HERAKLES: HIERARCHICAL SKILL COMPILATION FOR LLM AGENTS IN COMPOSITIONAL GOAL SPACES

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents fine-tuned with reinforcement learning exploit common-sense knowledge to act in interactive environments, but typically decide every elementary action. In compositional goal spaces, where reaching a goal requires reaching others first, this forces credit assignment over horizons that grow with goal depth. This in turn leads to considerable decrease of sample efficiency as goals get more complex. We posit that the compositionality of behaviours follows that of their linguistic descriptions, so that an LLM is best used to reason over goals, while low-level routines run without language. Existing hierarchies built on this division rely on skills supplied by humans, or grow them without direction. We introduce HERAKLES, in which an LLM planner and a lightweight controller acting from raw observations are trained concurrently by RL, without any pre-trained skill library. The goal space is given, but its prerequisite structure is not: the planner selects intermediate goals among those the controller is estimated to master, and trajectories reaching a goal are compiled into the controller, turning mastered goals into options for harder ones and amortizing past learning. On Crafter, HERAKLES keeps improving where flat and hierarchical baselines plateau, and degrades markedly less on compositional variants of known goals.

1 INTRODUCTION

Large language models (LLMs) encode broad commonsense knowledge about how the world works, and a growing body of work turns them into decision-making agents by fine-tuning them with reinforcement learning in interactive environments: text-based and embodied navigation (Carta et al., 2023; Szot et al., 2023), household and cooperative tasks (Tan et al., 2024), device control (Bai et al., 2024), or mathematical reasoning (Shao et al., 2024). Grounding a pretrained LLM through online RL has been shown to be far more sample-efficient than learning from scratch, as its prior already indicates which actions are sensible in a given situation (Carta et al., 2023).

These agents, however, typically act at the level of elementary actions: every step of a trajectory is a decision taken by the LLM. This becomes a bottleneck in environments whose goals are not independent, where reaching one goal requires having reached others first. In a crafting environment, for instance, mining stone requires a pickaxe, which requires wood and a crafting table, which itself requires wood. In such compositional spaces, the number of elementary actions separating an agent from a goal grows with the goal's depth, and with it the exploration burden and the variance of the learning signal. An LLM choosing each of these actions must assign credit over the whole horizon, pays a forward pass per step, and reasons in language about motor details where language offers little leverage. Yet the structure that makes these tasks hard is precisely the one language describes well. We build on the hypothesis that the compositionality of the behaviours to adopt follows that of their linguistic descriptions, so that an LLM is most useful when reasoning over goals rather than over elementary actions.

This suggests a division of labour that mirrors human skill acquisition, where deliberate, cognitively demanding composition progressively gives way to automatic execution (Fitts & Posner, 1967; Tsay et al., 2024): once mastered, a sequence is re-encoded at a lower level, driven by low-level signals rather than explicit reasoning, and invoked as a single unit. Only the planner needs language; the routines it calls do not. Hierarchies pairing an LLM planner with low-level controllers exist, but their subgoals mostly come from human supervision, either as a predefined library of pre-trained

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

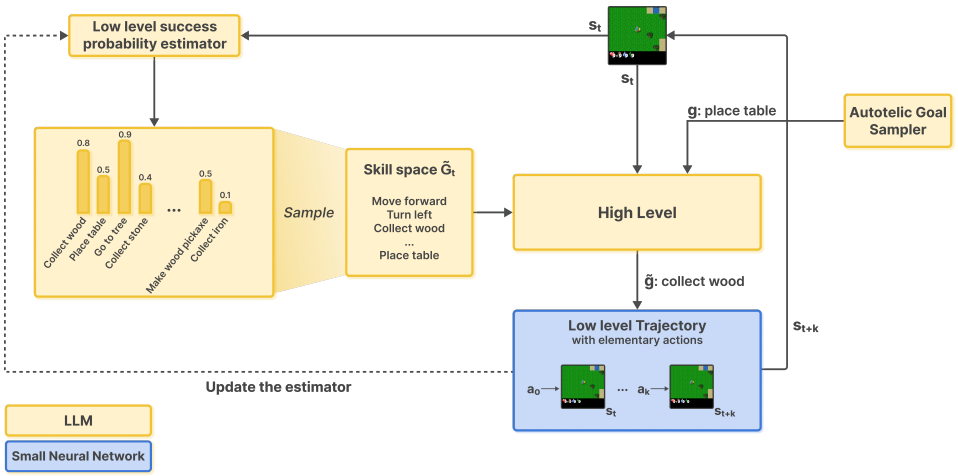


Figure 1: **HERAKLES.** An LLM-based high-level policy selects subgoals among those the low-level policy is estimated to reliably achieve; the low-level policy executes them with elementary actions. Once a goal is mastered, its full trajectory is compiled into the low-level policy as a single skill, which the high-level policy can then invoke directly.

skills (Ahn et al., 2022) or as subgoals annotated by humans solving the tasks (Ahuja et al., 2023). BOSS (Zhang et al., 2023b) lets the library grow instead, by chaining skills proposed by a frozen LLM and relabelling the resulting trajectories as new skills. This growth remains undirected, however: chains are assembled bottom-up and named after the fact, rather than decomposed from a goal the agent is trying to reach, and the planner never learns from their outcomes.

We introduce HERAKLES (**HiERA**rchical s**K**ill compi**LE**r**S**), a hierarchical agent in which an LLM planner and a lightweight controller acting from raw observations are trained concurrently by RL, without any pre-trained skill library. The goal space G is given, but the prerequisite relations structuring it are not: the agent must learn which goals help reach which. To pursue a goal, the planner selects intermediate goals from G among those the controller currently masters, so that every subgoal is itself a verifiable goal of the space. Whenever the hierarchical agent reaches its target, the whole trajectory is relabelled with that target and compiled into the controller, which thereby learns to reach it directly; the goal then becomes a single option for pursuing harder ones. The skill repertoire thus grows from the goals the agent pursues, rather than being assembled bottom-up. Each compiled goal shortens the horizon over which the planner must assign credit when pursuing harder ones, so that past learning is amortized rather than repeated; this is the source of the method's sample efficiency. Which goals the planner may invoke is decided at every step by an online estimate of the controller's competence from the current observation, so that the planner is neither confined to elementary actions nor exposed to subgoals the controller cannot yet execute. The only prior over how goals relate is the commonsense encoded in the pretrained LLM.

Our contributions are:

- A hierarchical framework in which high- and low-level policies are learned jointly by RL, without an expert-defined skill set, and in which mastered goals are continuously compiled into the low-level policy and become callable options for the high-level policy.

- A formulation of this coupling as a bilevel, Stackelberg-like problem, and an adaptive subgoal filtering rule that keeps the high-level action space aligned with the low-level's current competence.

- An empirical study in Crafter (Hafner, 2022), whose goals are organized along an achievement tree that makes compositional depth explicit. For a given interaction budget, HERAKLES reaches deeper goals: it keeps improving where flat and hierarchical baselines plateau after the easy ones, and generalizes to compositional, synonymous, and trajectory-similar goals unseen during training.

2 Related Work

LLM agents trained with RL. Fine-tuning pretrained LLMs with online RL grounds them in interactive environments (Tan et al., 2024; Szot et al., 2023), and is markedly more sample-efficient than learning task-specific behaviours from scratch (Carta et al., 2023). In these agents, the LLM selects every elementary action, so credit must be assigned over full trajectories and across the tokens composing each action. POAD (Wen et al., 2024) decomposes action-level credit over tokens; ArCHer (Zhou et al., 2024) pairs an utterance-level critic with token-level updates, a hierarchy over the granularity of credit assignment rather than between a planner and a controller. Herakles instead lets the LLM reason over goals, and delegates their execution to a controller acting on raw observations. We use POAD both to train the high-level policy of Herakles and, restricted to elementary actions, as our flat baseline.

Hierarchical RL. Temporal abstraction has long been used to shorten the horizon of credit assignment Dayan & Hinton (1992); Sutton et al. (1999); Precup & Sutton (2000); Bacon et al. (2017). Deep HRL methods such as h-DQN Kulkarni et al. (2016), FuN Vezhnevets et al. (2017) and HIRO Nachum et al. (2018b) express subgoals as points in a state or learned latent space, whose quality strongly conditions performance Nachum et al. (2018a); such subgoals carry no description the planner could relate to other goals or reuse. Language provides such descriptions Luketina et al. (2019); Jiang et al. (2019); Hu et al. (2019), and LLMs have been used as planners over language-conditioned skills. These skills are, however, supplied beforehand: a library of pre-trained skills under a frozen LLM Ahn et al. (2022), controllers trained on annotated demonstrations Shi et al. (2025); Intelligence et al. (2025), or subgoals annotated by humans solving the tasks Ahuja et al. (2023). Voyager Wang et al. (2023) grows a skill library, but its skills are programs written by a frozen LLM over a high-level API rather than learned controllers. Closest to our approach, BOSS Zhang et al. (2023b) grows a library pre-trained on demonstrations by chaining skills proposed by a frozen LLM and relabelling the resulting trajectories as new skills. Herakles differs in three respects: its planner is trained by RL; skills are compiled top-down, from trajectories reaching a goal the agent was pursuing, rather than assembled bottom-up and named afterwards; and the options offered to the planner are filtered at every decision by the controller's estimated competence. Finally, the coupling between levels can be cast as a bilevel, Stackelberg-like problem Fiez et al. (2019); Shen et al. (2024); Chakraborty et al. (2024); our training exploits the corresponding timescale separation between the LLM planner and the lightweight controller.

Goal generation and curricula. Open-ended learning is commonly formalized as the continual production of novel and learnable goals Sigaud et al. (2024); Edward et al. (2024), by autotelic agents that generate and select their own Colas et al. (2022). LLMs have been used to select or propose goals Du et al. (2023); Zhang et al. (2023a), to create goals together with their reward functions Wang et al. (2023); Faldor et al. (2025); Pourcel et al. (2024), and to prioritize, within a given goal space, those with the highest expected learning progress Gaven et al. (2025), which we use as our goal sampler. The goal spaces such systems produce typically exhibit the precedence structure Herakles exploits, as new goals are built from previously reached ones Pourcel et al. (2024); Wang et al. (2023). These approaches decide which goals to pursue; Herakles addresses the complementary question of how to keep them learnable as their compositional depth grows, by amortizing past learning through skill compilation.

3 The Herakles Method

The Herakles mechanism is based on a hierarchical agent composed of a high-level policy that directs a low-level policy, as depicted in Figure 1. The latter aims to gradually internalize the behavior of the hierarchical agent. In this way, the hierarchical agent learns to solve a goal g by relying on the low-level policy to achieve a set of sub-goals required for g. Once g is considered mastered by the low-level policy, the high-level policy gains access to a new option—namely, invoking the low-level policy to execute g. This facilitates the achievement of more complex goals by reducing the corresponding signal-to-noise ratio during training Park et al. (2023). Ultimately, this yields a global policy where the high-level component acts as a planner, focused on understanding situations and reasoning about composing skills to achieve goals, while the low-level component serves as a skill executor.

3.1 PROBLEM STATEMENT

Let $\mathcal{M} = (S, A, \mathcal{T}, G, r, \Omega, O, \gamma)$ be a goal-augmented Partially Observable Markov Decision Process, with S the state space, A the primitive action space, $\mathcal{T}$ the transition function, G a pre-generated goal space with an associated sparse reward function $r : S \times A \times G \to \{0; 1\}$, Ω the observation space, $O : S \to \Omega$ the observation function that maps states to observations, and $\gamma \in [0, 1)$ the discount factor. Given an initial state distribution P_{s_0} over $\mathcal{S}$ and a goal distribution $P_{\mathcal{G}}$ over $\mathcal{G}$, we seek a goal-conditioned policy $\pi : \Omega \times \mathcal{G} \to \Delta(\mathcal{A})$ maximizing the expected discounted return $\pi^* = \arg\max_\pi \mathbb{E}_{s_0 \sim P_{s_0},\, g \sim P_{\mathcal{G}}} \mathbb{E}_{\tau \sim \pi(\cdot|s_0,g)}, \big[R_g(\tau)\big]$, where $\pi(\tau \mid s_0, g)$ denotes the probability of trajectory τ under π conditioned on goal g and starting state s_0, and the goal-conditioned return is defined as $R_g(\tau) = \sum_{t=0}^{|\tau|} \gamma^t \, r_t^g$, with $r_t^g = r(s_t, a_t, g)$. Depending on the setting, $P_{\mathcal{G}}$ may either be static or evolve over time according to an exogenous process reflecting the distribution of goals encountered during open-ended exploration. In order to guide the agent and extract semantic relationships between goals, we also consider that the agent has access to a prompting function $\phi : \Omega \times G \to \mathcal{V}^{L_p}$ that transforms any pair (observation, goal) into a textual prompt of L_p tokens from a given vocabulary $\mathcal{V}$.

