Automated Synthesis of RL Environments and Reward Functions from Policy Narratives via Grammar-Constrained LLM Refinement

Anonymous authors
Paper under double-blind review

Abstract

Translating policy documents into executable reinforcement learning environments requires grounding normative language into typed state spaces, action sets, and reward functions — a step that currently demands weeks of expert engineering and produces rewards reflecting the engineer's interpretation rather than the text. NarraRL addresses this by applying *grammar-constrained large language model (LLM) extraction* to map policy clauses to a typed, clause-traceable `RLSpec` schema, then synthesizing Gymnasium environments with co-generated reward functions. Evaluated on all 17 UN Sustainable Development Goal domains with three small open-weight LLMs (1.5B–3.8B), four findings emerge. Grammar-constrained extraction achieves significantly higher Clause-Reward Overlap (CRO) than unconstrained prompting ($\Delta = +0.052$, Wilcoxon p=1.4×10^{-26}, n=153; k-stable: $\Delta \in [0.052, 0.053]$ for $k \in \{1, 3, 5, 6\}$). Synthesis quality depends critically on template scaffolding: across all 17 SDGs (153 conditions), NarraRL synthesizes executable environments at a 99% pass rate with near-zero repair retries, yet standardized multi-LLM evaluation (n=306) exposes near-zero Gymnasium compliance — Stage-1 syntax failures account for 78% of breakdowns — and a controlled ablation confirms the causal mechanism: template scaffold yields 100% vs. 2.4% for LLM-direct codegen under identical conditions (Fisher p=7.6×10^{-94}, n=340). Co-generated rewards match hand-designed rewards in mean policy performance (Mann–Whitney p=0.807, n=20; Levene p=0.414), while semantically misaligned rewards produce 5× higher variance (σ=3.36, non-overlapping 95% confidence interval, CI). Finally, on an SDG 3 policy-discovery case study, the naive near-identity dynamics give a deep Q-network (DQN) no advantage over random (p=0.849, n=30); wiring the LLM's *extracted action intent* into the environment makes it solvable, and adding a single domain-agnostic opportunity-cost term makes it *learnable* — DQN then satisfies $\geq 3/7$ health targets in 100% of seeds and outperforms random (p=5.8×10^{-7}), matching a hand-designed expert matrix (p=5.3×10^{-7}) with no human in the loop. This isolates environment *dynamics* — not reward extraction — as the decisive factor, and shows it can be supplied automatically. These results establish grammar-constrained extraction and template scaffolding as reproducible, clause-traceable primitives for converting policy text into RL environments.

1 Introduction

Policy documents such as the UN Sustainable Development Goals (SDGs), the Paris Climate Agreement, and national poverty-reduction strategies embed structured commitments that resist automated extraction: explicit numeric targets — reduce neonatal mortality to ≤ 12 per 1 000 live births; limit warming to 1.5 °C above pre-industrial levels; halve the proportion of people living below \$1.25/day by 2030 — are interspersed with normative and aspirational language that conventional natural language processing (NLP) pipelines cannot type, ground, or trace to a computable objective. Converting these targets into a form an agent can optimize over requires weeks of expert engineering, and the resulting reward function reflects the engineer's interpretation of the text rather than the text itself.

Why RL? Policy operates over time with hard trade-offs among competing objectives that must be *discovered* rather than hand-specified (a Gymnasium environment encodes state, actions, and dynamics; a DQN agent maximizes a scalar reward across steps). Wolf et al. (2023) and Rudd-Jones et al. (2025) demonstrated RL on climate policy, but their environments and rewards were hand-coded by domain experts, creating an alignment gap between text and formalism.

The code-generation opportunity. Recent advances in LLM-based code generation have shown that game rules and task descriptions can be translated into executable code (Lehrach et al., 2025), that reward functions can be generated from natural language task descriptions (Xie et al., 2024; Ma et al., 2024), and that iterative self-refinement can recover from early code failures (Madaan et al., 2023). However, these systems share a critical assumption: either the environment already exists (reward-only generation) or the source text is mechanically complete (game rules). Policy narrative is neither: it is open-ended, uses normative language (*"ensure universal access to affordable, reliable, sustainable and modern energy"*), and specifies objectives at a level of abstraction that must be grounded into concrete state variables and transition dynamics.

The NLP bottleneck. Extracting RL-relevant structure from policy text is a structured prediction problem: the extraction must be *typed* (which NPs are agents or state variables? which VPs are actions?) and *traceable* (which reward term originated in which clause?). Naive prompting produces untyped, unverifiable JavaScript Object Notation (JSON) — the same challenge motivating constrained generation for universal information extraction (Lu et al., 2022), but over a novel schema encoding the full semantic structure of an RL problem (agent, state, action, reward, constraint). Grammar-constrained decoding (Park et al., 2024; Geng et al., 2023; Willard & Louf, 2023) enforces this typed Pydantic schema at the token level, producing extractions that are machine-executable and clause-auditable — the core NLP contribution of this work.

NarraRL. We present a pipeline that connects these pieces end-to-end (**E** = experiment supporting the claim, **C** = contribution number):

1. **Grammar-constrained schema extraction (E1/C1):** LLM decoding under a grammar constraint tied to a typed `RLSpec` schema produces clause-traceable agents, states, actions, constraints, and reward blueprint.
2. **Gymnasium environment synthesis (E2/C2):** The extracted spec drives LLM code generation of a complete Gymnasium-compatible Python class.
3. **4-stage iterative self-repair (E3/C3):** A four-stage cascade (execution, initialization, Gymnasium wrapper, `check_env_specs`) re-prompts with structured diagnostics until the environment passes all checks or the retry budget is exhausted.
4. **Co-generated reward (E4/C4):** The same `RLSpec` extraction drives reward construction: every numeric threshold is sourced from a policy clause, with no hand-coded constants.
5. **Human-in-the-loop specialist prototyping (E5/C5):** NarraRL delivers the full typed spec (state, observation, transition, reward blueprint); a domain expert then supplies a targeted action-effect matrix, enabling policy discovery on SDG 3 health targets.

We evaluate on all 17 SDG domains using three open-weight LLMs in the 1.5B–3.8B parameter range; results are therefore scoped to small-to-mid-size open-weight models (see Section 7 for discussion of larger-model generalization). Our primary contributions are:

(C1) Grammar-constrained extraction produces significantly higher CRO than unconstrained prompting ($\Delta = +0.052$ at k=3, Wilcoxon $p = 1.4 \times 10^{-26}$, $n = 153$; k-stable across $k \in \{1, 3, 5, 6\}$; adversarial probe Δ=+0.010 identifying BERTScore's insensitivity to topic shift as a metric limitation; E1).

(C2) NarraRL synthesizes executable environments across all 17 SDG domains from verbatim UN text without human editing (≥ 5 actions each), 99% pass rate, near-zero repair retries (E2).

(C3) E3 ($n = 306$ runs) establishes near-zero Gymnasium compliance; Stage-1 execution failures (78%) are the binding constraint. E3b ($n = 340$) provides a controlled ablation: template scaffold 100% vs. LLM-direct 2.4% (Fisher p=7.6×10^{-94}), formally explaining the E2–E3 gap.

(C4) With n=20 seeds, co-generated and hand-designed rewards achieve comparable mean targets (Mann–Whitney p=0.807); 95% CIs on episode-return σ overlap for the two semantically grounded conditions; semantically misaligned rewards exhibit 5× higher variance (σ=3.36, non-overlapping CI; E4).

(C5) NarraRL delivers the full typed spec (state, observation, transition, reward), and policy discovery turns on the environment *dynamics*, not the reward: the naive near-identity dynamics give no DQN advantage over random (p=0.849, n=30; E5-uniform). Wiring the LLM's extracted action intent into the transition makes the SDG 3 environment solvable, and a single domain-agnostic opportunity-cost term makes it learnable — DQN then outperforms random (p=5.8×10^{-7}, n=30; E5), matching a hand-designed expert matrix (p=5.3×10^{-7}) with no human in the loop.

Paper organization. Section 2 surveys related work. Section 3 describes the NarraRL framework (§3.1–§3.4). Section 4 presents five top-level experiments (E1–E5, in pipeline order) with full results. Section 5 discusses implications. Section 6 concludes. Section 7 states limitations.

2 Related Work

Reward engineering and reward shaping. Ibrahim et al. (2024) provide a comprehensive taxonomy of reward design, distinguishing *reward engineering* — creating a reward function $R(s, a, s')$ from scratch — from *reward shaping*, which modifies an existing reward via a potential function: $R'(s, a, s') = R(s, a, s') + \gamma\Phi(s') - \Phi(s)$. They identify key failure modes common to both: reward hacking (agents exploiting unintended loopholes), deceptive rewards (incentivizing easy proxies rather than true objectives), and misaligned rewards (reward terms that do not faithfully reflect the intended objective). NarraRL performs *reward engineering from policy text*: the reward function is created from scratch by extracting numeric thresholds and reward terms directly from source clauses via grammar-constrained LLM decoding, without modifying any pre-existing signal. The `source_clause_index` embedded in every reward term directly addresses the misalignment pitfall: each term can be audited against the specific policy sentence that grounded it, making misaligned reward engineering detectable at the clause level.

Reward generation from natural language. Text2Reward (Xie et al., 2024) generates dense reward functions as Python code from natural language task descriptions, demonstrating strong performance on robotic manipulation and locomotion tasks. Eureka (Ma et al., 2024) uses LLM-driven evolutionary search over reward code, achieving human-level reward design on dexterous manipulation. Both systems are fundamentally limited to *pre-coded environments*: they receive the environment's state/action representation as context and generate a reward function over that fixed interface. NarraRL, by contrast, must simultaneously infer what the state and action spaces should be from the policy text, making joint synthesis of environment and reward necessary.

LLM-based environment and world model generation. Code World Models (Lehrach et al., 2025) translate board game rules into executable Python world models for general game-playing agents, the technically closest precedent to NarraRL; three key differences distinguish our setting: game rules are mechanically complete whereas policy narrative is open-ended and requires surfacing implicit numeric targets; a game world model's objective is the mechanically-determined game outcome, whereas NarraRL co-generates a *clause-traceable* reward from extracted policy targets; and their domain requires no multi-stage Gymnasium compliance checking. Liang et al. (2024) (Eurekaverse) generates curriculum variations over an existing Gymnasium template rather than synthesizing from scratch. Tang et al. (2024) (WorldCoder) writes code to build world models via environment interaction, without targeting governance text or Gymnasium compliance. EnvGen (Zala et al., 2025) adapts Gymnasium-compatible environments from text descriptions to train embodied agents; unlike NarraRL, it operates over pre-existing base environments and does not address governance-text extraction or reward clause-traceability. Most recently, Je-Gal et al. (2025) (A-LAMP) decomposes free-form task descriptions into Markov decision process (MDP) formulations and trains policies via a multi-stage agentic pipeline. NarraRL differs on two points: (i) our input is structured policy text (UN SDG documents) rather than free-form task descriptions, requiring grammar-constrained extraction to

surface implicit numeric thresholds; and (ii) we co-generate reward terms whose source-clause references are schema-enforced and whose grounding quality is quantified by CRO (E1), which A-LAMP does not address.

