RAISE: REINFORCING ACCESS CONTROL POLICY SYNTHESIS IN LLMS VIA SYMBOLIC EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Translating natural-language access-control requirements into policies requires careful reasoning about permissions, constraints, and exceptions, and even frontier LLMs often produce policies that violate the intended authorization semantics. We construct CedarInstruct, to our knowledge the first dataset that supports both training and semantic evaluation for formally verifiable Cedar policy synthesis. It contains 5,800 scenarios across 44 domains and 1,408 representing a single synthetic organization, each with a verified target policy and an executable verification plan. On this data we introduce RAISE, which trains policy synthesizers from formal verification in two stages, verified supervised fine-tuning (SFT) followed by a reinforcement learning (RL) stage that learns from verifier signal. We find that SFT succeeds largely by letting models express authorization logic they already have, since untrained models rarely write valid Cedar but often reason correctly when they do. After SFT, how the verifier's information is used matters more than how much of it is used. Of six RL instantiations that consume progressively richer verifier signal, only RAISE-OC improves meaningfully on SFT; it turns failed checks and symbolic counterexamples into guided exploration and learns from the result with off-context GRPO. With about 5.4K verified scenarios and LoRA fine-tuning, RAISE-OC trains Qwen3.5-9B to surpass zero-shot GPT-6 Astra and Claude Opus 5 by 13.33 and 16.26 percentage points in semantic success on held-out scenarios, and training transfers to the independently constructed CedarBench. The implementation is available at https://anonymous.4open.science/r/raise-424A/.

1 INTRODUCTION

Access control policies decide which principals may perform which actions on protected resources, and writing them precisely is hard. Manual policy configuration is error-prone, and deployed authorization often grants more access than intended (Xiao et al., 2025; Cao et al., 2024). Translating natural-language requirements into policies adds a further difficulty, because a policy must be both syntactically valid and faithful to the requirement's authorization semantics (Cheng et al., 2026). We study this problem in Cedar, a declarative authorization language whose schemas and symbolic analysis tools support validation, policy comparison, and counterexample generation. (Cutler et al., 2024) Appendix C.7 shows the following policy in Cedar and a short guide to reading Cedar policies. Consider a requirement from our dataset: finance managers may issue payments only during business hours, and only for approved invoices unless the request comes from the corporate network. A natural-looking policy that grants the network exception as a separate rule is valid Cedar, but it also lets managers on the corporate network issue payments outside business hours, which the requirement forbids. Mistakes like this, where an exception quietly overrides a restriction it was never meant to lift, are easy to make and hard to spot by reading the policy.

Even frontier models find it hard to write a correct policy in a single attempt. We check each generated policy against a *verification plan*, an executable set of symbolic checks that bound the access a policy may grant and specify the access it must grant. Against these plans, GPT-6 Astra and Claude Opus 5 produce valid Cedar for over 97% of our test scenarios but satisfy the full plan for only about a third (Section 5.1). Verifier-guided repair can close part of this gap (Vatsa et al., 2026), but it needs the verifier, the plan, and several model calls at deployment time. This paper asks whether a small open model can instead be trained to generate correct policies directly.

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

Answering that question requires verified demonstrations and executable semantic criteria, so we first construct CedarInstruct, which contains 5,800 scenarios across 44 domains and 1,408 representing a single synthetic organization, each pairing a requirement and a Cedar schema with a verified target policy and a verification plan. Training on CedarInstruct follows a standard two-stage recipe. We first fine-tune the model on the verified target policies (SFT), and then continue with reinforcement learning (RL), rewarding each sampled policy that passes verification. The two stages turn out to behave very differently. SFT carries most of the improvement. Before training, many policies are not valid and the smaller model's valid policies are rarely correct, while after SFT nearly every policy is valid and more of them are correct. SFT also plateaus quickly, since another epoch changes nothing (Section 6.1). RL, in contrast, has little to learn from. With nearly every policy now valid, the remaining errors are semantic, and for about a third of the inputs sampled during RL, every sampled policy fails verification, so a pass-or-fail reward gives the model no signal. Yet the verifier knows far more than pass or fail. It can identify which property each policy violates and produce a concrete request that witnesses the violation, and when we show these diagnostics to the model, it repairs many of its failures (Section 6.1). The missing signal is there to be used.

We therefore introduce RAISE, which keeps this two-stage recipe but builds the RL stage around the verifier's diagnostics rather than its pass-or-fail verdict. We instantiate the RL stage six ways, each adapted to Cedar synthesis and each consuming more verifier information than the last, from binary outcomes up to failed-check descriptions. The strongest instantiation, RAISE-OC, explores each verified failure in its own guided context and learns from it with off-context GRPO (Agrawal et al., 2026). It is the only instantiation that improves meaningfully on SFT, even though two others receive the same failed-check descriptions, which suggests that how verifier information is used matters more than how much of it is used. The resulting 9B model surpasses zero-shot frontier models on held-out scenarios, and training transfers to the independently constructed CedarBench. Within a single organization's fixed schema, training on about a thousand cases lets the model solve nearly all new requirement combinations.

This paper makes three contributions. We construct **CedarInstruct** (Section 3), to our knowledge the first dataset supporting both training and semantic evaluation of formally verifiable Cedar policy synthesis. We introduce **RAISE** (Section 4), a framework for learning from formal verification, instantiated six ways along a ladder of verifier information, including RAISE-OC's construction of guidance from verification. Finally, we analyze what each training stage contributes (Sections 5 and 6), showing that SFT makes policies valid and plateaus quickly, that RL stalls where the verifier has most to say, and that only verifier-guided exploration recovers that signal.

2 Background

Two ideas underpin the rest of the paper, namely how we decide whether a generated policy is correct and why standard reinforcement learning can fail to improve a model that is already reasonably good.

Policy correctness through symbolic verification. A Cedar request asks whether a principal may perform an action on a resource in a given context. A schema Σ declares the entity types, attributes, actions, and context fields available, and a policy bundle P is a set of `permit` and `forbid` rules that authorizes a request if and only if some `permit` rule applies and no `forbid` rule does. Our task maps a requirement D and a schema Σ, written $q = (D, \Sigma)$, to a bundle P, and we judge P by the set of requests it authorizes, written $\mathrm{sem}_\Sigma(P)$.

To decide correctness, we use boundary plans generated by AutoCedar (Vatsa et al., 2026). A plan Π is a set of checks. Each check c pairs a reference policy R_c with a request domain D_c and has one of three types. A *ceiling* check ensures that no request authorized by P within D_c falls outside the bound defined by R_c. A *floor* check requires P to authorize every request authorized by R_c within D_c. A *liveness* check requires P to authorize at least one such request, ensuring that the corresponding workflow remains possible. Formally, the three check conditions are

$$\begin{aligned} \text{Ceiling:} &\quad \mathrm{sem}_\Sigma(P) \cap D_c \subseteq \mathrm{sem}_\Sigma(R_c), \\ \text{Floor:} &\quad \mathrm{sem}_\Sigma(R_c) \cap D_c \subseteq \mathrm{sem}_\Sigma(P), \\ \text{Liveness:} &\quad \mathrm{sem}_\Sigma(P) \cap \mathrm{sem}_\Sigma(R_c) \cap D_c \neq \varnothing. \end{aligned} \tag{1}$$

A symbolic checker, which we call the *verifier*, decides these conditions using Cedar's symbolic compiler (`cedar symcc`), which reduces each check to an SMT query (Cutler et al., 2024). When a check fails, the verifier returns its natural-language description and, for ceiling and floor failures, a counterexample request whenever the solver produces one. A policy is *correct* if it parses, validates against Σ, and satisfies every check, and we call the percentage of generated policies that are correct *semantic success*, our primary metric. Correctness is therefore relative to the plan, which is used only for training and evaluation and is never given to the model as input.

