12th INTERNATIONAL SCIENTIFIC CONFERENCE ON DEFENSIVE TECHNOLOGIES
OTEH 2026
Serbia, Tara, 21 - 24 October 2026

MODULARITY OR COMPRESSION? A UNIFIED COMPARISON OF MIXTURE-OF-EXPERTS AND POLICY DISTILLATION AS EXECUTION MODULES FOR LLM-PLANNED AGENTS

IGOR CIGANOVIĆ
School of Computing, Union University, Belgrade, iciganovic217draf.rs

ALEKSANDAR PLUŠKOSKI
School of Computing, Union University, Belgrade, apluskoski317d@raf.rs

MILOŠ JOVANOVIĆ
Technical Test Center, Belgrade, milosjovanovic1967@gmail.com

JELENA VASILJEVIĆ
School of Computing, Union University, Belgrade, jvasiljevic@raf.rs

Abstract: *Hierarchical agents that delegate long-horizon planning to a large language model (LLM) and low-level control to a learned execution module have recently been shown to solve sparse-reward tasks that defeat monolithic reinforcement learning (RL) agents. Two distinct designs of the execution module have been proposed independently for the same environment: a mixture-of-experts (MoE) architecture that retains one specialized policy per sub-task and routes between them, and a single network obtained by distilling several expert policies into one. Both were reported to outperform an end-to-end baseline, but they have never been compared to each other, and the published numbers were obtained under different expert-training procedures, which makes a direct reading of them unsafe. This paper places both designs in a single architectural framework, re-analyses the published results under a common notation, and identifies precisely which comparisons the existing evidence does and does not support. We find that the two execution modules are separated by 0.02 average reward, far below the resolution of single-seed experiments, while the expert policies they were built from differ by up to 18 percentage points on identical sub-tasks purely because of how they were initialized. We further show that under both designs the planner contributes essentially no errors, so the execution module is the binding constraint in both systems.*

Keywords: *reinforcement learning, hierarchical agents, mixture of experts, policy distillation, large language models, MiniGrid, sparse reward*

1. INTRODUCTION

Reward sparsity remains one of the practical limits on applying reinforcement learning (RL) to tasks of realistic length [1], [2]. When the environment returns a scalar reward only after a long sequence of interdependent actions has been executed correctly, the credit-assignment problem becomes intractable for a single monolithic policy, and training either stagnates or converges to a policy that solves only the shortest missions. The difficulty compounds with horizon: the probability of a randomly explored trajectory completing every required sub-task falls off geometrically with the number of sub-tasks.

A family of recent architectures attacks this problem by refusing to learn planning and control in the same network. Planning, which requires abstract reasoning over a long horizon, is delegated to a large language model (LLM) that is queried once per episode and returns an ordered list of sub-goals. Control, which requires short-horizon navigation and manipulation, is left to an execution module trained by standard RL on individual sub-tasks, each of which carries a sufficiently dense reward to be learnable. Because the LLM never participates in training and is not queried at every step, the training pipeline remains a conventional RL procedure, and the inference cost of the language model is amortized over an entire episode.

Within this shared skeleton, two different designs of the execution module have been proposed independently and evaluated in the same extended MiniGrid environment. The first keeps the specialists separate: one PPO policy is trained per sub-task, and a gating network performs hard routing, selecting exactly one expert whose output is forwarded as the system action [3]. The second compresses the specialists: the trained experts act as teachers, generate a dataset of state-policy pairs, and a single student network is trained by supervised policy distillation to reproduce their behaviour across all sub-tasks [4]. One design proposes that capacity should stay partitioned, and the other that it can be shared without loss.

Both were reported to outperform an end-to-end PPO agent on multi-step missions, and both used the same planner, the same environment generator, the same reward

Corresponding author: I. Ciganović;
E-mail: iciganovic217d@raf.rs

DOI: 10.5937/OTEH26101C

function, and the same PPO implementation. Neither study set out to compare the two against each other, and neither had reason to: each was answering whether its own design beat a flat baseline. Having built both, we are now in a position to ask the question that neither could. The two designs imply opposite engineering decisions about memory footprint, inference cost, and the effort required to extend the agent with a new skill, and a practitioner choosing between them currently has no evidence to consult.

