sapling, or torch on
the tile in front. It succeeds only if the player has the required material or item and the
destination is valid and unoccupied. |
| Craft | Produces the chosen item only if the player has its ingredients and all required
workstations are on surrounding tiles. |
| Ascend or descend | Changes floor only while the player is standing on the corresponding
ladder and that direction of travel is available. |
| Shoot | Fires only if the player owns a bow, carries at least one arrow, and fewer than
three player projectiles are active on the current floor. A successful shot consumes one
arrow. |
| Cast a spell | Fires only if the spell has been learned, at least 2 mana is available,
and fewer than three player projectiles are active on the current floor. A successful cast
consumes 2 mana. |
| Use an item | Drinking a potion or reading a book requires the corresponding item and
consumes it. Each potion colour has its own drinking action. |
| Sleep or rest | Begins only when the relevant recovery condition is unmet. Once begun, it
temporarily overrides other chosen actions until the player has recovered or the state is
interrupted. |
| Improve an attribute | Requires one unspent experience point and an attribute below its
maximum level. |
| Enchant equipment | Requires the relevant equipment, the correct enchantment table on the
tile directly in front of the player, the corresponding gem, and sufficient mana. |

### Regions and travel

The nine regions form one descending sequence. Each has a recognisable environment, set of
resources, and family of creatures, while the detailed layout is generated anew for every
episode.

| Floor | Region | Environment and notable features | Creatures |
|---:|---|---|---|
| 0 | Overworld | An outdoor landscape of grass, trees, sand, water, stone, and lava. Coal
and iron occur here, while diamond is exceptionally rare. | Cow; Zombie; Skeleton |
| 1 | Dungeon | Illuminated rooms and corridors containing chests and fountains. | Snail;
Orc Soldier; Orc Mage |
| 2 | Gnomish Mines | Dark caves with stone, stalagmites, water, lava, and all five ores. |
Bat; Gnome Warrior; Gnome Archer |
| 3 | Sewers | Illuminated rooms and corridors containing chests, water, and an
ice-enchantment table. | Snail; Lizard; Kobold |
| 4 | Vault | Illuminated rooms and corridors containing chests, fountains, and a
fire-enchantment table. | Snail; Knight; Archer |
| 5 | Troll Mines | Dark caves rich in coal, iron, diamond, sapphire, and ruby, with stone,
stalagmites, water, and lava. | Bat; Troll; Deep Thing |
| 6 | Fire Realm | A bright volcanic region of fire grass, fire trees, extensive lava,
stone, coal, and ruby. | Bat; Pigman; Fire Elemental |
| 7 | Ice Realm | A dark frozen region of ice grass, ice shrubs, water, stone, diamond, and
sapphire. | No passive creature; Frost Troll; Ice Elemental |
| 8 | Graveyard | A dark, enclosed landscape of paths, walls, and graves built around the
Necromancer encounter. | Encounter waves drawn from the earlier regions |

The Dungeon, Sewers, and Vault contain fixed torches in room corners and occasional mossy
sections among the walls bordering their accessible spaces. The Graveyard can also contain
mossy walls. Fixed torches are part of the generated environment and cannot be collected
into the inventory; mossy walls are impassable like ordinary walls.

On each floor from the Dungeon through the Ice Realm, the route down remains locked until
the player has defeated eight hostile creatures on that floor. The Overworld's downward
ladder does not have this kill requirement. Both melee and ranged enemies count; passive
creatures do not. The count belongs to the floor and persists if the player leaves and
returns. Routes upward have no kill requirement.
```

```
Using a ladder is deliberate: the player must stand on it and choose ascend or descend. Descending places the player at the upward ladder of the next region, and ascending returns it to the downward ladder of the preceding region. The Graveyard is the final floor and has no further destination below it.

### Gathering and changing the terrain

The player gathers from the tile directly in front of it. Ordinary trees, fire trees, and ice shrubs can be cut without a tool and yield one wood. Interacting with grass has a 10% chance of finding a sapling without removing the grass. Stone, stalagmites, coal, and the metal or gem deposits require progressively stronger pickaxes:

| Block | Minimum pickaxe | Yield |
|---|---|---:|
| Stone or stalagmite | Wood | 1 stone |
| Coal | Wood | 1 coal |
| Iron | Stone | 1 iron |
| Diamond | Iron | 1 diamond |
| Sapphire | Diamond | 1 sapphire |
| Ruby | Diamond | 1 ruby |

A successful mining action removes the block and adds its resource to the inventory. Using an inadequate pickaxe leaves the block unchanged.

Placement allows the player to alter its immediate surroundings. A crafting table costs 2 wood and a furnace costs 1 stone. These workstations can be placed directly without another workstation. They can later be removed by direct interaction, but removing them does not return their materials. A carried torch can be placed on grass, sand, a path, fire grass, or ice grass. It remains in the world, permanently lights a small area for the rest of the episode, and cannot be recovered into the inventory.

One stone can be placed on eligible ground or used to replace a water or lava tile. The result is a solid stone block, not a traversable path; it must be mined before the player can occupy that tile.

A sapling can be planted on an empty grass tile. It begins as an unripe plant and ripens after 600 environment steps. Eating the ripe plant restores 4 food and returns it to the unripe state, allowing the same plant to grow again.

### Crafting and ordinary equipment

Crafting uses carried materials together with nearby workstations. A crafting table or furnace counts as nearby when it occupies any of the eight tiles around the player. The recipes are:

| Item produced | Materials consumed | Required nearby workstation |
|---|---|---|
| Wood pickaxe | 1 wood | Crafting table |
| Stone pickaxe | 1 wood, 1 stone | Crafting table |
| Iron pickaxe | 1 wood, 1 stone, 1 iron, 1 coal | Crafting table and furnace |
| Diamond pickaxe | 1 wood, 3 diamond | Crafting table |
| Wood sword | 1 wood | Crafting table |
| Stone sword | 1 wood, 1 stone | Crafting table |
| Iron sword | 1 wood, 1 stone, 1 iron, 1 coal | Crafting table and furnace |
| Diamond sword | 1 wood, 2 diamond | Crafting table |
| One iron armour piece | 3 iron, 3 coal | Crafting table and furnace |
| One diamond armour piece | 3 diamond | Crafting table |
| 2 arrows | 1 wood, 1 stone | Crafting table |
| 4 torches | 1 wood, 1 coal | Crafting table |

The player carries one pickaxe and one sword, each represented by its highest acquired tier. Crafting or finding a lower-tier version does not replace better equipment. Tools, weapons, and armour do not wear out.

Armour has four slots, handled in the order helmet, chestplate, leggings, and boots. Crafting iron armour fills the first empty slot in this order. Crafting diamond armour fills or upgrades the first slot that is not already diamond. Every iron piece reduces incoming physical damage by 10%, and every diamond piece reduces it by 20%, so complete iron and diamond sets provide 40% and 80% physical defence respectively.

A bow is not craftable. It is obtained from the first chest the player opens in the Dungeon. Arrows can be crafted at any time or found in chests, but they cannot be fired until the bow has been obtained.

### Survival and recovery

Health, food, drink, and energy link survival to the passage of time. Outside the Graveyard, food, drink, and energy are gradually depleted. Health slowly recovers while food and drink are above zero and the player either has energy or is sleeping; otherwise it gradually declines. In the Graveyard, survival needs do not decrease and empty needs do not cause health loss. Reaching zero health ends the episode.

Direct interaction with water or a fountain restores up to 1 drink and resets thirst accumulation without consuming the source. A ripe plant restores up to 4 food. Defeating a passive creature with a direct attack restores up to 6 food; killing it with a projectile does not.
```

```
Sleeping can begin while energy is below its maximum. The player then remains asleep until
energy is full or an attack wakes it. Sleep restores energy, speeds health and mana
recovery, and slows hunger and thirst, but a sleeping player takes substantially more
damage from melee attacks.