Algorithm 1: HERAKLES (Sketched)

Initialise empty buffers $\mathcal{B}_{\text{HL}}$, $\mathcal{B}^{\text{h}}_{\text{LL}}$, $\mathcal{B}^{\text{sg}}_{\text{LL}}$;
Set policies π_{HL} and π_{LL};
Competency estimator C_θ;
for NbIterations **do**
  $o \leftarrow env.reset()$; Sample $g \sim P_G$;
  Set $\tau_g \leftarrow \emptyset$; $t \leftarrow 0$; $k \leftarrow 0$; $t_k \leftarrow 0$;
  while $(k < N_{HL}) \wedge !done_g$ **do**
    Construct $\tilde{G}$ using C_θ (Section 3.4)
    Sample $\tilde{g} \sim \pi_{\text{HL}}^{\tilde{G}(o)}(\tilde{g}|\phi(o, g))$;
    $t_k \leftarrow t$; $k \leftarrow k + 1$; $h \leftarrow 1$;
    while $(t - t_k < N_{LL}) \wedge !done_{\tilde{g}}$ **do**
      $a \leftarrow \pi_{\text{LL}}(\tilde{g}, o)$;
      $o_{next}, r^g, r^{\tilde{g}} \leftarrow env.step(a)$
      $\tau_g \leftarrow \tau_g \oplus (o, h, \tilde{g}, a, r^g, r^{\tilde{g}})$;
      $o \leftarrow o_{next}$; $t \leftarrow t + 1$; $h \leftarrow 0$;
    end while
  end while
  $\tau_g \leftarrow \tau_g \oplus o$;
  Insert (τ_g, g) in buffers (Section 3.3);
  Update π^{LL} on $\mathcal{B}^{\text{h}}_{\text{LL}} \cup \mathcal{B}^{\text{sg}}_{\text{LL}}$;
  Update C_{θ_k} on $\mathcal{B}^{\text{sg}}_{\text{LL}}$;
  Update π^{HL} on $\mathcal{B}_{\text{HL}}$;
end for

Let the goal space G be endowed with a precedence relation $\prec$, such that for some pairs of goals $(g', g) \in G^2$, $g' \prec g$ indicates that achieving g' is a prerequisite for the completion of g. We assume that this relation defines a hidden hierarchical structure over G, whereby complex goals can be decomposed into prerequisite subgoals. Accordingly, for any pair (g', g) satisfying $g' \prec g$, any trajectory $\tau_g = \{s_0, a_0, s_1, a_1, \ldots, s_n\}$ that achieves g can be decomposed into a prefix trajectory $\tau_{g'}$ that achieves g' at some step $k \in (0, n)$, followed by a suffix $\tau_{g' \to g} = \{s_k, a_k, \ldots, s_n\}$ leading from g' to g. To anchor $\prec$, we define a minimal set of atomic goals by associating, with each primitive action $a \in \mathcal{A}$, an action-matching goal $g_a \in \mathcal{G}$ such that $R(s, a, g_a) = 1$ for all $s \in \mathcal{S}$. This is a mild requirement: it amounts to naming each primitive action as a goal, and guarantees that the minimal elements of $\prec$ are reachable in a single step. In the following, we exploit this latent structure by assuming a correspondence between linguistic compositionality, goals expressed as natural-language prompts through ϕ, and skill compositionality (similarly to Ahn et al. (2022); Ahuja et al. (2023)), allowing the agent to discover and reuse prerequisite chains wherever they exist in $\mathcal{G}$.

3.2 HIERARCHICAL COMPILATION

To exploit the compositional structure of G, we consider a hierarchical agent $\pi = (\pi_{\text{HL}}, \pi_{\text{LL}})$ operating at two levels of temporal abstraction. The low-level policy $\pi_{\text{LL}} : \Omega \times G \to \Delta(A)$ is a goal-conditioned controller mapping observations and goals to primitive actions. The high-level policy $\pi_{\text{HL}} : \phi(\Omega, G) \to \Delta(G)$, implemented as a language model, selects subgoals based on textual prompts. At execution time, π_{HL} proposes subgoals that are executed by π_{LL} until termination, inducing a Semi-Markov Decision Process (SMDP). A subgoal terminates when it is achieved or after N_{LL} steps, while the high-level policy selects at most N_{HL} subgoals.

A trajectory for a goal g is defined as: $\tau_g = (s_0, h_0, g_0, a_0, r_0^g, r_0^{g_0}, \ldots, s_T)$, where s_t is the state, $h_t \in \{0, 1\}$ indicates high-level decision steps ($h_t = 1$ triggers a query of π_{HL}), $g_t \in G$ is the subgoal executed by the low-level policy, and a_t is the primitive action. All trajectories respect $h_0 = 1$, as we impose the agent starting with a high-level call. Each step contains two rewards: r_t^g is the global (high-level) reward, while $r_t^{g_t}$ is the reward associated with the current subgoal g_t (used for off-policy training of π_{LL}). Let $I(\tau_g) = \{t_k\}_{k=0}^{K-1}$ be the ordered set of steps where $h_t = 1$. The

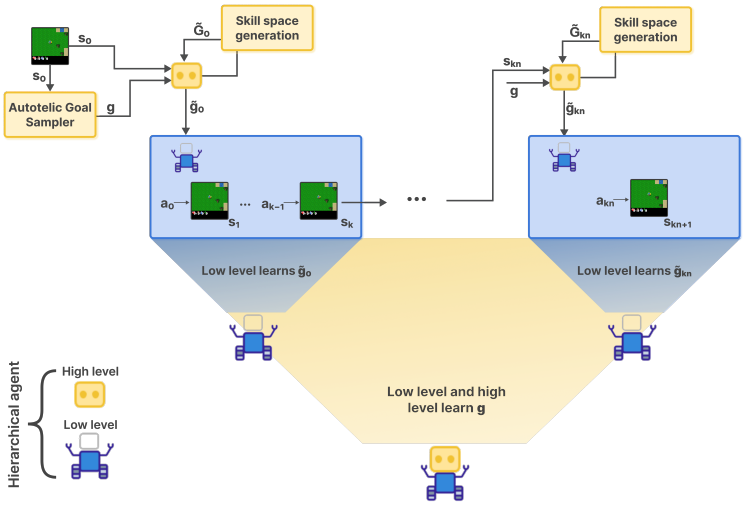


Figure 2: **Skill learning and compilation in HERAKLES**: Given a goal g and a state s_0, π_{HL} generates its set of options $\tilde{G}$ and samples one option $\tilde{g}_0$, where an option corresponds to a goal from G that is considered as mastered by the LL policy. This option is executed by π_{LL} using primitive actions in the environment, leading to state s_k. Given s_k and g, π_{HL} then generates a new set of options, samples a new option g_k and passes it to π_{LL}. This process repeats until g has been achieved or the limited number of high-level steps has been reached. π_{HL} is trained on the high-level trajectory consisting in the selection of options. π_{LL} is trained on all the low-level trajectories (one for each option) and on the **compiled trajectory** that is the concatenation of all the low-level trajectories to reach the goal g passed to the hierarchical agent. Doing so, π_{LL} progressively internalizes the sequence of primitive actions required to directly solve g.

projected trajectory over high-level decision steps is thus: $P_g(\tau_g) = \left((s_{t_k}, g_{t_k}, R_k^g)\right)_{k=0}^{K-1} \oplus (s_T)$, with $\oplus$ the concatenation operator and $R_k^g = \sum_{t=t_k}^{t_{k+1}-1} \gamma^{t-t_k} r_t^g$. Let us define $\gamma_{\text{HL}} = \gamma^{c_{\text{HL}}}$ as the high-level discount factor, with c_{HL} a cost for high-level decisions. Given a low level policy π_{LL}, the high-level (SMDP) objective is:

$$\pi_{\text{HL}}^* = \arg\max_{\pi_{\text{HL}}} \mathbb{E}_{(s_0,g)} \mathbb{E}_{\tau_g \sim (\pi_{\text{HL}}, \pi_{\text{LL}})} \left[\sum_k \gamma_{\text{HL}}^k R_k^g \right]. \tag{1}$$

Bilevel structure of the optimal policy Unlike standard hierarchical RL, where subgoals are transient and defined by the high-level policy, our setting considers a shared goal space G in which both task goals and intermediate subgoals reside, enabling reusable skill learning and explicit compositionality (as we assume a precedence relation between goals $\prec$ exists). Concretely, π_{LL} stands as a universal goal-conditioned policy expected to directly solve a large subset of goals from states reached by the hierarchical policy. Consequently, this objective does not define a single-level policy search, but induces a bilevel coupling between π_{HL} and π_{LL} through the state-subgoal distribution induced by $\pi = (\pi_{\text{HL}}, \pi_{\text{LL}})$, defined as $d_h^\pi(s, g') = \mathbb{E}_{g \sim P_G, s_0 \sim P_{s_0}} \left[P^\pi(s_t = s \mid h_t = 1, s_0, g)\, \pi_{\text{HL}}(g' \mid \phi(s, g)) \right]$. In particular, given π_{HL}, the optimal π_{LL} is given as:

$$\pi_{\text{LL}}^* = \arg\max_{\pi_{\text{LL}}} \mathbb{E}_{(s,g) \sim d_h^{(\pi_{\text{HL}}, \pi_{\text{LL}})}} \mathbb{E}_{\tau \sim \pi_{\text{LL}}(\cdot|s,g)} \left[\sum_t \gamma^t r_t^g \right]. \tag{2}$$

Coupled optimality We can interpret the joint optimum as a fixed point of coupled best-response operators:

$$\pi^* = (\pi_{\text{HL}}^*, \pi_{\text{LL}}^*) \quad \text{such that} \quad \begin{cases} \pi_{\text{HL}}^* \text{ maximizes equation 1 given } \pi_{\text{LL}}^* \\ \pi_{\text{LL}}^* \text{ maximizes equation 2 given } \pi_{\text{HL}}^*. \end{cases}$$

This Stackelberg-like coupling Fiez et al. (2019); Shen et al. (2024); Chakraborty et al. (2024) reflects that π_{HL} shapes the distribution of states and subgoals seen by π_{LL}, while low-level execution determines the optimality of high-level decomposition. Beyond this fixed-point view, π_{HL} guides

exploration through subgoal decomposition and composes low-level skills when needed, yielding a hybrid policy: in the limit of full compilation, simple goals reduce to a single high-level decision, while complex ones fall back to hierarchical sequencing. The parameter c_{HL} controls this trade-off—penalizing high-level decisions encourages temporally extended behaviors in π_{LL} and promotes skill compilation, while still allowing hierarchical reasoning. Importantly, π_{HL} remains part of the final policy, acting as a residual composer for goals that π_{LL} cannot directly solve.

3.3 RL TRAINING OF THE HL AND LL POLICIES

Given the coupled optimality defined in previous section, π_{HL} and π_{LL} are learned simultaneously (see Figure 2), following the sketched Algorithm 1. The full training procedure is given in Appendix C.1.

In the following, let $\mathcal{D}$ denote a dataset of recent trajectories paired with their task goals. From $\mathcal{D}$, we construct three replay buffers corresponding to different levels of abstraction. The high-level buffer contains projected trajectories at the high-level decision scale: $\mathcal{B}_{HL} := \{(P_g(\tau_g), g) \mid (\tau_g, g) \in \mathcal{D}\}$. The low-level buffer of hierarchical rollouts contains full (flattened) trajectories labelled by the task goal: $\mathcal{B}^h_{LL} := \{((s_t, a_t, r^g_t)_{t=0}^{T-1} \oplus s_T, g) \mid (\tau_g, g) \in \mathcal{D}\}$. Finally, the subgoal-conditioned buffer contains segments between high-level decisions: $\mathcal{B}^{sg}_{LL} := \{((s_t, a_t, r_t^{g_{t_k}})_{t=t_k}^{t_{k+1}-1} \oplus s_{t_{k+1}}, g_{t_k}) \mid (\tau_g, g) \in \mathcal{D},\ t_k \in I(\tau_g)\}$.

High-level policy training The high-level policy π_{HL} is implemented as a pre-trained language model equipped with a LoRA adapter Hu et al. (2022). This adapter is fine-tuned using on-policy RL over trajectories sampled from $\mathcal{B}_{HL}$.

Low-level policy training The low-level policy π_{LL} is a lightweight neural network operating on raw observations. It is trained off-policy using both $\mathcal{B}^h_{LL}$ and $\mathcal{B}^{sg}_{LL}$, which serve complementary roles.

Type 1 — Hierarchical trajectories. Samples from $\mathcal{B}^h_{LL}$ correspond to full rollouts generated by the hierarchical policy. These trajectories provide supervision at the task level, allowing π_{LL} to *compile* successful hierarchical behaviors into direct goal-conditioned policies using only primitive actions.

Type 2 — Subgoal trajectories. Samples from $\mathcal{B}^{sg}_{LL}$ correspond to segments collected during the execution of subgoals. These trajectories provide denser and more localized feedback, enabling π_{LL} to reinforce previously acquired skills and maintain performance across the goal space, while training it on states induced by high-level decisions, i.e., samples from the distribution d^{π}_h induced by the current policy.

Together, both training signals allow π_{LL} to acquire new skills through behavioral compilation and to stabilize existing ones through repeated subgoal execution. As the hierarchical agent successfully solves a goal g, the corresponding behaviors are progressively compiled into π_{LL}. Consequently, π_{LL} becomes able to directly reach g from relevant states, enabling π_{HL} to reuse it as a primitive for more complex goals. As training progresses, a growing fraction of execution is delegated to π_{LL}, reducing reliance on high-level decisions. This improves efficiency and facilitates exploration for π_{HL}, which operates over temporally extended and more predictable transitions.

3.4 ADAPTIVE SUBGOAL FILTERING FOR THE HIGH-LEVEL POLICY

The general principle of our approach is to progressively acquire new skills that can be compiled into π_{LL}. Once reliably executable, these skills form higher-level actions that π_{HL} can invoke to more efficiently achieve its goals.