The learning cliff and off-context guidance. Correctness also provides a natural reward. GRPO samples K candidates per input, rewards each with $r_k = 1$ if P_k is correct and 0 otherwise, and normalizes rewards within the group to obtain advantages (Shao et al., 2024). The normalization is also its weakness, because when every candidate fails, all advantages vanish and the input contributes no gradient. This is known as the *learning cliff* (Zhang et al., 2026; Agrawal et al., 2026), and several methods break it by adding privileged guidance, such as hints or prefixes of a reference solution, to the prompt so that hard inputs yield some successes (Zhang et al., 2026; Qu et al., 2024).

Training on the guided prompt, however, optimizes a different objective from the unguided one used at deployment. Off-context GRPO (OC-GRPO) (Agrawal et al., 2026) corrects this mismatch. It applies guidance g only to inputs the policy cannot solve, samples under (q, g), and reweights each sampled token by

$$\rho_t(\theta) = \frac{\pi_\theta(o_t \mid q, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t \mid q, g, o_{<t})}, \tag{2}$$

which keeps the update unbiased for the unguided objective. Its analysis adds two properties that matter later. The correction's variance grows with the length of the guidance, which favors the shortest guidance that breaks the cliff, and a guided success keeps credit only to the extent that the model could have produced it unguided. What OC-GRPO leaves open is where good guidance should come from, since its experiments use prefixes of reference solutions.

3 CedarInstruct

Learning from verification requires data in which every requirement comes with a checkable definition of correct behavior. CedarInstruct provides this in two parts, 5,800 scenarios across 44 domains, which this section describes and our main experiments use, and 1,408 scenarios representing a single synthetic organization. Each scenario consists of a requirement D, a schema Σ, a verification plan Π, reference policies, and a verified target policy $P^\star$.

Each multi-domain scenario starts from a structured specification, a machine-readable list of the actions the scenario governs and the rules that apply to each. Specifications are sampled from hand-written domain skeletons, which declare a domain's roles, resources, attributes, and actions, and the 44 skeletons span 19 application areas. The rules combine constraints such as separation of duty, lifecycle gates, and overrides with conditions such as roles, thresholds, and network or time restrictions, and the sampler rejects duplicates so that every scenario is a distinct combination. GPT-5.5 then renders each specification as a natural-language requirement, and AutoCedar (Vatsa et al., 2026) builds the schema, plan, reference policies, and target policy. We keep a scenario only if its policies validate and $P^\star$ satisfies every check.

We split the multi-domain scenarios into 4,386 for SFT, 1,039 for RL, and 375 for testing. No test requirement, schema, target policy, or plan appears in training, but the test split draws on the same 44 domain skeletons and the same construction pipeline, so it measures in-distribution performance. To measure transfer, we also evaluate on CedarBench (Vatsa et al., 2026), whose scenarios were constructed independently. Appendix C details the construction pipeline, sampling rules, and review procedures.

4 RAISE: Learning from Verifier Signal

CedarInstruct supplies verified policies to learn from, and the verifier of Section 2 can judge any new policy the model writes. RAISE uses both, training a policy synthesizer in two stages (Figure 1). The first teaches the model to write verified policies, and the second continues training from the

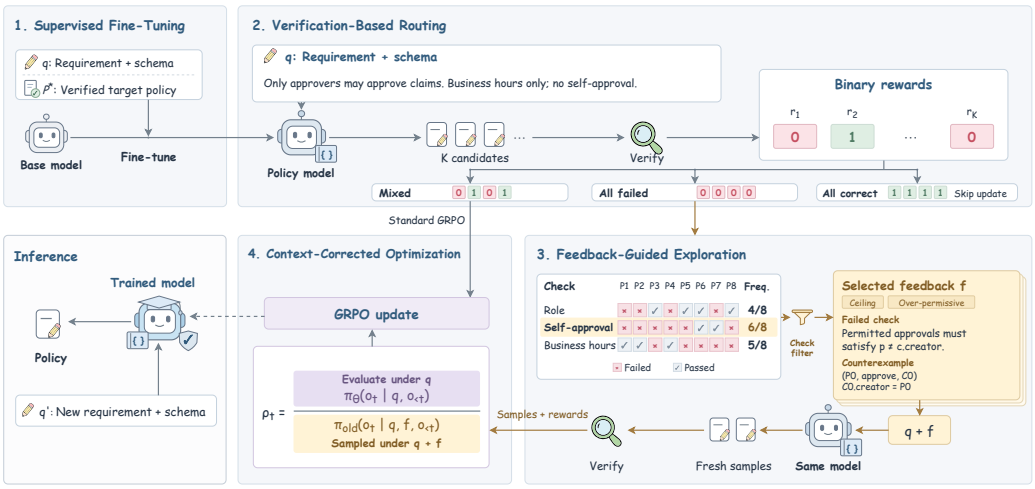

Figure 1: Overview of RAISE. Verified SFT is followed by an RL stage that learns from verifier signal. Shown is RAISE-OC, which routes all-failure groups to verifier-guided exploration and learns from the guided samples with an off-context update. At inference time, the model receives only the requirement and schema.

Table 1: Six instantiations of the RAISE RL stage along the verifier-feedback ladder.

Method	Verifier signal	Training use
Binary Reward GRPO	Plan outcome	Binary reward; zero gradient for all-failure groups
Coverage Reward GRPO	Check-pass fraction	Graded reward with a degeneracy guard
Formal Dominance GRPO	Passed-check set	Set-dominance preferences within all-failure groups
Critique-GRPO	Failure descriptions	Verifier descriptions guide candidate refinement
SDPO	Failure descriptions	Self-teaching uses feedback and a successful sibling
RAISE-OC	Descriptions and counterexamples	Separate guided contexts with off-context updates

verifier's feedback on the model's own attempts. In both stages, and at inference, the model sees only the requirement and schema.

4.1 Training Stages

Verified supervised fine-tuning. The first stage is supervised fine-tuning on verified target policies. The model learns to generate $P^\star$ from q, and only the policy tokens are supervised, with no intermediate reasoning. We hypothesize that this generalizes rather than memorizes. Lin et al. (2025) trace much of SFT's apparent failure to generalize to frozen prompts, which let a model ignore its instructions, and find that SFT generalizes to new instructions when each prompt declares its own vocabulary. CedarInstruct has this property by construction, because every input declares its vocabulary through its schema and a valid policy must use it.

Learning from verifier signal. SFT still leaves many inputs unsolved, and the second stage learns from the verifier on exactly those. Since no established method trains access-control synthesizers on sound verifier feedback, we adapt six verifier-guided training methods and order them by how much of the verifier's output each one uses (Table 1). They range from a binary pass-or-fail reward, through the fraction and identity of the checks a candidate passes, to the descriptions of the checks it fails. We ensure that all six start from the same SFT checkpoint and see only q at inference.

4.2 RAISE-OC

Failure-based routing and guidance construction. The strongest instantiation, RAISE-OC, targets inputs for which binary rewards provide no within-group learning signal. For each input q, it samples K candidates. Mixed-outcome groups receive a standard GRPO update, while all-

correct groups are skipped. For an all-failure group, let $\mathcal{V}_k$ be the checks failed by candidate k, $\mathcal{V} = \bigcup_{k=1}^{K} \mathcal{V}_k$, and $m = \min(M, |\mathcal{V}|)$. Guidance is constructed as

$$h(c) = \sum_{k=1}^{K} \mathbf{1}[c \in \mathcal{V}_k], \qquad \mathcal{S} = \mathrm{Top}_m(\mathcal{V}; h), \qquad \mathcal{B} = \{f_c = (d_c, x_c) \mid c \in \mathcal{S}\}, \tag{3}$$

where $h(c)$ counts how many candidates fail check c, and f_c contains its description d_c and an available counterexample x_c. Each $f_c \in \mathcal{B}$ defines a separate guided context. RAISE-OC distributes K fresh rollouts as evenly as possible across these contexts. Restricting $M \leq \lfloor K/2 \rfloor$ ensures at least two rollouts per context. Every guided candidate is evaluated against the complete verification plan, with advantages normalized within its context.