Resting can begin while health is below its maximum. It lasts until health is full, food or
drink reaches zero, or an attack interrupts it. Resting prevents other actions but does not
accelerate recovery in the way sleep does. Mana also regenerates gradually over time
without requiring an item.

The maximum survival values depend on the player's attributes:

| Quantity | Maximum value |
|---|---:|
| Health | `8 + strength` |
| Food | `7 + 2 × dexterity` |
| Drink | `7 + 2 × dexterity` |
| Energy | `7 + 2 × dexterity` |
| Mana | `6 + 3 × intelligence` |

### Creatures and combat

Every region from the Overworld through the Ice Realm has one hostile melee creature and
one hostile ranged creature. Passive creatures also appear from the Overworld through the
Fire Realm, but not in the Ice Realm. Creatures spawn stochastically over time on the floor
the player occupies. Passive creatures wander without attacking. Melee enemies pursue a
nearby player and attack from an adjacent cardinal tile; ranged enemies generally try to
keep some distance while launching projectiles. Hostiles appear more frequently until the
current floor's eight-kill requirement has been met. Creatures sufficiently far from the
player can disappear, so the population near the player changes during exploration.

Movement depends on creature type. Cows and snails remain on land, while bats, Fire
Elementals, and Ice Elementals can fly over otherwise impassable terrain. Lizards can cross
both land and water, and Deep Things live in water. Projectiles travel in cardinal
directions until they hit a creature, the player, or an obstructing block. They can pass
over water and lava, and a hostile projectile destroys a crafting table or furnace that it
hits.

A direct interaction attacks a creature on the tile in front of the player. The physical
damage before strength scaling is determined by the current sword:

| Weapon | Base physical damage |
|---|---:|
| No sword | 1 |
| Wood sword | 2 |
| Stone sword | 3 |
| Iron sword | 5 |
| Diamond sword | 8 |

A bow shot has 5 base physical damage before dexterity scaling. Shooting consumes one arrow
and creates a projectile travelling in the direction the player faces.

Damage has physical, fire, and ice components. Armour and creature defences reduce
components separately rather than reducing all attacks by one common amount. The ordinary
creatures have the following health, attacks, and innate defences:

| Region | Passive creature | Melee enemy | Ranged enemy | Enemy defences |
|---|---|---|---|---|
| Overworld | Cow: 3 health | Zombie: 5 health; 2 physical | Skeleton: 3 health; 2 physical
| None |
| Dungeon | Snail: 6 health | Orc Soldier: 9 health; 3 physical | Orc Mage: 6 health; 3
fire | None |
| Gnomish Mines | Bat: 4 health | Gnome Warrior: 7 health; 4 physical | Gnome Archer: 5
health; 2 physical | None |
| Sewers | Snail: 6 health | Lizard: 11 health; 5 physical | Kobold: 8 health; 4 physical |
None |
| Vault | Snail: 6 health | Knight: 12 health; 6 physical | Archer: 12 health; 5 physical |
Knight and Archer: 50% physical defence |
| Troll Mines | Bat: 4 health | Troll: 20 health; 6 physical, 1 fire, 1 ice | Deep Thing: 4
health; 4 physical, 3 fire, 3 ice | Troll: 20% physical defence; Deep Thing: none |
| Fire Realm | Bat: 4 health | Pigman: 20 health; 3 physical, 5 fire | Fire Elemental: 14
health; 3 physical, 5 fire | Pigman and Fire Elemental: 90% physical defence and complete
fire defence |
| Ice Realm | None | Frost Troll: 24 health; 4 physical, 5 ice | Ice Elemental: 16 health;
4 physical, 5 ice | Frost Troll and Ice Elemental: 90% physical defence and complete ice
defence |

Fire creatures lack defence against ice damage, while ice creatures lack defence against
fire damage.

### Experience and attributes

The first entrance to each floor below the Overworld grants one experience point. Returning
to a previously visited floor does not grant another, and ordinary creature kills do not
grant experience. One point can be spent at any location to increase dexterity, strength,
or intelligence by one. Each attribute begins at 1 and has a maximum of 5.

Attributes affect different parts of play:
```

```
- Each dexterity level above 1 adds 2 to maximum food, drink, and energy; slows the
accumulation of hunger, thirst, and fatigue by 12.5% of their base rate; and increases
arrow damage by 20%.
- Each strength level above 1 adds 1 to maximum health and increases physical melee damage
by 25%.
- Each intelligence level above 1 adds 3 to maximum mana, increases fireball and iceball
damage by 50%, increases the elemental damage of an enchanted sword by 5%, and accelerates
mana recovery.

### Chests, potions, and magic

Opening a chest removes it from the world. Each ordinary chest can independently contain
several categories of random loot:

| Loot category | Chance | Possible amount |
|---|---:|---|
| Torches | 60% | 4-7 torches |
| Ore | 60% | 1-3 coal, 1-2 iron, or 1 diamond, sapphire, or ruby |
| Potion | 50% | 1-2 potions of one colour |
| Arrows | 25% | 1-4 arrows |
| Tool or weapon | 20% | One wood, stone, iron, or diamond pickaxe or sword |

Coal and iron are more common than the other ores in chest loot. A looted pickaxe or sword
upgrades the player's current item only if its tier is higher.

Some floors add a guaranteed discovery to the first chest opened there. The Dungeon's first
chest supplies a bow. The first chest opened in the Sewers and the first opened in the
Vault each supply one book.

There are six potion colours, but their meanings are shuffled whenever a new world is
created. The colours are assigned one-to-one to six effects: restore 8 health, lose 3
health, restore 8 mana, lose 3 mana, restore 8 energy, or lose 3 energy. The assignment
remains fixed within an episode but changes between episodes.

Books teach the two available spells: fireball and iceball. Reading a book consumes it and
teaches one spell not already known, selected at random when both remain available. A
fireball deals 3 base fire damage and an iceball 3 base ice damage before intelligence
scaling. Either spell costs 2 mana per successful cast.

### Enchanting

The ice-enchantment table is found in the Sewers and the fire-enchantment table in the
Vault. To enchant an item, the player must stand next to and face the appropriate table. An
ice enchantment consumes 1 sapphire and a fire enchantment consumes 1 ruby; either also
consumes 9 mana.

An enchanted sword adds elemental damage equal to half of the sword's base physical damage.
Strength scales the physical part of the attack, while intelligence scales its elemental
part. An enchanted bow similarly adds an elemental component equal to half of an arrow's
base physical damage, after which dexterity scales the shot.

Armour enchantment is recorded separately for each of the four armour slots. A
fire-enchanted slot reduces incoming fire damage by 20%, and an ice-enchanted slot reduces
incoming ice damage by 20%. The action requires the player to own at least one armour
piece, but the affected slot is selected randomly among slots that have not yet been
enchanted, including currently empty slots. An enchantment on an empty slot still provides
its elemental defence. Once all four slots have an enchantment, a new enchantment can
replace one of the opposite type.

### The Graveyard and the Necromancer

The Graveyard contains a multi-round final encounter. Its eight rounds draw on the hostile
melee and ranged creature pairs introduced across the preceding floors, beginning with
Zombies and Skeletons and ending with Frost Trolls and Ice Elementals. Enemy attacks deal
1.5 times their ordinary damage in the Graveyard.

During each round, the Necromancer remains invulnerable while enemies are spawning or any
spawned melee or ranged enemy remains alive. Once the wave is over, the Necromancer becomes
vulnerable. Directly interacting with it in that state advances the encounter and begins
the next round; ordinary weapon damage does not. Eight successful vulnerable interactions
complete the encounter and end the episode.

### Variation between episodes

The broad structure and rules above remain stable, but their concrete realisation changes
from episode to episode. Terrain generation changes routes and distances; resources,
ladders, rooms, chests, and other structures appear in different places; and creature
spawning and movement remain stochastic throughout play. Chest contents, potion-colour
meanings, the order in which books teach spells, and the armour slot selected for
enchanting also involve randomness. Consequently, the same objective can require different
navigation, resource gathering, combat, and recovery decisions in different worlds.
```

Goal-suite context.

