Toward Neurosymbolic Reinforcement Learning via Editable Specifications

Vedant Khandelwal[1], **Hong Yung Yip**[1], **Amit Sheth**[1,2]

[1]Artificial Intelligence Institute, University of South Carolina, Columbia
[2]Indian AI Research Organisation (IAIRO), India
vedant, hyip@email.sc.edu, amit@sc.edu

Abstract

Reinforcement learning systems are commonly adapted to new settings by retraining or fine-tuning policies. This default is costly, difficult to audit, and poorly aligned with structured requirement changes such as revised safety rules, new operational constraints, or updated user preferences. We argue for an alternative abstraction: adaptation via edits to an external, human-readable specification that the agent consults at execution time. We propose conditioning decision-making on an editable knowledge graph encoding (i) applicability rules capturing action preconditions and high-level effects, (ii) hard constraints defining feasibility, and (iii) soft preferences shaping tradeoffs among feasible behaviors. Requirement changes become graph edits, not policy rewrites, operationalized through constraint-based action shielding and preference-driven objective shaping. This enables immediate, auditable behavior updates with zero or minimal gradient-based adaptation. We outline edit-based evaluation protocol and highlight open problems in specification design, edit inference, and guarantees.

Introduction

Reinforcement learning (RL) offers a clean training narrative, optimizing interaction to learn a policy that performs well in an environment (Sutton, Barto et al. 1998). Yet many deployment failures arise not from changes in dynamics, but from changes in *requirements*: safety rules update, operational constraints tighten, and user preferences shift. These updates are often local, legible, and easy for humans to state (e.g., "do not use this tool when the system is in maintenance mode"), but RL systems typically respond with retraining or fine-tuning.

Retraining is a poor adaptation primitive for such changes. It is expensive in data and compute, complicates assurance during adaptation, and makes behavior difficult to attribute to intended updates. After fine-tuning, behavioral shifts may reflect incidental correlations in new data rather than requirement changes. In contrast, humans handle updates by editing specifications and expecting immediate consequences.

This approach does involve a deliberate tradeoff. For constraint edits, the shield enforces compliance immediately without touching policy parameters. For preference edits and edits that substantially shrink the feasible action set, the policy may behave suboptimally in underexplored regions; a bounded fine-tuning budget can recover performance. We treat this as an explicit, traceable cost rather than a solved problem, and argue it is preferable to opaque uncertainty introduced by unconstrained fine-tuning on new data.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Requirement Change	Desired Response
New safety rule: "Do not pick fragile items with the gripper above force threshold τ"	Immediate compliance without retraining
Updated preference: "Prefer energy-efficient actions when deadline slack > 5min"	Shift tradeoffs predictably
Revised policy: "Tool X now requires supervisor approval"	Block action until condition met

Table 1: Examples of structured requirement changes. Each is local, human-legible, and should not require policy retraining.

Gaps in Existing Approaches

Several RL paradigms address non-stationarity, but preserve parameter updates as the primary adaptation mechanism. We summarize closest lines of work and gaps our position targets.

Goal-conditioned and meta-RL. Universal value function approximators condition on goals (Schaul et al. 2015), hindsight experience replay improves learning under sparse rewards (Andrychowicz et al. 2017), and meta-RL trains agents to adapt rapidly via learned update mechanisms (Duan et al. 2016) or parameters amenable to fine-tuning (Finn, Abbeel, and Levine 2017). **Gap:** Conditioning improves generalization but does not guarantee compliance when requirements change. If a new rule is introduced (e.g., disallow a class of actions), a learned conditioning mechanism provides no execution-time enforcement. The adaptation primitive remains gradient updates, not structured edits.