This paper does not report new training runs. Its contribution is to make the existing evidence legible and to establish what it can and cannot support. Concretely, we (i) express both architectures in one notation and identify the exact point at which they diverge, (ii) re-tabulate all published results under a common metric and flag the confounds that prevent a direct reading, (iii) show that the two execution modules are separated by an effect far smaller than the experimental resolution that produced them, (iv) demonstrate that in both systems the planner is not the limiting component.

2. RELATED WORK

2.1. Sparse reward and hierarchical control

Deep reinforcement learning handles dense-reward control well [5]; the difficulty here is specific to sparse reward. Classical remedies for reward sparsity include curriculum learning, which orders sub-tasks from easy to hard while retaining a single policy [6], and temporal abstraction, which introduces explicitly represented sub-policies. The options framework [7] and its differentiable successor, the option-critic architecture [8], learn both the sub-policies and the termination conditions end to end. FeUdal Networks [9] and h-DQN [10] instead separate a manager that emits goals from a worker that reaches them. These methods share the ambition of the present work, but they learn the decomposition jointly with the policy, which reintroduces a difficult optimization problem precisely where the reward is sparsest.

2.2. Mixture of experts

The mixture-of-experts formulation dates to Jacobs et al. [11], where a gating network combines the outputs of specialized sub-networks. Modern large-scale variants use sparse gating to activate a small subset of experts per input, yielding conditional computation at constant cost [12], [13]. The architecture considered here occupies the extreme of that spectrum: gating is hard, exactly one expert is active, and the experts are complete policies trained in isolation rather than sub-networks of a jointly optimized model.

2.3. Knowledge and policy distillation

Distillation transfers the behaviour of one or more teachers into a student by supervised regression on the teachers' outputs [14]. In the RL setting, policy distillation [15] and actor-mimic [16] showed that several task-specific agents can be compressed into a single network, often with improved stability, because the student learns from a fixed target distribution rather than from non-stationary environment interaction. The variant used in the system studied here is multi-teacher distillation with an important additional degree of freedom: the student may be initialized either randomly or from a policy already trained by RL on all sub-tasks simultaneously.

2.4. Comparing modular and monolithic policies

Direct comparisons between modular and compressed policies are rare, partly because the two are usually proposed as answers to different questions: mixtures of experts for capacity scaling, distillation for deployment efficiency. Where both appear in one study, the comparison is typically incidental. The RL literature also lacks agreed conventions for equalizing the training budget of a modular system against a monolithic one, since the natural unit of accounting, environment steps, is consumed per policy rather than per system. This paper treats that accounting question as a first-class obstacle rather than a detail.

2.5. Language models as planners

Following the emergence of instruction-following language models [17], [18], several systems have used them to decompose tasks for embodied agents. Zero-shot planners extract action sequences directly from an LLM [19]; SayCan grounds the language output in learned affordances [20]; ReAct interleaves reasoning and acting [21]; Voyager maintains a growing skill library in an open-ended environment [22]. Systems that query the model at every step incur an inference cost proportional to episode length, and systems that place the model inside the training loop inherit that cost multiplied by the number of training episodes. The architecture examined here queries the model exactly once per episode, before the first action.

3. UNIFIED ARCHITECTURE

We describe both systems as instances of a single three-stage architecture, which allows the point of divergence to be stated precisely. Figure 1 shows the shared pipeline and the two execution variants.

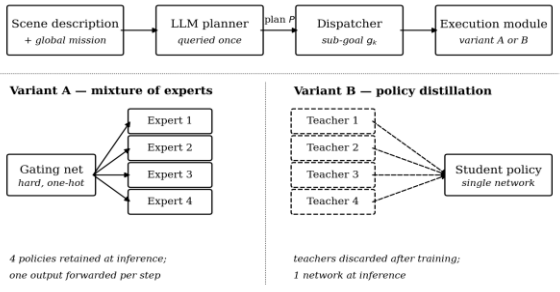


Figure 1. The unified three-stage architecture. Planner and dispatcher are shared; the two systems differ only in the execution module, which either routes between retained specialists (variant A) or replaces them with a single distilled student (variant B).