Off-context optimization. Following OC-GRPO (Agrawal et al., 2026), RAISE-OC uses the off-context ratio $\rho^{\mathrm{OC}}_{c,\ell,t}(\theta)$ from Equation 2 in a clipped surrogate:

$$\begin{aligned} L_\epsilon(\rho, A) &= \min\{\rho A,\ \mathrm{clip}(\rho, 1-\epsilon, 1+\epsilon)A\}\,, \\ \mathcal{J}_{\mathrm{guided}}(\theta) &= \mathbb{E}_{\mathrm{guided}}\left[L_\epsilon\left(\rho^{\mathrm{OC}}_{c,\ell,t}(\theta), \widehat{A}_{c,\ell}\right) - \beta \mathcal{K}_{c,\ell,t}(\theta)\right]. \end{aligned} \tag{4}$$

Here, $\mathbb{E}_{\mathrm{guided}}$ averages over guided inputs and their candidates, with per-candidate token normalization as formalized in Appendix B.7. $\mathcal{K}_{c,\ell,t}$ is the token-level KL penalty to the fixed reference policy, weighted by β. Because the numerator of ρ^{OC} conditions only on q, guided samples update the policy under the original input; no verifier feedback is therefore needed at inference time. Appendix B.7 gives the full objective, Algorithm 1 the complete procedure, and Appendix B.8 a feedback example.

Design rationale. RAISE-OC derives guidance from the verification plan and the behavior of the model's own candidates, without consulting the target policy $P^\star$. Pooling failures prioritizes checks that block the most candidates, while assigning each selected check a separate context avoids combining unrelated diagnostics. Because every guided candidate is evaluated against the complete plan, fixing one check while violating another receives no reward. The off-context ratio and clipped surrogate follow OC-GRPO; RAISE-OC contributes the verifier-based aggregation, selection, and grouping of guidance.

5 Experiments

5.1 Experimental Setup

Models and training. We now test whether this recipe delivers a small model that writes correct policies, and whether the way RAISE-OC uses the verifier matters. We train Qwen3.5-9B and Qwen3.8-27B with LoRA in verl (Sheng et al., 2025) on four NVIDIA H200 GPUs. Both backbones undergo three primary SFT epochs, followed by two epochs of each RL instantiation on the RL split, using a single training seed. For Qwen3.5-9B, we additionally run a fourth SFT epoch to assess saturation; all reported SFT results and subsequent RL runs use the epoch-3 checkpoint. RAISE-OC samples $K = 8$ candidates per input and turns at most $M = 4$ failed checks into guided contexts (Section 4.2).

Baselines and prompts. For comparison, we evaluate GPT-6 Astra (Max) (OpenAI, 2026b) and Claude Opus 5 (Max) (Anthropic, 2026) with the zero-shot prompt of Appendix H, and we also report GPT-5.5, which helped construct the dataset, in Appendix F.1. The untrained Qwen models receive a structured Cedar prompt instead, because with the plain prompt's brief guidance only 0.00% and 0.27% of their policies are valid, as most generations fail parsing or schema validation before any semantic check. The structured prompt adds syntax patterns, schema-grounding constraints, and an output scaffold, which makes it the stronger baseline.

Evaluation protocol. We evaluate the final checkpoint of every training run, and no prompt, hyperparameter, or checkpoint was chosen using the test split. Every model generates one policy per scenario, greedily for Qwen, with no verifier feedback.

Evaluation metrics. Alongside semantic success we report syntax, the percentage of valid policies, and two measures of partial correctness based on individual checks. The macro score averages, over

Table 2: Results on the CedarInstruct test split (%). Every RL instantiation starts from the SFT checkpoint, and arrows give the change in semantic success relative to it. "Among valid" is semantic success divided by syntax; for untrained models, valid policies are a selected subset, so this rate describes the shift rather than isolating it.

Method	Syntax	Semantic	Among valid	Macro	Micro
Frontier models, zero-shot					
GPT-6 Astra (Max)	97.87	33.60	34.3	71.10	68.70
Claude Opus 5 (Max)	100.00	30.67	30.7	73.12	72.82
Qwen3.5-9B					
Structured zero-shot	45.07	2.67	5.9	23.51	22.25
RAISE: SFT	99.20	42.93	43.3	78.93	77.92
+ Binary Reward GRPO	98.93	43.47$_{\uparrow 0.54}$	43.9	78.94	77.92
+ Coverage Reward GRPO	98.40	42.13$_{\downarrow 0.80}$	42.8	77.56	76.36
+ Formal Dominance GRPO	97.33	43.73$_{\uparrow 0.80}$	44.9	77.99	77.12
+ Critique-GRPO	99.73	42.67$_{\downarrow 0.26}$	42.8	79.37	78.09
+ SDPO	97.07	43.73$_{\uparrow 0.80}$	45.0	78.22	76.81
+ **RAISE-OC**	99.20	**46.93**$_{\uparrow 4.00}$	**47.3**	**80.78**	**79.34**
Qwen3.8-27B					
Structured zero-shot	79.73	28.80	36.1	60.54	58.51
RAISE: SFT	99.73	46.40	46.5	81.02	80.03
+ Binary Reward GRPO	99.73	44.80$_{\downarrow 1.60}$	44.9	81.14	80.21
+ Coverage Reward GRPO	99.73	46.93$_{\uparrow 0.53}$	47.1	**81.70**	**81.01**
+ Formal Dominance GRPO	98.67	43.47$_{\downarrow 2.93}$	44.1	80.23	79.65
+ Critique-GRPO	99.73	46.13$_{\downarrow 0.27}$	46.3	80.44	79.31
+ SDPO	98.93	45.60$_{\downarrow 0.80}$	46.1	79.97	78.44
+ **RAISE-OC**	99.73	**48.00**$_{\uparrow 1.60}$	**48.1**	81.66	80.83

scenarios, the fraction of each scenario's checks that the policy passes, so every scenario counts equally. The micro score pools all checks across scenarios and reports the fraction passed, so scenarios with more checks count more. An invalid policy passes no checks.

5.2 Main Results

In Table 2, five of the six RL instantiations improve semantic success by at most 0.80 points over SFT on either backbone, and four of them fall below SFT on Qwen3.8-27B, including Critique-GRPO and SDPO, which see the same failed-check descriptions as RAISE-OC. RAISE-OC alone improves meaningfully, by 4.00 and 1.60 points, which corresponds to 15 and 6 additional scenarios solved and a margin of 12 and 4 scenarios over the strongest other instantiation (Appendix F.3). Its validity is unchanged, so the gains come from getting the semantics right. Scores based on individual checks tell a less clear story, because passing more checks does not guarantee a correct policy. For Qwen3.8-27B, Coverage Reward GRPO scores marginally higher than RAISE-OC in macro and micro checks while solving fewer plans. We also report, interestingly, that Claude Opus 5 passes more individual checks than GPT-6 Astra but fewer complete plans for each scenario.

Against frontier systems, both trained models come out ahead. With RAISE-OC, Qwen3.5-9B surpasses GPT-6 Astra and Claude Opus 5 by 13.33 and 16.26 percentage points in semantic success, and Qwen3.8-27B by 14.40 and 17.33, without verifier feedback at inference, using LoRA and about 5.4K verified scenarios. These test scenarios are held out but in-distribution (Section 3), and Section 6.2 tests transfer to independently constructed ones. The trained models also run at lower per-scenario latency and estimated cost (Appendix F.7).