Rather than allowing π_{HL} to consider all subgoals in G, which is inefficient when only a subset is achievable by π_{LL}, we introduce a pre-filtering step, as depicted in Figure 1 which illustrates the hierarchical decision process. This filtering step restricts the set of admissible subgoals to $\tilde{G}(o) \subseteq G$, containing only goals that the low-level policy is expected to reliably execute from the current state, given its observation o. Given this restricted set, π_{HL} is implemented as an LLM agent that selects subgoals from $\tilde{G}(o)$ as options to trigger. Following Carta et al. (2023), the probability of sampling a subgoal $\tilde{g} \in \tilde{G}(o)$, with textual description $\phi(\tilde{g})$, given a goal g and observation o, is defined as: $\pi_{\mathrm{HL}}(\tilde{g} \mid \phi(o, g)) \propto P_{LLM}(\phi(\tilde{g}) \mid \phi(o, g))$, where $P_{LLM}(\cdot \mid \phi(o, g))$ denotes the probability

induced by the LLM under constrained decoding, restricted to sequences corresponding to valid goal descriptions in $\tilde{G}(o)$. For a given goal g and observation o, we denote as $\pi_{\text{HL}}^{\tilde{G}(o)}(.|\phi(o,g))$ our HL policy restricted to subgoals from $\tilde{G}(o)$.

Constructing $\tilde{G}_k$(.) A crucial aspect of our method is how to define, at any state reached by the high-level controller, the set of callable options compatible with the *current* capabilities of π_{LL}. Constructing $\tilde{G}(o)$ requires balancing two competing objectives: (i) avoiding exposure of π_{HL} to options not yet mastered by π_{LL}, which would destabilize learning, and (ii) introducing new options to π_{HL} as early as possible so that it can rapidly adapt to them. Let $\tilde{G}_k(o)$ denote the set $\tilde{G}(o)$ at iteration k of the learning process. We can bootstrap $\tilde{G}_0(\cdot)$ as the set containing only the primitive actions, which by definition π_{LL} always masters, by using the fact that G contains at least one goal g_a for each primitive action $a \in A$.

The set of available actions for π_{HL} is built at any step k of the learning process based on a competency estimator C_{θ_k}, that we learn from a buffer of low-level trajectories $\mathcal{B}_{\text{LL}}^{\text{sg}}$ (see section 3.3) by minimizing the binary cross-entropy loss:

$$\mathcal{L}(\theta_k) = \mathbb{E}_{(\tau,g)\in\mathcal{B}_{\text{LL}}^{\text{sg}}} \mathbb{E}_{o\in\tau} \left[BCE(I(R_g(\tau) > 0), C_{\theta_k}(o,g)) \right],$$

with $\mathcal{B}_{\text{LL}}^{\text{sg}}$ containing recent pairs (τ, g), with τ any sub-trajectory sampled from the distribution $\pi_{\text{LL}}(.|g)$ of trajectories following the LL policy conditioned on goal g (i.e., starting from an HL call to g). $I(R_g(\tau) > 0)$ is the indicator function that returns 1 if goal g is achieved in τ, 0 otherwise. Similarly to Gaven et al. (2025), $C_{\theta_k}(o, g)$ is defined as an MLP on top of the representation produced by an LLM for the last token of the prompt $\phi(o, g)$. θ_k stands as parameters of this MLP and those of a specific LORA Hu et al. (2022) adapter of a base LLM.

Given an observation o_t, $C_{\theta_k}(o_t, g)$ predicts the probability that π_{LL} successfully achieves g from o_t. At any decision step t where $h_t = 1$, each goal $g \in G$ is independently included in $\tilde{G}_k(o_t)$ via a Bernoulli draw with probability: $p_g = \max\left(C_{\theta_k}(o_t, g), \epsilon\right)$,where ϵ is an exploration hyper-parameter. This construction directly addresses both objectives: $C_{\theta_k}(o_t, g)$ keeps the inclusion probability low for goals π_{LL} has not yet mastered from o_t, satisfying (i); meanwhile, the floor ϵ guaranties that every goal retains a minimal selection probability, ensuring that new or under-practiced goals are still occasionally proposed to π_{HL}, satisfying (ii). The exploration parameter ϵ avoids a self-reinforcing loop in which unmastered goals receive low scores, are never practiced, and consequently prevent both π_{LL} and C_{θ_k} from improving on them.

4 Experiments

4.1 Experimental setup

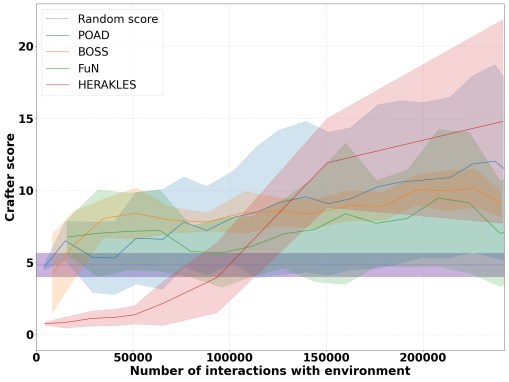


Figure 3: Crafter score as a function of the number of steps in the environment. Shaded area represent standard deviation using 5 seeds.

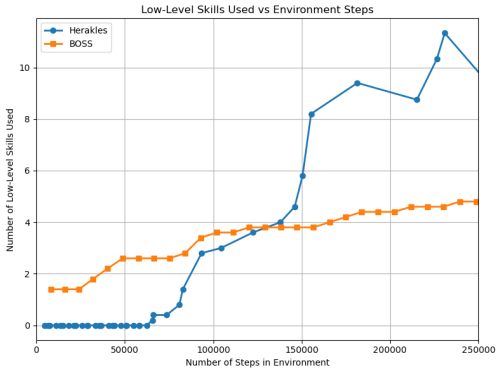


Figure 4: Evolution of the average number (over 5 seed) of low level skills that π_{HL} can call depending on the number of steps.

Environment description We evaluate our method in Crafter Hafner (2022) using the version proposed by Du et al. (2023) (more details in Appendix A.2). It is a procedurally generated and

partially observable Minecraft-like 2D environment that enables collecting and creating a set of artifacts organized along an achievement tree of 17 goals (see Appendix A.1). Most of the goals require the agent to reuse discovered or generated artifacts from previous goals. For instance, to "*make a wood pickaxe*", the agent needs to have built a crafting table. While the Crafter goal space is finite, it captures a locally open-ended setting by presenting an expanding set of learnable goals through environment interactions. This property enables us to study the scaffolding dynamics of hierarchical agents—specifically, how the high-level policy mobilizes and recombines an increasingly rich repertoire of low-level skills to achieve progressively complex goals. Consequently, the Crafter goal space is an excellent candidate for testing our method's ability to identify and exploit the underlying goal structures within a goal space. Such structures are found in the goal spaces generated by open-ended agentsEdward et al. (2024)Wang et al. (2023).For our experiments, we equip the environment with a prompting function ϕ, that generates textual description of the scene. Thus, an observation $o \in \Omega$ is a tuple made from visual and textual observations $o = (o_v, o_{text})$ (see Appendix A.3 for more details). π_{HL} receives the linguistic description o_{text} and π_{LL} uses the visual observation o_v.

Goal Sampler In Crafter, the agent is encouraged to master as many goals as possible. Goal difficulty is not fixed but conditioned on the current environment state s_k. For instance, *'place table'* is substantially easier if the agent already carries wood. To select goals that maximize hierarchical policy competence across the large goal space G, we draw inspiration from Learning Progress (LP) methods, which prioritize goals expected to yield the greatest future improvement (Oudeyer et al., 2007). Specifically, for all approaches compared in the experiments, we use the MAGELLAN algorithm Gaven et al. (2025) to define P_G. This goal sampling strategy, based on LLM models to predict and generalize competences across the goal space, simulates the evolution of the goal distribution that open-ended agents' policies may encounter within an open-ended exploration setting. Implementation details are provided in Appendix G.

Training HL and LL policies We model π_{HL} using Mistral 7B and train it with the POAD on-policy RL algorithm Wen et al. (2024). For π_{LL}, we adapt the ResNet-based architecture given by Moon et al. (2023) ($2M$ parameters). We train π_{LL} using the AWR off-policy algorithm Peng et al. (2019) and fill the buffer with both types of trajectories described in Section 3.3. We approximate the timescale separation required for convergence of the Stackelberg-like coupling by training the lightweight π_{LL} faster than the LLM-based π_{HL}. Appendices D and E give more details on π_{HL}, π_{LL}, and their training.

Baselines To measure the impact of HRL in HERAKLES, we compare with POAD Wen et al. (2024), which corresponds to the high-level policy of HERAKLES restricted to elementary actions A (see Appendix H.2). To evaluate the benefit of a language-structured skill space, we compare with FUN Vezhnevets et al. (2017), a feudal HRL method where the high-level policy generates subgoals in a latent embedding space. To match HERAKLES's automatic curriculum, we make FUN's high-level policy goal-conditioned by appending a goal embedding (produced by the same LLM as HERAKLES) to the current observation (see the Appendix H.1). Finally, we compare against BOSS Zhang et al. (2023b), which also uses an LLM as a frozen high-level policy to compose skills hierarchically, but relies on a pre-trained skill library initialized from demonstrations and chains skills bottom-up via trajectory relabelling, in contrast to HERAKLES's continuous skill compilation. Training details and adaptations are provided in Appendix H.3. In Appendix B, we present further experiments and more details about the effects of our methods.

4.2 SAMPLE EFFICIENCY

We train our agents for $250{,}000$ interaction steps in Crafter and report the Crafter score (Hafner, 2022): $S_c = \exp\left(\frac{1}{N}\sum_{i=1}^{N} \ln(1 + sr_i)\right) - 1$, where $sr_i \in [0; 100]$ is the success rate on goal i and $N = 17$. The geometric mean emphasizes rare and difficult achievements, yielding a difficulty-aware metric without prior knowledge of goal complexity. Success rates are estimated on 40 held-out environments at regular intervals.

Figure 3 shows that baselines initially progress faster than HERAKLES but plateau after mastering easy goals (e.g. *go to tree*, *place table*), making only marginal progress on harder ones. In contrast, HERAKLES improves gradually as more goals are compiled into π_{LL} (Figure 4), enabling the

hierarchical policy to reach complex goals and steadily increase its score. BOSS quickly masters a few easy low-level skills but stagnates, failing to discover more complex ones through its exploration process: upon manual inspection, the most complex skill reached by one of its seeds is *place table*, against *make stone pickaxe* for HERAKLES. Further analyses are reported in Appendix B.1.

4.3 GENERALIZATION TO NEW GOALS

Table 1: Success rates on unseen goals. Goals with a star (*) were seen during training. We give the average results and the standard deviation over 5 seeds. More results are given in Appendix B.2.

Generalization Type	Goal	HERAKLES	FuN	BOSS	POAD
Compositional	collect wood*	$\mathbf{0.92 \pm 0.05}$	0.73 ± 0.18	0.88 ± 0.07	0.80 ± 0.39
	collect 2 woods	$\mathbf{0.92 \pm 0.05}$	0.35 ± 0.15	0.72 ± 0.07	0.41 ± 0.48
	collect 4 woods	$\mathbf{0.91 \pm 0.06}$	0.08 ± 0.07	0.55 ± 0.08	0.07 ± 0.27
Synonyms	collect wood*	$\mathbf{0.92 \pm 0.05}$	0.73 ± 0.18	0.88 ± 0.07	0.80 ± 0.39
	acquire wood	$\mathbf{0.92 \pm 0.05}$	0.80 ± 0.19	0.87 ± 0.02	0.79 ± 0.02
	make wood pickaxe*	$\mathbf{0.47 \pm 0.27}$	0.11 ± 0.06	0.37 ± 0.08	0.75 ± 0.07
	create wood pickaxe	$\mathbf{0.44 \pm 0.39}$	0.16 ± 0.20	0.39 ± 0.07	0.71 ± 0.11
Similar trajectory in skill space	make wood pickaxe*	$\mathbf{0.47 \pm 0.27}$	0.11 ± 0.06	0.37 ± 0.08	0.75 ± 0.07
	make wood sword	$\mathbf{0.40 \pm 0.37}$	0.00 ± 0.00	0.33 ± 0.05	0.00 ± 0.05

As HERAKLES uses an LLM as a high-level policy, we examine its ability to generalize to three types of new goals:

• **Compositional generalization**: goals repeating a known goal k times (*e.g.*, *'collect k woods'*).

• **Synonymous goals**: goals in which verbs are replaced by synonyms.

• **Trajectory-similar goals**: we evaluate on the unseen goal *'make wood sword'* which shares the same prerequisite as the training goal *'make wood pickaxe'*, both require crafting a table and holding wood, differing only in the final elementary action (*craft* `wood sword` vs. *craft* `wood pickaxe`).

Table 1 reports success rates on unseen goals after $250{,}000$ training steps; pretrained goals are marked with a star (*). All language-based methods generalize to varying degrees, but HERAKLES suffers the least: it shows a negligible drop from *'collect wood'* to *'collect* 4 *woods'*, unlike the baselines. The 'Synonyms' row shows all methods handle synonymous verbs, yet baselines fail to reuse skills for non-synonymous goals sharing a similar trajectory. In contrast, HERAKLES leverages the linguistic structure of its skill space, retaining strong performance ($0.47 \rightarrow 0.40$). Additional generalisation results are in Appendix B.2.