```
The supplied goals belong to a custom goal suite and are not limited to Craftax's native
achievements. Treat each supplied `goal` description as the authoritative definition of its
success condition. The supplied `id` is only an output identifier. Infer goal meaning and
relationships from the `goal` descriptions, not from the naming or structure of the IDs.
```

Learner context.

Policy restricted to primitive actions.

```
* The learner is one shared goal-conditioned reinforcement-learning policy trained across
all goals.
* In each episode, a single selected goal defines the reward. Reward is 1 only on the
transition where that goal succeeds and 0 otherwise. Native Craftax rewards, progress
toward the selected goal, and completion of other goals or achievements provide no reward.
* Selected-goal success immediately ends the episode and resets the environment.
* Each episode is limited to 4,096 environment steps.
* The selected textual goal is provided to the policy through a frozen language encoder.
* The language encoder may provide useful semantic structure, but the policy must learn
from environment interaction how goal descriptions relate to Craftax observations,
dynamics, actions, and goal success. Do not assume that the policy can reason in language
or has prior knowledge of Craftax mechanics.
* The same trainable policy parameters and learned representations are used for every goal.
* Environment state resets between episodes. Inventory, location, health, encountered
entities, and other episode-specific state do not persist; only changes to the learned
policy persist.
* The learner has no external hierarchy or library of named skills and cannot call a
previously learned goal as a subroutine.
```

Policy with subroutine calls.

```
The learner is one shared goal-conditioned reinforcement-learning policy trained from
scratch across all goals. At the start of each episode, it is assigned a goal to achieve.
Besides choosing primitive environment actions, it can choose to call itself with a goal it
has already learned to achieve with some success. A goal becomes available for calling when
its estimated success rate over completed episodes assigned that goal reaches at least 1/3.
This allows behavior learned while training on one goal to be used while pursuing another.

When the policy calls a goal, it temporarily chooses actions using that goal instead of the
goal assigned for the episode. It uses the behavior it has learned for the called goal in
the same ongoing world, with the current inventory, location and other environment state.
After 32 further environment steps, it switches back to choosing actions for the episode's
assigned goal, unless the episode has ended sooner.