6 Further Analysis

The main results raise two questions. Why does SFT account for most of the gain, and why does RAISE-OC improve on it when other ways of using the verifier do not? We take them in turn, and then ask how far the results extend beyond the test split.

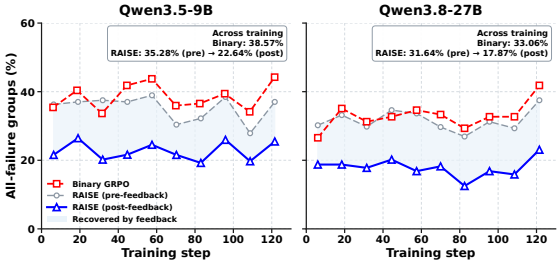

Figure 2: All-failure group rates for Binary GRPO and RAISE-OC before and after verifier-guided exploration. Shading indicates recovered learning signal.

Table 3: Component ablations: semantic success (%) on the test split.

Configuration	9B	27B
No SFT, binary RL	0.00	27.20
SFT	42.93	46.40
+ Binary GRPO	43.47$_{\uparrow 0.54}$	44.80$_{\downarrow 1.60}$
+ Guided, uncorrected	44.27$_{\uparrow 1.34}$	47.20$_{\uparrow 0.80}$
+ Guided, OC-GRPO	**46.93**$_{\uparrow 4.00}$	**48.00**$_{\uparrow 1.60}$

Notes. 9B: Qwen3.5-9B; 27B: Qwen3.8-27B. Arrows show changes from SFT; the last row is RAISE-OC. Syntax exceeds 99% with SFT; without SFT, it is 0.00% (9B) and 74.13% (27B).

6.1 What SFT and RAISE-OC Each Contribute

Semantic success requires validity, and before training many policies are not valid. Under structured zero-shot prompting, only 45.07% of Qwen3.5-9B's policies and 79.73% of Qwen3.8-27B's are valid, and correctness among the valid ones differs sharply between Qwen3.5-9B and Qwen3.8-27B. Only 5.9% of Qwen3.5-9B's valid policies satisfy the full plan, whereas 36.1% of Qwen3.8-27B's do, comparable to GPT-6 Astra (34.3%) and Claude Opus 5 (30.7%) on the same measure (Table 2). Our interpretation is that Qwen3.8-27B already brings authorization reasoning from pretraining, likely from code in other languages and other policy formats, but cannot yet express it reliably in Cedar. The SFT data teaches that expression, meaning Cedar's syntax and how its constructs interact, such as `permit` and `forbid` rules, `when` and `unless` conditions, and schema-typed attributes. After SFT, validity rises above 99% for both Qwen3.5-9B and Qwen3.8-27B, and correctness among valid policies rises to 43.3% and 46.5%. For Qwen3.8-27B, this is consistent with the view that fine-tuning largely teaches models to express capabilities acquired in pretraining (Zhou et al., 2023), and it would help explain why about 4.4K demonstrations suffice. Qwen3.5-9B has less to unlock, since its valid policies are rarely correct before training, so for Qwen3.5-9B, SFT likely teaches reasoning as well as expression. The gap between Qwen3.5-9B and Qwen3.8-27B among valid policies accordingly narrows from 30.2 points before training to 3.2 points after. Both frontier models achieve near-perfect syntactic validity (97.87% and 100.00%), so their errors are unlikely to stem from unfamiliarity with Cedar and instead reflect the difficulty of the authorization reasoning itself.

If SFT mainly teaches a language, more of it should not help much, and it does not. A further SFT epoch on the original 4,386 scenarios does not improve Qwen3.5-9B, and a further epoch on the 1,039 RL-partition scenarios, the same data RAISE-OC learns from, lowers its semantic success by 5.60 points (Appendix D). Further progress, therefore, needs a different training signal.

Once nearly every policy is valid, the remaining failures are semantic, and under binary-reward GRPO, 38.57% and 33.06% of sampled groups contain no correct candidate for the 9B and 27B models (Figure 2), so they contribute nothing to learning. These are the inputs where the model is most wrong, and they are also where the verifier has the most to say, since beyond pass or fail it can report the fraction and identity of the checks a candidate passes and, for each failed check, a description of the violated property and often a concrete counterexample. At inference time, giving the SFT model its failed-check descriptions and counterexamples and asking it to repair its policy raises semantic success by 21.07 points on Qwen3.5-9B, from 42.93% to 64.00%, whereas a retry that reports only failure adds 2 (Appendix F.5). The richest information is therefore directly useful to the model, and because its content rather than the extra attempt does most of the work, it serves, in the terms of Section 2, as guidance that raises the chance of success where the unguided model fails, without any reference solution.

Table 3 isolates each component of RAISE-OC, with every change measured relative to SFT. Without SFT, binary-reward GRPO fails on both backbones. Qwen3.5-9B produces no valid policies, so every reward is zero, and Qwen3.8-27B reaches 27.20%, below the 28.80% it achieves with the structured prompt alone. After SFT, binary-reward GRPO changes semantic success by $+0.54$ points on Qwen3.5-9B and -1.60 on Qwen3.8-27B. With verifier-guided exploration and an uncorrected update, the changes become $+1.34$ and $+0.80$. This variant samples under feedback-augmented prompts but computes the loss under the original input q, without the off-context correction. With the correction, which gives RAISE-OC, they become $+4.00$ and $+1.60$, so the guidance

Table 4: CedarBench (221 scenarios, inputs converted to natural-language requirements; %) for Qwen3.5-9B and Qwen3.8-27B, excluded from training and model selection. Arrows give the change in semantic success relative to SFT.

	9B		27B	
Method	Syntax	Semantic	Syntax	Semantic
Zero-shot	0.00	0.00	1.81	1.36
SFT	64.71	39.37	72.40	49.77
+ Binary GRPO	66.97	42.53$_{\uparrow 3.17}$	72.40	49.77$_{0.00}$
+ **RAISE-OC**	66.52	**43.44**$_{\uparrow 4.07}$	68.78	**52.49**$_{\uparrow 2.71}$

Table 5: Semantic success (%) of Qwen3.5-9B on 256 organizational test cases. "Before" is the CedarInstruct SFT checkpoint; "After" is that checkpoint after training on the 1,024 organizational training cases.

Input	Before	After
Requirement + schema	7.81	**94.92**
+ plan	58.20	**96.88**

helps with either update and most with the correction. Within the guidance, selecting checks by frequency, exploring each in its own context, and pooling failures across candidates each outperform their alternatives (Appendix F.4). Figure 2 shows where the gain comes from. Guided exploration lowers RAISE-OC's all-failure rate from 35.28% to 22.64% on Qwen3.5-9B and from 31.64% to 17.87% on Qwen3.8-27B, recovering a learning signal for about 13% of all sampled groups. Part of this recovery comes from drawing K additional candidates per all-failure input, but the repair result above suggests that the diagnostics' content, not the extra attempts, does most of the work.

6.2 Beyond the Test Split

Our test scenarios are new, but they come from the same 44 domains as the training data, with the same roles, resources, and actions. We therefore evaluate on two further settings. The first is CedarBench, whose scenarios were built independently of CedarInstruct. The second is a single organization that keeps the same schema and actions while its access rules change, tested on rule combinations unseen in training.