5 CONCLUSION

We introduced HERAKLES, a hierarchical framework for LLM agents in compositional goal spaces, in which an LLM-based high-level policy and a lightweight low-level policy are trained concurrently. As the agent masters increasingly complex goals, the corresponding behaviours are compiled into the low-level policy as reusable skills for the high-level policy. This dynamic skill expansion enables effective operation in evolving goal spaces without expert-defined skills or pre-training. In the Crafter environment, HERAKLES scales with goal complexity, improves sample efficiency over prior HRL and LLM-only approaches, and generalizes to compositional, synonymous, and trajectory-similar goals. Several limitations remain. The high-level policy operates on textual goals only; extending it to visual goals via Vision-Language Models (Liu et al., 2024) would broaden its applicability. The low-level policy's periodic retraining on previously learned skills also incurs computational overhead we would like to reduce. Finally, our setting assumes an externally provided goal space; integrating HERAKLES with goal-generation (Faldor et al., 2025; Pourcel et al., 2024) and skill-discovery methods is a promising direction toward fully autonomous open-ended agents.

AI USE STATEMENT

In this work, we used generative AI tools only for language editing and polishing of the manuscript. All scientific content, analyses, and conclusions are the authors' own. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects, personal data, or newly collected datasets. All experiments are conducted in Crafter, a procedurally generated simulated environment, using publicly available models (Mistral 7B) and benchmarks. The main foreseeable concerns are indirect. First, the high-level policy inherits the biases and knowledge gaps of its pretrained LLM; in our setting these only affect which subgoals are proposed within a fixed, verifiable goal space, but deploying similar agents in open-world settings would require auditing these priors. Second, methods that let agents acquire skills autonomously raise oversight questions as they scale; in HERAKLES, the planner's decisions are sequences of subgoals expressed in natural language, which makes its behaviour directly inspectable, a property we see as favourable to monitoring.

REPRODUCIBILITY STATEMENT

To help reproducibility, we release our code in the supplementary material of the paper. The global architecture is described in Appendix C.2, the training procedure is given in Algorithm 2 (Appendix C.1), and the architectures and training of the high-level policy, low-level policy, and competence estimator are detailed in Appendices D, E, and F, including the exact subgoal inclusion probability (Appendix F.2), the low-level update schedule (Appendix E.2), and all prompts (Appendices D.2 and F.3). Our modifications to Crafter, its achievement tree, and the observation space are described in Appendix A; our adaptation of MAGELLAN in Appendix G; and the implementation of all baselines, including the changes required to fit our setting, in Appendix H. Hyper-parameters and compute resources are listed in Appendices I and J. All results are averaged over 5 seeds and evaluated on 40 held-out environments.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

Arun Ahuja, Kavya Kopparapu, Rob Fergus, and Ishita Dasgupta. Hierarchical reinforcement learning with natural language subgoals, 2023. URL https://arxiv.org/abs/2309.11564.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. doi: 10.1609/aaai.v31i1.10916. URL https://ojs.aaai.org/index.php/AAAI/article/view/10916.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In Andreas

Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 3676–3713. PMLR, 2023. URL http://dblp.uni-trier.de/db/conf/icml/icml2023.html#CartaRWLSO23.

Souradip Chakraborty, Amrit Singh Bedi, Alec Koppel, Huazheng Wang, Dinesh Manocha, Mengdi Wang, and Furong Huang. PARL: A unified framework for policy alignment in reinforcement learning from human feedback. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ByR3NdDSZB.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Cédric Colas, Tristan Karch, Clément Moulin-Frier, and Pierre-Yves Oudeyer. Language and culture internalization for human-like autotelic ai. *Nature Machine Intelligence*, 4(12):1068–1076, December 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00591-4. URL http://dx.doi.org/10.1038/s42256-022-00591-4.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles (eds.), *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8657–8677. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/du23f.html.

Hughes Edward, Dennis Michael D, Parker-Holder Jack, Behbahani Feryal, Mavalankar Aditi, Shi Yuge, Schaul Tom, and Rocktäschel Tim. Position: Open-endedness is essential for artificial superhuman intelligence. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20597–20616. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/hughes24a.html.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017. URL https://arxiv.org/abs/1702.03118.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code, 2025. URL https://arxiv.org/abs/2405.15568.

Tanner Fiez, Benjamin Chasnov, and Lillian J. Ratliff. Convergence of learning dynamics in stackelberg games. *CoRR*, abs/1906.01217, 2019. URL http://arxiv.org/abs/1906.01217.

Paul M Fitts and Michael I Posner. *Human performance.* Brooks/Cole, 1967.

Loris Gaven, Thomas Carta, Clément Romac, Cédric Colas, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Magellan: Metacognitive predictions of learning progress guide autotelic llm agents in large goal spaces. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2502.07709.

Danijar Hafner. Benchmarking the spectrum of agent capabilities, 2022. URL https://arxiv.org/abs/2109.06780.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/7967cc8e3ab559e68cc944c44b1cf3e8-Paper.pdf.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

YiDing Jiang, Shixiang (Shane) Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/0af787945872196b42c9f73ead2565c8-Paper.pdf.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, 2016. URL https://arxiv.org/abs/1604.06057.

Hanchao Liu, Wenyuan Xue, Yifei Chen, Dapeng Chen, Xiutian Zhao, Ke Wang, Liping Hou, Rongjun Li, and Wei Peng. A survey on hallucination in large vision-language models. *arXiv preprint arXiv:2402.00253*, 2024.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language, 2019. URL https://arxiv.org/abs/1906.03926.

Seungyong Moon, Junyoung Yeom, Bumsoo Park, and Hyun Oh Song. Discovering hierarchical achievements in reinforcement learning via contrastive learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 63674–63686. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/c919a2b5ec1de69f2629f9119676e336-Paper-Conference.pdf.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *CoRR*, abs/1810.01257, 2018a. URL http://dblp.uni-trier.de/db/journals/corr/corr1810.html#abs-1810-01257.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018b. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/e6384711491713d29bc63fc5eeb5ba4f-Paper.pdf.

Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007. ISSN 1941-0026. doi: 10.1109/TEVC.2006.890271. Conference Name: IEEE Transactions on Evolutionary Computation.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *Advances in Neural Information Processing Systems*, 36:34866–34891, 2023.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

Guillaume Pourcel, Thomas Carta, Grgur Kovač, and Pierre-Yves Oudeyer. Autotelic LLM-based exploration for goal-conditioned RL. In *IMOL@NeurIPS 2024 - Intrinsically Motivated Open-ended Learning Workshop at NeurIPS 2024*, Vancouver, Canada, December 2024. URL https://inria.hal.science/hal-04861896.

Doina Precup and Richard S. Sutton. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts, 2000. AAI9978540.

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

Han Shen, Zhuoran Yang, and Tianyi Chen. Principled penalty-based methods for bilevel reinforcement learning and RLHF. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 44774–44799. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/shen24g.html.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models, 2025. URL https://arxiv.org/abs/2502.19417.

Olivier Sigaud, Gianluca Baldassarre, Cedric Colas, Stephane Doncieux, Richard Duro, Pierre-Yves Oudeyer, Nicolas Perrin-Gilbert, and Vieri Giuliano Santucci. A definition of open-ended learning problems for goal-conditioned agents, 2024. URL https://arxiv.org/abs/2311.00344.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999. URL http://dblp.uni-trier.de/db/journals/ai/ai112.html#SuttonPS99.

Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Rin Metcalf, Walter Talbott, Natalie Mackraz, R Devon Hjelm, and Alexander T Toshev. Large language models as generalizable policies for embodied tasks. In *The Twelfth International Conference on Learning Representations*, 2023.

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowledge comes from practice: Aligning large language models with embodied environments via reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 54281–54328, 2024.

Jonathan S Tsay, Hyosub E Kim, Samuel D McDougle, Jordan A Taylor, Adrian Haith, Guy Avraham, John W Krakauer, Anne GE Collins, and Richard B Ivry. Fundamental processes in sensorimotor learning: Reasoning, refinement, and retrieval. *eLife*, 13:e91839, aug 2024. ISSN 2050-084X. doi: 10.7554/eLife.91839. URL https://doi.org/10.7554/eLife.91839.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017. URL https://arxiv.org/abs/1703.01161.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Muning Wen, Ziyu Wan, Jun Wang, Weinan Zhang, and Ying Wen. Reinforcing llm agents via policy optimization with action decomposition. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 103774–103805. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/bc09efb501c801ed92e181e26a885c2d-Paper-Conference.pdf.

Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. Omni: Open-endedness via models of human notions of interestingness. In *The Twelfth International Conference on Learning Representations*, 2023a.

Jesse Zhang, Jiahui Zhang, Karl Pertsch, Ziyi Liu, Xiang Ren, Minsuk Chang, Shao-Hua Sun, and Joseph J Lim. Bootstrap your own skills: Learning to solve new tasks with large language model guidance. In *7th Annual Conference on Robot Learning*, 2023b. URL https://openreview.net/forum?id=a0mFRgadGO.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.

APPENDICES

This supplementary material provides additional results, discussion, and implementation details.

- Section A details our environment.
  - Section A.1 provides our environment's achievement tree with the list of all possible goals and their requirements.
  - Section A.2 details the modifications we performed to Crafter.
  - Section A.3 details the observation space used by our agents.
- Section B provides complementary results.
  - Section B.1 shows the per-task success rate for each method across training.
  - Section B.2 analyzes the generalization abilities of each method. We study both generalization to synonyms and n-compositionality.
  - Section B.3 provides insights on HERAKLES' high-level sampling strategy. We notably study the evolution fo sampling strategy for multiple goals.
- Section C details HERAKLES' interactions between the high-level and low-level policy.
  - Section C.1 provides HERAKLES' full algorithm.
  - Section C.2 gives details about our implementation of HERAKLES. We also provide the link to our code.
- Section D details the implementation of our high-level policy.
  - Section D.1 explains how our LLM backbone is used and trained to select skills.
  - Section D.2 details our prompting strategy.
- Section E details the implementation of our low-level policy.
  - Section E.1 explains the architecture of our lightweight neural networks used for the low-level policy.
  - Section E.2 details the low-level policy's training strategy based on AWR Peng et al. (2019).
  - Section E.3 presents a method to mitigate loss of plasticity and catastrophic forgetting during training of the low-level policy.
  - Section E.4 explains why it is necessary to filtrate a part of compiled trajectories and how our method do that.
- Section F gives details about our low-level success rate estimator.
  - Section F.1 explains how the low level success rate estimator is trained.
  - Section F.2 gives the exact formula used to calculate the probability for adding an option in the option set $\tilde{G}$ at step k.
  - Section F.3 shows the prompt given to our estimator.
  - Section F.4 provides details on the training strategy for the low-level success rate estimator.
  - Section F.5 explains how the estimation is used to generate the set of skills the high-level policy samples from.
- Section G describes how our autotelic goal sampler reimplements MAGELLAN Gaven et al. (2025).
  - Section G.1 provides details on how we have adapted MAGELLAN to our problem.
  - Section G.2 shows the prompt given to MAGELLAN's success rate estimator.
- Section H provides implementation details of our baselines.
  - Section H.1 describes our implementation of FUN Vezhnevets et al. (2017).
  - Section H.2 details our implementation of POAD's Wen et al. (2024) training strategy.
  - Section H.3 gives the practical implementation and training for the BOSS's algorithm Zhang et al. (2023b).
- Section I details the hyper-parameters we used.
- Section J gives the compute resources used for performing the experiments.

A Environment detail

A.1 Achievement tree

We build upon the Crafter environment introduced by Hafner (2022), introducing several modifications to emphasize learning in a heterogeneous and compositional goal space. Specifically, our environment features an achievement tree structure that explicitly defines the prerequisite relationships among goals, thereby encouraging agents to discover and exploit goal dependencies. The environment comprises 17 distinct goals, categorized into three types: (i) 5 movement-related goals requiring navigation to specific locations; (ii) 3 collection-based goals that require prior construction of the appropriate tools —e.g., collecting stone and coal is only feasible after crafting the corresponding equipment; and (iii) 3 crafting goals, which are achievable only when the agent possesses a sufficient quantity of the required resources in its inventory. This design introduces structured complexity that challenges agents to plan and act over extended temporal horizons.

Here is the list of all the goals and their requirements:

- go to tree: facing a tree
- collect wood: facing a tree, no tool needed
- place table: facing grass, sand, path and having 2 woods in inventory
- go to table: facing a table
- make wood pickaxe: facing a table and having 1 wood in inventory
- go to stone: facing a stone
- collect stone: facing a stone, having a wood pickaxe
- go to coal: facing a coal
- collect coal: facing a coal, having a wood pickaxe
- place furnace: facing grass, sand, path and having 4 stones in inventory
- go to furnace: facing a furnace
- make stone pickaxe: facing a table and having 1 wood and 1 stone in inventory
- go to iron: facing iron
- collect iron: facing iron, having a stone pickaxe
- make iron pickaxe: facing a furnace and having 1 iron 1 coal and 1 wood in inventory
- go to diamond: face diamond
- collect diamond: face diamond with an iron pickaxe

A.2 Modifications

Similarly to Du et al. (2023) we modify the original Crafter environment to isolate and emphasize the compositional aspects of goal-directed behaviour, removing elements primarily associated with survival dynamics. First, we eliminate hostile entities such as zombies and skeletons, as well as the need for the agent to eat or drink, thereby reducing perturbation-related distractions. Second, we convert the *'do'* action that originally represents several types of actions (meaning *'attack'* in front of a zombie but *'eat'* in front of a plant) into distinct actions (such as *'eat plant'* or *'attack zombie'*). To improve the agent's navigational capabilities, we augment the goal space with *"go to X"* objectives. These are always feasible for $X \in$ tree, stone, coal, but may not be realizable for $X \in$ table, furnace depending on the current world state, introducing ambiguity that poses a challenge for the low-level success rate estimator (see Appendix F). Furthermore, we enrich the observation space by integrating a captioning module (see Appendix A.3). The hierarchical policy operates under temporal constraints: the high-level controller is allowed up to 96 steps (i.e., skill invocations), each of which may be executed by the low-level controller in up to 64 steps. To enable multi-goal episodes while ensuring bounded episode length, we cap the environment at 155 high-level steps before a reset occurs.