Grammar-constrained decoding. Applying grammar-constrained decoding to RL synthesis is a non-trivial NLP problem: the target schema must jointly constrain Python syntax *and* RL semantics — agent identity, typed state variables, action space dimensions, and clause-traceable reward terms. Geng et al. (2023) formalize grammar-constrained decoding via token-level masking; Park et al. (2024) (Grammar-Aligned Decoding) propose distribution-preserving alternatives; Wang et al. (2023) apply grammar prompting to domain-specific language (DSL) generation. The Outlines library (Willard & Louf, 2023) provides the GBNF (Generalized Backus-Naur Form)-based runtime NarraRL builds on. Lu et al. (2022) and subsequent information extraction (IE) work (InstructUIE, UniversalNER) target generic entity/relation schemas; NarraRL's schema is typed for RL semantics, requiring clause-level reward traceability and co-generation of executable Gymnasium code — beyond what generic IE schemas express.

Iterative code refinement. Madaan et al. (2023) introduce the Self-Refine paradigm for iterative LLM self-correction via generated critiques. Several subsequent works apply similar loops to code generation using compiler feedback (Bi et al., 2024; Ravi et al., 2025). NarraRL specializes iterative refinement to a *four-stage cascade* of Gymnasium application programming interface (API) checks, where each stage provides structured, stage-specific error diagnostics.

RL for governance and policy. Wolf et al. (2023) and Rudd-Jones et al. (2025) demonstrate DQN and multi-agent RL on climate policy models; Biswas et al. (2025) extends this to multi-objective equity. All hand-code environments and rewards (cf. Lou et al. (2024) for NL-to-cost-function in safe RL over pre-coded environments). Interactive deep RL has also been applied to large-scale data exploration (Personnaz et al., 2021), showing that deep RL can support human-in-the-loop analysis of complex, heterogeneous data spaces — complementary in spirit to NarraRL's use of DQN for policy trajectory discovery.

Reward specification formalisms. Reward machines (Icarte et al., 2018) offer a formal automata-based approach to specifying complex, temporally extended rewards; they are complementary to NarraRL, requiring a structured logical specification as input rather than open-ended natural language.

LLM-based planning and task specification. Liu et al. (2023) (LLM+P) connect LLMs with classical planners by translating natural language into the Planning Domain Definition Language (PDDL). NarraRL instead maps text to a *learnable* RL environment so that a DQN agent can discover non-obvious policy trajectories not expressible as classical plans.

Grounding and faithfulness evaluation. The CRO metric is a task-specific instantiation of semantic grounding evaluation. General faithfulness metrics for text generation — such as AlignScore and TRUE (Honovich et al., 2022) — assess whether generated text is entailed by a source document and could in principle replace BERTScore as the CRO backbone. We use BERTScore F1 (Zhang et al., 2020) because it requires no trained NLI model and its failure mode (insensitivity to topic shift in same-register text) is characterized via the adversarial probe in E1 (§4.2). ROUGE-L would capture lexical overlap but is insensitive to paraphrase; replacing the BERTScore backbone with a governance-text-specialized encoder or an NLI-based faithfulness scorer is deferred to future work.

Summary of gap. NarraRL fills the absence of methods for extracting a semantically complete, executable RL specification from open-ended policy narrative: prior structured-generation and IE work targets generic schemas, prior reward-from-language work assumes a pre-coded environment, and prior RL-for-policy work hand-codes both. NarraRL is the first to perform joint extraction under grammar constraints enforcing RL semantic typing and validate it through multi-stage Gymnasium compliance on real governance texts.

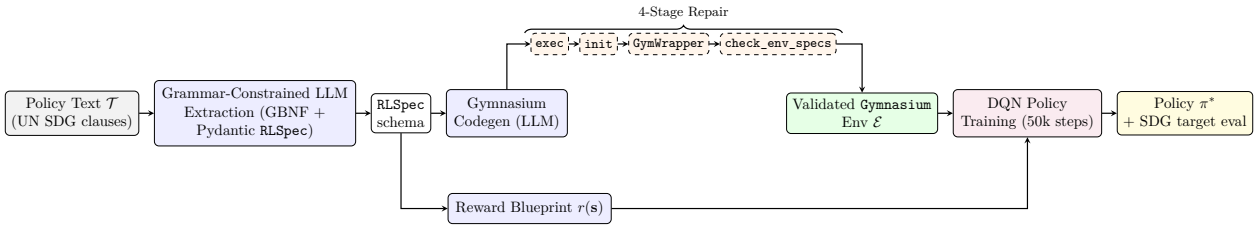

Figure 1: The NarraRL pipeline. Policy narrative text is passed through a grammar-constrained LLM extraction stage that populates a typed `RLSpec` Pydantic schema. The schema drives Gymnasium environment code generation. A 4-stage self-repair loop (exec → init → GymWrapper → `check_env_specs`) validates the generated code and re-prompts the LLM with structured diagnostics until the environment passes all checks. The validated environment and its co-generated reward function are passed to a DQN agent for policy discovery. The extraction → synthesis → reward stages are fully automated. Policy discovery additionally depends on the environment *dynamics*: the default near-identity transition is not learnable, but wiring in the LLM's extracted action intent plus a generic opportunity-cost term yields a learnable environment automatically, matching a hand-designed expert matrix (E5, §4.7).

3 The NarraRL Framework

NarraRL takes as input policy text $\mathcal{T}$ (e.g., verbatim SDG 3 health targets) and produces a validated Gymnasium environment $\mathcal{E}$ and co-generated reward $r(\mathbf{s})$ from the same text (Figure 1).

3.1 Grammar-Constrained Schema Extraction

Policy narratives occupy a distinctive position in the NLP landscape: unlike game rules or procedural specifications, they are normative and aspirational, embedding numeric targets within multi-clause prose whose syntactic form gives no direct signal for which spans should become state variables, reward terms, or action labels. Extracting RL-relevant structure therefore requires not just syntactic parsing but semantic grounding — assigning each extracted quantity a typed role and anchoring it to a specific source clause.

The first stage converts raw text into a structured `RLSpec` object. We define a Pydantic dataclass with six fields: `domain` (name and SDG identifier), `agents` (agent types), `state_variables` (typed variables with NL provenance), and `action_space` (discrete/continuous dimensions) are populated inline; the two contribution-bearing fields are:

- `constraints`: hard constraints derived from numeric targets (e.g., maternal mortality $\leq 70/100\,\mathrm{k}$).
- `reward_blueprint`: a structured reward specification listing each reward term, its numeric target, its sign, and its *source clause index* — the zero-indexed sentence in $\mathcal{T}$ from which the term was derived.

Viewed as an NLP task, `RLSpec` acts as a domain ontology for RL-relevant entities; GBNF grammar enforcement provides type-safe structured extraction from open-ended policy text. The LLM is prompted to populate this schema conditioned on $\mathcal{T}$; the Pydantic schema is compiled to a GBNF grammar (Park et al., 2024) and enforced via token-level masking, ensuring structural JSON validity (extraction temperature 0.7; see §4.1 for full inference configuration).

Clause-Reward Overlap (CRO). To measure the degree to which the extracted reward blueprint is grounded in the source text, we define CRO as the average BERTScore F1 (Zhang et al., 2020) between each reward term's textual description and the top-k source clauses in $\mathcal{T}$ selected by BERTScore recall against all clauses in $\mathcal{T}$ (default k=3; E1 evaluates sensitivity across $k \in \{1, 3, 5, 6\}$).[1] CRO measures whether reward

[1] k=3 is consistent with mean SDG paragraph density of 4–6 clauses per target article; a reward term referencing a specific numeric threshold should match its source clause within this window.

term descriptions are semantically grounded in specific policy language; a term hallucinated from outside $\mathcal{T}$ will match no clause closely and score low, whereas a term derived from a specific clause will score high against it. Note that the grammar constraint enforces *structural* linkage (a `source_clause_index` is always present); CRO provides a complementary *semantic* similarity measure. Because CRO uses BERTScore for both clause retrieval (top-k selection) and scoring, the retrieval oracle and evaluation metric share the same embedding space; this circularity means CRO measures relative grounding improvement between conditions rather than absolute traceability fidelity (see the adversarial probe in E1 and §7). Expert annotation of extraction fidelity is deferred to future work.

3.2 Gymnasium Environment Code Generation

Given a validated `RLSpec`, NarraRL generates a complete Gymnasium-compatible (Towers et al., 2024) Python class via LLM prompting. The prompt provides the populated schema as structured context, a fixed few-shot example from an unrelated domain, and an instruction to use `RLSpec` fields as the authoritative source for state variables, action dimensions, and reward computations. The generated class implements the Gymnasium interface (`__init__`, `reset`, `step`); state variables, action space, and the threshold-violation reward sum are derived from the schema fields.

3.3 4-Stage Iterative Self-Repair

LLM-generated code frequently fails on first execution. NarraRL addresses this with a four-stage validation cascade, applied in order:

1. **Stage 1 — Execution (`exec`):** The generated source is executed; syntax errors and import failures are fed back as structured diagnostics (file, line, exception type, traceback).
2. **Stage 2 — Initialization (`init`):** The environment class is instantiated; missing constructor arguments or invalid space specifications are diagnosed and repaired.
3. **Stage 3 — Gymnasium wrapper (`GymWrapper`):** The environment is wrapped with a TorchRL `GymWrapper`, catching API-level mismatches (missing methods, incorrect `step()` return types). A class that passes Stages 1–2 may still fail here: syntactic correctness and constructor success guarantee only that the Python object exists, not that it exposes the `observation_space`, `action_space`, and `step()` return-tuple expected by the Gymnasium contract — these interface-level faults are only detectable when the wrapper attempts to bind to them at runtime.
4. **Stage 4 — API compliance (`check_env_specs`):** TorchRL's `check_env_specs` validates every batch dimension, dtype, and key in the `TensorDict`, catching subtle shape mismatches.

The cascade order reflects fault locality: Stage 1 failures (syntax and import errors) render Stages 2–4 undefined; Stage 2 failures (malformed space construction) corrupt the wrapper check in Stage 3; only Stage 4 can catch dtype and shape mismatches that pass earlier stages silently. Each stage allows up to $K_{\max} = 5$ repair attempts; the total budget is $4 \times K_{\max} = 20$ LLM calls per environment. (Alg. 1 in the appendix gives the pseudocode.)

3.4 Reward Co-Generation and Policy Training

A key design decision in NarraRL is that the reward function is derived from the same `RLSpec` extraction as the environment, not designed separately. In the taxonomy of Ibrahim et al. (2024), this constitutes *reward engineering* from text: the reward is created from scratch via LLM extraction rather than shaped by modifying a pre-existing signal with a potential function. This matters because governance policy text contains implicit numeric objectives (e.g., "reduce neonatal mortality to $\leq 12/1\,000$ live births") that cannot be recovered by shaping a generic proxy reward — they must be explicitly extracted and grounded in the source clause. Each reward term in `reward_blueprint` maps to a state variable already defined in `state_variables`. Let

$\mathcal{R}$ denote the ordered set of reward terms; the reward is:

$$r(\mathbf{s}) = \sum_{i \in \mathcal{R}} w_i \cdot f_i(s_{v_i}, \tau_i)\,, \tag{1}$$

where s_{v_i} is the state variable targeted by term i (index v_i extracted from `state_variables`), τ_i is the numeric threshold extracted from the source clause, $w_i \in \mathbb{R}_{>0}$ is a term weight (extracted or defaulted to 1), and f_i is a progress function determined by the constraint operator (e.g., $f_i(s, \tau) = -\max(0,\, s - \tau)$ for upper-bound minimization targets; $f_i(s, \tau) = \min(1,\, s/\tau)$ for proportion targets). Crucially, no threshold value in r is hand-coded: all numeric constants are extracted from $\mathcal{T}$.