CedarBench (Vatsa et al., 2026) was built independently of CedarInstruct. To match CedarInstruct's input format, we convert CedarBench's inputs into natural-language requirements using GPT-5.5 with the rendering prompt of Listing 10, and we keep CedarBench's original verification plans and scoring. None of its 221 scenarios is used for training or model selection. Training transfers substantially (Table 4). SFT raises semantic success from 0.00% to 39.37% on Qwen3.5-9B and from 1.36% to 49.77% on Qwen3.8-27B. RAISE-OC is best on both backbones, reaching 43.44% and 52.49%, which is 4.07 and 2.71 points above SFT, whereas binary-reward GRPO gains 3.17 and 0.00. On Qwen3.8-27B, RAISE-OC's gain comes despite lower syntax validity than SFT, so it improves the correctness of valid policies rather than their validity. Frontier models were not evaluated on CedarBench, so the frontier comparison applies only to CedarInstruct.

A deployed synthesizer faces a different kind of shift. An organization typically keeps its schema and actions fixed while its requirements change, and we study this setting with the 1,408 CedarInstruct scenarios that represent a single synthetic organization. They share one document-and-approval schema and six actions (read, edit, share, approve, delete, and manage access), and we split them by rule combination into 1,024 training, 128 development, and 256 test cases, so every test case is a combination of rules unseen in training. Starting from the SFT checkpoint, we train Qwen3.5-9B for one epoch with a variant of the SDPO instantiation whose self-teacher is conditioned on the verified reference policy and on verified repair histories. We use this variant rather than RAISE-OC because the setting is narrow. With the schema and actions fixed, the same rules and checks recur across cases, much like a small game, so we expect correct policies to be easier to reach and the all-failure inputs that RAISE-OC targets to be rarer, while a teacher that sees the verified reference provides a dense learning signal. One model receives the requirement and schema, and a second also receives the verification plan, as an organization that maintains its requirements as checkable properties could provide. After training, the model given the requirement and schema solves 243 of 256 test cases and the model also given the plan solves 248 (Table 5), so modest organization-specific training lets Qwen3.5-9B handle new requirement combinations almost without error. Before organization-specific training, the SFT checkpoint solves only 7.81% of these test cases, against 42.93% on the CedarInstruct test split, so a single organization's rules can differ substantially from our multi-domain data. Adding the plan raises its success to 58.20%, which suggests that the plan's atomic properties are useful scaffolding.

7 Discussion

Four instantiations use more verifier information than a binary outcome, and two use the same failed-check descriptions as RAISE-OC, yet only RAISE-OC improves meaningfully over SFT. This suggests that the difference lies in how the feedback is used. Critique-GRPO refines candidates that have already failed, SDPO distills a feedback-conditioned teacher into the student's own, mostly failing, candidates, and neither corrects for the guided context. RAISE-OC instead samples fresh candidates under short, targeted guidance and learns from them off-context, which is where Section 2 predicts guidance is most useful.

This guidance is exact because Cedar is deliberately not Turing-complete, so whether a policy satisfies a check is decided by an SMT solver rather than estimated with tests (Cutler et al., 2024). RAISE should therefore transfer most directly to other policy and configuration languages that can be analyzed the same way. The organizational study suggests a practical deployment path, in which a broadly trained model is adapted to one organization's fixed schema with about a thousand verified cases (Section 6.2). Since SFT here is answer-only, the atomic properties of each plan also offer a natural reasoning scaffold, a direction supported by Lin et al. (2025), who find that chain-of-thought supervision helps on harder instances.

Our study has several limitations. Correctness is defined by generated verification plans, whose fidelity symbolic checking cannot establish and which we assess only through sampled human and LLM review. Because the same pipeline defines the reward and the test grading, trained models may partly learn its reading of ambiguous requirements.All results use a single training seed, so margins as small as 4 of 375 test scenarios (RAISE-OC over the strongest other instantiation on Qwen3.8-27B) could partly reflect run-to-run variation. Guided exploration also draws extra samples, and we did not compare verifier guidance with reference-solution prefixes or LLM-written hints (Appendix G).

8 Related Work

Prior work applies LLMs to access-control policy generation through prompting and fine-tuning (Lawal et al., 2024; Vatsa et al., 2025b; Yang et al., 2025; Jayasundara et al., 2026). AutoCedar instead uses symbolic verification for inference-time repair (Vatsa et al., 2026), whereas we use verification to train the generator. In reinforcement learning with verifiable rewards (Le et al., 2022; Guo et al., 2025), GRPO estimates advantages from within-group reward differences (Shao et al., 2024). Uninformative groups have been addressed through filtering and resampling (Yu et al., 2026), adaptive task difficulty (Zeng et al., 2025), and hierarchical hints (Zhang et al., 2026). Feedback has supported both inference-time refinement (Madaan et al., 2023; Gou et al., 2024) and training in Critique-GRPO and SDPO (Zhang et al., 2025; Hübotter et al., 2026), which we adapt as RAISE instantiations. Closest to RAISE-OC, adaptive OC-GRPO retries failed prompts with progressively stronger tutor-generated hints (Agrawal et al., 2026). RAISE-OC instead derives guidance from formal verification of the model's own candidates, pools failures across candidates, and explores each selected property in its own context. Appendix A provides a fuller review.

9 Conclusion

Small open models can learn to translate access-control requirements into verified Cedar policies better than zero-shot frontier models, from a few thousand verified scenarios and without verifier feedback at inference. Verified SFT does most of the work, making nearly every policy valid and more often correct, but it plateaus, and more supervised data does not help. Beyond that point, the verifier's diagnostics supply the missing signal, and how they are used decides whether they help. Of six ways of learning from verification, only exploring each verified failure in its own guided context and learning from it off-context improves meaningfully on SFT. Training transfers to independently built scenarios, and a synthesizer adapts to a single organization's requirements from about a thousand examples.

AI USE STATEMENT

The authors used large language models (LLMs) to assist with manuscript polishing and the construction of CedarInstruct. Specifically, LLMs were used to render synthetic access-control requirements from structured specifications and to assist with faithfulness review during dataset construction. Language polishing focused on proofreading, grammatical correction, and stylistic refinement. The authors conceived the research ideas, designed the methodology, and conducted the experimental evaluation and analysis.

ETHICS STATEMENT

This research adheres to the ICLR Code of Ethics. Our objective is to improve the reliability of access control policy synthesis by integrating formal verification with language model training. We acknowledge the security risks of incorrectly generated policies, including unintended permissions and denial of legitimate access. Our symbolic verification guarantees are limited to the specified verification plans. CedarInstruct is constructed from synthetic scenarios, without real-world personal or sensitive authorization data. We encourage deploying generated policies with independent validation and human oversight.

REPRODUCIBILITY STATEMENT

The supplementary material contains the complete source code, including the implementation of RAISE and all six RL-stage instantiations, the CedarInstruct construction and verification pipeline, and all training and evaluation scripts. The implementation is available at https://anonymous.4open.science/r/raise-424A/.

REFERENCES

Mahmoud Abdelgawad, Indrakshi Ray, Saja Alqurashi, Videep Venkatesha, and Hosein Shirazi. Synthesizing and analyzing attribute-based access control model generated from natural language policy statements. In *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, pp. 91–98, 2023.

Priyank Agrawal, Ankur Samanta, Shervin Ghasemlou, Jalaj Bhandari, Kavosh Asadi, Daniel Jiang, and Aditya Modi. Off-context grpo: Learning to reason on hard problems using privileged information. *arXiv preprint arXiv:2607.19313*, 2026.

Anthropic. Introducing Claude Opus 5, 2026. URL https://www.anthropic.com/news/claude-opus-5.

John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek, Kasper Luckow, Neha Rungta, Oksana Tkachuk, and Carsten Varming. Semantic-based automated reasoning for aws access policies using smt. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pp. 1–9. IEEE, 2018.

Leo Cao, Luoxi Meng, Deian Stefan, and Earlence Fernandes. Stateful least privilege authorization for the cloud. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 3477–3494, 2024.