Safe and constrained RL. Constrained MDPs and safe exploration (Achiam et al. 2017; Ray, Achiam, and Amodei 2019) optimize under cost thresholds, providing formal frameworks for trading off reward against constraint satisfaction. **Gap:** Constraints are typically fixed at training time; when safety rules change post-deployment, these approaches require re-optimization rather than immediate enforcement. Moreover, soft preferences, contextual tradeoffs like energy efficiency vs. speed, are not represented as editable objects

distinct from the reward function. Our framing makes both constraint edits (changing $\mathcal{A}_G(s)$) and preference edits (shifting tradeoffs via shaping) effective at runtime.

Graph-augmented RL. Knowledge graphs have been used to improve representation and decision-making: KG-based agents learn world-state graphs for action pruning in text games (Ammanabrolu and Riedl 2019), relational inductive biases improve generalization (Zambaldi et al. 2018), and knowledge-infused learning motivates injecting structured knowledge for interpretability (Sheth et al. 2022). **Gap:** In most KG-augmented RL, the graph is a *learned feature* or *state memory*, not an *editable specification* with explicit semantics for rules, constraints, and preferences. Requirement changes are handled through further learning, not through graph edits that deterministically alter admissible actions, undermining auditability in deployed systems. Without an explicit specification artifact, behavioral changes cannot be attributed to specific requirement updates, a critical limitation for systems requiring compliance verification.

Model-based RL and world models. Model-based approaches learn dynamics models or latent simulators for planning and imagination, including compact world models for control (Ha and Schmidhuber 2018) and latent-imagination agents (Hafner et al. 2020). **Gap:** Adaptation proceeds by updating parametric models and re-optimizing behavior; symbolic edits to an external specification are not the primitive operation. In domains where changes are stated as requirement updates, edit-driven adaptation provides a direct abstraction.

Shielding and formal methods. Runtime shields enforce formal constraints via monitors synthesized from temporal-logic specifications (Alshiekh et al. 2018), temporal-logic objectives can guide learning (Hasanbeig, Kroening, and Abate 2020), and planning formalisms like PDDL provide explicit symbolic structure (Fox and Long 2003). **Gap:** Prior shielding work establishes that runtime enforcement of fixed safety properties is feasible and sound. Our contribution is not a new enforcement mechanism; action masking and safe-set projection are well understood. The novelty is in treating the *specification itself* as a first-class editable object: a structured knowledge graph that unifies rules, constraints, and preferences, supports local edits with stable semantics, and connects each edit to an immediate, attributable behavioral change through standard enforcement and shaping.

Position. RL should treat an *editable specification* as a first-class interface for adaptation. We propose conditioning decision-making on an editable knowledge graph encoding: **(i)** applicability rules describing action preconditions and high-level effects, **(ii)** hard constraints defining forbidden behavior, and **(iii)** soft preferences expressing tradeoffs. The knowledge graph is not an auxiliary feature vector; it is an *operational interface* with execution-time semantics. When requirements change, the system adapts by editing the graph, not rewriting the policy.

This leads to **edit-based generalization**: the agent maintains competence across a family of specifications and adapts rapidly when the specification is edited, ideally with zero gradient updates. Across existing approaches, either (i) conditioning is learned but not enforceable, (ii) graphs serve as features rather than editable interfaces, or (iii) symbolic specifications exist but are not integrated with learned policies. We target that missing combination: *editable specifications with execution-time semantics* that make many edits effective immediately. We focus on abstraction levels where rules apply to actions, skills, or options, including discrete decision-making and robotics settings that select among structured behaviors.

Editable Specifications as a First-Class Object

We model the environment as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \gamma)$ (Sutton, Barto et al. 1998) and introduce a *specification object* G that policies are conditioned on: $\pi(a \mid s, G)$. Unlike standard goal conditioning, G is editable and carries execution-time semantics, determining what actions are *allowed* and how tradeoffs among allowed actions are *ranked*.

Specification Contents

We define an editable specification as a triple:

$$G \;=\; (\Psi,\, \mathcal{C},\, \mathcal{P}), \tag{1}$$

where Ψ are **applicability rules**, $\mathcal{C}$ are **hard constraints**, and $\mathcal{P}$ are **soft preferences**. Each component has distinct semantics and edit behavior.