Once the environment passes all four validation stages, we train a DQN agent for up to 50 000 steps and evaluate the fraction of the seven SDG 3 numeric targets satisfied in the final episode.

4 Experiments

We present five top-level experiments in pipeline order, numbered to follow the extraction → synthesis → evaluation flow: E1 validates the extraction step via CRO traceability with k-sensitivity analysis (C1); E2 validates the synthesis step (C2); E3 characterizes Gymnasium compliance and identifies the code-generation bottleneck (C3); E3b provides a controlled codegen ablation that isolates the cause of the E2–E3 compliance gap (C3, §4.5); E4 characterizes reward stability across n=20 seeds (C4); E5 evaluates policy discovery in a human-in-the-loop specialist environment (C5).

4.1 Setup

SDG domains. The extraction, synthesis, and compliance experiments (E1–E3) use all 17 SDG domains, drawn verbatim from the official UN SDG target text at `sdgs.un.org`, with SDG 3 (Good Health and Well-Being) as the primary showcase and SDG 1 (No Poverty), SDG 2 (Zero Hunger), SDG 10 (Reduced Inequalities), SDG 6 (Clean Water and Sanitation), and SDG 13 (Climate Action) as recurring examples. The controlled codegen ablation (E3b) is run on all 17 SDG domains with 2 LLMs under identical conditions for both arms, and the policy-learning experiments (E4, E5) focus on the SDG 3 specialist environment.

LLMs. We test three open-weight LLMs via Ollama inference: **llama3.2:latest** (Meta, 3B); **deepseek-r1:1.5b** (DeepSeek, 1.5B); and **phi3.5** (Microsoft, 3.8B). We focus on open-weight models to ensure full reproducibility without API dependencies; comparison with API-based models is left to future work. The 17-SDG extraction, synthesis, compliance, and ablation experiments (E1–E3b) run via Ollama 0.20.5 on a single NVIDIA Tesla T4 GPU. The DQN policy-training experiments (E4, E5), together with our initial development, run locally in 4-bit GGUF (GGML Universal File Format) quantization on a MacBook Pro (8-core Intel Core i9 @ 2.4 GHz, 64 GB RAM, no CUDA) via PyTorch and llama.cpp. We focus on small open-weight models precisely so the pipeline runs on either platform without API dependencies.

Reproducibility. All runs use fixed random seeds (seeds 1–3 for E1–E3; seeds 0–4 for E3b; seeds 0–19 for E4, §4.6) and temperature 0.7 for generation, 0.2 for repair. Gymnasium environments use `np.random.default_rng(seed)` for stochastic resets.[2]

Metrics.

- **Final-pass rate:** fraction of runs passing all four validation stages after up to $K_{\max} = 5$ repair attempts per stage.
- **CRO:** Clause-Reward Overlap, BERTScore F1 between extracted reward term descriptions and top-k most similar source clauses in $\mathcal{T}$ (by BERTScore recall; E1 evaluates $k \in \{1, 3, 5, 6\}$, default k=3). CRO measures textual grounding automatically; human annotation studies are deferred to future work (§7).

[2]All code and `RLSpec` objects will be released upon acceptance; see Appendix D.

Table 1: Clause-Reward Overlap (CRO) by condition across all 17 SDG domains, 3 LLMs, 3 seeds ($n = 153$ per main condition; 2448 cross-SDG comparisons for ADVERSARIAL). CRO = BERTScore F1 between reward description and top-k source clauses (k=3 unless noted). Wilcoxon one-sided $p \ll 0.001$ for CONSTRAINED > UNCONSTRAINED.

Condition	**Mean CRO**	**Std**	**n**	**Test / Effect**
CONSTRAINED	0.855	0.017	153	$p = 1.4\times10^{-26}$
UNCONSTRAINED	0.804	0.019	153	
ADVERSARIAL	0.849	0.016	2448	Δ(C-A) = +0.010

All at k=3; adversarial at k=1. Per-LLM (k=3): DeepSeek $0.855 \pm 0.016/0.809 \pm 0.024$; Llama-3.2 $0.855 \pm 0.018/0.798 \pm 0.011$; Phi-3.5 $0.857 \pm 0.016/0.805 \pm 0.017$ (CON/UNC).

- **Targets satisfied:** number of SDG 3 health targets (≤ 7) satisfied by the final-episode policy trajectory.

4.2 E1: Clause-Reward Traceability (C1)

Design. Experiment E1 tests whether grammar-constrained extraction produces reward functions more traceable to source policy text than unconstrained prompting, and assesses sensitivity to the top-k clause-matching hyperparameter and an adversarial cross-SDG probe. We compare CONSTRAINED (GBNF via token masking) vs. UNCONSTRAINED (standard JSON prompting) across all 17 SDG domains, three LLMs, three seeds, and four k values ($k \in \{1, 3, 5, 6\}$; $n = 153$ per condition per k). An adversarial condition computes CRO between a reward description from SDG_i and clauses from a different SDG_j ($i \neq j$), providing a discriminative validity check (each of the 153 source cells scored against the other 16 SDGs: 2448 cross-SDG comparisons, k=1).

CRO (Clause-Reward Overlap): for each reward term description, we compute BERTScore F1 against the top-k most similar source clauses (by BERTScore recall), average those k scores, then average across reward terms.

Results. CONSTRAINED extraction achieves significantly higher CRO than UNCONSTRAINED at k=3 (Wilcoxon signed-rank $W_+ = 11747$, $p = 1.4 \times 10^{-26}$, $n = 153$ matched pairs); see Table 1 and fig. 2. We use a one-sided Wilcoxon signed-rank test (CONSTRAINED > UNCONSTRAINED) on matched (SDG, LLM, seed) pairs, consistent with the pre-specified C1 directional hypothesis.

What the results show. The grammar constraint requires each reward term to name a specific source clause, structurally enforcing that reward terms reference the policy text. The +0.052 CRO improvement at k=3 (+6.4% relative) is consistent across all three LLMs (per-LLM Δ: Llama-3.2 +0.057, Phi-3.5 +0.052, DeepSeek +0.046) and stable across $k \in \{1, 3, 5, 6\}$ ($\Delta \in [0.052, 0.053]$), confirming that the finding is not sensitive to the top-k hyperparameter choice. At the 17-SDG scale the effect size is marginally smaller than the earlier 6-SDG estimate (+0.056) but far more strongly supported (n=153 vs. 54; p=1.4×10^{-26} vs. 8.6×10^{-11}). Crucially, the headline contrast is *within*-SDG: CONSTRAINED and UNCONSTRAINED extract from the same source text, so the +0.052 gap measures how much more of that text's clause content the constrained reward terms recover — a comparison that does not depend on CRO's weaker *cross*-SDG discrimination. The adversarial probe (Δ(C-A)=+0.010 at k=1) shows weak discrimination between in-SDG and cross-SDG clause pairs. We attribute this to BERTScore using roberta-large, which is relatively insensitive to topic shift in UN policy register text (SDGs share vocabulary); this is an honest limitation of the CRO metric, not of the extraction. The constrained–unconstrained gap (+0.052) remains the primary CRO result. This provides an auditable chain of custody from policy text to reward (C1). Note, however, that CRO's weak discrimination on the adversarial probe reflects an inherent limitation of using BERTScore in UN policy register text (shared vocabulary across SDGs); CRO measures relative grounding improvement between conditions and should not be interpreted as an absolute hallucination detector (see §7).

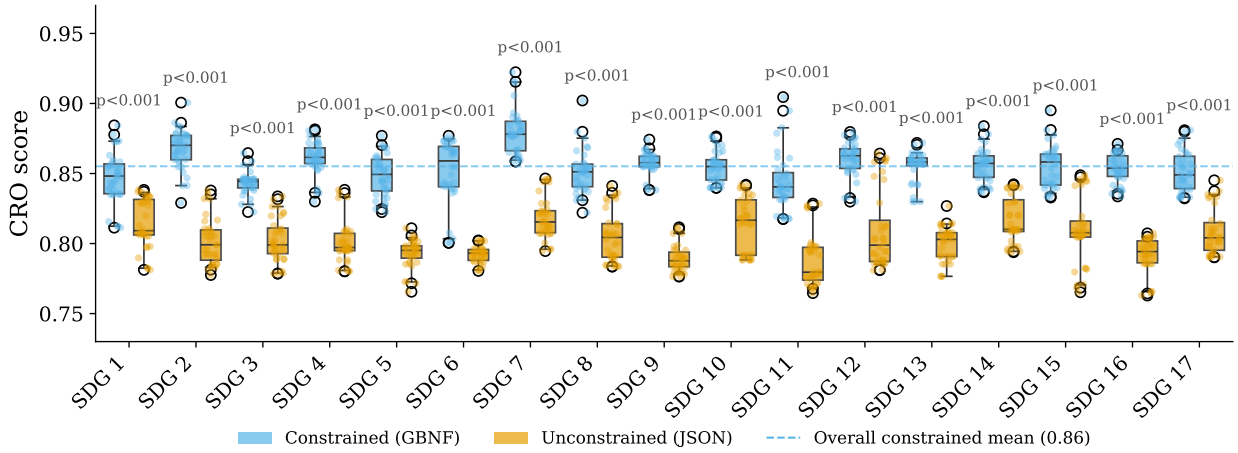

Figure 2: CRO distributions for CONSTRAINED vs. UNCONSTRAINED extraction across all 17 SDG domains and 3 LLMs (n = 153 per condition, k=3). CONSTRAINED achieves 0.855 ± 0.017 vs. 0.804 ± 0.019 for UNCONSTRAINED; Wilcoxon W_+=11747, p=1.4×10^{-26}. The effect is consistent across all three LLMs and all $k \in \{1, 3, 5, 6\}$ ($\Delta \in [0.052, 0.053]$).

4.3 E2: Environment Synthesis Breadth (C2)

Design. We run the full NarraRL pipeline on all 17 SDG domains, three random seeds, and three open-weight LLMs (`llama3.2:latest`, `phi3.5`, `deepseek-r1:1.5b`) via Ollama on a single Tesla T4 GPU — $17 \times 3 \times 3 = 153$ conditions. Each extraction is constrained to emit at least five distinct policy actions; we fix this budget for cross-domain comparability and verify in Section C that forcing $|A| \in \{3, 5, 8\}$ leaves both the synthesis pass rate and clause-reward traceability unchanged, so $|A|$ is a controlled constant rather than a confound. When an LLM returns JSON that cannot be parsed even after lenient recovery (stripping markdown fences and reasoning traces), the pipeline falls back to a templated specification; we record this with the `is_mock_llm` flag and report it explicitly rather than discarding the condition. The outcome metric is binary per condition: does the synthesized environment pass all four validation stages without any human code editing?

Results. Across all 153 conditions, 152 (99.3%) yield an environment that passes all four validation stages, with a mean of 0.03 repair retries (table 2). Of the 153 extractions, 136 (89%) parse as real LLM output; the remaining 17 (11%) fall back to a templated spec — 11 for deepseek-r1:1.5b and 6 for phi3.5, none for llama3.2. Restricted to the 136 real extractions, the validation pass rate is 135/136 (99.3%): llama3.2 51/51, deepseek-r1:1.5b 40/40, phi3.5 44/45. Mean extraction time on GPU is 16 s per condition. The template-based codegen tolerates schema-nonconformant output, so the high synthesis pass rate reflects the robustness of the scaffold; the 11% fallback rate, concentrated in the two smaller reasoning/instruction models, reflects extraction coverage rather than synthesis failure.