* Only the goal assigned at the start of the episode determines rewarded success. Reward is
1 on the transition where that goal succeeds and 0 otherwise. That success immediately ends
the episode and resets the environment.
* Completing any other goal provides no reward and does not end the episode or cause a call
to finish early. Native Craftax rewards and progress toward goals also provide no reward.
* Each episode is limited to 4,096 environment steps and can also end through the
environment's ordinary terminal conditions.
* Goal descriptions are provided to the policy through a frozen language encoder. The
language encoder may provide useful semantic structure, but the policy must learn from
environment interaction how goal descriptions relate to Craftax observations, dynamics,
actions and goal success. Do not assume that the policy can reason in language or has prior
knowledge of Craftax mechanics.
* Environment state resets between episodes. Inventory, location, health, encountered
entities and other episode-specific state do not persist; only changes to the learned
policy persist.
```

C EXPERIMENTAL SETUP

C.1 BENCHMARK GOAL SET

The benchmark pairs textual goal descriptions with success conditions evaluated directly from environment states or transitions. The complete set of 256 goals is presented in Table 1. For spatial objectives, "See" requires the target to be visible within the agent's local observation window, accounting for lighting, whereas "Be adjacent" requires it to occupy one of the four cardinally neighbouring tiles. Inventory thresholds specify the quantity that must be held at one time, rather than the total collected over an episode.

Table 1: Complete 256-goal benchmark. A row containing n represents one goal for each listed value; countable nouns take the singular form when $n = 1$. All other rows represent one goal. The cumulative count gives the number of goals represented up to and including each row. Goal specifications reproduce the canonical goal descriptions, except that scalar threshold values are replaced by n.

Goal specification	Values of n	Cumulative count
Have at least n wood.	$1, 2, 4, 8, 16, 32, 64$	7
Have at least n stone.	$1, 2, 4, 8, 16, 32, 64$	14
Have at least n coal.	$1, 2, 4, 8, 12, 24, 48$	21
Have at least n iron.	$1, 2, 3, 6, 12, 24$	27
Have at least n diamonds.	$1, 2, 3, 6, 12$	32
Have at least n sapphires.	$1, 2, 4, 8$	36
Have at least n rubies.	$1, 2, 4, 8$	40
Have at least n saplings.	$1, 2, 4, 8$	44
Have at least n torches.	$4, 8, 16, 32, 64$	49
Have at least n arrows.	$2, 4, 8, 16, 32$	54
Have at least n books.	$1, 2$	56
Obtain a wood pickaxe or better.	—	57
Obtain a stone pickaxe or better.	—	58
Obtain an iron pickaxe or better.	—	59
Obtain a diamond pickaxe.	—	60
Obtain a wood sword or better.	—	61
Obtain a stone sword or better.	—	62
Obtain an iron sword or better.	—	63
Obtain a diamond sword.	—	64
Obtain a bow.	—	65
Obtain one piece of iron armour or better.	—	66
Obtain one piece of diamond armour.	—	67
Obtain an iron chestplate or better.	—	68
Obtain a diamond chestplate.	—	69
Obtain iron leggings or better.	—	70
Obtain diamond leggings.	—	71
Obtain iron boots or better.	—	72
Obtain diamond boots.	—	73
Apply a fire enchantment to a sword.	—	74
Apply an ice enchantment to a sword.	—	75
Apply a fire enchantment to a bow.	—	76
Apply an ice enchantment to a bow.	—	77
Apply a fire enchantment to one piece of armour.	—	78
Apply an ice enchantment to one piece of armour.	—	79
Reach dexterity level n.	$2, 3, 4, 5$	83
Reach strength level n.	$2, 3, 4, 5$	87
Reach intelligence level n.	$2, 3, 4, 5$	91
Reach floor 1, the Dungeon.	—	92
Reach floor 2, the Gnomish Mines.	—	93
Reach floor 3, the Sewers.	—	94
Reach floor 4, the Vault.	—	95
Reach floor 5, the Troll Mines.	—	96
Reach floor 6, the Fire Realm.	—	97
Reach floor 7, the Ice Realm.	—	98
Reach floor 8, the Graveyard.	—	99
Place a crafting table.	—	100
Place a furnace.	—	101
Plant a sapling.	—	102
See coal.	—	103
See iron.	—	104
See diamond.	—	105
See lava.	—	106
See a mossy wall.	—	107
See sapphire.	—	108

Continued on next page

Table 1 continued

Goal specification	Values of n	Cumulative count
See ruby.	—	109
See a chest.	—	110
See a fountain.	—	111
See a fire enchantment table.	—	112
See an ice enchantment table.	—	113
See the downward ladder on floor 0, the Overworld.	—	114
See the downward ladder on floor 1, the Dungeon.	—	115
See the downward ladder on floor 2, the Gnomish Mines.	—	116
See the downward ladder on floor 3, the Sewers.	—	117
See the downward ladder on floor 4, the Vault.	—	118
See the downward ladder on floor 5, the Troll Mines.	—	119
See the downward ladder on floor 6, the Fire Realm.	—	120
See the downward ladder on floor 7, the Ice Realm.	—	121
See a cow.	—	122
See a zombie.	—	123
See a skeleton.	—	124
See a snail.	—	125
See an orc soldier.	—	126
See an orc mage.	—	127
See a bat.	—	128
See a gnome warrior.	—	129
See a gnome archer.	—	130
See a lizard.	—	131
See a kobold.	—	132
See a knight.	—	133
See an archer.	—	134
See a troll.	—	135
See a Deep Thing.	—	136
See a pigman.	—	137
See a fire elemental.	—	138
See a frost troll.	—	139
See an ice elemental.	—	140
See the Necromancer.	—	141
Be adjacent to water.	—	142
Be adjacent to stone.	—	143
Be adjacent to a path.	—	144
Be adjacent to sand.	—	145
Be adjacent to coal.	—	146
Be adjacent to iron.	—	147
Be adjacent to diamond.	—	148
Be adjacent to lava.	—	149
Be adjacent to a ripe plant.	—	150
Be adjacent to a torch.	—	151
Be adjacent to a wall.	—	152
Be adjacent to a mossy wall.	—	153
Be adjacent to a stalagmite.	—	154
Be adjacent to sapphire.	—	155
Be adjacent to ruby.	—	156
Be adjacent to a chest.	—	157
Be adjacent to a fountain.	—	158
Be adjacent to fire grass.	—	159
Be adjacent to a fire tree.	—	160
Be adjacent to a fire enchantment table.	—	161
Be adjacent to ice grass.	—	162
Be adjacent to an ice shrub.	—	163
Be adjacent to an ice enchantment table.	—	164
Be adjacent to a grave.	—	165
Be adjacent to the downward ladder on floor 0, the Overworld.	—	166
Be adjacent to the downward ladder on floor 1, the Dungeon.	—	167
Be adjacent to the downward ladder on floor 2, the Gnomish Mines.	—	168

Continued on next page

Table 1 continued

Goal specification	Values of n	Cumulative count
Be adjacent to the downward ladder on floor 3, the Sewers.	—	169
Be adjacent to the downward ladder on floor 4, the Vault.	—	170
Be adjacent to the downward ladder on floor 5, the Troll Mines.	—	171
Be adjacent to the downward ladder on floor 6, the Fire Realm.	—	172
Be adjacent to the downward ladder on floor 7, the Ice Realm.	—	173