A.3 Observations

The modified Crafter environment used in our experiments generates both a standard visual observation and a structured textual description. The visual observation consists of an RGB image of dimension $3 \times 32 \times 32$. The textual description acts as a sparse captioning of the visual input and is composed of multiple informative components, each providing structured context relevant to the agent's decision-making. Specifically, the textual description includes:

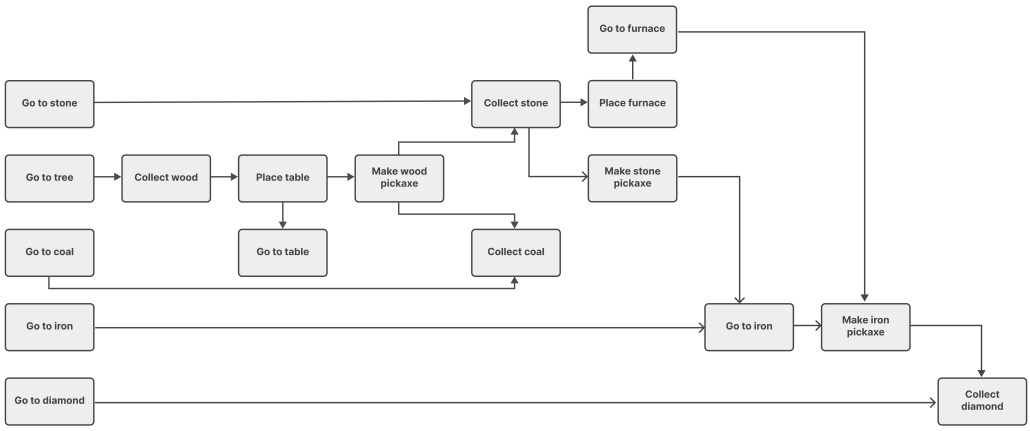


Figure 5: Achievement tree used in our experiments, the goal on the right necessitate some composition of the goals on the left.

- A general sentence describing the overall nature of the game.
- The agent's current goal, along with the number of remaining high-level steps to achieve it.
- The player's current position in the environment.
- A description of all visible elements in the field of view, each identified by type and relative position using cardinal directions (e.g., "a tree is three steps north"). Each type of element is mentioned only once, even if multiple instances are visible.
- A description of the block directly in front of the player.
- A list of items currently in the player's inventory.
- The positions of environment elements that have been placed by the player.
- The available elementary actions.
- If applicable, the list of admissible skills that can be invoked by the high-level policy.
- The last action executed by the agent.

This detailed textual observation allows the high-level policy to operate with a semantically rich input space, facilitating more informed and context-aware decision-making. An example of such textual observation is given in Figure 9, Figure 10.

	HERAKLES	POAD	FUN	BOSS
Original goals	39.1	23.6	7.8	27.6
Synonyms	32.8 (**-16%**)	18.2 (**-23%**)	5.7 (**-26%**)	24.8 (**-10%**)

Table 2: Generalization for synonyms goals.

B COMPLEMENTARY RESULTS

B.1 EVOLUTION OF SUCCESS RATE

In the Figure 6, we analyze the evolution of the success rate (SR) during training for each method. We measure the success rate every 28000 interaction steps by testing the agent on 40 different held-out environments. For the test, the agent has to complete the goal in a maximum of 1000 interaction steps with the environment and under 64 high level steps. For the simple tasks such as *"go to tree"* or *place table* the baselines tend to perform better than HERAKLES as they do not need to learn a low level agent (POAD) or a classifier to propose the relevant subgoals to the high level policy (FUN, BOSS). However, for more complex goals such as *collect coal* or *place furnace* that requires long chain of actions, baselines tend to stagnates (BOSS) or are not able to learn in the given time (POAD). HERAKLES exhibits rapid improvement on more complex compositional tasks. For instance, the SR when HERAKLES started to master *make wood pickaxe* around 150000 steps it simultaneously starts to master *collect stone* and *place furnace* that reuses the *make wood pickaxe*.

B.2 GENERALIZATION RESULTS

The ability to generalize to novel goals, particularly those expressed differently in natural language, is essential for open-ended agents to learn efficiently. In many cases, goals generated within a linguistic space may be semantically equivalent, either due to synonymy or because they correspond to behaviours already mastered under different phrasings.

Synonym generalization We assess the generalization performance of HERAKLES, POAD, and FUN on a set of synonym-based goals. For each original goal, such as *"collect wood"*, we define a synonym set by selecting five alternative formulations (e.g., *"gather wood"*, *"harvest wood"*, *"procure wood"*, *"acquire wood"*, and *"amass wood"*), and compute the average Crafter score across these variants. Table 3 lists the complete set of synonym categories used in our evaluation. Of note, certain words such as *"acquire"* appear across multiple categories (e.g., for both *"collect"* and *"make"*), requiring the agent to rely on contextual cues to disambiguate the intended task—a key competency in linguistically grounded goal spaces.

We quantify generalization performance by extending the Crafter score for synonym reformulations as:

$$S_c^{syn}(t) = \exp\left(\frac{1}{|G|}\sum_{g\in G}\frac{1}{|Syn(g)|}\sum_{g'\in Syn(g)}\ln(1+sr_{g'}(t))\right) - 1,$$

where $Syn(g)$ stands for the set of possible reformulations of goal $g \in G$ given the list of synonyms given in Table 3, and $sr_{g'}(t)$ is the average success rate over a reformulated goal $g' \in \bigcup_{g\in G} Syn(g)$ at step t of the training. The rate $sr_{g'}(t)$ is estimated by performing 8 runs using the model at training step t, with the high-level policy conditioned on the goal g'.

Table 2 contains the value of S_c^{syn} at the end of training. As illustrated in Table 2 HERAKLES preserves a high Crafter score in the synonym goal space, demonstrating a robust linguistic generalization. While HERAKLES experiences only a 16% drop in average score relative to the original goal space, POAD, FUN and BOSS exhibit more substantial decreases of 23%, 26% and 10%, respectively. These results highlight the superiority of HERAKLES in handling semantic variability in goal specification.

n-Compositionality A second form of generalization we investigate is *n-compositionality*, which evaluates an agent's ability to scale learned behaviours to repeated instances of the same subgoal. In this setting, we construct new environments where each original goal must be achieved n times consecutively, with $n \in 2, 3, 4$. For example, the original goal *"collect wood"* is reformulated as *"collect 2 woods"* in the 2-compositional environment. This setup tests the compositional depth of the agent's policy and its capacity to generalize beyond one-shot achievement.

We exclude *"go to X"* goals from the n-compositional environment, as repeating these goals lacks meaningful interpretation—one cannot meaningfully go to the same object n times in succession.

Table 4 illustrates that the Crafter score decreases for all methods as n increases, reflecting the growing complexity of the environment. However, HERAKLES exhibits a significantly smaller relative performance

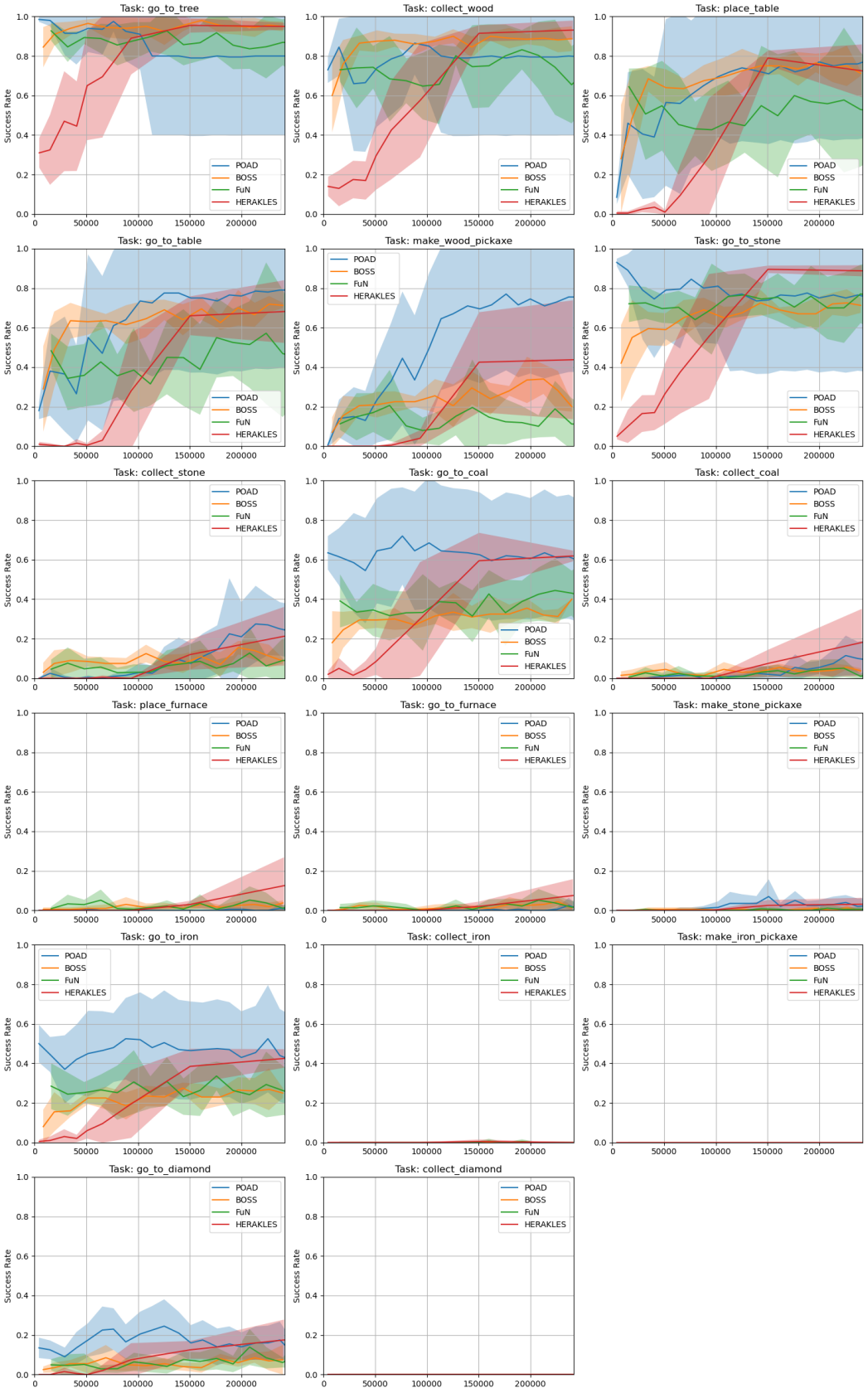


Figure 6: Success rate of each method per task depending on the number of interaction steps done in the environment. Shaded area represent standard deviation based on 5 seeds.

Table 3: Synonyms used for the synonym generalization experiment.

Original words	Synonyms		
collect	gather	acquire	procure
	harvest	amass	
make	craft	construct	build
	acquire	create	
place	put	putdown	install
	deploy	position	
go	move	walk	proceed
	travel	run	

Table 4: Generalization for n-compositional goals.

	HERAKLES	POAD	FUN	BOSS
Original goals	39.1	23.6	7.8	27.6
2 Compositional	33.2 (**-15%**)	0.396 (**-68%**)	5.11 (**-34%**)	18.8 (**-32%**)
3 Compositional	26.6 (**-32%**)	0.231 (**-82%**)	2.51 (**-67%**)	11.6 (**-58%**)
4 Compositional	19.2 (**-51%**)	0.102 (**-92%**)	1.40 (**-82%**)	9.94 (**-64%**)

drop compared to the baselines. For instance, at $n = 4$, HERAKLES incurs a 51% reduction in Crafter score, whereas POAD suffers a 92% drop, FUN a 82% and BOSS 64%. These results indicate that HERAKLES is better equipped to generalize through compositional repetition, a key property for agents operating in scalable, open-ended settings.

When analyzing performance across both generalization settings—synonym-based and n-compositional—we observe complementary strengths in the two baseline methods. Although POAD exhibits a lower overall average Crafter score, it outperforms FUN on synonym generalization tasks. This is likely attributable to the linguistic capabilities embedded in its large language model (LLM) backbone, in contrast to FUN, which relies solely on a static embedding for goal representation. Conversely, FUN's hierarchical architecture enables stronger performance in the n-compositional setting, where the reuse and chaining of subskills are crucial. Notably, HERAKLES, which integrates the strengths of both approaches—leveraging both a linguistically grounded high-level policy and a hierarchical skill structure—consistently outperforms both baselines across all generalization regimes.