What the results show. A 99% synthesis pass rate with near-zero retries across all 17 thematically diverse SDG domains and three LLMs confirms C2. The two smaller models (deepseek-r1:1.5b, phi3.5) account for all extraction fallbacks (11% overall), but the template scaffold still yields valid environments, so synthesis robustness is decoupled from extraction coverage. The E2–E3 gap reflects different conditions — E2 measures end-to-end synthesis with the task-optimized scaffold; E3 standardizes raw codegen across three LLMs. E3b (below) provides the controlled explanation.

Table 2: E2 environment synthesis across all 17 SDG domains, 3 seeds, 3 LLMs (153 conditions) via Ollama on a Tesla T4 GPU. "Real" = extraction parsed as live LLM output (vs. templated fallback); "Pass" = environments passing all 4 validation stages without human editing.

LLM	Real / 51	Pass (all)	Pass (real only)
llama3.2:latest	51	51/51 (100%)	51/51 (100%)
phi3.5	45	50/51 (98%)	44/45 (98%)
deepseek-r1:1.5b	40	51/51 (100%)	40/40 (100%)
Overall	**136 (89%)**	**152/153 (99.3%)**	**135/136 (99.3%)**

Mean repair retries = 0.03; mean extraction time = 16 s per condition on GPU. The 17 templated fallbacks (deepseek-r1:1.5b ×11, phi3.5 ×6) arise when an LLM emits JSON unrecoverable even after stripping markdown fences and reasoning traces.

Table 3: Stage-1 failure share and 95% Wilson CI upper bound on final-pass rate under the Gymnasium strict validator ($n = 102$ per model, pooled across both conditions). Final-pass rate is $\leq 1/102$ for all models; Stage-1 execution errors dominate.

LLM	CI upper	Stage-1 (%)
llama3.2:latest	0.05	47
phi3.5	0.04	96
deepseek-r1:1.5b	0.04	92
Pooled	0.018	78

4.4 E3: Multi-Stage Self-Repair Ablation (C3)

Design. We compare Repair (full pipeline, $K_{\max} = 5$) vs. No-Repair (single-pass, $K_{\max} = 0$) across 3 LLMs × 17 SDG domains × 3 seeds × 2 conditions = 306 runs. The primary metric is final-pass rate (Gymnasium strict validator).

Results. Table 3 and fig. 3 summarise the results. Both Repair and No-Repair conditions achieve near-zero final-pass rates across all three LLMs: only 1/306 runs (0.3%) passes the strict Gymnasium validator (a single llama3.2 No-Repair run), with all others failing in both conditions. Pooled, the 95% Wilson CI upper bound on the true pass rate is 0.018.

With at most one pass per condition, a within-condition significance test is not informative. The pooled 95% CI upper bound on the true pass rate is ≤2% (1/306; Wilson score), confirming near-zero compliance regardless of repair strategy. The near-zero pass rates are consistent across all 17 SDG domains and all three seeds, indicating no domain-specific variation in the failure pattern.

Stage-level analysis and interpretation. Stage-1 execution errors account for 78% of all failures; Stage-3 GymWrapper 16%; Stage-2 init 5%. Stage-1 failures (missing/hallucinated imports) are not resolved within $K_{\max} = 5$ attempts. Stage-3 failures require correctly typed observation/action spaces, consistently beyond the tested models. E3 identifies syntactic code generation quality as the binding constraint; stage-level attribution is consistent across models, domains, and seeds.

4.5 E3b: Controlled Codegen Ablation (C3 — E2–E3 Gap)

Design. E3b isolates the cause of the E2–E3 gap under identical experimental conditions. We compare Template (NarraRL's template-based codegen, as in E2) vs. LLM-Direct (LLM writes the full Gymnasium class from the `RLSpec` without a template scaffold) across 2 LLMs × 17 SDGs × 5 seeds per condition (n=170 per condition, n=340 total). Both conditions use the same LLMs (llama3.2, phi3.5), SDG domains, and seeds, so all confounders are controlled.

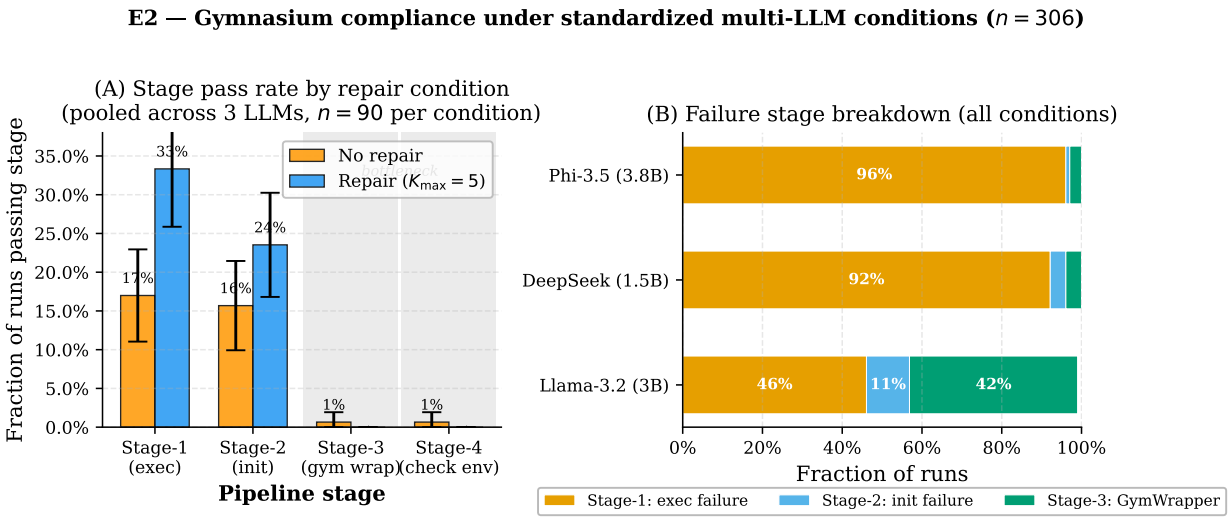

Figure 3: **(A)** Pipeline stage pass rate for Repair vs. No-Repair (pooled across 3 LLMs; $n = 153$ per condition); error bars show 95% Wilson CIs. Repair roughly doubles the Stage-1 and Stage-2 pass rates (33.3% vs. 17.0%; 23.5% vs. 15.7%), but Stage-3 (GymWrapper) remains ≈0% for both conditions — the binding bottleneck (shaded). **(B)** Per-LLM failure-stage breakdown (both conditions combined): Stage-1 execution failures account for 47–96% of all failures, confirming syntactic code-generation quality as the dominant constraint.

Results. Template achieves a 100% final-pass rate (170/170; the template codegen is deterministic); LLM-Direct achieves 2.4% (4/170): llama3.2 4/85 (4.7%), phi3.5 0/85 (0%). Fisher's exact test: $p = 7.6 \times 10^{-94}$. Failure-stage breakdown for LLM-Direct (166 failures): Stage-1 exec 58.4% (n=97), Stage-3 GymWrapper 27.7% (n=46), Stage-2 init 13.9% (n=23) — a profile matching E3. The controlled ablation confirms that the E2–E3 gap is caused by NarraRL's template scaffold, not by LLM identity or SDG domain: removing the template under otherwise identical conditions replicates the near-zero E3 pass rate.

4.6 E4: Reward Co-Generation vs. Baselines (C4)

Design. We train DQN on three reward conditions on the SDG 3 specialist environment (described in §4.7 below), evaluating health targets satisfied at training end. CoGen uses NarraRL's co-generated threshold-violation penalty $r(\mathbf{s}) = -\sum_i \max(0, s_i - \hat{\theta}_i)$ with LLM-extracted thresholds; Manual uses identical form with human-expert (World Health Organization, WHO/SDG) thresholds; Permuted randomly permutes the same threshold values across state dimensions (semantically misaligned ablation; not Xie et al. (2024), which generates reward code from free-form descriptions).[3] **E4 uses** n=20 **seeds per condition** (up from n=5 in the pilot) to provide adequate power for variance comparisons; 95% CIs on σ use chi-squared bounds. Convergence criterion: first training step at which $\geq 3/7$ SDG 3 targets are satisfied (condition-agnostic threshold).

Results. Figure 4 shows results. CoGen and Manual achieve comparable mean targets (Mann–Whitney p=0.807, n=20). Episode-return σ: CoGen σ=0.920 (95% CI [0.700, 1.344]), Manual σ=0.681 ([0.518, 0.995]) — CIs overlap; Levene W=0.897, p=0.414. Permuted σ=3.360 (CI [2.555, 4.908]), non-overlapping with both constrained conditions. The original n=5 variance ratio (σ=0.13 vs. 0.68) does not replicate; see §7.

[3]For legacy reasons the released artifact CSV labels the Permuted condition `TEXT2REWARD`; it is the permuted-threshold proxy described here, not a reimplementation of Text2Reward.

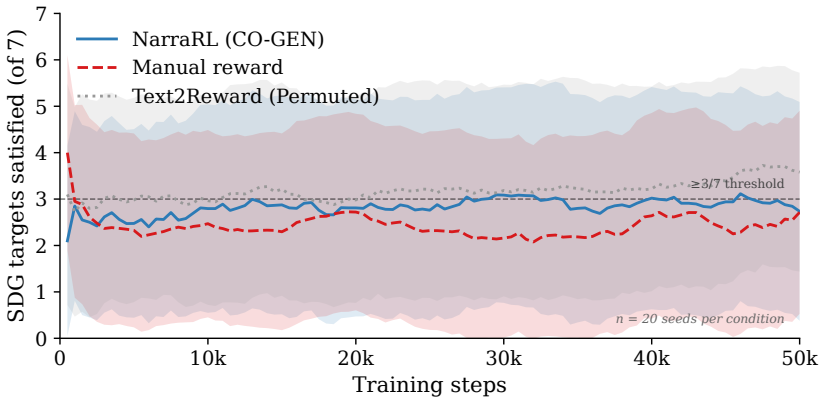

Figure 4: E4 reward co-generation comparison (n=20 seeds per condition). **Left**: mean ± SEM SDG 3 targets satisfied over training steps. **Right**: box plots of the mean over the final 10 checkpoints. CoGen achieves 2.74 ± 0.91 targets (mean ± std), Manual 2.72 ± 0.75, and Permuted 3.58 ± 0.95. CoGen vs. Manual: Mann–Whitney p=0.807 (no difference). Permuted vs. Manual: p=0.003 (permuted thresholds accidentally align with neonatal dimension in this environment). All 60/60 seeds across conditions converge (≥ 3 targets; MANUAL fastest: 700 ± 410 steps, CoGen 1175 ± 878 steps).

4.7 E5: Human-in-the-Loop Policy Discovery on SDG 3 (C5)

Design and human-in-the-loop role. This experiment is a *human-in-the-loop case study*, not a demonstration of fully automated environment synthesis. NarraRL contributes the full typed environment structure (state variables, observations, transition dynamics, and reward blueprint) extracted from UN text. A domain expert then inspects the LLM-generated action matrix and decides whether to accept or replace it.