3.1. Formal setting

The environment is a partially observable Markov decision process. At step t, the agent receives an observation consisting of a 7x7x3 egocentric view, a scalar direction, and a tokenized mission string, and emits an action a_t from the discrete MiniGrid action set. A mission M is solvable by an ordered plan $P = (g_1, ..., g_K)$ of sub-goals drawn from a fixed vocabulary G = {go to goal, go to <object>, toggle <object>, pick up <object>}. The reward is granted only on completion of the global mission and is discounted by the number of steps taken,

$$R = 1 - 0.9\frac{num_steps}{max_steps} \quad (1)$$

so that a perfect score of 1.0 is unattainable and all reported values are bounded well below it.

3.2. Stage 1: planner

A structured textual description of the scene, generated by the environment during scene construction, is combined with the global mission and submitted once to a locally hosted LLM. The model returns an ordered list of sub-goals from G. The planner is not queried again for the remainder of the episode. It takes no part in training, and its parameters are frozen.

3.3. Stage 2: dispatch

Sub-goals are issued to the execution module one at a time as the current mission string. Control returns to the dispatcher when the execution module emits the done action, at which point the next sub-goal in P becomes active. No external oracle verifies that the sub-goal was in fact achieved; the module's own done signal is the only completion event. If the plan is exhausted or the step budget expires, the episode terminates.

3.4. Stage 3: execution, variant A (mixture of experts)

Variant A maintains four independent policies, one per sub-goal type in G, each trained by PPO in isolation on episodes whose entire mission is that sub-goal. A gating network, consisting of an embedding layer, a GRU, and a linear layer, consumes the current mission string and emits a one-hot vector selecting the active expert. Formally, the module computes

$$f(x) = sum_i(w_i(x)f_i(x)) \quad (2)$$

With w_i representing the weight corresponding to the i-th expert, and f_i the output of the said expert. w(x) is constrained to the vertices of the simplex, so exactly one expert's output survives. The gating network is trained by supervised learning on the fixed vocabulary.

3.5. Stage 3: execution, variant B (policy distillation)

Variant B trains the same four experts, then discards them at inference time. Each expert is rolled out in the environment to produce a dataset of pairs (o, p_i_expert(o)), the datasets are pooled, and a single student network with the architecture of one expert is trained by supervised learning to reproduce the pooled action distributions. Two initializations are considered: a randomly initialized student, and a student initialized from a PPO policy previously trained on all sub-tasks simultaneously, referred to below as the all model. Parameter count is constant in the number of skills, and no routing decision is made at inference time.

3.6. Cost model

The two variants differ in a way that is visible before any experiment is run. Let θ denote the parameter count of one expert policy, γ that of the gating network, and N the number of skills. Variant A stores N*θ + γ parameters and must keep all N policies resident, since the identity of the next sub-goal is not known until the dispatcher issues it. Variant B stores θ regardless of N. With the network described in Section 4.2 and N = 4, variant A therefore carries roughly four times the execution-time footprint of variant B, plus a small gating network dominated by its 32 x 1024 embedding layer.

The per-step compute tells a different story. Because gating is hard, exactly one expert is evaluated per step, so the forward-pass cost of variant A equals that of variant B plus one gating evaluation, which is amortized because the gate need only be recomputed when the sub-goal changes. The asymmetry is therefore in memory, not in arithmetic. Extending the agent with a new skill costs one additional PPO training run and one additional resident policy under variant A, against one additional teacher plus a full re-distillation of the student under variant B. Neither published study measured any of these quantities, and Section 5.5 records them as open.

3.7. Baselines

The end-to-end model, denoted no LLM, is a single PPO policy trained directly on complete multi-step missions with no planner at all; it isolates the benefit of the hierarchy itself.

4. EXPERIMENTAL SETUP

Because this paper re-analyses previously published experiments, the setup below is the one common to both source studies [3], [4]. Where the two studies diverge, the divergence is stated explicitly, since it is the substance of Section 5.4.