Ye Cheng, Minghui Xu, Yue Zhang, Kun Li, Hao Wu, Yechao Zhang, Shaoyong Guo, Wangjie Qiu, Dongxiao Yu, and Xiuzhen Cheng. "say what you mean": Natural language access control with large language models for internet of things. *IEEE Transactions on Information Forensics and Security*, 2026.

Joseph W Cutler, Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, Eleftherios Ioannidis, John Kastner, Anwar Mamat, et al. Cedar: A new language for expressive, fast, safe, and analyzable authorization. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):670–697, 2024.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, et al. Stepcoder: improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4571–4585, 2024.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Bethel Hall, Owen Ungaro, and William Eiers. Cloudfix: Automated policy repair for cloud access control policies using large language models. In *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 322–332. IEEE, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Sakuna Harinda Jayasundara, Nalin Asanka Gamagedara Arachchilage, and Giovanni Russello. Agentvlm: Access control policy generation and verification framework with language models. *Journal of Information Security and Applications*, 98:104379, 2026.

Difan Jiao, Qianfeng Wen, Blair Yang, Zhenwei Tang, and Ashton Anderson. Thinktwice: Jointly optimizing large language models for reasoning and self-refinement. *arXiv preprint arXiv:2604.01591*, 2026.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 54523–54549, 2025.

Sherifdeen Lawal, Xingmeng Zhao, Anthony Rios, Ram Krishnan, and David Ferraiolo. Translating natural language specifications into access control policies by leveraging large language models. In *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, pp. 361–370. IEEE, 2024.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022.

Xiaofeng Lin, Hejian Sang, Zhipeng Wang, and Xuezhou Zhang. Debunk the myth of sft generalization. *arXiv preprint arXiv:2510.00237*, 2025.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. Rltf: Reinforcement learning from unit test feedback. *arXiv preprint arXiv:2307.04349*, 2023.

Xiaoyuan Liu, Tian Liang, Zhiwei He, Jiahao Xu, Wenxuan Wang, Pinjia He, Zhaopeng Tu, Haitao Mi, and Dong Yu. Trust, but verify: A self-verification approach to reinforcement learning with verifiable rewards. *Advances in Neural Information Processing Systems*, 38:130475–130501, 2026.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Youssef Mroueh. Reinforcement learning with verifiable rewards: GRPO's loss, dynamics, and success amplification. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=z9Bo4eXSpW.

OpenAI. GPT-5.5 Model. https://developers.openai.com/api/docs/models/gpt-5.5, 2026a.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026b. URL https://openai.com/index/gpt-6-astra/.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *Advances in Neural Information Processing Systems*, 37:55249–55285, 2024.

Chi Ruan, Dongfu Jiang, Yubo Wang, and Wenhu Chen. Critique-coder: Enhancing coder models by critique reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 19678–19692, 2026.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

John Slankas, Xusheng Xiao, Laurie Williams, and Tao Xie. Relation extraction for inferring access control rules from natural language artifacts. In *Proceedings of the 30th annual computer security applications conference*, pp. 366–375, 2014.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Adarsh Vatsa, Bethel Hall, and William Eiers. Exploring large language models for access control policy synthesis and summarization. *arXiv preprint arXiv:2510.20692*, 2025a.

Adarsh Vatsa, Pratyush Patel, and William Eiers. Synthesizing access control policies using large language models. In *2025 IEEE/ACM International Workshop on Natural Language-Based Software Engineering (NLBSE)*, pp. 13–16. IEEE, 2025b.

Adarsh Vatsa, Sachi Shome, Yingming Zhou, and William Eiers. Autocedar: An agentic framework for verifier-guided access control policy synthesis. *arXiv preprint arXiv:2607.03656*, 2026.

Ka Lok Wu, Christa Jenkins, Scott D Stoller, and Omar Chowdhury. Automatically tightening access control policies with restricter. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 110–129. Springer, 2026.

Lifang Xiao, Hanyu Wang, Aimin Yu, Lixin Zhao, and Dan Meng. Caspr: Context-aware security policy recommendation. In *NDSS*, 2025.

Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta, and Tao Xie. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pp. 1–11, 2012.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. *Advances in Neural Information Processing Systems*, 38:117157–117186, 2026.

Mian Yang, Vijayalakshmi Atluri, Shamik Sural, and Ashish Kundu. Extraction of machine enforceable abac policies from natural language text using llm knowledge distillation. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies*, pp. 157–168, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, et al. Rlve: Scaling up reinforcement learning for language models with adaptive verifiable environments. *arXiv preprint arXiv:2511.07317*, 2025.

Xiaoying Zhang, Yipeng Zhang, Hao Sun, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback. *arXiv preprint arXiv:2506.03106*, 2025.

Xichen Zhang, Sitong Wu, Yinghao Zhu, Haoru Tan, Shaozuo Yu, Ziyi He, and Jiaya Jia. Scaf-grpo: Scaffolded group relative policy optimization for enhancing llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 131946–131974, 2026.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, 2023.

TABLE OF CONTENTS

A Comprehensive Review of Related Work **15**

B Formal Definitions and Training Details **16**

B.1 Training Procedure 16
B.2 Verification 16
B.3 Verified SFT Objective 17
B.4 Group Relative Policy Optimization 17
B.5 RAISE-OC's Guidance Construction 17
B.6 When a Guided Group Yields a Learning Signal 18
B.7 Full Off-Context Objective 18
B.8 Illustrative Verifier Feedback 18

C Construction of CedarInstruct **20**

C.1 Stage 1: Controlled Scenario Space and Sampling 20
C.2 Stage 2: Natural-Language Requirement Rendering 22
C.3 Stage 3: Cedar Artifact Generation 22
C.4 Stage 4: Verification and Faithfulness Filtering 22
C.5 Dataset Characterization 23
C.6 Example Retained Scenario 24
C.7 Reading a Cedar Policy 26

D Implementation Hyperparameters **27**

E Baseline Methods **28**

F Additional Experimental Results **29**

F.1 Construction-Model Reference 29
F.2 Metric Definitions 29
F.3 Significance Tests 29
F.4 Guidance Ablations 29
F.5 Verifier Feedback at Inference Time 30
F.6 Authorization Failure Patterns 31
F.7 Inference Efficiency 31

G Limitations **31**

H Prompt Templates **32**

A Comprehensive Review of Related Work

Access-Control Policy Synthesis and Verification. Earlier work on natural-language policy authoring extracted access-control rules from textual requirements (Xiao et al., 2012; Slankas et al., 2014) and constructed authorization models from policy statements (Abdelgawad et al., 2023). Recent studies investigate LLM prompting for policy extraction and synthesis, including whether generated policies preserve the intended authorization behavior (Lawal et al., 2024; Vatsa et al., 2025b;a). Beyond prompting, Yang et al. (2025) fine-tune CodeT5 on GPT-4-generated data to extract ABAC rules from natural-language text. AGentVLM uses small open-source language models for structured policy extraction and provides verification feedback to support policy revision (Jayasundara et al., 2026). Formal analysis complements policy generation by checking authorization behavior against specified properties. Zelkova and Cedar support symbolic reasoning about policies, including comparisons of the requests they allow (Backes et al., 2018; Cutler et al., 2024). Verification also supports policy modification. Restricter tightens Cedar policies against access logs, while CloudFix combines formal fault localization with LLM-generated repairs checked by SMT solvers (Wu et al., 2026; Hall et al., 2026). AutoCedar constructs reviewed schemas and boundary plans specifying required permissions, upper bounds on permitted access, and liveness conditions, and uses symbolic verification to guide policy generation and repair (Vatsa et al., 2026). Our focus is on using the resulting verification feedback to train the generator, rather than solely to assess and repair individual candidate policies.