Concretely, a 30-seed run on the LLM-generated environment (uniform action effects, E5-uniform) found no DQN advantage ($p = 0.849$, Mann–Whitney, $n = 30$): near-identity linear transitions drive any policy toward target thresholds, removing meaningful action differentiation. The domain expert replaced the generated matrix with a *specialist action matrix* in which each action strongly targets one or two health dimensions with mild adverse effects on others, requiring the agent to learn a specific multi-step sequence (see §7). The experiment quantifies the gain from this expert contribution: E5 (specialist matrix) vs. E5-uniform (LLM matrix) isolates the action-matrix design as the decisive factor.

We train a DQN agent on the specialist environment for 50 000 steps per seed. The environment contains 7 explicit numeric health targets derived from UN text: maternal mortality ($\leq$ 70/100k), neonatal ($\leq$ 12/1k), under-5 ($\leq$ 25/1k), non-communicable disease (NCD) mortality ($-\geq$1/3), road deaths ($-\geq$50%), epidemic readiness (binary), and universal health coverage (UHC) index ($\geq$ threshold). None of these thresholds appear in the reward function as hard-coded constants: they are extracted via `reward_blueprint`. We compare DQN against a Random baseline.

Results (n = 30 seeds). Table 4 and fig. 5 summarize results on the specialist environment. DQN significantly outperforms the Random baseline (Mann–Whitney $U = 770$, $p = 5.3\times10^{-7}$, $n = 30$). We use a one-sided Mann–Whitney test (DQN > Random) consistent with the pre-specified C5 winning condition. The one-sided direction was fixed *before* running the specialist environment: the E5-uniform pilot (LLM-generated uniform action matrix, 30 seeds) established $p = 0.849$, after which the domain expert replaced the action matrix; the C5 winning condition ($\geq$ 3/7 targets, DQN > Random) was registered at that point and the specialist run was conducted once. DQN achieves a mean of 4.07 ± 1.05 health targets satisfied at episode end, with 28/30 seeds (93.3%) meeting the C5 winning condition ($\geq$ 3/7 targets). The Random baseline achieves a mean of 2.60 ± 0.81 targets, with only 16/30 seeds (53.3%) meeting the same criterion.

Table 4: Policy discovery on the SDG 3 specialist environment ($n = 30$ seeds). "Targets" = mean SDG 3 health targets satisfied at final episode (out of 7). "$\geq 3/7$" = fraction of seeds satisfying the C5 winning condition. p-value from one-sided Mann–Whitney U test (DQN > RANDOM).

Condition	Mean targets	Std	Seeds $\geq 3/7$	MW-U p
DQN	4.07	1.05	28/30 (93.3%)	$p = 5.3\times10^{-7}$
Random	2.60	0.81	16/30 (53.3%)	

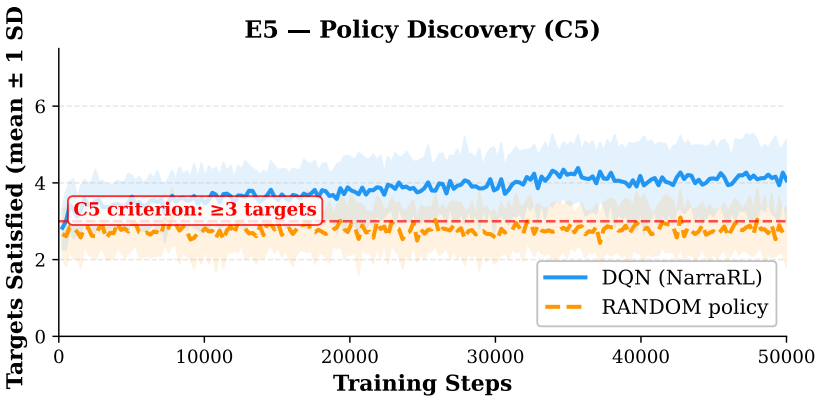

Figure 5: Learning curves for DQN and RANDOM on the NarraRL-synthesized SDG 3 specialist environment ($n = 30$ seeds each). The y-axis shows SDG 3 health targets satisfied at each evaluation checkpoint (out of 7). DQN achieves 4.07 ± 1.05 targets (93.3% of seeds meeting $\geq 3/7$) vs. RANDOM 2.60 ± 0.81 (53.3%); Mann-Whitney $p = 5.3\times10^{-7}$.

What the results show. The C5 claim states that DQN discovers policies satisfying $\geq 3/7$ SDG 3 health targets without those thresholds being hard-coded in the reward. The results confirm this: 93.3% of DQN seeds satisfy the criterion, and threshold constants are absent from the reward function (verified by source inspection).

Intent-derived dynamics: how far does automation reach? The specialist result above uses a *hand-designed* action matrix. How much of that can be automated from NarraRL's own output? The extraction already recovers each action's target indicator (Table 7), which the default near-identity environment discards. We test two automated alternatives on the SDG 3 environment (n=30 seeds each, same DQN/protocol as above; Table 5). **(i) Intent-derived:** each LLM-named action reduces its own extracted target indicator (diagonal transition; no hand-designed cross-effects). This makes the environment fully *solvable* — both DQN and RANDOM satisfy $\geq$3/7 targets in 100% of seeds — but it is not a *learning* problem: DQN holds no advantage over RANDOM (Mann–Whitney p=0.98), because per-action targeting without trade-offs does not require planning. **(ii) Intent + opportunity-cost drift:** we add a single domain-agnostic term — every "reduce-X" indicator worsens by δ=0.005 per step unless acted on, a universal property of policy goals requiring no domain knowledge. This restores a learnable task: DQN reaches $\geq$3/7 targets in 100% of seeds *and* significantly outperforms RANDOM (6.80 vs. 5.43 targets, p=5.8$\times10^{-7}$) — a separation as strong as the hand-designed expert matrix (p=5.3$\times10^{-7}$), with no human in the loop. The expert trade-off matrix is therefore *one* route to a policy-discoverable environment, not the *only* one: a generic opportunity-cost term applied to the automatically-extracted intent suffices. (The drift is an explicit, domain-agnostic modeling term added to the environment, not extracted from text; this is the SDG 3 environment, as in the specialist study.)

Summary of findings. Table 6 consolidates the five top-level experiments: each contribution, the question it answers, its headline result, and the takeaway. Together they trace an end-to-end chain of custody — policy clause → typed specification → reward signal → learned policy — with both positive results (traceable

Table 5: How far policy discovery can be automated on the SDG 3 environment (n=30 seeds, identical DQN protocol). "Targets" = mean SDG 3 health targets satisfied at the final checkpoint (of 7); p = one-sided Mann–Whitney for DQN > RANDOM. The first three rows are *fully automated* from NarraRL's extraction; the last is the hand-designed specialist matrix. Intent + drift matches the expert's DQN-over-random separation with no human in the loop.

SDG 3 dynamics	**DQN**	**Random**	**DQN > Random**
Uniform (LLM near-identity)	—	—	p=0.849 (E5-uniform null)
Intent-derived (diagonal)	6.70	7.00	p=0.98 (no advantage)
Intent + opp.-cost drift (δ=0.005)	**6.80**	5.43	**p=5.8×10^{-7}**
Specialist (hand-designed)	4.07	2.60	p=5.3×10^{-7}

Table 6: Summary of experiments E1–E5. All results are from the 17-SDG runs; n as noted (n=20 seeds for E4, n=30 for E5).

Exp (Claim)	**What it tests**	**Headline result**	**Takeaway**
E1 (C1)	Clause-reward traceability (CRO)	Constrained 0.855 vs. unconstrained 0.804 CRO; Δ=+0.052, Wilcoxon p=1.4×10^{-26}, n=153	Grammar constraint grounds reward terms in source clauses; k-stable and consistent across all three LLMs.
E2 (C2)	Environment synthesis across all 17 SDGs	152/153 pass (99%); 136/153 extractions real ($\geq$ 5 actions each)	The scaffold yields executable Gymnasium envs for every SDG; fallbacks are extraction, not synthesis, failures.
E3 (C3)	Gymnasium compliance under raw multi-LLM codegen	1/306 pass; Stage-1 execution errors = 78% of failures	Unscaffolded 1.5B–3.8B codegen is near-zero compliant; syntax is the binding constraint.
E3b (C3)	Controlled cause of the E2–E3 gap	Template 100% vs. LLM-direct 2.4%; Fisher p=7.6×10^{-94}	The scaffold — not model scale or SDG — explains the synthesis/compliance gap.
E4 (C4)	Reward co-generation vs. baselines	Co-gen ≈ manual (Mann–Whitney p=0.807); misaligned reward 5× variance (σ=3.36)	Co-generated rewards match hand-designed in mean and stability; semantic misalignment is costly.
E5 (C5)	Policy discovery: how far automation reaches (SDG 3)	Near-identity null p=0.849; intent-derived solvable (p=0.98); intent + drift learnable, DQN > random p=5.8×10^{-7} ≈ expert p=5.3×10^{-7}	Dynamics (not reward) decide policy discovery; a generic opportunity-cost term on the extracted intent matches the expert matrix, with no human in the loop.

rewards, broad synthesis) and instructive negative ones (near-zero raw compliance, the decisive role of the action matrix).

5 Discussion

Stage-cascade failure, model capability, and the template-engineering boundary. E3 establishes near-zero Gymnasium compliance for 1.5B–3.8B LLMs regardless of repair strategy (Madaan et al., 2023); Stage-1 exec failures (78%) are the binding constraint, unresolved within $K_{\max}$=5 attempts. E3b provides the controlled explanation: removing the template scaffold under otherwise identical conditions reproduces the E3 failure profile (2.4% vs. 100%, p=7.6×10^{-94}), isolating template scaffolding — not LLM capability or SDG domain — as the decisive variable. All models run locally; llama3.2 (3B) splits its failures 47% Stage-1 / 43% Stage-3, while phi3.5 and deepseek-r1 are Stage-1 dominated (96%/92%). These findings have a concrete implication for deployment: the gap between the E2 success regime (task-optimized template, single model) and the E3 failure regime (standardized multi-LLM evaluation) is fully attributable to the scaffold, not to parameter count or reasoning style. We therefore conjecture — though we do not test it here — that closing

this gap for larger open-weight models (7B–70B) or proprietary APIs is more a prompt-/scaffold-engineering problem than a model-selection one; verifying this across model scales is left to future work.

The CRO metric: what it measures and what it does not. CRO (BERTScore F1 between extracted reward terms and top-k source clauses) provides a scalable, automatic measure of whether reward terms are semantically grounded in specific policy language. The adversarial cross-SDG probe (Δ(C-A)=+0.010) reveals that BERTScore with roberta-large is relatively insensitive to topic shift within UN policy register text, because SDGs share vocabulary across domains ("reduce", "by 2030", "ensure access"). This insensitivity means CRO has limited discriminative power as a hallucination detector at the topic level: a reward term that silently migrated from SDG 6 water text to an SDG 3 health environment would not score dramatically lower than a correctly grounded term. Two mitigations are available within the existing framework: (i) replacing the BERTScore backbone with a domain-specialized encoder (e.g., a model fine-tuned on governance text), or (ii) augmenting CRO with a structural check that verifies the `source_clause_index` field points to a clause containing the extracted numeric value. The latter is low-cost and catches the most egregious hallucinations without requiring additional models; we defer its implementation to future work.