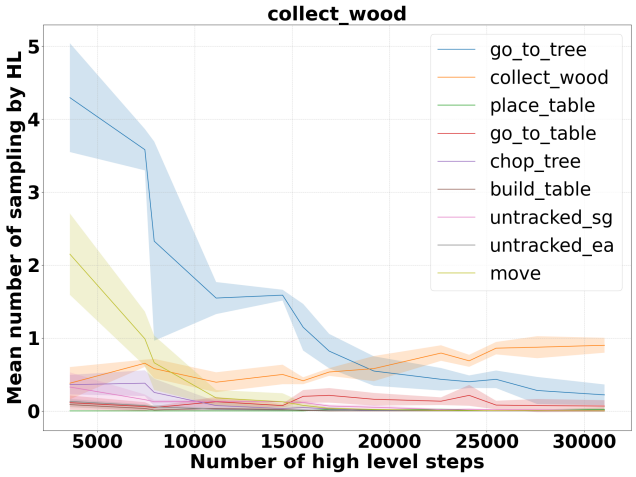
B.3 EVOLUTION OF HIGH-LEVEL SAMPLING STRATEGY

HERAKLES leverages the continuous compilation of mastered goals into low-level skills, gradually transferring control from the hierarchical agent to the low-level controller. In this section, we analyze how this mechanism shapes the high-level policy's sampling strategy throughout training. According to the learning dynamics described in Section 3, we hypothesize a three-phase progression: initially, the high-level policy broadly samples elementary actions to guide the undertrained low-level policy; subsequently, as the low-level improves, skill calls become more frequent; and finally, the high-level converges to invoking the compiled skill corresponding to the goal directly, relying fully on the low-level policy for execution. For instance, for the goal "collect wood", the high level would sample the *"collect wood"* skill.

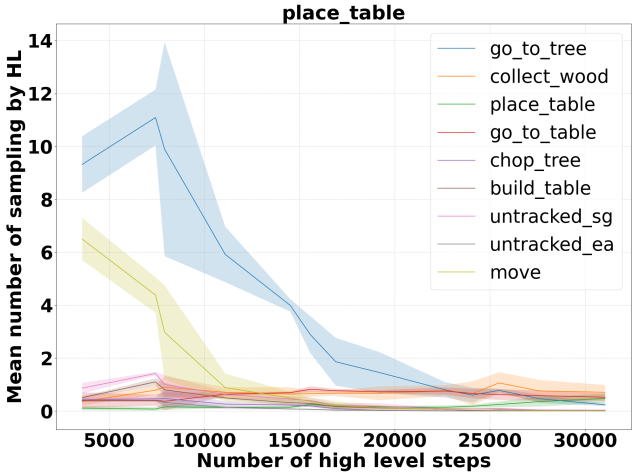
In Figure 7, we report the mean number of skill or action calls per trajectory. For clarity, we aggregate all untracked compiled skills and untracked elementary actions into *untracked_sg* and *untracked_ea*, respectively, and average all *"move"* actions under a single *"move"* category.

Across the three subplots of Figure 7, we observe that before $10{,}000$ high-level steps, the policy predominantly samples elementary actions of type *"move"* and the skill *"go to tree"*. Since gathering wood is a prerequisite for nearly all goals, this early focus is expected. Notably, the skill *"go to tree"* is oversampled—approximately 10 times per trajectory—because the low-level policy has yet to master it, despite the skill being highly accessible (random agents succeed 90% of the time in 128 steps).

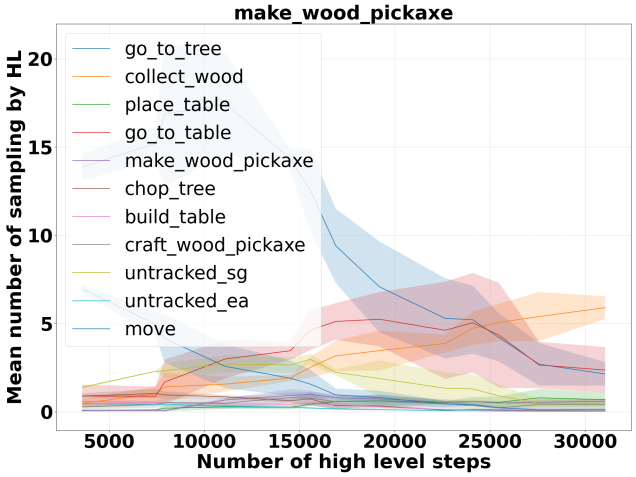
As shown in Figure 7a, the elementary action *"chop tree"* is also sampled about 0.5 times per trajectory in the same early phase. Between $10{,}000$ and $20{,}000$ steps, elementary action calls decrease and the high-level policy increasingly delegates to compiled skills. This shift is evident in Figure 7c, where *"collect wood"* and *"go to table"* are more frequently invoked.



(a) Collect wood.



(b) Place table.



(c) Make wood pickaxe.

Figure 7: Evolution of sampling strategy of high level.

By 25,000 steps, the high-level policy either directly selects compiled skills, as in Figure 7a with *"collect wood"*, or replaces simpler skills with more complex ones, as seen in Figures 7b and 7c, where *"collect wood"* replaces the earlier calls to *"go to tree"*.

These observations confirm that HERAKLES adheres to the hypothesized learning dynamics, effectively leveraging the hierarchy to bootstrap new skills from previously acquired ones, thereby enabling efficient goal acquisition over time.

C HIERARCHICAL AGENT

C.1 HERAKLES ALGORITHM

In this section, we provide the pseudocode of the HERAKLES algorithm (see Algorithm 2), showing the core training loop and its hierarchical decision-making structure. For clarity and brevity, we omit the update mechanisms for the MAGELLAN goal-sampling module Gaven et al. (2025) detail in Appendix G.

Algorithm 2 HERAKLES

Require: π_{HL} the high-level policy, π_{LL} the low-level policy, M the MAGELLAN module Gaven et al. (2025), G the goal space to be learned, $\mathcal{A}$ set of low level sensory motor commands, $\mathcal{B}_{\text{HL}}$ π_{HL} buffer, $\mathcal{B}^{\text{h}}_{\text{LL}}$ π_{LL} buffer for trajectories collected from the hierarchical policy, $\mathcal{B}^{\text{sg}}_{\text{LL}}$ π_{LL} buffer for trajectories collected from π_{LL} policy alone executing a goal decided by π_{HL}, C_{θ_k} the π_{LL} competency estimator and env the environment.

$(obs^{visual}, obs^{textual}) \leftarrow env.reset()$ ▷ see Appendix A.3
$g \leftarrow M(obs^{textual})$ ▷ Choose a goal g in an autotelic way
$done_g, r^g \leftarrow env.verify(g)$ ▷ verify checks if g is done, r^g reward for g
while $done_g$ not `True` **do**
  $\tilde{G} \leftarrow \{\}$ ▷ Instantiate the skill set
  for $g' \in G$ **do** ▷ Adding a skill with probability $p_{g'}$
    $\tilde{G} \leftarrow \tilde{G} \cup \{g'\} \sim p_{g'}$ ▷ $p_{g'}$ expression in Appendix F.2
  end for
  $\tilde{g} \leftarrow \pi^{HL}(g, (obs^{textual}, \tilde{G}))$
  $done_g, r^g \leftarrow env.verify(g)$
  $done_{\tilde{g}}, r^{\tilde{g}} \leftarrow env.verify(\tilde{g})$
  while $done_{\tilde{g}}$ not `True` and $done_g$ not `True` **do** ▷ π_{LL} executes the skill in the environment
    $a \leftarrow \pi^{LL}(\tilde{g}, obs^{visual})$ ▷ $a \in \mathcal{A}$
    $(obs^{visual}_{next}, obs^{textual}_{next}) \leftarrow env.step(a)$
    $done_g, r^g \leftarrow env.verify(g)$
    $done_{\tilde{g}}, r^{\tilde{g}} \leftarrow env.verify(\tilde{g})$
    $\mathcal{B}^{\text{sg}}_{\text{LL}} \leftarrow (\tilde{g}, obs^{visual}, a, r^{\tilde{g}})$ ▷ Transition seen as targeting $\tilde{g}$
    $\mathcal{B}^{\text{h}}_{\text{LL}} \leftarrow (g, obs^{visual}, a, r^{g})$ ▷ Transition seen as targeting g
    $obs^{visual} \leftarrow obs^{visual}_{next}$
    if enough new transitions in $\mathcal{B}_{\text{LL}}$ **then**
      Update π_{LL} on $\mathcal{B}^{\text{h}}_{\text{LL}} \cup \mathcal{B}^{\text{sg}}_{\text{LL}}$ ▷ Using AWR Peng et al. (2019)
      Update C_{θ_k} using $\mathcal{B}^{\text{sg}}_{\text{LL}}$ ▷ Appendix F
    end if
    if $done_{\tilde{g}}$ or $done_g$ **then**
      $\mathcal{B}_{\text{HL}} \leftarrow (g, (obs^{textual}, \tilde{G}), a, r^g)$
    end if
  end while
  Update π_{HL} $\mathcal{B}_{\text{HL}}$ ▷ Using POAD Wen et al. (2024)
  $obs^{textual} \leftarrow obs^{textual}_{next}$
end while

C.2 HERAKLES IMPLEMENTATION

To instantiate the hierarchical agent described in Section 3, we employ a large language model (LLM)-backboned agent as the high-level policy (detailed in Appendix D) and a goal conditioned lightweight neural networks as the low-level policy (see Appendix E). The Figure 8 gives an overview of the overall architecture.

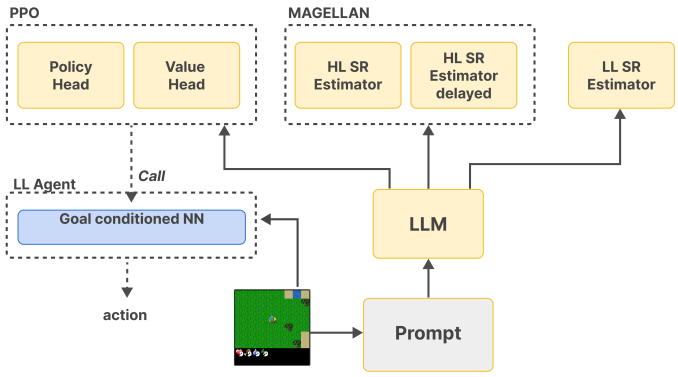


Figure 8: HERAKLES architecture. In yellow are the parts that used a LLM backbone and are trained with LoRA adapters Hu et al. (2022). In blue is the simple goal conditioned neural network that represent the low level policy, more details on its architecture in Appendix E.1.

In accordance with the formulation in Section 3, we assume that $\forall(a, s) \in A \times S, ; \pi^{LL}(a|s, a) = 1$. Consequently, when the high-level policy selects a skill that corresponds to an elementary action—e.g., *"move right"*—we bypass the low-level controller and directly execute the action in the environment.

HERAKLES further incorporates a low-level success rate estimator to manage skill inclusion in the filtered skill set $\tilde{G}$, and leverages MAGELLAN to guide goal sampling. These components are described in Sections F and G, respectively.

Each module—the high-level policy, the individual low-level networks, the success rate estimator, and MAGELLAN—is trained independently using separate Adam optimizers. Detailed hyper-parameter configurations for each optimizer are provided in Appendix I.

D HIGH-LEVEL POLICY

D.1 HIGH-LEVEL POLICY ARCHITECTURE

The high-level policy architecture in HERAKLES builds upon GLAM Carta et al. (2023), which employs an actor-critic framework grounded in a large language model (LLM) backbone. The value function is implemented as a two-layer multilayer perceptron (MLP) with 1024 sigmoid-activated units, applied to the final hidden state of the decoder. Both the policy and the value head utilize shared LoRA adapters Hu et al. (2022), enabling parameter-efficient fine-tuning of the LLM.

In contrast to GLAM, skill selection in HERAKLES is achieved through constrained decoding over the set of admissible skills $\tilde{G}$. Instead of sampling tokens from the entire vocabulary, token generation is restricted to sequences that correspond to valid skills in $\tilde{G}$, ensuring both syntactic validity and semantic precision.

Training of the high-level policy is conducted using POAD Wen et al. (2024), an algorithm that adapts PPO to operate at the token level for language agents. The policy is updated every 2496 high-level transitions, using POAD with 4 training epochs per update cycle. In addition to the POAD loss we add a penalty term based on Kullback-Liebler divergence:

$$\beta_{KL} KL(\pi^{HL}, \pi^{HL}_{original}),$$

with β_{KL} an hyper-parameter controlling for the importance of this penalty term. It ensures that the high-level policy does not drift too much from the original policy and retains some plasticity, allowing it to adapt to the apparition of new usable skills during training.

D.2 PROMPTS FOR THE HIGH-LEVEL POLICY

In this section, we provide an illustrative example of a prompt given to the high-level policy in HERAKLES. In the depicted scenario, the environment has not been reset since the previous interaction, a new goal, *"make furnace"* has been sampled and the agent has already done one high level step. The prompt includes a description of the current state, the previously executed skill, and the list of admissible skills $\tilde{G}$. This setup allows the high-level

policy to condition its next action on both the recent interaction history and the updated goal. Such prompt structure ensures that the LLM-based policy can contextualize its decision-making process in a temporally coherent and semantically grounded manner.

Example Prompt for the high-level policy (1/2)

```
You are playing a Minecraft like game.  You can use elementary
actions or, if available, more efficient low-level policies.
Your task:  Make to furnace in 64.0 steps
You have already done 1 step.
Your coordinates:  (29,10)
You see:
     • grass 1 step to your west
     • path 5 steps to your south-west
     • sand 4 steps to your south-east
     • tree 3 steps to your east
     • table 1 step to your north
     • plant 6 steps to your south-west
You face table at your front.
Your inventory:
     • sapling:  9
     • wood:  4
     • coal:  2
     • wood pickaxe:  4
You placed table at (4,3) You placed table at (23,13) You placed
table at (29,9) You placed plant at (0,12) You placed plant at (0,0)
You placed stone at (13,0)
Elementary actions you can take:
     • move left
     • move right
     • move up
     • move down
     • sleep
     • consume cow
     • consume plant
     • attack zombie
     • attack skeleton
     • attack cow
     • chop tree (require facing tree)
     • chop bush (require facing bush)
```

Figure 9: Example of a prompt given to the agent high-level policy (1/2).