Applicability Rules Ψ. Rules capture structured preconditions and consistency conditions over (i) predicates derived from state and (ii) facts and relations in G. A minimal template is an implication:

$$\rho: \;\; \phi(s) \wedge \psi(G) \;\Rightarrow\; \mathrm{Applicable}(a; s, G), \tag{2}$$

where $\phi(s)$ is a condition on the current state and $\psi(G)$ is a query over the specification. Rules encode human-meaningful requirements, e.g., "only place items marked `dishwasher_safe` in the dishwasher" or "tool X requires material type Y." Unlike hard constraints, rules provide *semantic guidance* for generalization and policy conditioning without requiring runtime enforcement. If unconditional enforcement is required, the rule is promoted to $\mathcal{C}$.

Hard Constraints $\mathcal{C}$. Hard constraints specify non-negotiable requirements that must never be violated:

$$\mathcal{C}(s, a, G) \in \{0, 1\}, \quad \mathcal{C}(s, a, G) = 1 \Rightarrow (s, a) \text{ is forbidden.} \tag{3}$$

Constraints are *enforced at decision time* via shielding (Alshiekh et al. 2018), not merely penalized during training. This ensures immediate compliance when $\mathcal{C}$ is edited, regardless of the policy's learned preferences.

Soft Preferences $\mathcal{P}$. Preferences encode tradeoffs among feasible behaviors as an explicit cost signal:

$$c_{\mathcal{P}}(s, a, G) \in \mathbb{R}_{\geq 0}, \tag{4}$$

capturing statements such as "avoid noisy actions at night" or "prefer energy-efficient actions when time permits." Unlike constraints, preferences may be overridden when necessary. Classical results on reward shaping (Ng, Harada, and Russell 1999) motivate exposing such guidance so updates have controlled, interpretable effects. We note that editing $\mathcal{P}$ shifts the shaping term immediately, but stable realignment of the policy's action distribution may require bounded fine-tuning when the edit substantially changes relative action values.

Execution-Time Semantics

The specification induces a *feasible action set* at each state:

$$\mathcal{A}_G(s) \;=\; \{a \in \mathcal{A} \mid \mathcal{C}(s, a, G) = 0\}. \tag{5}$$

Execution enforces feasibility by restricting selection $\mathcal{A}_G(s)$:

$$\tilde{\pi}_\theta(a \mid s, G) \;\propto\; \pi_\theta(a \mid s, G)\, \mathbf{1}[a \in \mathcal{A}_G(s)]. \tag{6}$$

This makes a concrete commitment: *constraint compliance is an execution-time property*. When $\mathcal{C}$ is edited, $\mathcal{A}_G(s)$ changes by construction, yielding immediate behavioral changes without modifying policy parameters θ.

Preferences incorporated by shaping the effective objective:

$$r_G(s, a) = r(s, a) - \lambda\, c_{\mathcal{P}}(s, a, G), \tag{7}$$

or by reweighting feasible actions as a tie-breaker. The design goal is *predictability*: editing $\mathcal{P}$ produces an auditable, directional shift in tradeoffs without entangling policy parameters.

Key Insight: Edit Semantics. Unlike feature-only graph conditioning (Schaul et al. 2015), our specification has *operational semantics*:

- Editing $\mathcal{C}$ changes $\mathcal{A}_G(s)$ *by construction*, with immediate and guaranteed effect
- Editing $\mathcal{P}$ shifts the shaping term *by construction*, with predictable directional effect

This enables **auditability**: after an edit, compliance and preference shifts can be evaluated directly without attributing behavior differences to opaque parameter updates.

Edit-Based Adaptation

We model requirement updates as edits to the specification:

$$G' = \Delta(G), \tag{8}$$

where Δ is an edit operator (e.g., adding or removing constraints, modifying preference weights). The central generalization objective is **edit-based adaptation**: after observing G', the agent should satisfy updated constraints and track updated preferences with zero or minimal gradient updates.