Be adjacent to a cow.	—	174
Be adjacent to a zombie.	—	175
Be adjacent to a skeleton.	—	176
Be adjacent to a snail.	—	177
Be adjacent to an orc soldier.	—	178
Be adjacent to an orc mage.	—	179
Be adjacent to a bat.	—	180
Be adjacent to a gnome warrior.	—	181
Be adjacent to a gnome archer.	—	182
Be adjacent to a lizard.	—	183
Be adjacent to a kobold.	—	184
Be adjacent to a knight.	—	185
Be adjacent to an archer.	—	186
Be adjacent to a troll.	—	187
Be adjacent to a Deep Thing.	—	188
Be adjacent to a pigman.	—	189
Be adjacent to a fire elemental.	—	190
Be adjacent to a frost troll.	—	191
Be adjacent to an ice elemental.	—	192
Be adjacent to the Necromancer.	—	193
Damage a cow.	—	194
Damage a zombie.	—	195
Damage a skeleton.	—	196
Damage a snail.	—	197
Damage an orc soldier.	—	198
Damage an orc mage.	—	199
Damage a bat.	—	200
Damage a gnome warrior.	—	201
Damage a gnome archer.	—	202
Damage a lizard.	—	203
Damage a kobold.	—	204
Damage a knight.	—	205
Damage an archer.	—	206
Damage a troll.	—	207
Damage a Deep Thing.	—	208
Damage a pigman.	—	209
Damage a fire elemental.	—	210
Damage a frost troll.	—	211
Damage an ice elemental.	—	212
Defeat a cow.	—	213
Defeat a zombie.	—	214
Defeat a skeleton.	—	215
Defeat a snail.	—	216
Defeat an orc soldier.	—	217
Defeat an orc mage.	—	218
Defeat a bat.	—	219
Defeat a gnome warrior.	—	220
Defeat a gnome archer.	—	221
Defeat a lizard.	—	222
Defeat a kobold.	—	223
Defeat a knight.	—	224
Defeat an archer.	—	225
Defeat a troll.	—	226
Defeat a Deep Thing.	—	227
Defeat a pigman.	—	228

Continued on next page

Table 1 continued

Goal specification	Values of n	Cumulative count
Defeat a fire elemental.	—	229
Defeat a frost troll.	—	230
Defeat an ice elemental.	—	231
Kill at least n hostile monsters on floor 1, the Dungeon.	$1, 2, 4, 8$	235
Kill at least n hostile monsters on floor 2, the Gnomish Mines.	$1, 2, 4, 8$	239
Kill at least n hostile monsters on floor 3, the Sewers.	$1, 2, 4, 8$	243
Kill at least n hostile monsters on floor 4, the Vault.	$1, 2, 4, 8$	247
Kill at least n hostile monsters on floor 5, the Troll Mines.	$2, 4, 8$	250
Kill at least n hostile monsters on floor 6, the Fire Realm.	$2, 8$	252
Kill at least n hostile monsters on floor 7, the Ice Realm.	$2, 8$	254
Kill at least n hostile monsters on floor 8, the Graveyard.	$2, 8$	256
	Total	256

The success evaluator can sometimes fail to register a lethal projectile or spell attack on a cow, bat, or snail. For the corresponding damage and defeat goals, the agent may therefore need to repeat the interaction with another creature of the same type before success is registered.

C.2 Learner architecture

The observation encoder processes the spatial and non-spatial components of the symbolic Craftax observation separately. For each spatial cell, learned embeddings represent blocks, items, creatures and projectiles, and visibility. These embeddings are concatenated and projected to 32 features, then processed by two 3×3 convolutional layers with 32 channels each. The non-spatial features pass through a fully connected layer with 64 units. The flattened spatial features and encoded non-spatial features are concatenated and projected to a 512-dimensional observation embedding. GELU activations follow each fully connected and convolutional layer in the observation encoder.

Each goal description is encoded offline using a frozen Qwen3-Embedding-4B model, with embeddings truncated to 512 dimensions and normalized to unit length. These embeddings remain fixed throughout training. Both learners use a Gated Transformer-XL backbone consisting of two blocks with a hidden dimension of 256, four attention heads, and a memory length of 64 steps. Gate biases are initialized to 2. During training, backpropagation is truncated to subsequences of 64 steps, with preceding cached memory states treated as constants. The actor and critic use separate MLP heads, each with two hidden layers of 512 units and ReLU activations.

For the policy restricted to primitive actions, the observation and goal embeddings are concatenated and passed through a fully connected layer with 512 units and a GELU activation, followed by a linear projection to the Transformer hidden dimension. The same goal embedding is also concatenated with the Transformer output before each actor and critic head. The actor assigns probabilities to the 43 primitive actions. The complete actor–critic network contains approximately 6.17 million trainable parameters, excluding the frozen text encoder.

For the policy with subroutine calls, the observation embedding is projected directly to the Transformer hidden dimension, without goal conditioning. The actor receives the Transformer output concatenated with the embedding of the goal assigned to the episode or, during a subroutine, the currently invoked goal. The critic additionally receives the embedding of the goal assigned to the episode and the remaining delegation time, expressed as a fraction of the maximum delegation duration. The actor assigns probabilities to the 43 primitive actions and 256 call actions, one per benchmark goal. The complete actor–critic network contains approximately 6.04 million trainable parameters, excluding the frozen text encoder.

C.3 Training protocol

At the start of each curriculum iteration, all environments are reset. Each environment then runs for 4,096 steps with a fixed assigned goal. Episodes are limited to 4,096 steps. After every 256 steps per environment, we perform one epoch of PPO optimization over eight minibatches.

We use PPO with a clipping parameter of 0.2 and the same threshold for value-function clipping. Advantages are computed using generalized advantage estimation (GAE) with $\gamma = 0.999$ and $\lambda = 0.97$. For the policy restricted to primitive actions, advantages are normalized to zero mean and unit variance across the complete rollout batch. The value-loss coefficient is 0.5, and the entropy coefficient is 0.005. The changes to the actor loss, advantage normalization, and regularization for the policy with subroutine calls are described in Appendix C.4.

We optimize the network using Adam with a constant learning rate of 2×10^{-4} and $\epsilon = 10^{-5}$. Gradients are clipped to a maximum global norm of 1.

C.4 Policy with subroutine calls