Example Prompt for the high-level policy (2/2)

- chop grass (require facing grass)
- extract stone (require 1 wood pickaxe in your inventory while facing stone)
- extract coal (require 1 wood pickaxe in your inventory while facing coal)
- extract iron (require 1 stone pickaxe in your inventory while facing iron)
- extract diamond (require 1 iron pickaxe in your inventory while facing diamond)
- drink water
- put stone (require 1 stone in your inventory)
- build table (require 2 woods in your inventory)
- build furnace (require 4 stones in your inventory)
- put plant (require 1 sapling in your inventory)
- craft wood pickaxe (require 1 wood in your inventory while facing a table)
- craft stone pickaxe (require 1 wood, 1 stone in your inventory while facing a table)
- craft iron pickaxe (require 1 wood, 1 coal, 1 iron in your inventory while facing a furnace)
- craft wood sword (require 1 wood in your inventory while facing a table)
- craft stone sword (require 1 wood, 1 stone in your inventory while facing a table)
- craft iron sword (require 1 wood, 1 coal, 1 iron in your inventory while facing a furnace)

Low-level policies you can call:

- go to tree
- collect wood
- place table
- go to table

The last action you took: collect wood
Your action:

Figure 10: Example of a prompt given to the agent high-level policy (2/2) .

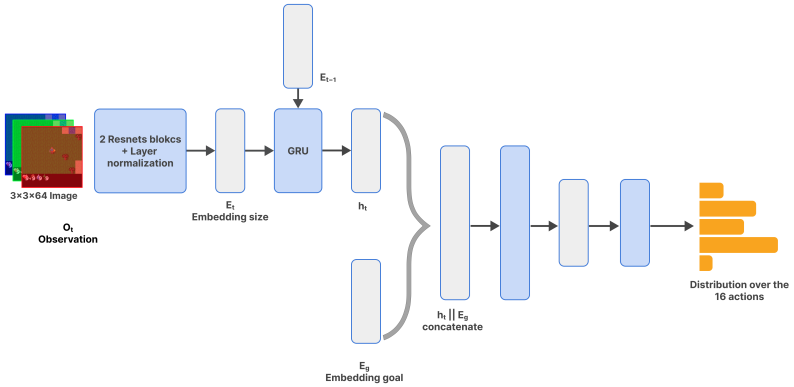


Figure 11: Low-level policy architecture. The blue rectangles represent neural network layers and the grey rectangles the outputted embeddings.

E LOW-LEVEL POLICY

E.1 LOW-LEVEL POLICY ARCHITECTURE

The low-level policy in HERAKLES is designed to be lightweight compared to the high-level language model and operates directly on raw sensory input, specifically RGB observations from the environment. To extract features and generate elementary actions, we adapt the architecture proposed by Moon et al. (2023). This model employs a two-block ResNet He et al. (2015) with channel dimensions $[64, 128, 128]$ to encode visual information. The resulting feature map is flattened and passed through a GRU layer Cho et al. (2014) to build a history of the trajectory embedded in h_t. h_t is concatenated with the embedding of the goal E_g and passes through two fully connected layers of sizes 1024 and 128, respectively. The final embedding is then fed into a sigmoid-activated output layer of dimension 16 to produce a probability distribution in the discrete space of elementary actions. A low-level policy architecture diagram is shown in Figure 11. The critic architecture is similar to the actor but the last layer has an output dimension of 1. The low-level agent contains approximately 2 million parameters, which constitutes only 0.03% of the total parameter count of the high-level policy, ensuring computational efficiency.

E.2 LOW-LEVEL TRAINING

We reimplement the Advantage-Weighted Regression (AWR) algorithm Peng et al. (2019) to train the low-level policy in HERAKLES. AWR is selected for its simplicity and robustness, as well as its flexibility in integrating heterogeneous data sources into a unified replay buffer. Specifically, we leverage two types of transitions: *(i)* those collected from the low-level policy executing a given skill, enabling continual skill refinement, and *(ii)* transitions sampled from successful trajectories of the hierarchical agent, used to distil composite behaviours into the low-level policy.

This dual sourcing enables multiplicative relabelling, analogous to Hindsight Experience Replay (HER) Andrychowicz et al. (2017). For instance, a transition such as *"facing a tree and chopping wood"* can simultaneously reinforce the skill *"collect wood"* and contribute to the higher-level objective *"place table"*. This mechanism allows the low-level agent to acquire new skills by imitating hierarchical behaviours while refining its competence on previously mastered ones.

We employ sparse binary rewards to signal skill completion and goal achievement. For failed trajectories, i.e., when a skill is not completed within the 64-step budget, we substitute the zero reward with the critic's value estimate as a proxy signal. This effectively smooths the discontinuity introduced by the fixed time horizon and provides a more informative learning signal.

Determining *when* to update the low-level policy requires balancing sample efficiency with overall training throughput. As the agent progresses, it gains access to an increasingly large set of subgoals, each generating a growing stream of transitions. Naively updating the low-level policy at a fixed transition interval would cause it to over-train on frequently visited, elementary goals, such as *go to wood*, which is invoked as a prerequisite for nearly every higher-level objective like *place table*, while starving the system of new experience. Conversely, updating too infrequently wastes the information carried by newly collected transitions.

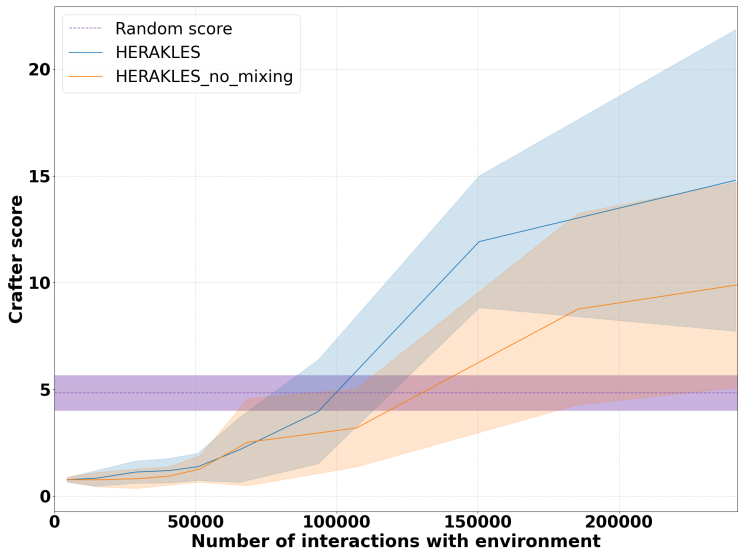


Figure 12: Evolution of the crafter score with and without doing the update of π_{LL} with a mixture of goal as regularisation.

To reconcile these competing pressures, we condition each update on three criteria: *(i)* a minimum number of new transitions must have accumulated since the last update, ensuring a sufficiently large and fresh batch; *(ii)* the goal must have a non-trivial success rate in the buffer, filtering out goals for which the positive signal is too sparse to provide a reliable learning gradient; and *(iii)* the update probability decays with the number of low-level updates performed since the last high-level update, preventing runaway specialisation on easy goals. Formally, the update probability for goal g is:

$$p_{\text{update}}(g) \;=\; \mathbf{1}_{\Delta n(g) > 2496} \;\cdot\; \mathbf{1}_{SR_\tau(g) > \alpha} \;\cdot\; \big(1 - SR_\tau(g)\big)^{N_{\text{ll}}(g)}, \tag{3}$$

where:

- $\Delta n(g)$ denotes the number of new transitions for goal g added to the replay buffer $\mathcal{B}_{LL}(g)$ since its last update;
- $SR_\tau(g)$ is the fraction of successful trajectories for goal g currently stored in the buffer $\mathcal{B}_{LL}(g)$ with $\alpha = 0.1$; and
- $N_{\text{ll}}(g)$ is the number of low-level updates performed on goal g since the most recent high-level policy update.

The second indicator suppresses updates for goals that are too difficult and lack sufficient positive examples, avoiding noisy gradient estimates. The exponential decay in the third term ensures that goals which are mastered early, such as *go to tree*, are not over-trained at the expense of broader exploration.

E.3 Multigoal buffer composition

In our method, π_{LL} is trained to achieve multiple goals in a continual learning setting, where new goals are introduced progressively during training. To mitigate loss of plasticity and catastrophic forgetting, we augment the training process as follows: whenever π_{LL} is updated using the replay buffer $\mathcal{B}_{LL}(g_i)$ associated with a goal g_i we include up to a fraction β_{sample} of transitions uniformly sampled from buffers corresponding to other goals g_j with $i \neq j$. In our experiments, we set $\beta_{sample} = 0.2$. The idea is that π_{LL} would not optimize only for g_i forgetting its previous trainings on other goals.

In Figure 12 we propose an ablation where there is no mixing of goals in the updates of π_{LL} during training. We observe that without such mixing the overall performances decrease strongly with a score of 10 when there is no mixing instead of 15 after 250000 steps in the environment. Regularising the π_{LL} updates by adding a mixt of transitions from other goals help the low level to learn in a more robust manner.

E.4 BUFFER TRANSITIONS FILTRATION

When the hierarchical agent attempts a new goal (g'), the high-level policy explores by sampling a large number of options. The resulting relabelled trajectories added to the replay buffer of π_{LL} for (g') tend to be extremely long in terms of primitive actions—often several times longer than (N^{LL}), the maximum number of steps allowed for π_{LL}. Moreover, these trajectories are typically unrewarded, generating substantial noise during the early stages of training π_{LL} on (g').

To mitigate this issue, we prune a subset of these failed trajectories based on the hierarchical agent's learning progress, denoted ($LP^{hierarchical}$). Specifically, we retain a compiled failed trajectory only if its length ($l_{compiled_failed}$) satisfies

$$l_{compiled_failed} < \frac{1}{1 - LP^{\text{hierarchical}}}, N^{LL}.$$

We focus particularly on failed trajectories when the competency of the hierarchical policy is evolving, as such failures are more likely to carry meaningful learning signals.

F LOW-LEVEL SUCCESS RATE ESTIMATOR

F.1 LOW-LEVEL SUCCESS RATE ESTIMATOR TRAINING

To measure the probability of success $\mathbb{E}_{\pi^{LL}(.|g),s_k\sim D_{\pi^{LL}}}[r^g|O(s_k)]$, we leverage the LLM used by π_{HL} by learning the parameters θ_k of a competence estimator $C_{\theta_k}(s_k, g) \approx \mathbb{E}_{\pi^{LL}(.|g),s_k\sim D_{\pi^{LL}}}[r^g|O(s_k)]$, following a strategy similar to that of Gaven et al. (2025). We pass in the LLM a prompt describing the pair $(s, g) \in S \times G$ which produces a latent representation on top of its final decoder block for the last token. The representation is passed to a multilayer perceptron (MLP) that outputs the estimated success probability $C_{\theta_k}(s, g)$. We train both the LLM and the MLP by minimizing the binary cross-entropy: $\mathcal{L}(\theta_t) = \mathbb{E}_{((s,g),r)\sim\mathcal{D}_t}[BCE(r, C_{\theta_t}(s, g))]$, with $\mathcal{D}_t$ a buffer containing the most recent $((s, g), r)$ pairs. In practice, we train two separate versions of the same initial LLM using LoRA adapters Hu et al. (2022): one for π_{HL} and one for our competence estimator. In Figure 8 presents how the competence estimator is integrated inside to the model.

F.2 p_g NUMERICAL CALCULATION

In Section 3.4, each goal $g \in G$ is included in $\tilde{G}_k(o_t)$ via an independent Bernoulli draw with probability $p_g = \max\big(C_{\theta_k}(o_t, g), \epsilon\big)$. In practice, two additional filtering conditions are applied. The full inclusion probability used in our implementation is:

$$p_g \;=\; \max\big(C_{\theta_k}(o_t, g),\, \epsilon\big) \;\cdot\; \mathbf{1}_{SR_\tau(g)>\alpha} \;\cdot\; \mathbf{1}_{u_g^{LL}>L}, \tag{4}$$

where u_g^{LL} denotes the number of low-level updates performed on goal g so far, L is a minimum-update threshold, $SR_\tau(g)$ is the fraction of successful trajectories for g currently stored in the buffer $\mathcal{B}_{\text{LL}}^{\text{sg}}$, and α is a minimum-success-rate threshold.

The two indicators play distinct roles. The first, $\mathbf{1}_{SR_\tau(g)>\alpha}$, is a direct consequence of the schedule used to update π_{LL} (see Eq. 3): π_{LL} is updated on g only once a non-trivial fraction of successful trajectories has been collected in the buffer, so applying the same condition to p_g ensures that $\tilde{G}_k$ is aligned with the goals on which π_{LL} is actually being trained. The second, $\mathbf{1}_{u_g^{LL}>L}$, is a simple heuristic: if π_{LL} has not been updated on g a sufficient number of times, it is unlikely to solve g regardless of the state, so proposing g as a callable option to π_{HL} would only destabilize high-level training. Finally, ϵ retains its role from the main paper: it ensures that goals satisfying both indicators keep a minimal inclusion probability, sustaining exploration and breaking the self-reinforcing loop between C_{θ_k} and $\tilde{G}_k$.