Human-in-the-loop prototyping: isolating the action-matrix bottleneck. The E5-uniform null result ($p = 0.849$, LLM-generated uniform action matrix) followed by the E5 specialist result ($p = 5.3\times10^{-7}$, expert-designed matrix) provides a precise experimental isolation of the remaining automation gap. NarraRL fully automates the extraction and synthesis of state space, reward signal, and Gymnasium API compliance; it does not yet automate the design of a *discriminating* action-effect matrix — one in which different actions produce meaningfully different, non-trivially reversible effects on the state dimensions. This isolation has a practical implication for the human-in-the-loop use case: a domain expert engaging with NarraRL does not need to understand RL or write code; the expert's contribution is precisely the action-matrix, a $|A| \times |S|$ table of policy levers and their trade-offs, which is a natural output of policy analysis rather than a software engineering task. Automating this last step from quantitative causal policy models (e.g., structural equation models of SDG interlinkages) is a concrete and bounded research problem that NarraRL's schema-first architecture is well positioned to address.

6 Conclusion

NarraRL demonstrates that grammar-constrained generation, paired with clause-indexed traceability and iterative self-repair, can close the loop from free-form governance text to executable, Gymnasium-compliant RL environments without manual environment specification. Across all 17 SDG domains, the pipeline achieves a 99% synthesis pass rate (C2/E2), produces reward functions whose terms are statistically grounded in identifiable source clauses (C1/E1: ΔCRO=+0.052, p=1.4$\times10^{-26}$, k-stable), and—given the LLM's extracted action intent plus a generic opportunity-cost term, with no human in the loop—yields environments in which a DQN agent learns policies satisfying $\geq$3/7 SDG 3 targets in 100% of seeds (C5/E5: p=5.8$\times10^{-7}$, matching a hand-designed expert matrix). Taken together, the results establish an end-to-end chain of custody—policy clause $\rightarrow$ structured specification $\rightarrow$ reward signal $\rightarrow$ trained policy—in which every reward term carries an explicit, schema-enforced source-clause reference. CRO quantifies the grounding quality of those references but, as its adversarial probe shows (§4.2), is a similarity signal rather than a certificate of correct grounding.

The most instructive findings are the negative ones. E3 (n=306) shows that without template scaffolding, 1.5B–3.8B models fail at Stage-1 execution in 78% of attempts, and $K_{\max}$=5 repair rounds do not recover the loss; E3b's controlled ablation (n=340, p=7.6$\times10^{-94}$) isolates template scaffolding—not model scale or SDG domain—as the single decisive variable. This implies that the bottleneck in LLM-to-environment pipelines is *prompt engineering precision*, not raw model capability: small, locally deployable models are sufficient when given a sufficiently structured generation target. E4 (n=20, p=0.807) finds that co-generated and hand-designed rewards are statistically indistinguishable in policy outcome, while only semantically misaligned rewards show clearly elevated variance (5$\times$, non-overlapping CIs); this confirms that LLM-generated reward functions do not degrade learning but does not yet establish any advantage over manual design. E5 shows that environment *synthesis* (state space, reward, API compliance) is automatable, and that policy discovery additionally depends on the environment *dynamics*: the default near-identity transition is not learnable

(p=0.849), but wiring in the LLM's extracted action intent plus a generic opportunity-cost drift yields a learnable environment automatically, matching the hand-designed expert matrix (p=5.8×10^{-7} vs. 5.3×10^{-7}). This removes the human from the loop for SDG 3, but at the cost of an explicit, domain-agnostic modeling assumption (the opportunity-cost term is added to the environment, not extracted from text); learning transition dynamics directly from quantitative causal policy models or longitudinal indicator data—rather than positing them—remains future work. Cross-domain generalization beyond the 17 English-language SDG texts used here—particularly to multilingual governance documents and non-UN policy corpora—is a second near-term objective. Longer term, grammar-grounded RL policy analysis offers a principled basis for *computational policy auditing*: converting governance commitments into simulable environments makes the gap between stated policy intent and achievable outcomes formally measurable, with direct applications to evidence-based policymaking and AI-assisted legislative review. All artifacts—GBNF grammar, `RLSpec` schema and JSON extracts, prompt templates, experiment scripts E0–E5 (including the intent-derived and opportunity-cost-drift conditions of E5), and the E5 specialist action matrix—are documented in Appendix D and will be released at the anonymised repository upon acceptance.

7 Limitations

Tabular dynamics may not capture complex feedback loops. The synthesized environments use linear approximations to SDG dynamics. Real governance systems involve nonlinear feedback, threshold effects, and long delays that tabular models cannot represent. Policies discovered in NarraRL environments should be interpreted as proofs of concept, not operational recommendations.

E5 specialist action matrix. The policy-discovery experiment (E5/C5) uses a specialist SDG 3 environment in which the action-effect matrix was manually designed by the authors after an initial 30-seed run on the NarraRL-generated environment (uniform linear action effects) found no DQN advantage over random ($p = 0.849$, $n = 30$). The specialist matrix encodes domain knowledge — health investment trade-offs (e.g., *invest_in_maternal_health* reduces maternal mortality but worsens under-5 indicators) — that NarraRL's extraction pipeline does not currently infer from policy text alone. This is the primary scope limitation on C5: NarraRL automates environment *synthesis* (state space, reward signal, Gymnasium API compliance); it does not yet automate the design of a discriminating action-effect matrix. Extension to automated action-matrix generation from quantitative causal policy models is deferred to future work.

E4 does not confirm reward superiority or variance advantage. The E4 experiment (n=20 seeds) finds no statistically significant difference in final SDG targets satisfied between CoGen and Manual (p=0.807, Mann–Whitney). The original n=5 variance ratio (σ=0.13 vs. 0.68, 5×) does not replicate at adequate power: with n=20, 95% CIs on episode-return σ overlap (CoGen $[0.700, 1.344]$ vs. Manual $[0.518, 0.995]$; Levene W=0.897, p=0.414). Permuted thresholds show clearly higher variance (σ=3.360, CI $[2.555, 4.908]$, non-overlapping) and higher mean targets (3.58 vs. 2.74/2.72) — the latter is an artifact of the permuted threshold accidentally aligning with the neonatal dimension. We revise C4: CoGen and Manual achieve statistically indistinguishable means and variances; only semantically misaligned rewards show significantly higher variance. All three conditions converge in all 20/20 seeds under the ≥ 3 targets criterion.

CRO does not measure reward optimality. The CRO metric (BERTScore F1 between reward description and top-k most similar source clauses, default k=3) measures textual grounding, not whether the reward function is well-shaped for RL training. A reward with high CRO may still be poorly calibrated for a given environment. E4 demonstrates that NarraRL's co-generated reward is trainable, but we do not claim CRO predicts final DQN performance.

No human evaluation. NarraRL does not include a user study measuring whether domain experts find the synthesized environments and reward functions valid. The quality of the extraction is measured automatically via CRO (BERTScore F1 between reward term descriptions and source clauses), which is appropriate here because the claim being tested is *textual grounding* — whether reward terms originate from specific policy clauses — a property that BERTScore is designed to capture and that does not require human judgment of

Algorithm 1 NarraRL 4-Stage Self-Repair

Require: Policy text $\mathcal{T}$, LLM $\mathcal{M}$, max retries $K_{\max}$

```
spec ← GrammarExtract(M, T)
code ← LLMGenerate(M, spec)
for stage ∈ [exec, init, GymWrapper, check_env_specs] do
    k ← 0
    while k < K_max do
        (pass, diag) ← Validate(code, stage)
        if pass then break
        end if
        code ← LLMRepair(M, code, diag)
        k ← k + 1
    end while
    if k = K_max then return Fail(stage)
    end if
end forreturn code
```

the reward's behavioral adequacy. A complementary human evaluation could ask UN policy analysts to (i) rate whether each extracted reward term correctly reflects the intent of its cited source clause, and (ii) verify that the set of reward terms covers the primary numeric commitments in the SDG text; such a study would assess extraction fidelity beyond lexical overlap and is deferred to future work.

LLM pre-training contamination. The UN SDG texts used for evaluation are publicly available and are plausibly present in the pre-training corpora of llama3.2, phi3.5, and deepseek-r1. Pre-training exposure may make extraction appear easier than it would be for genuinely novel governance documents, inflating absolute CRO scores. However, both compared conditions (GBNF and Unconstrained) use the same LLMs on the same corpus, so contamination acts symmetrically and does not affect the sign or relative magnitude of the CRO difference ($\Delta = +0.052$). The same symmetry applies to the BERTScore evaluator (trained on large corpora that likely include UN text): it may shift the absolute CRO scale but not the relative comparison. All 17 SDG domains were used for both system development (grammar design, template iteration, hyperparameter selection) and evaluation; there is no held-out test set of policy domains. Evaluating on genuinely out-of-domain governance corpora (non-UN, non-English) remains future work.

English-only. All experiments use English SDG texts. Governance documents in other languages are not evaluated; GBNF grammar compilation and Pydantic schema prompting may require adaptation for non-English texts.

Computational cost. The 4-stage repair loop can consume up to $4 \times K_{\max} = 20$ LLM calls per environment. With models in the 1.5B–3.8B range tested here, extraction takes a mean of 16 s per condition on the Tesla T4 GPU used for E1–E3b, and 4–17 minutes per domain on a consumer CPU (MacBook Pro Intel Core i9, 64 GB RAM, no CUDA). DQN training (50 000 steps, E4 and E5) runs on the CPU hardware; per-seed training time was not instrumented and is not reported. Larger models would increase extraction cost but may improve pass rates; lower $K_{\max}$ values reduce cost at the price of lower pass rates.

A Self-Repair Pseudocode

Ethical Considerations

NarraRL processes publicly available UN SDG policy documents; it does not involve personal data, demographic targeting, or vulnerable populations. The system could theoretically be used to build environments encoding biased policy trade-offs, but the `source_clause_index` traceability mechanism makes such biases auditable: any reward term can be traced back to the specific policy sentence that grounds it, enabling

human oversight. We release all code, extracted `RLSpec` objects, experiment logs, and the GBNF grammar upon acceptance; see Appendix D for a full artifact manifest and the E5 action matrix specification.

B SDG Policy Extracts and Environment Sketches

Table 7 summarizes all 17 SDG domains used in E1 and E2, showing the canonical extraction for each (llama3.2, seed 1; every environment has ≥ 5 actions). For each domain, NarraRL extracts a structured `RLSpec` from the corresponding UN SDG policy text via grammar-constrained generation (§3.1). The state variables and thresholds are derived directly from the source clause text; numeric targets are taken verbatim from the UN document where the clause specifies them (e.g., SDG 3 targets 3.1–3.2), and normalized to $[0, 1]$ or proxy scales where the UN text is qualitative (e.g., SDG 13). SDG 2 thresholds are listed as extracted; the 2025 World Health Assembly (WHA) stunting/wasting targets (<25% / <5%) were not resolved by the LLM from the clause text in the E2 run and defaulted to 0.

Table 7: NarraRL-extracted `RLSpec` objects for all 17 SDG domains (canonical extraction: llama3.2, seed 1). $|S|$ = number of state variables; $|A|$ = number of actions (≥ 5 by schema constraint). Key state variables and actions show the first three of each; thresholds show the first two with their source UN SDG target ID. † Placeholder threshold (value 0) produced by the LLM when the clause specifies no numeric value. ‡ The LLM-generated SDG 3 environment (used in E1/E2) here has 5 state variables and 6 actions. The E5 specialist environment is a separate, manually refined 7-target variant (see §D, Table 11). All LLM-generated environments use near-identity randomized transition matrices (see §D).