The curriculum assigns each episode a goal $g \in \mathcal{T}$. The agent receives a reward of one upon achieving g and zero otherwise, with success ending the episode. This policy variant augments the primitive Craftax action set $\mathcal{A}_{\text{prim}}$ with one call action per benchmark goal, giving $\mathcal{A}_{\text{combined}} = \mathcal{A}_{\text{prim}} \cup \{\text{call}(g') : g' \in \mathcal{T}\}$. The actor assigns probabilities over this combined action space, so invoking a goal is selected in the same way as a primitive action. These calls provide an explicit mechanism for reusing behavior learned while training on other goals.

Selecting a primitive action executes it directly in the environment. Selecting $\text{call}(g')$ executes one no-op environment step, after which the policy switches its conditioning to g'. It returns to conditioning on g after 32 further environment steps. During these 32 steps, the policy can select either primitive actions or further call actions. Each call consumes a no-op step and replaces the goal used for the remaining steps, without extending the original deadline. Execution then returns directly to g. A complete call block therefore occupies 33 environment steps, regardless of calls made within it. All steps count toward the episode and training budgets. The same network generates actions throughout. Environment state and Transformer memory persist across changes in goal conditioning. Achieving a goal other than g provides no reward and does not cause an early return. Episode termination ends any ongoing subroutine.

At the start of curriculum iteration i, calls are restricted to goals in $\mathcal{E}_i = \{g' \in \mathcal{T} : \widehat{P}_i(g') \geq 1/3\}$ to avoid spending time invoking goals on which the policy has demonstrated little competence. This set remains fixed throughout the iteration and its PPO updates. Primitive actions are always available, and the curriculum remains free to select any goal for training.

The PPO actor loss uses only decisions made outside subroutines, including actions that initiate calls, and actor advantages are normalized over these decisions. GAE and critic learning use all environment steps, with returns defined by the episode's assigned goal g. Rewards obtained during a subroutine can therefore inform the update to the decision that invoked it, while actions taken within the subroutine receive no direct actor loss.

Assigning equal probability to every available action would make subroutine calls more frequent as more goals become eligible. It would also fail to account for the fact that a complete call occupies 33 environment steps, whereas a primitive action occupies one. We therefore define a reference distribution μ_i that, when calls are available, assigns total probability $33/34$ to primitive actions and $1/34$ to eligible calls, distributed uniformly within each group. Assuming calls run to completion, this choice gives equal expected environment time to direct primitive actions and call blocks. When no goals are eligible, μ_i is uniform over primitive actions.

The actor learns preferences relative to this reference distribution. Given its output logit $\ell_t(a)$ for action a, action probabilities are $\pi_t(a) \propto \mu_i(a) \exp(\ell_t(a))$. Equal logits recover the reference distribution, while learned preferences can change the allocation between primitive actions and calls. The same construction is used both outside and during subroutines.

The ordinary entropy bonus favors uniform probabilities over available actions. To make regularization consistent with the chosen reference distribution, we replace this bonus with a term equivalent to penalizing $D_{\text{KL}}(\pi_t \| \mu_i)$. This penalty is averaged over decisions made outside subroutines and uses the same coefficient as the entropy bonus specified in Appendix C.3. When no goals are eligible, this recovers ordinary entropy regularization.

C.5 EVALUATION PROCEDURE

For the reported results, policies are evaluated every 15 curriculum decisions in Section 5.3.1 and every 30 in Section 5.3.2. At each evaluation point, each goal's success rate is estimated from 128 separate evaluation episodes for each training seed, with actions sampled from the current policy. Each comparison uses five training seeds paired across methods. Aggregate performance curves show the mean across seeds, with shaded regions indicating one standard deviation.

For the policy with subroutine calls, the set of callable goals is determined from the latest training performance estimates and remains fixed throughout evaluation. Evaluation outcomes do not update these estimates.

Evaluating every benchmark goal as a separate single-target objective was computationally prohibitive. After the uniform-weight experiments for the policy restricted to primitive actions, we selected candidate targets spanning qualitatively different objectives and judged to be challenging but plausibly learnable within the available training budget. For each candidate, we ran TARGET, ALP, ALP+D, ALP+F, and ALP+D+F with a shared training seed and retained targets for which at least one method achieved nonzero success at the final evaluation. This yielded the Dungeon, Skeleton, Wood, and Furnace targets; "See an orc soldier" and "Obtain a stone sword or better" were excluded under this criterion. These initial runs were included in the reported five-seed comparisons.

C.6 HYPERPARAMETER SELECTION

Most hyperparameters were fixed without tuning. All hyperparameter tuning was conducted using the ALP baseline on the complete 256-goal benchmark, with uniform task weights and the policy restricted to primitive actions. The selected values were then shared across curriculum methods and learner variants wherever applicable. Selection proceeded through successive comparisons against a reference configuration, usually changing one setting at a time. Most candidates were evaluated on one development seed, with selected comparisons repeated on a second seed. Runs were generally allocated approximately two billion environment steps, and all development seeds were distinct from those used in the main experiments. Tuning used a smaller variant of the final network; the architecture used in the main experiments was evaluated separately. Table 2 reports the candidate and selected values.

Table 2: Numerical hyperparameters considered during tuning. Values were evaluated through successive comparisons rather than a joint grid search.

Hyperparameter	Values evaluated	Selected
Learning rate	3×10^{-5}, 5×10^{-5}, 10^{-4}, 2×10^{-4}	2×10^{-4}
Entropy coefficient	0.005, 0.01	0.005
Discount factor γ	0.995, 0.999, 1	0.999
GAE parameter λ	0.95, 0.97, 0.99	0.97
PPO minibatches	4, 8	8
Rollout length per environment	256, 512	256