4.1. Environment

MiniGrid v3.0.0 [23] was extended to allow control over mission complexity. The grid is fixed at 11x11, giving at most four rooms. Object placement is constrained so that generated scenes are always solvable and never obstruct the agent, the goal tile, or a doorway. Doors carry colors and may be locked, in which case a key of the same color is required; keys may be placed inside boxes, adding a level of nesting. The environment additionally emits a

structured textual description of the scene for the planner. Training and evaluation scenes were sampled from the same generator but drawn as disjoint sets.

4.2. Policy and training

All policies use the same feature extractor: a three-branch network processing the image, direction, and mission inputs, whose latent vectors are concatenated and fed to actor and critic heads. Training uses PPO [2] as implemented in Stable-Baselines3 [24], with hyperparameters obtained by population-based bandit search [25]. Each run consists of 20 million steps with 16 parallel environments, and training proceeds in consecutive runs with decreasing learning rate and epoch count until improvement stops. The number of runs was chosen per policy by that stopping rule, not fixed in advance.

4.3. Planner configuration

The planner is qwen3:30b served locally through Ollama at temperature 0.0 with a fixed system prompt describing the available instructions, objects, colors, and the key-door rule. We benchmarked alternative open models on a fixed set of scenes; the selected model matched all reference solutions, while gemma2:27b, deepseek-r1:32b, and llama3:8b matched 13, 11, and 9, respectively [4].

4.4. Metrics

Two quantities are reported. Average reward is the mean of R over an evaluation set of 1000 freshly sampled episodes. Success rate is the fraction of episodes in which the global mission was completed, computed on a manually verified subset of 100 episodes and disaggregated by the number of sub-tasks in the plan. Sub-task-level results are reported as the average reward obtained by a policy when the sub-task is the entire mission.

5. RESULTS

This section re-tabulates the published results of both systems under the common notation of Section 3.

5.1. Sub-task level

Table 1 reports the performance of individual policies on each sub-task. Two facts stand out. First, specialization is clearly beneficial: every specialist outperforms the shared all model on its own sub-task, and every specialist scores exactly zero on sub-tasks it was not trained for, confirming that the experts carry no incidental cross-task competence. Second, and more consequentially, the pick-up and toggle experts differ substantially between the two studies. In [3] they reach 0.57 and 0.47; in [4], where they were fine-tuned from the all model rather than trained from scratch, they reach 0.68 and 0.65.

Table 1. Sub-Task Performance Of Individual Policies (Average Reward)

Policy	Goal	Object	Pick	Togg
Go to goal	0.86	0.00	0.00	0.00
Go to object	0.00	0.72	0.00	0.00
Pick up [3]	0.00	0.00	0.57	0.00
Toggle [3]	0.00	0.00	0.00	0.47
Pick up [4]*	0.00	0.00	0.68	0.00
Toggle [4]*	0.00	0.00	0.00	0.65
All (shared)	0.75	0.65	0.59	0.58

* fine-tuned from the all model rather than trained from scratch.

The gap matters. On two of four sub-tasks, the distillation system started from teachers 11 to 18 points stronger than the experts the MoE system routes between. Neither paper had cause to flag this, since neither was comparing across the two, but any end-to-end comparison that ignores it inherits it as a confound.

5.2. Multi-task execution

Table 2 reports execution modules evaluated across all sub-tasks simultaneously, without a planner. The ordering is informative. Distillation from a randomly initialized student reaches 0.56, which is worse than the all model it was meant to improve upon; distillation from the all model reaches 0.67, which is better. The identity of the initialization, rather than the distillation procedure itself, accounts for a swing of 0.11.

The MoE module has no directly comparable entry in this table, because hard routing makes its aggregate multi-task performance a weighted average of the per-expert values of Table 1 rather than a separately measured quantity. Computing that average over the four sub-task types gives an implied value of approximately 0.66, but this is an arithmetic construction and not a measurement, and it is reported here only to indicate the order of magnitude.