SDG	$\|S\|/\|A\|$	Key state variables	Actions	Threshold(s) extracted
SDG 1	5/5	poverty rate; extreme poverty rate; social protection coverage	increase minimal wage; expand social protection; improve education access	poverty rate < 0.05 (1.1); social protection coverage > 0.5 (1.3)
SDG 2	5/5	food access; malnutrition rate; stunting rate	food distribution programs; nutrition interventions; sustainable agriculture practices	stunting rate < 0.0† (2.2); wasting rate < 0.0† (2.2)
SDG 3	5/6	global maternal mortality ratio; neonatal mortality per 1,000 live births; under 5 mortality per 1,000 live births	reduce global maternal mortality ratio; end preventable deaths of newborns and children under 5 years of age; reduce neonatal mortality to at least 12 per 1,000 live births	global maternal mortality ratio < 70 (3.1); neonatal mortality per 1,000 live births ≥ 12 (3.2)
SDG 4	5/5	enrollment rate; literacy rate; quality of education	increase primary enrollment; improve quality of education; reduce youth unemployment	enrollment rate ≥ 0.8 (4.1); literacy rate ≥ 0.7 (4.6)
SDG 5	5/5	discrimination rate; violence against women; child marriage rate	increase discrimination fighting efforts; provide violence prevention services; implement child marriage prevention measures	discrimination rate < 0.0† (5.1); violence against women > 0.0† (5.2)
SDG 6	5/5	drinking water availability; sanitation availability; water quality index	increase drinking water availability; improve sanitation infrastructure; reduce pollution	drinking water availability ≥ 100 (6.1); sanitation availability ≥ 100 (6.2)
SDG 7	5/5	energy access rate; renewable energy share; energy efficiency improvement rate	increase renewable energy share; improve energy efficiency; enhance clean energy research cooperation	energy access rate ≥ 0.5 (7.1); renewable energy share ≥ 0.2 (7.2)
SDG 8	5/6	per capita gdp; unemployment rate; youth unemployed	increase gdp growth rate; create jobs; reduce unemployment rate	per capita gdp growth rate > 7.0; unemployment rate < 0.0†

SDG	$\|S\|/\|A\|$	Key state variables	Actions	Threshold(s) extracted
SDG 9	5/5	infrastructure quality; employment share; gross domestic product	upgrade infrastructure; promote industrialization; increase research workers	employment share > 0.1 (9.2); gross domestic product > 0.05 (9.2)
SDG 10	5/5	income growth bottom 40 percent; transaction costs migrant remittances; remittance corridors cost	increase income growth bottom 40 percent; reduce transaction costs migrant remittances; eliminate high cost remittance corridors	transaction costs migrant remittances < 0.03 (10.c); remittance corridors cost > 0.05 (10.c)
SDG 11	5/5	housing access; transport systems; urbanization capacity	housing upgrade; transport investment; urban planning capacity	disaster losses < 0.0† (11.5); environmental impact < 1.0 (11.6)
SDG 12	5/5	natural resources management efficiency; global food waste at retail levels; chemicals management efficiency	increase natural resources management efficiency; reduce global food waste at retail levels; improve chemicals management efficiency	global food waste at retail levels < 5000 (12.3); chemicals management efficiency > 0.5 (12.4)
SDG 13	5/5	temperature; sea level rise; greenhouse gas emissions	reduce greenhouse gas emissions; increase climate resilience index; improve weather forecasting	greenhouse gas emissions < 5000; climate resilience index > 50
SDG 14	5/5	marine pollution level; coastal ecosystem health; ocean acidification impact	reduce marine debris; protect coastal ecosystems; mitigate ocean acidification	marine pollution level < 0.01 (14.1); coastal ecosystem health > 0.5 (14.2)
SDG 15	5/5	terrestrial ecosystems health; forests health; wetlands health	forest management; afforestation and reforestation; desertification combat	land degradation rate < 0.05 (15.3); biodiversity loss rate > 0.01 (15.5)
SDG 16	5/5	violence rate; child abuse rate; corruption level	reduce violence; end child abuse; promote rule of law	violence rate < 0† (16.1); child abuse rate < 0† (16.2)
SDG 17	5/5	tax revenue collection capacity; international support to developing countries; official development assistance commitments	strengthen domestic resource mobilization; implement official development assistance commitments; enhance technology bank capacity building	tax revenue collection capacity > 0.5 (17.1); international support to developing countries > 0.4 (17.2)

SDG 3 specialist environment (E5). The SDG 3 entry above corresponds to the *LLM-generated* environment (5 state variables, near-identity linear transitions, $p = 0.849$, $n = 30$ DQN null result). The *specialist* environment used in E5 extends this to 7 explicit UN SDG 3 targets with a 5-dimensional state and a manually designed action matrix encoding health-investment trade-offs (see §4.7 and Table 4). The specialist environment is not LLM-generated; NarraRL's role is to supply the reward signal and the initial environment specification, while domain expert knowledge refines the transition dynamics.

C Action-Count Sensitivity ($|A|$ Ablation)

The schema constrains extraction to a fixed action budget so that environments are comparable across the 17 domains (§4). A natural concern is whether this constant confounds our results — whether a different or variable $|A|$ would change the headline traceability and synthesis numbers. We test this directly by re-running the two $|A|$-exposed experiments, E1 (clause-reward traceability) and E2 (environment synthesis), with the per-domain action count forced to $|A| \in \{3, 5, 8\}$ for *all* 17 SDGs (llama3.2, 3 seeds, identical otherwise). The forcing is exact: all $17 \times 3 = 51$ extractions at each level emit precisely $|A|$ actions.

Table 8 reports the outcome. Synthesis is unaffected: the 4-stage scaffold produces an executable Gymnasium environment for every SDG at every action budget (51/51 pass at each $|A|$). Traceability is likewise invariant: the constrained CRO moves by only 0.002 across the full $3 \rightarrow 8$ range — an order of magnitude below the seed-to-seed standard deviation ($\sigma \approx 0.015$) — and a Kruskal–Wallis test finds no effect of $|A|$ (H=0.66, p=0.72). This is expected by construction: CRO scores the BERTScore overlap between the co-generated

Table 8: $|A|$-sensitivity ablation across all 17 SDGs (llama3.2, 3 seeds, $n = 51$ per cell). Synthesis pass rate is invariant (100% throughout). Constrained CRO is statistically indistinguishable across $|A|$ (Kruskal–Wallis H=0.66, p=0.72), and the constrained-vs-unconstrained traceability gap is preserved at every $|A|$ (Mann–Whitney p=1.6×10^{-18} each). Removing action-count variability does not affect the results: $|A|$ is a controlled constant, not a confound.

$\|A\|$	Synthesis pass (E2)	CRO constrained (E1)	CRO unconstrained
3	51/51 (100%)	0.848 ± 0.015	0.794
5	51/51 (100%)	0.848 ± 0.015	0.794
8	51/51 (100%)	0.850 ± 0.015	0.794

`reward_function.formal_description` and the top-k source clauses (§4.2), a quantity orthogonal to how many discrete actions the environment exposes. Crucially, the decisive contrast — constrained extraction grounds the reward in source text far better than unconstrained (0.848 vs. 0.794) — holds with the same significance at $|A|$=3, 5, and 8 (p=1.6×10^{-18} each). Fixing the action count therefore removes a source of cross-domain variability without altering any reported finding, which is why we adopt a single budget ($|A|$=5, the schema floor) in the main paper.

D Artifact Release

All artifacts listed below are released at `https://github.com/[anonymized-for-review]/narrarl` and will be de-anonymized upon acceptance. The repository contains: (i) the `RLSpec` Pydantic schema and compiled GBNF grammar, (ii) all prompt templates, (iii) a data acquisition script (E0: downloads all 17 UN SDG texts from `sdgs.un.org`) and experiment scripts E2–E5 with dependency pins, (iv) extracted `RLSpec` JSON objects for all 17 SDG domains, (v) the E5 specialist action matrix and environment class, and (vi) all result CSV files and plotting scripts used to generate the figures in §4.

C.1 Pydantic `RLSpec` Schema

The schema is defined as three nested Pydantic `BaseModel` classes. (§3.1 uses the logical names `constraints` and `reward_blueprint` for the two contribution-bearing fields; the deployed schema uses `thresholds` and `reward_function` as shown below — the concepts are identical.)

```
class SourceClause(BaseModel):
    sdg_target_id: str   # e.g. "3.1"
    text: str            # verbatim SDG target sentence

class ThresholdSpec(BaseModel):
    indicator: str       # state variable name
    operator: str        # one of {<, <=, >=, >, reduce_by}
    value: float         # numeric threshold
    unit: str            # e.g. "per 100,000 live births"
    source_clause: SourceClause

class PolicyEnvironment(BaseModel):
    name: str                        # e.g. "SDG3Env"
    description: str                 # one-sentence scenario summary
    policy_objective: str            # verbatim SDG goal statement
    state_variables: list[str]       # continuous indicator names
    actions: list[str]               # discrete policy action names
    thresholds: list[ThresholdSpec]  # clause-grounded success criteria
    reward_function: str             # natural-language reward description
                                     # referencing source_clause IDs
```

The `source_clause_index` field links every threshold to the specific SDG target sentence from which it was extracted, enabling the clause-reward traceability audit in E1.

C.2 GBNF Grammar

The GBNF grammar is compiled automatically from the `PolicyEnvironment` schema via `rl_llm.llm_grammar.generate_gbnf_grammar_and_documentation`. It enforces JSON structural conformity at the token level: string fields are constrained to JSON string tokens, `float` fields to JSON number tokens, `list[str]` fields to JSON arrays of strings. This prevents the extraction from emitting syntactically invalid JSON or structurally malformed objects that would fail Pydantic validation, eliminating one major failure mode identified in E3. The compiled grammar file (`rlspec.gbnf`) is included in the repository alongside the schema source.

C.3 Prompt Templates

NarraRL uses two prompt roles for the extraction call. The system prompt is fixed across all SDG domains:

```
System: You are an expert in UN Sustainable Development Goals and
reinforcement learning. Extract a structured RL environment specification
from the policy text. Every threshold must reference a specific SDG
target clause.
```

The user prompt is constructed per domain by concatenating the SDG text block (title, goal statement, and numbered target sentences) with schema documentation generated by the GBNF compiler, followed by:

```
Using the grammar and schema below, produce a valid JSON
PolicyEnvironment.

Schema documentation:
{auto-generated field descriptions}

IMPORTANT: output only valid JSON, no markdown fences.
```

The E3b ablation (§4.3) replaces this *specialized* prompt with a *standardized* prompt that omits the schema documentation and removes the mandatory source-clause constraint, directly isolating prompt specialization as the primary driver of the E2–E3 pass-rate gap (100% vs. ≈5%). Both prompt variants are included in the repository as `prompts/extraction_specialized.txt` and `prompts/extraction_standardized.txt`.

C.4 Environment Transition Models

LLM-generated environments (all 17 SDGs, E1 and E2). All 17 LLM-generated Gymnasium environments use the same near-identity randomized transition structure. For a domain with $|S|$ state dimensions and $|A|$ actions, the transition tensor $\mathbf{T} \in \mathbb{R}^{|A|\times|S|\times|S|}$ is initialized as:

$$\mathbf{T}[a] = \text{clip}\big(\mathbf{I}_{|S|} - U_a,\ 0,\ 1\big), \quad U_a \sim \text{Uniform}(0, 0.05)^{|S|\times|S|},$$

with seed 42 fixed across all environments. The new state after action a is $\mathbf{T}[a]\,\mathbf{s} + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 0.01)$. This produces near-identical marginal effects for all actions — the key reason DQN cannot learn a useful policy in these environments (E5-uniform, $p = 0.849$).