Reinforcement Learning with Verifiable Rewards. Verification results can serve as training rewards for policy generation. This follows the broader reinforcement learning with verifiable rewards (RLVR) paradigm, which trains language models using automatically assessed outcomes in mathematical reasoning and code generation (Guo et al., 2025; Team et al., 2025). Earlier code-generation methods use execution outcomes, learned critics, fine-grained unit-test feedback, and code-completion curricula to guide learning (Le et al., 2022; Liu et al., 2023; Dou et al., 2024). GRPO estimates advantages relative to responses sampled for the same prompt, avoiding a separately trained value model (Shao et al., 2024). Subsequent methods and analyses examine how reward normalization and policy regularization affect optimization (Liu et al., 2025; Mroueh, 2026). Sampling becomes particularly important when rewards offer little variation within a group. In GRPO with binary outcome rewards, groups containing only successful or only unsuccessful responses yield zero group-relative advantages. DAPO maintains informative training batches by filtering such groups and replenishing them through additional sampling (Yu et al., 2026), while RLVE adapts the difficulty of procedurally generated tasks to the evolving policy (Zeng et al., 2025). For policy generation, symbolic verification can identify violated authorization properties and, when available, return concrete requests that witness those violations. Descriptions of the violated properties and the corresponding counterexamples can guide subsequent policy generation, but this information is not conveyed by an aggregate correctness reward alone.

Feedback-Guided Training of LLMs. Beyond scalar rewards, natural-language critiques and tool feedback have been used to guide iterative refinement without updating model parameters (Madaan et al., 2023; Gou et al., 2024). Training-based approaches teach models to revise their outputs through supervised learning and multi-turn reinforcement learning (Qu et al., 2024; Kumar et al., 2025; Jiao et al., 2026). Related methods jointly train solution generation and correctness assessment using verifiable outcomes (Liu et al., 2026; Ruan et al., 2026). More directly related to learning from assisted generations, Critique-GRPO uses natural-language critiques to generate refinements and optimizes both initial responses and refinements during online training (Zhang et al., 2025). SDPO uses a feedback-conditioned version of the model as a self-teacher and distills its next-token distributions into the policy conditioned on the original prompt (Hübotter et al., 2026). Learning from assisted generations also requires accounting for how the training responses are obtained. LUFFY combines off-policy demonstrations with on-policy rollouts and uses policy shaping to balance imitation and exploration (Yan et al., 2026). OC-GRPO instead studies a mismatch in conditioning, generating rollouts with privileged guidance while optimizing an importance-corrected objective under the original prompt (Agrawal et al., 2026). Our work investigates how structured evidence from formal verification can be used during training to improve the semantic correctness of generated access-control policies.

Algorithm 1 RAISE with the RAISE-OC instantiation

Require: Initial parameters θ_0; partitions $\mathcal{E}_{\mathrm{SFT}}, \mathcal{E}_{\mathrm{RL}}$; rollout budget $K \geq 2$; feedback limit $1 \leq M \leq \lfloor K/2 \rfloor$; RL epochs N_{RL}; optimization steps U; ϵ, δ, β
Ensure: A policy $\pi_\theta(\cdot \mid D, \Sigma)$ that needs no verifier feedback at inference

$\theta \leftarrow \mathrm{SFT}(\theta_0, \mathcal{E}_{\mathrm{SFT}})$ ▷ Eq. 7
for $e = 1, \ldots, N_{\mathrm{RL}}$ **do**
  for each minibatch $\mathcal{Q} \subset \mathcal{E}_{\mathrm{RL}}$ **do**
    $\theta_{\mathrm{old}} \leftarrow \theta$; $\mathcal{G} \leftarrow [\,]$
    for each $(D, \Sigma, \Pi, \mathcal{R}) \in \mathcal{Q}$ **do**
      $q \leftarrow (D, \Sigma)$; sample $P_k \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid q)$, $k = 1, \ldots, K$
      $(r_k, z_k) \leftarrow \mathrm{Verify}(P_k; \Sigma, \Pi, \mathcal{R})$
      if $\sum_k r_k = K$ **then**
        continue ▷ all correct: skip
      else if $\sum_k r_k > 0$ **then**
        $\Gamma_q \leftarrow \{(q, \{(P_k, r_k)\}_{k=1}^{K})\}$ ▷ mixed: standard GRPO
      else
        $V \leftarrow \bigcup_k V(P_k)$; $h(c) \leftarrow |\{k : c \in V(P_k)\}|$
        if $V = \varnothing$ **then**
          $\mathcal{B} \leftarrow \{f_{\mathrm{syn}}\}$
        else
          $J \leftarrow$ the $\min(M, |V|)$ checks with largest $h(c)$; $\mathcal{B} \leftarrow \{f_c = (d_c, x_c)\}_{c \in J}$
        end if
        Choose $n_c \geq 2$ as evenly as possible with $\sum_{f_c \in \mathcal{B}} n_c = K$; $\Gamma_q \leftarrow [\,]$
        for each $f_c \in \mathcal{B}$ **do**
          Sample $\widetilde{P}_{c,\ell} \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid q, f_c)$, $\ell = 1, \ldots, n_c$; $\widetilde{r}_{c,\ell} \leftarrow \mathrm{Verify}(\widetilde{P}_{c,\ell})$
          Append $((q, f_c), \{(\widetilde{P}_{c,\ell}, \widetilde{r}_{c,\ell})\}_\ell)$ to Γ_q
        end for
      end if
      Append (q, Γ_q) to $\mathcal{G}$
    end for
    if $\mathcal{G} \neq [\,]$ **then**
      Compute group-wise advantages and frozen behavior probabilities ▷ Eq. 13
      for $u = 1, \ldots, U$ **do**
        $\theta \leftarrow \theta + \eta \nabla_\theta \mathcal{J}_{\mathcal{G}}(\theta)$ ▷ Eq. 15
      end for
    end if
  end for
end for
return π_θ

B Formal Definitions and Training Details

This appendix gives the formal definitions behind Sections 2 and 4: verification, the SFT and GRPO objectives, RAISE-OC's guidance construction and objective, an example, and the complete procedure.

B.1 Training Procedure

Algorithm 1 gives the complete procedure.

B.2 Verification

Let $\mathcal{X}_\Sigma$ denote the schema-conforming authorization inputs, each a request with the entity data needed to evaluate it. $\mathrm{Valid}_\Sigma(P)$ holds when P parses and every policy in it validates against Σ; for a valid bundle, $\mathrm{sem}_\Sigma(P) = \{x \in \mathcal{X}_\Sigma \mid P \text{ authorizes } x\}$. A check $c \in \Pi$ has a type $\tau_c \in \{\mathrm{ceil}, \mathrm{floor}, \mathrm{live}\}$, a reference policy R_c, and a domain $D_c \subseteq \mathcal{X}_\Sigma$ given by its declared principal, action, and resource scope. A valid candidate satisfies c, written $P \models_\Sigma c$, when

$$P \models_\Sigma c \iff \begin{cases} \mathrm{sem}_\Sigma(P) \cap D_c \subseteq \mathrm{sem}_\Sigma(R_c), & \tau_c = \mathrm{ceil}, \\ \mathrm{sem}_\Sigma(R_c) \cap D_c \subseteq \mathrm{sem}_\Sigma(P), & \tau_c = \mathrm{floor}, \\ \mathrm{sem}_\Sigma(P) \cap \mathrm{sem}_\Sigma(R_c) \cap D_c \neq \varnothing, & \tau_c = \mathrm{live}. \end{cases} \tag{5}$$