Table 2. Execution Modules Evaluated Across All Sub-Tasks, No Planner

Execution module	Reward	Seeds
All (PPO, shared policy)	0.65	1
Distilled, random init	0.56	1
Distilled, from all model	0.67	1
MoE, hard routing	~0.66*	1

* implied weighted average of Table 1, not an independent measurement.

5.3. End-to-end performance

Table 3 reports complete systems on multi-step missions.

Both hierarchical systems improve substantially on both baselines: against the end-to-end no-LLM agent the margin is 0.12 to 0.14 average reward, and against a non-hierarchical PPO agent it is 0.22 to 0.24. The benefit of delegating planning is therefore not in question.

The comparison between the two execution modules, however, is. The MoE system reports 0.57 and the distilled system 0.55, a difference of 0.02. All figures come from single runs at a fixed seed, which was a reasonable choice when the comparison of interest was against a flat baseline separated by 0.12 or more. It is not enough resolution for a 0.02 gap. RL training at fixed hyperparameters varies across seeds, and [3] reports exactly such a spread in final expert performance over four seeds. We read Table 3 as saying the two execution modules are indistinguishable on the evidence available, rather than as a narrow win for MoE.

Table 3. End-To-End Performance On Multi-Step Missions (1000 Episodes)

System	Reward	Seeds
PPO	0.33	1
End-to-end, no LLM	0.43	1
LLM + distilled [4]	0.55	1
LLM + MoE [3]	0.57	1

5.4. Complexity and the location of failure

Table 4 disaggregates success rate by plan length for the MoE system and its baselines. Performance degrades sharply with horizon: from 98.44 percent on single-sub-task missions to 28.57 percent at three sub-tasks and 14.29 percent at four. The end-to-end agent, by contrast, collapses to near zero as soon as more than one sub-task is required, solving one three-sub-task episode only through an incidental behaviour in which an object lying on the path happened to be the required key.

Crucially, manual verification of 100 planner outputs found no invalid plans in either study; every plan was a correct, if not always optimal, solution to the global mission. Since the planner introduced no failures, all observed degradation is attributable to the execution module. This holds for both variants and is the single most important shared finding: the two systems differ in how they organize execution, and execution is exactly what limits them.

A rough consistency check supports this. Treating sub-task failures as independent, the expected success rate of a three-step pick up, toggle, go-to-goal sequence under the MoE expert rates of Table 1 is 0.57 x 0.47 x 0.86, or about 0.23, against 0.2857 observed. The estimate is close enough to make execution failure the plausible dominant mechanism, though the independence assumption is untested and the observed value exceeds the prediction, which suggests failures are in fact mildly correlated with scene difficulty rather than independent.

Table 4. Success Rate By Plan Length (Percent) calculated on the first 100 episodes

System	1 sub	3 sub	4 sub
PPO	68.75%	0.00%	0.00%
End-to-end, no LLM	68.75%	4.76%	7.14%
LLM + distilled [4]	89.06%	23.81%	14.29%
LLM + MoE [3]	98.44%	28.57%	14.29%
sample size	64	21	14

5.5. Planner accuracy and cost

Because the planner is shared, its properties bound both systems equally. Table 5 reports the model-selection benchmark from [4]. The selected model matched every reference solution on the benchmark set, and two of the three alternatives were close behind, which indicates that on 11x11 scenes the planning task is near saturation for models of this size. That is convenient for the present comparison, since it holds the planner constant, but it also means the benchmark cannot discriminate between planners and gives no evidence about behaviour on larger scenes.

Planning cost, by contrast, is substantial. Reported inference time is 3 to 15 minutes per episode on a single consumer GPU, dominated by the thinking phase of the model, against a per-step execution cost measured in milliseconds. The hierarchy therefore trades a large fixed cost at the start of each episode for a much cheaper trajectory, and any deployment argument that turns on latency must account for this term, which is identical for both execution modules and larger than either.

Table 5. Planner Model Selection Benchmark [4]

Model	Matched	Params
qwen3:30b (selected)	14 / 14	30B
gemma2:27b	13 / 14	27B
deepseek-r1:32b	11 / 14	32B
llama3:8b	9 / 14	8B

Benchmark set of 14 reference scenes. Near-saturation at this scene complexity limits the discriminative power of the benchmark.