Transformer memory length	64, 128	64
Learning-progress exponent ρ	1/2, 2/3, 1	1/2
Retained observations for LP estimation N_{obs}	5, 7	7
Uniform-sampling weight in ALP	0.2, 0.3	0.2

Most of the evaluated hyperparameter changes had a modest effect on performance. Changes with more substantial negative effects included increasing the entropy coefficient to 0.01, extending Transformer memory to 128 steps, reducing the learning rate to 3×10^{-5}, and setting the discount factor to $\gamma = 1$. For the discount factor, $\gamma = 0.999$ yielded late-training success rates close to those obtained with $\gamma = 0.995$. We preferred $\gamma = 0.999$ because it discounts delayed goal achievement less heavily, better aligning training with the undiscounted goal-success objective.

C.7 SENSITIVITY ANALYSIS

We examine the sensitivity of ALP+D+F to the maximum path length L used to compute downstream potential and the downstream coefficient α. We consider uniform task weights and settings with all objective weight assigned to either Dungeon or Skeleton. Using the policy restricted to primitive actions, we vary one parameter at a time around the main configuration, keeping all other settings as in the corresponding main experiments. We conducted this analysis after fixing the hyperparameters used in the main experiments; its results did not inform their selection. All configurations and the ALP reference use the same training seeds within each setting: three under uniform weights and five per target. Evaluations follow the protocol in Appendix C.5. Table 3 reports success at selected training budgets, and Figure 4 shows the learning curves.

The effects of the two parameters depend on the setting and training budget. Under uniform task weights, performance varies little across the tested configurations; on Skeleton, differences mainly concern learning speed, with similar final mean success. Dungeon shows stronger sensitivity: $L = 1$ produces higher mean success early in training but subsequently levels off below $L = 2$ and $L = 3$, while both alternative values of α yield lower final mean success than the default. These Dungeon comparisons also exhibit substantial variation across seeds. Despite this sensitivity, all tested configurations have higher mean success than ALP at the first reported budget on Skeleton and at the final budget on Dungeon.

Table 3: Sensitivity of ALP+D+F to maximum path length L and downstream coefficient α. Entries are evaluation success percentages (mean $\pm$ one sample standard deviation across seeds), averaged over all tasks under uniform weights and measured on the target otherwise. Columns show one-third, two-thirds, and the full budget of $3{,}019{,}898{,}880$ steps under uniform weights or $1{,}509{,}949{,}440$ steps for the targets.

Uniform task weights (3 seeds)					
	L	α	$\approx$ 1.0B	$\approx$ 2.0B	$\approx$ 3.0B
Default	2	1	12.40 ± 0.49	13.10 ± 0.19	13.08 ± 0.46
	1	1	12.01 ± 0.33	12.72 ± 0.12	13.27 ± 0.21
	3	1	12.10 ± 0.45	12.78 ± 0.17	13.34 ± 0.06
	2	3	12.23 ± 0.07	13.13 ± 0.12	13.25 ± 0.11
	2	10	12.66 ± 0.29	12.75 ± 0.18	13.09 ± 0.13
Dungeon (5 seeds)					
	L	α	$\approx$ 0.5B	$\approx$ 1.0B	$\approx$ 1.5B
Default	2	1	1.88 ± 4.19	11.41 ± 5.02	43.75 ± 26.53
	1	1	11.88 ± 3.24	15.62 ± 1.99	15.94 ± 4.64
	3	1	0.62 ± 1.40	12.81 ± 7.40	53.75 ± 27.33
	2	0.3	0.00 ± 0.00	8.28 ± 12.30	26.25 ± 24.67
	2	3	1.09 ± 2.45	10.94 ± 3.54	25.16 ± 13.16
Skeleton (5 seeds)					
	L	α	$\approx$ 0.5B	$\approx$ 1.0B	$\approx$ 1.5B
Default	2	1	56.41 ± 7.15	64.06 ± 1.99	67.19 ± 2.65
	1	1	61.41 ± 4.23	64.53 ± 4.54	67.66 ± 3.15
	3	1	42.81 ± 24.76	63.44 ± 2.30	67.50 ± 4.71
	2	0.3	51.09 ± 28.97	66.09 ± 5.08	69.38 ± 4.54
	2	3	60.78 ± 3.38	65.78 ± 4.11	70.94 ± 8.58

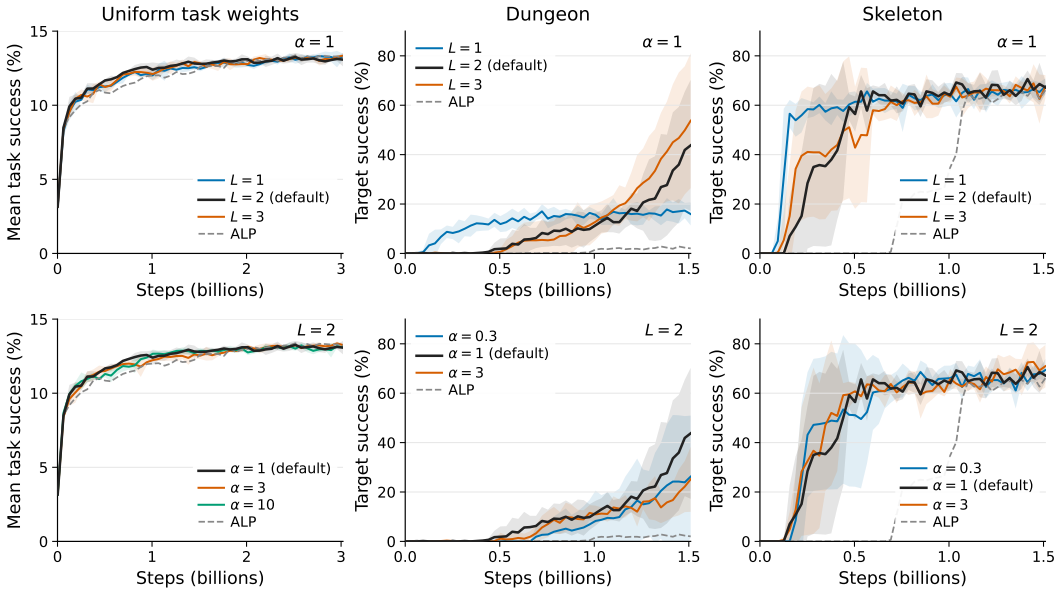

Figure 4: Sensitivity of ALP+D+F to L (top, $\alpha = 1$) and α (bottom, $L = 2$). Curves show mean evaluation success; shading indicates one sample standard deviation across three seeds under uniform weights and five seeds for each target task.

D BASELINES AND ABLATIONS

Within each comparison, all methods use the same learner and training protocol. Performance and learning-progress estimates are computed as described in Appendix A.2. Adaptive curricula use uniform sampling for the first 16 curriculum iterations. In the single-target experiments, each adaptive distribution governs the 60 task selections remaining after the four fixed target selections, as described in Section 5.

For compactness, we define the sampling distribution induced by a nonnegative task score S as

$$\mathcal{N}[S](m) = \begin{cases} \dfrac{S(m)}{\sum_{m' \in \mathcal{T}} S(m')}, & \displaystyle\sum_{m' \in \mathcal{T}} S(m') > 0, \\ q^{\mathrm{U}}(m), & \text{otherwise}, \end{cases}$$

where $q^{\mathrm{U}}(m) = 1/|\mathcal{T}|$ is uniform over the task set.

D.1 BASELINES

Uniform sampling (UNIF). UNIF samples uniformly over the task set, with $q_i = q^{\mathrm{U}}$.

Absolute learning progress (ALP). ALP is representative of conventional curricula guided by estimated absolute learning progress (Matiisen et al., 2019). It also serves as an ablation of our method with both downstream potential and feasibility-guided sampling removed. Its score and sampling distribution are

$$S_i^{\mathrm{ALP}}(m) = \left|\widehat{\mathrm{LP}}_i(m)\right|^{\rho} w(m),$$
$$q_i = 0.8\,\mathcal{N}[S_i^{\mathrm{ALP}}] + 0.2\,q^{\mathrm{U}},$$

with $\rho = \frac{1}{2}$, as in all LP-based variants.

Intermediate difficulty (I-DIFF). I-DIFF represents curricula that prioritize tasks of intermediate difficulty relative to the learner's current competence. We use success-rate thresholds of 0.1 and 0.9, matching those used by Florensa et al. (2018). The score and sampling distribution are

$$S_i^{\text{I-DIFF}}(m) = \mathbf{1}\Big[0.1 \le \widehat{P}_i(m) \le 0.9\Big]\, w(m),$$
$$q_i = 0.8\,\mathcal{N}[S_i^{\text{I-DIFF}}] + 0.2\,q^{\mathrm{U}}.$$

Direct target training (TARGET). In the single-target experiments, TARGET assigns the entire training budget to the designated target from the first curriculum iteration. It provides a reference for assessing the benefit of training on auxiliary tasks.

D.2 ABLATIONS

Removing feasibility-guided sampling (ALP+D). ALP+D retains the full method's LP-based score and omits the feasibility-guided component:

$$S_i^{\mathrm{ALP+D}}(m) = \left|\widehat{\mathrm{LP}}_i(m)\right|^{\rho} \left[\widetilde{w}_i^{\mathrm{LP}}(m) + \alpha\,\widetilde{D}_i^{\mathrm{LP}}(m)\right],$$
$$q_i = 0.8\,\mathcal{N}[S_i^{\mathrm{ALP+D}}] + 0.2\,q^{\mathrm{U}}.$$

We use $\alpha = 1$ and the normalization defined in Appendix A.3.