We distinguish two edit regimes:

- **In-schema edits (target regime):** Modify $\mathcal{C}$ or $\mathcal{P}$ under a fixed schema and action interface. Constraint edits take effect with zero gradient steps by re-evaluating $\mathcal{A}_{G'}(s)$; preference edits shift tradeoffs immediately, with bounded fine-tuning available to restore full competence.
- **Schema-extending edits:** Introduce new options, observable features, or entity types. Limited fine-tuning may be reasonable, but G provides structure for targeted adaptation rather than wholesale retraining.

Why Knowledge Graphs?

Why implement G as a knowledge graph rather than a latent context variable or natural-language rulebook?

Not latent context alone. Latent context helps generalization but is a weak interface for requirement change: it is not directly *editable* (constraints require retraining), not *auditable* (no explicit artifact captures changes), and doesn't provide *enforcement* (compliance is emergent, not guaranteed).

Not natural language alone. Natural language is easy to author but unstable for compliance: it is ambiguous, brittle for enforcement (shields require machine-checkable criteria), and difficult to validate for consistency. Natural language can serve as a front-end for authoring edits, but the execution layer benefits from structured representation.

Why graphs. Knowledge graphs provide: **(1)** local edits with stable semantics, add/delete facts, change attributes, update rules; **(2)** typed schemas enabling compositional edits; **(3)** auditability via provenance and change tracking (Hogan et al. 2021; Yip and Sheth 2024); and **(4)** compatibility with neural conditioning via graph embeddings $z = f_\phi(G)$, supporting a hybrid design where symbolic execution handles enforcement while neural policies provide competence.

Architecture

Figure 1 illustrates our reference architecture, which separates *what the agent should do* (Specification Layer) from *how the agent acts* (Execution Layer). The key property is that edits to the specification affect behavior immediately through execution-time mechanisms, not through retraining.

Specification Layer

Specification store. The system maintains a persistent knowledge graph $G = (\Psi, \mathcal{C}, \mathcal{P})$ encoding applicability rules, constraints, and preferences. The store supports local edits Δ that add or remove constraints, update preference weights, modify rule conditions, and record provenance for auditability (Hogan et al. 2021). When requirements change, edit produces $G' = \Delta(G)$ *without modifying policy parameter* θ.

Graph encoder. An optional encoder computes an embedding $z = f_\phi(G)$ using a graph neural network or relational encoder. This embedding *conditions* the policy (see Figure 1), enabling generalization across entity and relation combinations under a fixed schema. The encoder supports neural generalization but is not the mechanism for compliance.

Execution Layer

The Execution Layer implements the decision loop: State $\rightarrow$ Policy $\rightarrow$ Shield $\rightarrow$ Action $\rightarrow$ Environment. Constraints and preferences from G influence this loop at two points.

Policy conditioned on G. The agent uses a policy $\pi_\theta(a \mid s, G)$ conditioned on the specification embedding. Conditioning supports learning families of behaviors appropriate to different specifications, but *compliance is not guaranteed by conditioning alone*; it is ensured by the downstream shield.

Constraint enforcement (shield). Hard constraints $\mathcal{C}$ induce a feasible action set $\mathcal{A}_G(s) \subseteq \mathcal{A}$. The shield filters the policy's output, restricting selection to feasible actions:

$$a \sim \pi_\theta(\cdot \mid s, G) \quad \text{restricted to } \mathcal{A}_G(s). \tag{9}$$

In discrete domains, this is action masking; in continuous domains, option-level shielding or safe-set projection (Alshiekh et al. 2018). Editing $\mathcal{C}$ yields immediate behavioral change: $\mathcal{A}_{G'}(s)$ is recomputed on the next step without touching θ.