F.3 PROMPT OF THE LOW-LEVEL SUCCESS RATE ESTIMATOR

In this Figure 13, we give an example of prompts given to the low-level success rate estimator.

F.4 TRAINING OF THE LOW-LEVEL SUCCESS RATE ESTIMATOR

The low-level success rate estimator predicts the probability of successful skill execution given an observation. It is built upon an LLM backbone, from which we extract the final hidden state of the decoder. This representation is passed through two SiLU-activated hidden layers of size 1024 Elfwing et al. (2017), followed by a sigmoid-activated output layer of dimension 1, yielding a scalar success probability.

Example of prompt of the low-level policy success rate estimator

```
You are playing a Minecraft like game.  Your task is to evaluate
your success rate for the goal:  goal
Your coordinates:  (13,31)
You see:
     • water 7 steps to your north-west
     • grass 1 step to your west
     • stone 3 steps to your north-east
     • sand 3 steps to your north
     • table 1 step to your south
You face table at your front.
Your inventory:
     • sapling:  8
     • stone:  9
     • coal:  6
     • wood pickaxe:  3
     • wood sword:  1
You placed table at (39,40) You placed table at (3,32) You placed
table at (12,34) You placed table at (13,32) You placed table at
(10,30) You placed table at (37,18) You placed table at (40,17) You
placed table at (31,16)
Your success rate is:
```

Figure 13: Example of a prompt given to low-level policy success rate estimator.

During execution, whenever the high-level policy selects a skill, we record a tuple $(\text{obs}_{\text{ll_sr_estimator}}, \text{skill}, \text{outcome})$, where *outcome* is a binary indicator of whether the skill was successfully executed by the low-level policy. To augment training data and enhance the estimator's ability to interpret rich observations, we assume that starting from any of the initial 10% of states visited in a trajectory would not alter the success outcome. Accordingly, we collect $6 = \lfloor 0.1 \times 64 \rfloor$ such tuples per skill execution.

These transitions are aggregated during each high-level data collection cycle (consisting of 2496 high-level steps; see Appendix D) and stored in a buffer of size 3. The estimator is updated every 256 new transitions using a binary cross-entropy loss and trained for a single update epoch per cycle. These two hyper-parameters were selected based on empirical performance.

F.5 Composition of the set of skills

In Section 3.4, we define the probability of sampling a skill g into the admissible skill set $\tilde{G}$ at step k as:

$$p_g = \max\left(\mathbb{E}_{\pi^{LL}(\cdot|g),\, s_k \sim D_{\pi^{LL}}}\left[r^g \mid O(s_k)\right], \varepsilon_{explo}\right),$$

where ε_{explo} is an exploration term. We empirically found that ε_k should reflect the update frequency of the low-level policy corresponding to skill g. The rationale is that a frequently updated low-level policy is more likely to have improved, thus increasing the probability that the associated skill will succeed and should be selected.

We therefore define the exploration term as:

$$\varepsilon_{explo} = \min(\text{update_frequency_g},\, 0.1),$$

where `update_frequency_g` denotes the number of times the low-level policy for skill g has been updated in the past five high-level data collection cycles. This dynamic exploration term outperforms a fixed value, such as the constant $\varepsilon_k = 0.1$, in empirical evaluations.

G IMPLEMENTATION OF MAGELLAN

G.1 MAGELLAN ADAPTATION

We reimplemented MAGELLAN from Gaven et al. (2025) to make our autotelic goal sampler prioritize goals with maximum Learning Progress (LP) from G. Our implementation slightly differs from the original one to match our framework.

First, the setup in MAGELLAN involved a single goal per episode. As a result, their goal space was the combination of all possible initial states and instructions (i.e., the Multi-Armed Bandit had one arm per initial state-instruction pair). Here, multiple goals can be solved within a single episode. Therefore, whenever a goal has to be sampled, MAGELLAN estimates the LP of each goal from G only for the current state. While this does not change much from a practical point of view, our bandit is now a contextual Multi-Armed Bandit (i.e., the set of arms never changes but the current state conditions the LP estimation).

Second, instead of storing a single entry per goal (i.e. the goal, the state it was sampled from, and its associated outcome) in MAGELLAN's dataset ($\mathcal{D}$ in their paper), we store multiple copies of the same goal-outcome pair with different states seen during the trajectory. In particular, we store the first $N^{HL} * 0.1 = 6$ states observed by the high-level policy during the trajectory. This provides richer information for a goal and helps foster generalization.

Then, as a possibly varying number of entries are added to the dataset for each goal (i.e. up to 6), MAGELLAN's update frequency is now calculated based on the number of entries added instead of goals sampled. We perform this update every 128 new entries. Moreover, instead of keeping a dataset of fixed size, we use a dataset of varying size storing the entries from the last 3 data collection phases of the high-level policy (i.e. 2496 high-level steps, see Appendix D).

For the buffer of weights (named $\mathcal{B}$ in MAGELLAN), we only keep the last 3 success rate estimator copies. Finally, we use an exponential decay for ϵ with a rate of 3.34. All the other hyper-parameters not mentioned in the section are kept the same as in the original implementation of MAGELLAN.

G.2 PROMPT OF THE HIGH-LEVEL SUCCESS RATE ESTIMATOR

We show in 13 the prompt given to the LLM when estimating the competence, i.e. the success rate.

Example Prompt high-level policy success rate estimator

```
You are playing a Minecraft like game.  Your task is to evaluate
your success rate for the goal:  <goal>
Your coordinates:  (24,8)
You see:
     • grass 1 step to your west
     • stone 2 steps to your west
     • path 2 steps to your north-east
     • coal 2 steps to your south-west
     • iron 7 steps to your south-west
     • table 3 steps to your south-east
     • furnace 1 step to your south
You face a furnace at your front.
Your inventory:
     • sapling:  6
     • stone:  5
     • coal:  1
     • wood pickaxe:  3
     • wood sword:  1
You placed table at (26,36) You placed table at (16,24) You placed
table at (26,9) You placed table at (24,0)
You placed plant at (1,0) You placed plant at (4,0)
You placed stone at (5,17) You placed stone at (3,17) You placed
stone at (2,18)
You placed furnace at (24,9)
Your success rate is:
```

Figure 14: Example of a prompt given to high-level policy success rate estimator. This prompt is given for each goal in $g \in G$ with <goal> replaced g.

H BASELINES DETAILS

H.1 FUN

We implemented FeUdal Networks (FUN) a hierarchical RL architecture with two components: a Manager and a Worker, both modeled as recurrent networks. The Manager operates at a lower temporal resolution, producing directional goals $g_t \in \mathbb{R}^d$ in a learned latent state space. The Worker receives these goals and produces elementary actions at every time step.

Observations x_t are encoded into latent states z_t via a shared perceptual module. The Manager computes a latent embedding s_t, and generates g_t using a dilated LSTM that supports long-range dependencies. The Worker pools recent goals, maps them into a low-dimensional space via a projection ϕ, and combines them with action embeddings U_t to produce the policy:

$$\pi_t = \text{SoftMax}(U_t \, \phi \left(\sum_{i=0}^{c-1} g_{t-i} \right))$$

The Manager is trained using a *transition policy gradient* to align g_t with advantageous latent transitions $s_{t+c} - s_t$, while the Worker is trained via intrinsic reward based on goal-following:

$$r_t^I = \frac{1}{c} \sum_{i=1}^{c} \cos(s_t - s_{t-i}, g_{t-i})$$

and optimizes a weighted sum of intrinsic and extrinsic rewards.

In the original FUN implementation, only environment observations were passed to the perceptual module, as the evaluated environments were not goal-conditioned. In our version, we concatenate an embedding of the goal

to the output of the perceptual module. These goal embeddings are generated using the same language model employed for HERAKLES (Mistral 7B v0.3), enabling the agent to generalize to goals it has not encountered during training.

Table 5 summarizes the key hyper-parameters used in our implementation of FuN.

Table 5: Hyperparameters used in the FuN implementation.

Parameter	Value
Worker goal horizon	10
Hidden dimension of Manager	256
Hidden dimension of Worker	16
Discount factor for Worker	0.99
Discount factor for Manager	0.999
Intrinsic reward coefficient	0.5
Dilation factor for Manager's LSTM	10

H.2 POAD TRAINING

The POAD baseline Wen et al. (2024) corresponds to a simplified version of HERAKLES in which the hierarchical structure is removed by discarding the low-level policy. In this configuration, all elementary actions are directly selected by the high-level policy. The training procedure strictly follows the methodology and hyper-parameters detailed in Appendix D.

H.3 BOSS TRAINING

BOSS Zhang et al. (2023b) is a method for learning complex long-term tasks by autonomously growing a skill library. It proceeds in two phases: (1) pre-training a language-conditioned skill policy on a labelled demonstration dataset, and (2) iteratively bootstrapping that policy into longer-horizon behaviours. The two phases that composed the method are detailed below with the adaptations we have made to fit our experimental setup.

Phase 1 – Pre-training a Skill Policy. BOSS assumes access to a dataset $\mathcal{D}_L = \{\tau_{z_1}, \tau_{z_2}, \ldots\}$ of trajectories, where each trajectory τ_{z_i} is annotated with a free-form language description z_i of a primitive skill (*e.g.*, "collect wood"). A sparse reward function r signals task completion. A language-conditioned policy $\pi_{LL}^{BOSS}(a \mid s, z)$ and value function $V(s, z)$ are trained on $\mathcal{D}_L$ using Implicit Q-Learning (IQL) Kostrikov et al. (2022), yielding a repertoire of executable primitive skills $\mathcal{Z} = \{z_1, z_2, \ldots\}$.

Adaptations: In our experimental set up we do not have access to $\mathcal{D}_L$. We replace the primitive skills by elementary actions (*e.g.*, *move left*, *move right*, *chop tree*, ...) that can always be executed. For modeling $V(s, z)$, we use a value function with a LLM back bone similarly to the one use for π_{HL} described in Appendix D.1.

Phase 2 – Skill Bootstrapping. Starting from the primitive repertoire, BOSS iterates three steps to compose increasingly long-horizon skills without additional human supervision.

1. **Initial skill sampling.** At the start of each episode, a skill $z \in \mathcal{Z}$ is sampled proportionally to the pre-trained value function $V(s_1, z)$, favouring skills that are likely to succeed from the current state s_1.
   Adaptations: $V(s_1, z)$ is not pretrained in our set up but learned during training on collected trajectories. For elementary actions (*e.g. move left*) that are always successful in one step, we bypass V and assigned a value of 1 (the true value).
2. **LLM-guided skill chaining.** After a successful skill execution, a large language model (LLM) is prompted with the current skill repertoire and the skills executed so far, and proposes the next skill in natural language. This is repeated N times; the actual next skill is drawn from the distribution of LLM-assigned token likelihoods, encouraging diversity. The proposal is then mapped back to the closest skill in $\mathcal{Z}$ via a pre-trained sentence embedding model. $\pi_{LL}^{BOSS}(a \mid s, z)$ is executed to reach the chosen skill, if it is successful step 2 is repeated M times.
   Adaptations: the next skills is proposed through constrain decoding among the list of all possible skills in Crafter avoiding the mapping to a pretrained sequence. We use $N = 4$ in the experiments. In the original paper they used $M = 2$, however, as we start with elementary actions and not pretrained skills, we used $M = 4$ to help discovering complex skills.

3. **Skill library expansion.** At the end of each episode, the collected experience is added to the replay buffer with a sparse reward of 1 per completed sub-skill. The same LLM is used to generate composite language instructions describing multi-step chains (*e.g.*, "make wood pickaxe"); these composite instructions and associated trajectories are appended to both the replay buffer and the skill repertoire $\mathcal{Z}$ for further bootstrapping rounds. To mitigate catastrophic forgetting, offline data from $\mathcal{D}_L$ and online data are sampled at equal proportions during policy updates.

   Adaptations: We use an oracle to relabel the trajectories. Using an LLM leads to highly noisy relabeling in the Crafter environment that impedes the learning of more complex skills. Our oracle relabel the trajectory with the most complex skills reached at the end of it. We store the trajectories in $\mathcal{D}_L$ which is modeled as a queue with a maximum size of 100000 transitions. We train $V(s_1, z)$ and π_{LL}^{BOSS} $\mathcal{D}_L$ after collecting 48 trajectories in the environment.

At test time, the LLM is given a prompt similar to that of Appendix D.2.

I HYPER-PARAMETERS

Table 6: Hyper-parameters used in HERAKLES.

Parameter	**Value**
high-level policy max steps	64
low-level policy max steps	128
number of environment in parallel	48
number of high level step of data collection for POAD	2496
lr_{POAD}	10^{-5}
λ_{POAD}	0.9
γ_{POAD}	0.95
entropy coefficient POAD	0.01
β_{KL}	0.1
update epoch POAD	4
AWR buffer size	10^5
lr_{AWR}	10^{-4}
$lr_{\text{low level success rate estimator}}$	10^{-4}
udpate epoch low level success rate estimator	2

J COMPUTE RESOURCES

To run the various training sessions, we used H100 and V100 GPUs. Training HERAKLES was done with a cluster consisting of four H100 GPUs for 200 hours. To train POAD, a cluster comprising four H100s was required for 60 hours; to train BOSS, a cluster comprising four H100s was required for 100 hours. Training FUN necessitated 20 hours on a V100. It is important to note that a significant proportion of the time taken by HERAKLES is accounted for by loading time and buffer operations (approximately 33% of the total time). These operations were not optimised during these experiments