Removing downstream potential (ALP+F). ALP+F removes downstream potential from both scores by setting $\alpha = 0$, while retaining feasibility-guided sampling:

$$S_i^{f,0}(m) = \mathbf{1}[\widehat{P}_i(m) \le \tau_f]\, f_i(m)\, w(m),$$
$$q_i = 0.8\,\mathcal{N}[S_i^{\mathrm{ALP}}] + 0.1\,\mathcal{N}[S_i^{f,0}] + 0.1\,q^{\mathrm{U}},$$

with $\tau_f = 0.05$.

Permuting the learning-connectivity graph (PERM). PERM follows ALP+D+F but replaces the learning-connectivity weights $C(s,t)$ with $C(\sigma(s),\sigma(t))$, where σ is a random permutation of $\mathcal{T}$, sampled once per run and held fixed throughout training. Applying the same permutation to both endpoints preserves the graph's topology and edge weights while randomizing its correspondence to the goals. Task weights and the feasibility-evidence graph retain their original goal assignments; all other components and hyperparameters are unchanged.

E Additional results and analyses

E.1 Curriculum dynamics

Illustrative trajectories across methods. Figures 5–7 extend the Wood example in Figure 2 to Dungeon, Skeleton, and Furnace, using each method's run with the highest final target success.

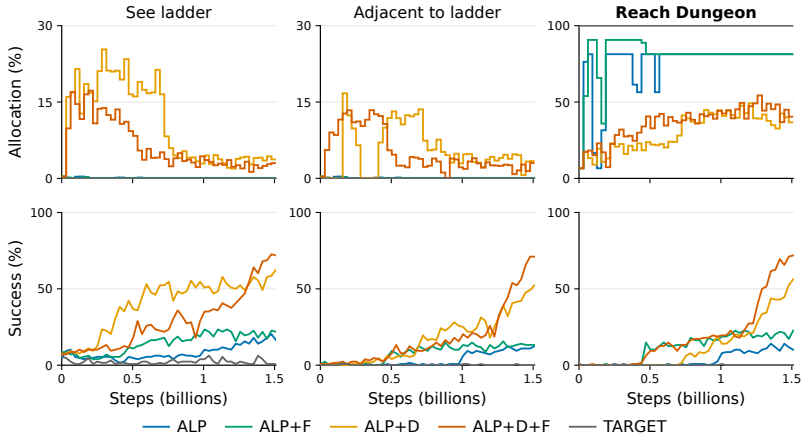

Figure 5: Illustrative curriculum dynamics with all objective weight assigned to reaching the Dungeon. The top row shows the probability mass allocated to the target and selected related auxiliary goals, averaged over non-overlapping blocks of 15 curriculum decisions. The bottom row shows the policy's success rate when evaluated on each goal. For each method, we show the run with the highest final target success among the five seeds in Figure 1.

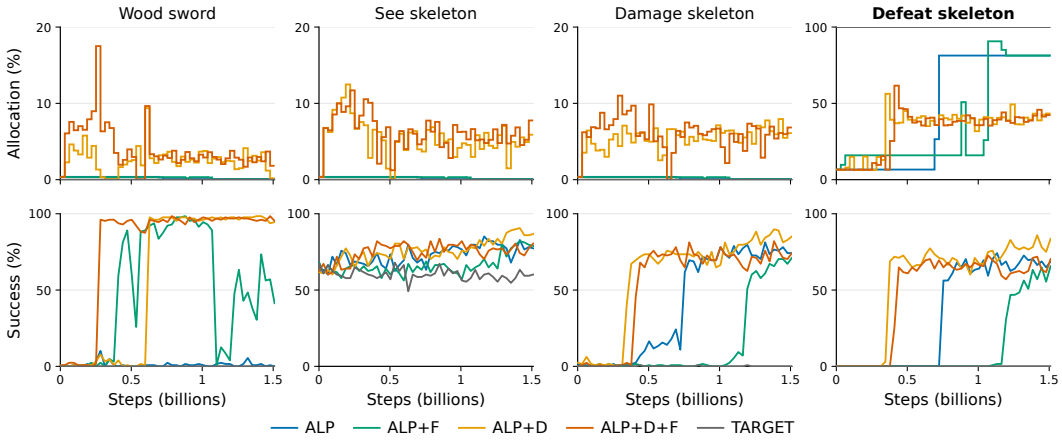

Figure 6: Illustrative curriculum dynamics with all objective weight assigned to defeating a skeleton. The top row shows the probability mass allocated to the target and selected related auxiliary goals, averaged over non-overlapping blocks of 15 curriculum decisions. The bottom row shows the policy's success rate when evaluated on each goal. For each method, we show the run with the highest final target success among the five seeds in Figure 1.

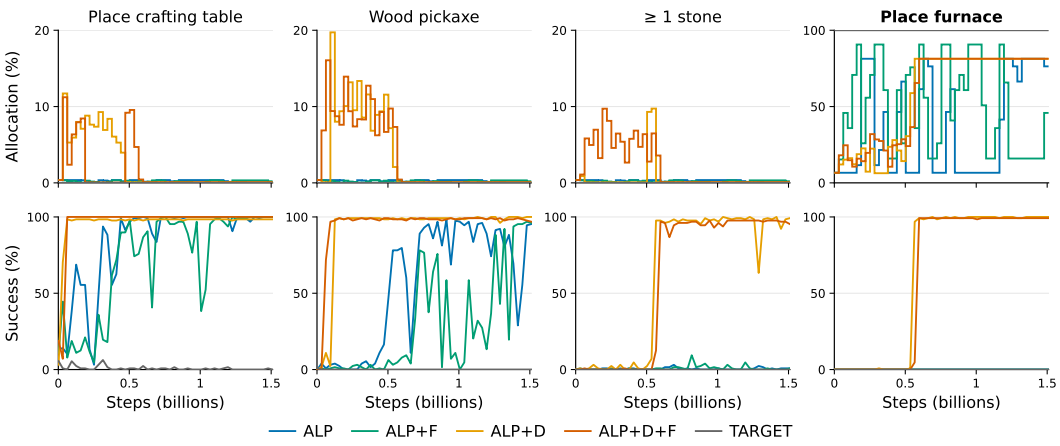

Figure 7: Illustrative curriculum dynamics with all objective weight assigned to placing a furnace. The top row shows the probability mass allocated to the target and selected related auxiliary goals, averaged over non-overlapping blocks of 15 curriculum decisions. The bottom row shows the policy's success rate when evaluated on each goal. For each method, we show the run with the highest final target success among the five seeds in Figure 1.

ALP+D+F across training seeds. Figures 8–11 show individual ALP+D+F trajectories for all five training seeds on each target.

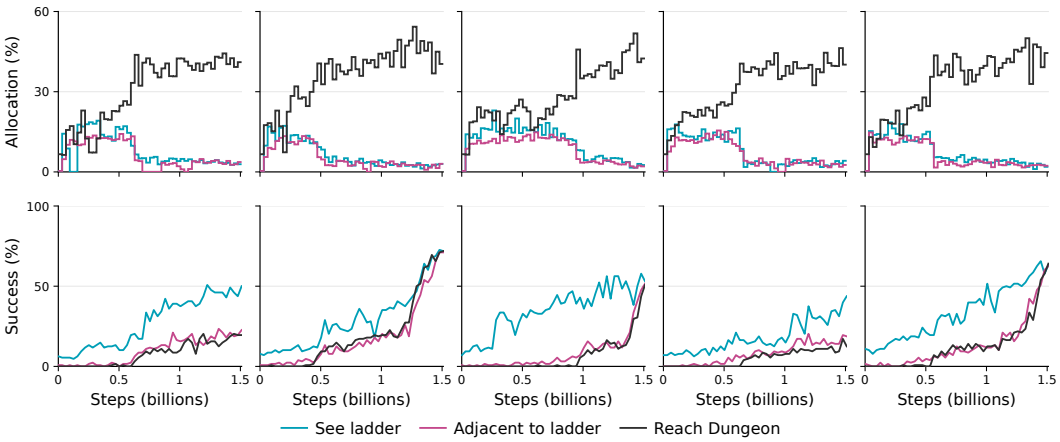

Figure 8: Curriculum dynamics of ALP+D+F with all objective weight assigned to reaching the Dungeon. Each column shows one of the five training seeds in Figure 1. The top row shows allocation probability averaged over non-overlapping blocks of 15 curriculum decisions; the bottom row shows evaluation success.

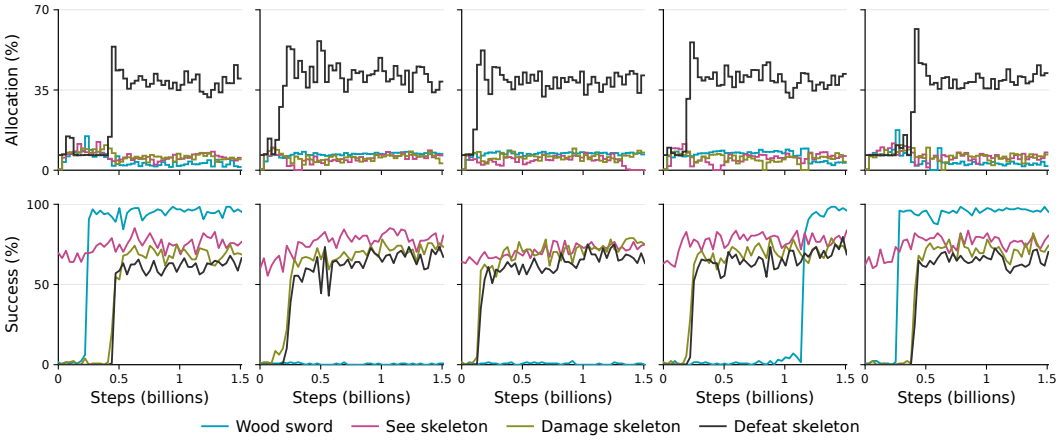

Figure 9: Curriculum dynamics of ALP+D+F with all objective weight assigned to defeating a skeleton. Each column shows one of the five training seeds in Figure 1. The top row shows allocation probability averaged over non-overlapping blocks of 15 curriculum decisions; the bottom row shows evaluation success.

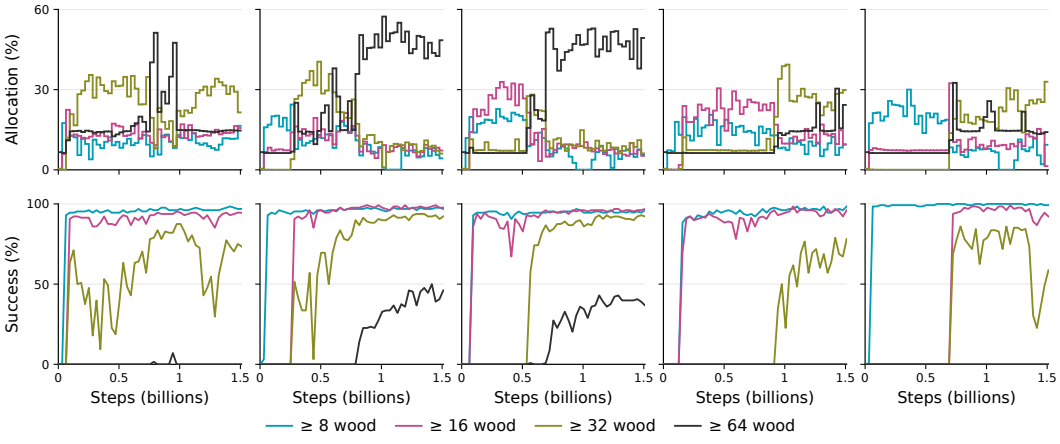

Figure 10: Curriculum dynamics of ALP+D+F with all objective weight assigned to having at least 64 wood. Each column shows one of the five training seeds in Figure 1. The top row shows allocation probability averaged over non-overlapping blocks of 15 curriculum decisions; the bottom row shows evaluation success.

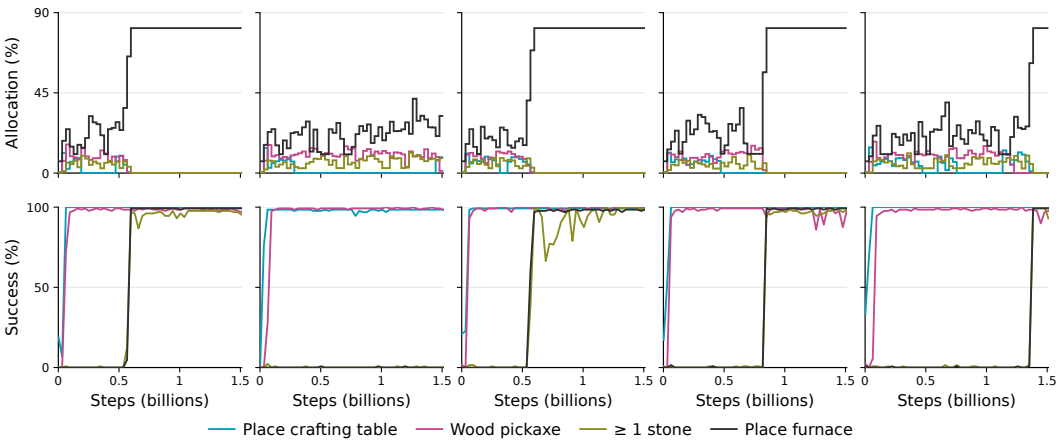

Figure 11: Curriculum dynamics of ALP+D+F with all objective weight assigned to placing a furnace. Each column shows one of the five training seeds in Figure 1. The top row shows allocation probability averaged over non-overlapping blocks of 15 curriculum decisions; the bottom row shows evaluation success.

E.2 Smoothing the target-centered objectives

With all objective weight assigned to the target, ALP cannot prioritize auxiliary tasks according to their learning progress. To test whether the benefits of ALP+D+F discussed in Section 5.3.1 persist when this restriction is removed, we evaluate ALP with 50% of the total objective weight assigned to the target and the remaining 50% distributed uniformly across the other tasks (Figure 12). Reweighting accelerates skeleton learning under ALP, but learning remains slower than under ALP+D+F. Moreover, ALP's Dungeon performance declines, and it remains unsuccessful on Wood and Furnace. Applying the same reweighting to ALP+D+F has its most pronounced effects on Wood and Furnace, where success rates are substantially reduced within the tested budget.

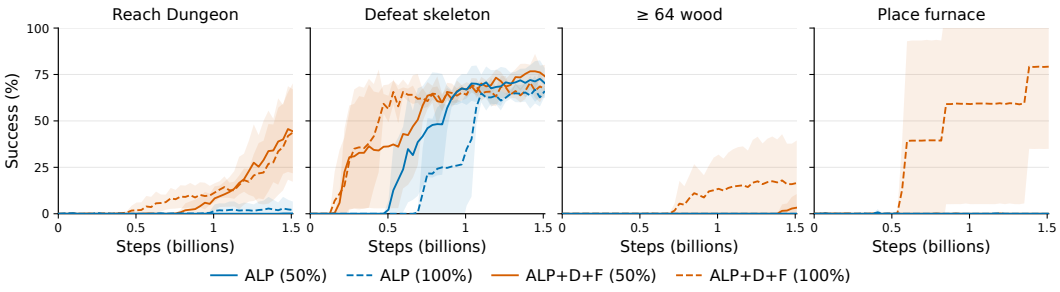

Figure 12: Evaluation success on selected target tasks during training, with 50% (solid) or 100% (dashed) of the objective weight assigned to the corresponding target. Lines show means across five seeds; shaded regions indicate one standard deviation.

E.3 Discussion of downstream potential

Downstream potential aggregation. The strongest-path construction is a heuristic for aggregating LLM judgments into a measure of potential downstream usefulness. It combines contributions from distinct downstream tasks according to their importance, retaining only the strongest discounted path to each task to avoid counting alternative routes separately. Multiplying edge weights and applying the path discount attenuates weakly supported connections and longer chains. These choices provide a simple aggregation rule, without assigning the resulting path scores a calibrated interpretation as transfer probabilities or performance gains. A single learning-connectivity edge can represent potential for either immediate transfer or more productive subsequent training on another task. Consequently, $L = 1$ already allows effects on future learning to influence downstream po-

tential. Increasing L extends the aggregation to possible effects through intermediate tasks; it limits the number of composed relationships without specifying how many subsequent training decisions their realization may require. The aggregation does not explicitly model the intermediate learning required to realize the opportunities suggested by these paths.