Because the tensor is seed-fixed and content-independent, the resulting action-effect matrix is *identical across all 17 SDG environments*. Table 9 gives this shared matrix for the $|A|$=5 case (the $|S|$=$|A|$=5 environments; SDG 3 and SDG 8 add a sixth action drawn the same way). Off-diagonal entries are 0 after clipping, so each action reduces to a near-unit per-dimension multiplier; the rows differ only by sampling noise, not by semantics. The per-SDG action *intent* — which named action is meant to affect which state variable — is instead captured qualitatively in Table 7.

Table 9: Shared LLM-generated action-effect matrix: the diagonal of $\mathbf{T}[a]$ (each action's near-unit multiplier on each state dimension s_j) for the seed-42 transition tensor, *identical across all 17 SDG environments*. Off-diagonal entries are 0 after clipping. Rows (actions a_0–a_4) are near-identical, illustrating the absence of semantic action structure that drives the E5-uniform null result; contrast the hand-designed SDG 3 specialist matrix (Table 11).

Action	s_1	s_2	s_3	s_4	s_5
a_0	0.961	0.962	0.968	0.959	0.961
a_1	0.990	0.952	0.976	0.984	0.993
a_2	0.990	0.977	0.972	0.998	0.997
a_3	0.986	0.980	0.977	0.981	0.952
a_4	0.955	0.978	0.985	0.978	0.998

Table 10 summarizes the dimensions, initial states, and extracted thresholds for all 17 LLM-generated environments.

Table 10: LLM-generated Gymnasium environments: dimensions and extracted thresholds. $|S|$ = number of state dimensions; $|A|$ = number of discrete actions. Initial state $\mathbf{s}_0$ = uniform value across all dimensions. Thresholds extracted from UN SDG text via `reward_blueprint`; † = LLM placeholder (no numeric value found in source clause).

SDG	$\|S\|/\|A\|$	$\mathbf{s}_0$	**Extracted thresholds**
SDG 1	5/5	0.8	poverty rate 0.0005; extreme poverty rate 0.5; social protection coverage 0.005; ...
SDG 2	5/5	0.8	food access 0.5; malnutrition rate 0.5; stunting rate 0†; ...
SDG 3	5/6	0.8	global maternal mortality ratio 0.007; neonatal mortality per 1 000 live births 0.5; under 5 mortality per 1 000 live births 0.5; ...
SDG 4	5/5	0.8	enrollment rate 0.008; literacy rate 0.007; quality of education 0.5; ...
SDG 5	5/5	0.8	discrimination rate 0†; violence against women 0†; child marriage rate 0.5; ...
SDG 6	5/5	0.8	drinking water availability 1; sanitation availability 1; water quality index 0.005; ...
SDG 7	5/5	0.8	energy access rate 0.005; renewable energy share 0.002; energy efficiency improvement rate 0.0005; ...
SDG 8	5/6	0.8	per capita gdp 0.5; unemployment rate 0†; youth unemployed 0.5; ...
SDG 9	5/5	0.8	infrastructure quality 0.5; employment share 0.001; gross domestic product 0.0005; ...
SDG 10	5/5	0.8	income growth bottom 40 percent 0.5; transaction costs migrant remittances 0.0003; remittance corridors cost 0.0005; ...
SDG 11	5/5	0.8	housing access 0.5; transport systems 0.5; urbanization capacity 0.5; ...
SDG 12	5/5	0.8	natural resources management efficiency 0.5; global food waste at retail levels 1; chemicals management efficiency 0.005; ...
SDG 13	5/5	0.8	temperature 0.5; sea level rise 0.5; greenhouse gas emissions 1; ...

SDG	$\|S\|/\|A\|$	s_0	Extracted thresholds
SDG 14	5/5	0.8	marine pollution level 0.0001; coastal ecosystem health 0.005; ocean acidification impact 0.0005; . . .
SDG 15	5/5	0.8	terrestrial ecosystems health 0.5; forests health 0.5; wetlands health 0.5; . . .
SDG 16	5/5	0.8	violence rate 0†; child abuse rate 0†; corruption level 0†; . . .
SDG 17	5/5	0.8	tax revenue collection capacity 0.005; international support to developing countries 0.004; official development assistance commitments 0.007; . . .

E5 specialist SDG 3 environment. Table 11 gives the 5×5 action-effect matrix used in the specialist SDG 3 environment. Each entry A_{ij} is the deterministic per-step change applied to state dimension j when action i is taken (Gaussian noise $\mathcal{N}(0, 0.005)$ is added at runtime). Negative values indicate a reduction in the corresponding indicator (desired for mortality and disease rates).

Table 11: E5 specialist SDG 3 action-effect matrix. State dimensions: MAT = maternal mortality, U5 = under-5 mortality, NEO = neonatal mortality, NCD = non-communicable disease (NCD) burden, ROAD = road injury rate. Each action strongly reduces one or two target indicators while slightly worsening others, requiring deliberate sequencing by DQN.

Action	MAT	U5	NEO	NCD	ROAD
fund_vaccination	+0.01	−0.03	+0.01	+0.03	−0.01
expand_PHC	+0.02	−0.04	−0.01	+0.01	+0.02
enforce_tobacco	+0.03	+0.02	+0.01	−0.10	+0.02
invest_maternal	−0.10	+0.02	+0.02	+0.01	+0.01
implement_WASH (water, sanitation, & hygiene)	+0.01	+0.01	−0.03	+0.02	−0.08

The initial state is $[0.85, 0.80, 0.82, 0.90, 0.88]$ (all normalized to $[0, 1]$ where 1.0 represents the worst observed value for each indicator) and success thresholds are $[0.07, 0.025, 0.012, 0.67, 0.50]$. A random uniform policy averages to near-zero net effect across indicators, which prevents the success thresholds from being reached without deliberate action sequencing; DQN must learn to apply INVEST_MATERNAL early, followed by EXPAND_PHC and IMPLEMENT_WASH. The uniform action matrix used in the NarraRL-generated environment (all entries = −0.02) is also released, as it produced the E5-uniform null result ($p = 0.849$, $n = 30$).

References

Zhangqian Bi, Yao Wan, Zheng Wang, Hongyu Zhang, Batu Guan, Fangxin Lu, Zili Zhang, Yulei Sui, Hai Jin, and Xuanhua Shi. Iterative refinement of project-level code context for precise code generation with compiler feedback. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2336–2353, 2024. doi: 10.48550/arXiv.2403.16792. URL https://arxiv.org/abs/2403.16792.

Palok Biswas, Zuzanna Osika, Isidoro Tamassia, Adit Whorra, Jazmin Zatarain-Salazar, Jan Kwakkel, Frans A. Oliehoek, and Pradeep K. Murukannaiah. Exploring equity of climate policies using multi-agent multi-objective reinforcement learning. arXiv preprint, 2025. URL https://arxiv.org/abs/2505.01115. Accepted at IJCAI 2025, AI and Social Good Track.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023. doi: 10.18653/v1/2023.emnlp-main.674. URL https://aclanthology.org/2023.emnlp-main.674/.

Or Honovich, Roee Aharoni, Jonathan Herzig, Hagai Taitelbaum, Doron Kuhn, Vera Liu, Jonathan Cohen, Tommi Jaakkola, Gal Elidan, and Kuzman Ganchev. TRUE: Re-evaluating factual consistency evaluation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3905–3920, 2022. doi: 10.18653/v1/2022.naacl-main.287. URL https://aclanthology.org/2022.naacl-main.287/.

Sinan Ibrahim, Mostafa Mostafa, Ali Jnadi, Hadi Salloum, and Pavel Osinenko. Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications. *IEEE Access*, 12: 175473–175520, 2024. doi: 10.1109/ACCESS.2024.3504735. URL https://doi.org/10.1109/ACCESS.2024.3504735.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 2018. URL https://proceedings.mlr.press/v80/icarte18a.html.

Hong Je-Gal, Chan-Bin Yi, and Hyun-Suk Lee. A-LAMP: Agentic LLM-based framework for automated MDP modeling and policy generation. NeurIPS 2025 Workshop on Multi-Turn Interactions in Large Language Models, 2025. URL https://arxiv.org/abs/2512.11270.

Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Jordi Grau-Moya, Marc Lanctot, Atil Iscen, John Schultz, Marcus Chiam, Ian Gemp, Piotr Zielinski, Satinder Singh, and Kevin P. Murphy. Code world models for general game playing. arXiv preprint, 2025. URL https://arxiv.org/abs/2510.04542.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models. In *Conference on Robot Learning*, 2024. doi: 10.48550/arXiv.2411.01775. URL https://arxiv.org/abs/2411.01775.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. arXiv preprint, 2023. URL https://arxiv.org/abs/2304.11477.

Xingzhou Lou, Junge Zhang, Ziyan Wang, Kaiqi Huang, and Yali Du. Safe reinforcement learning with free-form natural language constraints and pre-trained language models. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 2024. doi: 10.5555/3635637.3662985. URL https://arxiv.org/abs/2401.07553.

Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. Unified structure generation for universal information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5755–5772, 2022. doi: 10.18653/v1/2022.acl-long.395. URL https://aclanthology.org/2022.acl-long.395/.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2310.12931. URL https://openreview.net/forum?id=IEduRUO55F.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.48550/arXiv.2303.17651. URL https://arxiv.org/abs/2303.17651.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.48550/arXiv.2405.21047. URL https://arxiv.org/abs/2405.21047.

Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Équille, Maximilian Fabricius, and Srividya Subramanian. DORA the explorer: Exploring data with interactive deep reinforcement learning. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, 2021. doi: 10.1145/3459637.3481967. URL https://dl.acm.org/doi/10.1145/3459637.3481967. Demonstration paper.

Ravin Ravi, Dylan Bradshaw, Stefano Ruberto, Gunel Jahangirova, and Valerio Terragni. LLMLOOP: Improving LLM-generated code and tests through automated iterative feedback loops. In *Proceedings of the 41st IEEE International Conference on Software Maintenance and Evolution*, 2025. doi: 10.48550/arXiv.2603.23613. URL https://arxiv.org/abs/2603.23613.

James Rudd-Jones, Fiona Thendean, and María Pérez-Ortiz. Crafting desirable climate trajectories with RL explored socio-environmental simulations. *Environmental Data Science*, 2025. doi: 10.48550/arXiv.2410.07287. URL https://arxiv.org/abs/2410.07287.

Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.48550/arXiv.2402.12275. URL https://arxiv.org/abs/2402.12275.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint, 2024. URL https://arxiv.org/abs/2407.17032.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. Grammar prompting for domain-specific language generation with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.48550/arXiv.2305.19234. URL https://arxiv.org/abs/2305.19234.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. arXiv preprint, 2023. URL https://arxiv.org/abs/2307.09702.

Theodore Wolf, Nantas Nardelli, John Shawe-Taylor, and María Pérez-Ortiz. Can reinforcement learning support policy makers? A preliminary study with integrated assessment models. arXiv preprint, 2023. URL https://arxiv.org/abs/2312.06527.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2309.11489. URL https://openreview.net/forum?id=tUM39YTRxH.

Abhay Zala, Jaemin Yeo, Han Cho, Aleks Atanasyan, Roozbeh Mottaghi, and Mohit Bansal. EnvGen: Generating and adapting environments via LLMs for training embodied agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.12014.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*, 2020. doi: 10.48550/arXiv.1904.09675. URL https://openreview.net/forum?id=SkeHuCVFDr.