5.6. What the evidence does not support

Table 6 summarizes the comparisons required to choose between the two execution modules, together with their current status. Of seven, three are settled by published data, and four are open. None of the published runs report either module at a matched step budget, at matched parameter count, over multiple seeds, or with latency and memory figures. That is not an oversight so much as a consequence of scope: those quantities only become interesting once the two designs are placed side by side, which is what we are

doing here for the first time.

Table 6. Status Of The Comparisons Required To Choose A Module

Comparison	Status
Beats end-to-end baseline	Settled, both
Planner error contribution	Settled, both
Success by plan length	Settled, both
Matched step budget	Open
Matched parameter count	Open
Multi-seed with CIs	Open
Inference latency, memory	Open

6. DISCUSSION

Three things come out of this.

The hierarchy is settled. The execution module is not. Delegating planning to a language model queried once per episode produces a large and consistent improvement over end-to-end RL on multi-step missions, reproduced independently in both studies with different execution modules. We take that as settled. The choice between routing specialists and compressing them rests on a single 0.02 difference measured once, and we would treat it as open.

Initialization matters more than the distillation procedure does. The gap between distilling from random weights and distilling from a trained policy is 0.11, five times the gap between the two architectures. A randomly initialized student ends up worse than the model that would have supplied its initialization. That points to imitation of expert action distributions not being enough on its own to build the representation, and suggests distillation works here as a consolidation step on an already competent policy rather than as a way of training one from nothing.

Execution is what binds in both designs. Across the two studies, the planner produced no errors in 100 manually verified episodes, so every route to a better system runs through the execution module. That changes the question worth asking. Not which module scores higher today, but which improves faster per unit of extra training, and which degrades more gracefully once the planner starts making mistakes. A planner with no observed errors on 11x11 grids will not stay error-free on larger scenes.

The two designs are not just different implementations of the same capability. They fail differently, and that is testable. A routed mixture degrades sub-task by sub-task, since a weak expert damages only the plans that invoke it, and the remaining experts are untouched. A distilled student shares parameters across all skills, so degradation is expected to be diffuse, and a poorly represented sub-task may be compensated or overwritten by the others. The published data are consistent with this reading.

The published evidence supports a weak recommendation, and it follows from the architecture rather than from the reward numbers. Where the skill set is fixed and small, memory is not constrained, and skills are added rarely, MoE architecture is preferable because a new skill costs one training run and cannot disturb existing ones. Where the agent must run on constrained hardware, or where the skill set is large enough, a distilled network is preferable, provided the student is initialized from a multi-task policy rather than randomly. The 0.11 penalty for random initialization is the one effect in this analysis large enough to act on with confidence.

Resolving the comparison needs nothing conceptually new. Both modules should be trained under a declared step-accounting rule, reported under both conventions, since a four-expert MoE granted the same per-expert budget as a single student consumes four times the total experience. Both should be evaluated over at least five seeds with confidence intervals, on at least 200 episodes per complexity bracket, and reported alongside parameter count, memory footprint, and per-step inference latency.

7. REPRODUCIBILITY

All numbers re-analysed in this paper originate from the two source studies [3], [4], both of which published their source code and a video demonstration of the resulting agents. The environment is a modified MiniGrid v3.0.0, the RL implementation is Stable-Baselines3 v2.7.0, and the planner is served through Ollama v0.30.0 with the configuration reproduced in Section 4.3. Training was performed on a single consumer GPU, with each 20-million-step run requiring between four and seven hours of wall-clock time.

We note one obstacle to exact reproduction that affects both studies equally. The number of consecutive training runs per policy was determined by an improvement-based stopping rule applied during training rather than fixed in advance, so the total number of environment steps consumed by each published policy is a function of the observed learning curve. Reproducing the published numbers therefore requires reproducing that decision process, not merely the hyperparameters.