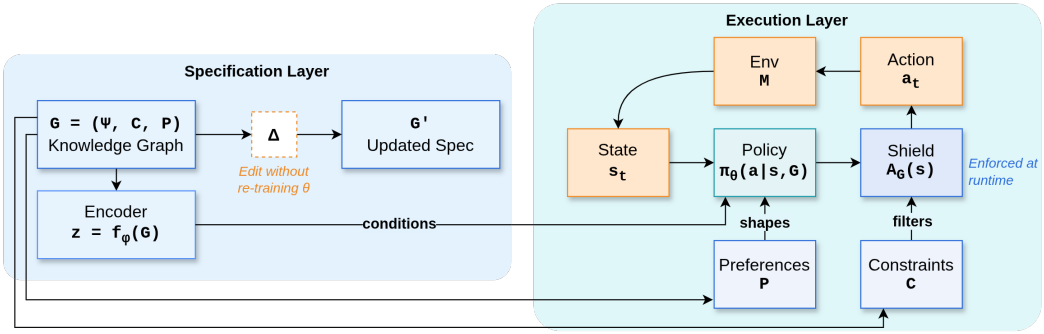

Figure 1: Reference architecture for neurosymbolic RL via editable specifications. The **Specification Layer** (left) maintains knowledge graph $G = (\Psi, \mathcal{C}, \mathcal{P})$ updated via edits Δ without modifying policy parameters θ. The **Execution Layer** (right) uses graph embedding to condition policy, enforces constraints $\mathcal{C}$ via shielding, and applies preferences $\mathcal{P}$ via reward shaping.

Preference application. Soft preferences $\mathcal{P}$ shape the policy's objective by biasing selection among feasible actions:

$$r_G(s,a) = r(s,a) - \lambda\, c_{\mathcal{P}}(s,a,G), \tag{10}$$

or equivalently by reweighting within $\mathcal{A}_G(s)$. Editing $\mathcal{P}$ shifts tradeoffs predictably (Ng, Harada, and Russell 1999); behavioral alignment may require a bounded fine-tuning budget when the edit substantially changes relative action values.

Execution loop. Each decision step proceeds as follows:

1. Retrieve current specification G
2. Derive feasible set $\mathcal{A}_G(s)$ from constraints $\mathcal{C}$
3. Compute shaped reward r_G from preferences $\mathcal{P}$
4. (Optional) Compute embedding $z = f_\phi(G)$
5. Sample action $a \sim \pi_\theta(\cdot \mid s, G)$ subject to $a \in \mathcal{A}_G(s)$
6. Execute a, observe next state from environment $\mathcal{M}$

When requirements change, an edit Δ produces G', and the same loop executes under G' with updated $\mathcal{A}G'(s)$ and rG'. No gradient updates are required for constraint edits; preference edits may benefit from a bounded fine-tuning budget.

Compatibility with standard RL. We do not require a new optimizer. Value-based and actor-critic methods integrate naturally: action masking implements the shield, and shaped rewards implement preferences. The novelty is the **edit-driven interface**: requirement changes are edits to G that take effect through enforcement and shaping, with parameter updates treated as secondary and budgeted.

Open Problems and Research Agenda

We highlight research directions that make specification editing a reliable adaptation primitive:

Edit inference from sparse feedback. How can we infer minimal, consistent edits Δ from violations, corrections, or preference comparisons? This connects to preference learning (Christiano et al. 2017) and cooperative inverse RL (Hadfield-Menell et al. 2016), but targets structured specification updates rather than reward function estimation.

Schema design and stable grounding. What schema designs ensure edits remain compositional and symbols ground reliably to state features and option-level actions? Grounding failure is a central risk: if the symbolic vocabulary does not map cleanly onto the agent's observable state space, even a correct edit may produce no behavioral change or an incorrect one. We regard reliable grounding as a prerequisite for the guarantees we claim (Hogan et al. 2021).

Guarantees for enforcement and preference compliance. Can we make $\mathcal{A}_G(s)$ provably sound (no forbidden actions taken) without being overly restrictive? For preferences, the challenge is harder: when does a preference edit produce stable behavioral alignment without retraining, and how quickly does a bounded fine-tuning budget recover performance after edits that substantially restrict the feasible set (Alshiekh et al. 2018; Hasanbeig, Kroening, and Abate 2020)?

Robotics abstractions via options. How should option libraries be co-designed with specification vocabularies so that edits constrain and prioritize stable behaviors rather than fragile low-level control (Sutton, Precup, and Singh 1999)?

Benchmarks for edit-based generalization. We need evaluation protocols measuring: (1) *post-edit success with zero updates*, can agent succeed under G' without retraining? (2) *constraint violations*, are constraints respected? (3) *recovery curves*, how does performance improve with bounded fine-tuning budget K? Candidate domains include decision-making (MiniGrid-style), PDDL-derived planning MDPs, and control (Safety Gym-style) (Chevalier-Boisvert et al. 2023; Ray, Achiam, and Amodei 2019).

Conclusion

We argued for treating an editable specification G, represented as a knowledge graph, as a first-class adaptation interface for RL. Requirement changes become edits $G \to G'$ that yield immediate, auditable behavior changes through constraint enforcement and preference shaping, with no gradient updates required for in-schema edits. This involves a deliberate tradeoff: the adaptation budget is explicit and bounded, and any suboptimality from restricted feasible sets is a traceable cost rather than an opaque one. We hope this positions edit-based adaptation as a benchmarkable form of generalization in structured decision-making, bridging the auditability requirements of deployed systems with the flexibility of learned policies.

Acknowledgments

This work is supported in part by NSF grants #2119654, "RII Track 2 FEC: Enabling Factory to Factory (F2F) Networking for Future Manufacturing" and SCRA grant "Enabling Factory to Factory (F2F) Networking for Future Manufacturing across South Carolina".

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *International conference on machine learning*, 22–31. PMLR.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Ammanabrolu, P.; and Riedl, M. 2019. Playing text-adventure games with graph-based deep reinforcement learning. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, Volume 1 (Long and short papers)*, 3557–3565.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Chevalier-Boisvert, M.; Dai, B.; Towers, M.; Perez-Vicente, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P. S.; and Terry, J. 2023. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36: 73383–73394.

Christiano, P. F.; Leike, J.; Brown, T.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Duan, Y.; Schulman, J.; Chen, X.; Bartlett, P. L.; Sutskever, I.; and Abbeel, P. 2016. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv preprint arXiv:1611.02779*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, 1126–1135. PMLR.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Ha, D.; and Schmidhuber, J. 2018. World Models. *CoRR*.

Hadfield-Menell, D.; Russell, S. J.; Abbeel, P.; and Dragan, A. 2016. Cooperative inverse reinforcement learning. *Advances in neural information processing systems*, 29.

Hafner, D.; Lillicrap, T.; Ba, J.; and Norouzi, M. 2020. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*.

Hasanbeig, M.; Kroening, D.; and Abate, A. 2020. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 1–22. Springer.

Hogan, A.; Blomqvist, E.; Cochez, M.; de Melo, G.; Gutierrez, C.; Kirrane, S.; et al. 2021. Knowledge Graphs. *ACM Computing Surveys*, 54(4).

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, 278–287. Citeseer.

Ray, A.; Achiam, J.; and Amodei, D. 2019. Benchmarking Safe Exploration in Deep Reinforcement Learning. *arXiv preprint arXiv:1910.01708*.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–1320. PMLR.

Sheth, A.; Gaur, M.; Roy, K.; Venkataraman, R.; and Khandelwal, V. 2022. Process knowledge-infused ai: Toward user-level explainability, interpretability, and safety. *IEEE Internet Computing*, 26(5): 76–84.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1–2): 181–211.

Yip, H. Y.; and Sheth, A. 2024. The EMPWR platform: Data and knowledge-driven processes for the knowledge graph lifecycle. *IEEE Internet Computing*, 28(1): 61–69.

Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; et al. 2018. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*.