8. CONCLUSION

We have placed two independently proposed execution modules for LLM-planned hierarchical agents, a hard-routed mixture of experts and a multi-teacher distilled policy, into a single architectural framework and re-analysed their published results under common notation. The hierarchical design itself is well supported: both variants improve on an end-to-end PPO agent by 0.12 to 0.14 average reward on multi-step missions, and in both the planner was found to contribute no failures, making the execution module the binding constraint.

The comparison between the modules, however, is not supported. They are separated by 0.02 average reward from single-seed runs, while the experts underlying them differ by up to 18 percentage points on identical sub-tasks

solely because of initialization, and the effect of initialization on the distilled student is itself five times larger than the architectural difference. We conclude that the choice between modularity and compression remains open.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, 2nd ed*, MIT Press, 2018.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "*Proximal policy optimization algorithms,*" arXiv, 2017.

[3] I. Ciganović, A. Pluškoski, M. Jovanović, and J. Vasiljević, "*Evaluation of multi-model architecture against single-model PPO in the MiniGrid environment,*" Electronics, vol. 15, no. 14, art. 3170, 2026.

[4] A. Pluškoski, I. Ciganović, M. Jovanović, and J. Vasiljević, "*Benefits of using LLM for long-term planning with distilled subtask model compared to end-to-end reinforcement learning in the MiniGrid simulator,*" Electronics, vol. 15, no. 9, art. 1921, 2026.

[5] V. Mnih et al., "*Human-level control through deep reinforcement learning*", Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[6] S. Narvekar et al., "*Curriculum learning for reinforcement learning domains: A framework and survey,*" arXiv, 2020.

[7] R. S. Sutton, D. Precup, and S. Singh, "*Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,*" Artificial Intelligence, vol. 112, no. 1–2, pp. 181–211, 1999.

[8] P.-L. Bacon, J. Harb, and D. Precup, "*The option-critic architecture,*" in Proc. AAAI, 2017, pp. 1726–1734.

[9] A. S. Vezhnevets et al., "*FeUdal networks for hierarchical reinforcement learning,*" in Proc. ICML, 2017, pp. 3540–3549.

[10] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "*Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,*" in Proc. NeurIPS, 2016.

[11] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "*Adaptive mixtures of local experts,*" Neural Computation, vol. 3, no. 1, pp. 79–87, 1991.

[12] N. Shazeer et al., "*Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,*" in Proc. ICLR, 2017.

[13] W. Fedus, B. Zoph, and N. Shazeer, "*Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity,*" J. Mach. Learn. Res., vol. 23, no. 120, pp. 1–39, 2022.

[14] G. Hinton, O. Vinyals, and J. Dean, "*Distilling the knowledge in a neural network,*" in NIPS Deep Learning Workshop, 2015.

[15] A. A. Rusu et al., "*Policy distillation,*" in Proc. ICLR, 2016.

[16] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "*Actor-mimic: Deep multitask and transfer reinforcement learning,*" in Proc. ICLR, 2016.

[17] A. Vaswani et al., "*Attention is all you need,*" in Proc. NeurIPS, 2017.

[18] T. Brown et al., "*Language models are few-shot learners,*" in Proc. NeurIPS, 2020.

[19] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "*Language models as zero-shot planners: Extracting actionable knowledge for embodied agents,*" in Proc. ICML, 2022.

[20] M. Ahn et al., "*Do as I can, not as I say: Grounding language in robotic affordances,*" in Proc. CoRL, 2022.

[21] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. ICLR, 2023.

[22] G. Wang et al., "*Voyager: An open-ended embodied agent with large language models,*" Trans. Mach. Learn. Res., 2024.

[23] M. Chevalier-Boisvert et al., "*Minigrid & Miniworld: Modular and customizable reinforcement learning environments for goal-oriented tasks,*" in Proc. NeurIPS Datasets and Benchmarks, 2023.

[24] A. Raffin et al., "*Stable-Baselines3: Reliable reinforcement learning implementations,*" J. Mach. Learn. Res., vol. 22, no. 268, pp. 1–8, 2021.

[25] J. Parker-Holder, V. Nguyen, and S. J. Roberts, "*Provably efficient online hyperparameter optimization with population-based bandits,*" in Proc. NeurIPS, 2020.