A ceiling failure has a witness in $\mathrm{sem}_\Sigma(P) \cap D_c \setminus \mathrm{sem}_\Sigma(R_c)$ and a floor failure a witness in $\mathrm{sem}_\Sigma(R_c) \cap D_c \setminus \mathrm{sem}_\Sigma(P)$; a liveness failure means the region contains no authorized input, so it has no witness. The diagnostic record of P consists of its validity, the set $V(P) \subseteq \Pi$ of failed checks ($\varnothing$ when P is invalid), and for each $c \in V(P)$ the description d_c and, when the solver returns one, a counterexample x_c. The reward is

$$r(P) = \mathbb{I}[\mathrm{Valid}_\Sigma(P) \wedge P \models_\Sigma c \, \forall c \in \Pi] \,. \tag{6}$$

B.3 Verified SFT Objective

With $o_i^\star = (o_{i,1}^\star, \ldots, o_{i,T_i}^\star)$ the tokens of the target policy for input q_i, SFT minimizes

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\frac{1}{\sum_{i=1}^N T_i} \sum_{i=1}^N \sum_{t=1}^{T_i} \log \pi_\theta\big(o_{i,t}^\star \mid q_i, o_{i,<t}^\star\big) \,, \tag{7}$$

so only policy tokens contribute, weighted equally without per-example length normalization.

B.4 Group Relative Policy Optimization

Given q, the policy $\pi_{\theta_{\mathrm{old}}}$ samples K responses $o_1, \ldots, o_K$ with rewards $r_1, \ldots, r_K$, and every token of o_k receives the advantage $\widehat{A}_k = (r_k - \bar{r})/(s + \delta)$, where $\bar{r}$ and s are the mean and sample standard deviation of the group's rewards and $\delta > 0$ ensures stability (Shao et al., 2024). With the per-token ratio $\rho_{k,t}(\theta) = \pi_\theta(o_{k,t} \mid q, o_{k,<t}) / \pi_{\theta_{\mathrm{old}}}(o_{k,t} \mid q, o_{k,<t})$, GRPO maximizes

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^K \frac{1}{|o_k|}\sum_{t=1}^{|o_k|}\Big(\min\Big(\rho_{k,t}\widehat{A}_k, \mathrm{clip}(\rho_{k,t}, 1-\epsilon, 1+\epsilon)\widehat{A}_k\Big) - \beta\,\mathrm{KL}_{k,t}\Big)\right], \tag{8}$$

where $\mathrm{KL}_{k,t} = D_{\mathrm{KL}}(\pi_\theta(\cdot \mid q, o_{k,<t}) \| \pi_{\mathrm{ref}}(\cdot \mid q, o_{k,<t}))$ and π_{ref} is the fixed SFT policy.

B.5 RAISE-OC's Guidance Construction

Given K candidates with rewards r_k and $n_+ = \sum_k r_k$ successes, RAISE-OC routes each group as in OC-GRPO (Agrawal et al., 2026):

$$\mathrm{Route}(n_+) = \begin{cases} \text{standard GRPO update}, & 0 < n_+ < K, \\ \text{skip}, & n_+ = K, \\ \text{verifier-guided exploration}, & n_+ = 0. \end{cases} \tag{9}$$

For an all-failure group, let $V = \bigcup_k V(P_k)$ be the distinct failed checks and $h(c) = |\{k : c \in V(P_k)\}|$ the number of candidates failing c; invalid candidates contribute no checks. RAISE-OC selects the set J of the $\min(M, |V|)$ checks with the largest $h(c)$, where $1 \le M \le \lfloor K/2 \rfloor$. Each selected check yields a feedback item $f_c = (d_c, x_c)$, omitting x_c when no counterexample is available. If no candidate is valid, so that $V = \varnothing$, RAISE-OC uses a single syntax-correction instruction f_{syn}. The feedback set is $\mathcal{B} = \{f_c\}_{c \in J}$, or $\{f_{\mathrm{syn}}\}$ when $J = \varnothing$.

Each $f_c \in \mathcal{B}$ receives $n_c \ge 2$ rollouts, as evenly as possible, with $\sum_c n_c = K$, and RAISE-OC samples

$$\widetilde{P}_{c,\ell} \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid q, f_c), \qquad \ell = 1, \ldots, n_c. \tag{10}$$

Each guided candidate receives the full-plan reward of Eq. 6, and advantages are normalized within the group of candidates that share f_c:

$$\widehat{A}_{c,\ell} = \frac{\widetilde{r}_{c,\ell} - \bar{r}_c}{s_c + \delta}, \qquad \bar{r}_c = \frac{1}{n_c}\sum_{\ell=1}^{n_c} \widetilde{r}_{c,\ell}, \tag{11}$$

where s_c is the sample standard deviation of the group's rewards. The initial K candidates serve only for routing and guidance, so an all-failure input costs $2K$ generations, of which the K guided ones enter optimization.

B.6 When a Guided Group Yields a Learning Signal

A group is outcome-informative when at least one advantage is nonzero; with binary rewards, exactly when it contains both a success and a failure. If candidates sampled under (q, f_c) are independent with common success probability p_c, the group is outcome-informative with probability

$$g_{n_c}(p_c) = 1 - (1 - p_c)^{n_c} - p_c^{n_c}, \tag{12}$$

since the number of successes is $\mathrm{Binomial}(n_c, p_c)$. Two consequences follow. Guidance helps only if it raises p_c above zero, which inference-time repair suggests verifier feedback does (Appendix F.5). And when four checks are selected, each group has $n_c = 2$ and is informative with probability $2p_c(1 - p_c) \le 1/2$, so exploration creates opportunities for learning signal rather than guaranteeing it.

B.7 Full Off-Context Objective

For a retained input q, let Γ_q be its groups. A mixed-reward input has one group of K candidates with behavior context $b_g = q$; an all-failure input has one group per feedback item, with $b_g = (q, f_c)$ and $n_g = n_c$ candidates, so that $\sum_{g\in\Gamma_q} n_g = K$. For each sampled token we freeze the behavior probability and define

$$\beta_{g,\ell,t} = \pi_{\theta_{\mathrm{old}}}(o_{g,\ell,t} \mid b_g, o_{g,\ell,<t}), \qquad \rho_{g,\ell,t}(\theta) = \frac{\pi_\theta(o_{g,\ell,t} \mid q, o_{g,\ell,<t})}{\beta_{g,\ell,t}}. \tag{13}$$

When $b_g = q$ this is the GRPO ratio; when $b_g = (q, f_c)$ it is the off-context ratio of Eq. 2. With $\ell_\epsilon(\rho, A) = \min(\rho A, \mathrm{clip}(\rho, 1-\epsilon, 1+\epsilon)A)$ and group-normalized advantages $\widehat{A}_{g,\ell}$, each candidate contributes

$$\mathcal{J}_{g,\ell}(\theta) = \frac{1}{|o_{g,\ell}|} \sum_{t=1}^{|o_{g,\ell}|} \left[\ell_\epsilon\Big(\rho_{g,\ell,t}(\theta), \widehat{A}_{g,\ell}\Big) - \beta\, \mathrm{KL}_{g,\ell,t}(\theta) \right], \tag{14}$$

with the KL term evaluated under q. For a minibatch $\mathcal{Q}$ of retained inputs, RAISE-OC maximizes

$$\mathcal{J}_{\mathcal{Q}}(\theta) = \frac{1}{|\mathcal{Q}|} \sum_{q\in\mathcal{Q}} \frac{1}{K} \sum_{g\in\Gamma_q} \sum_{\ell=1}^{n_g} \mathcal{J}_{g,\ell}(\theta). \tag{15}$$

Each retained input contributes exactly K optimized candidates. For mixed-reward inputs the objective reduces to GRPO; for all-failure inputs it averages separately normalized groups without mixing their advantages. Because the numerator is always evaluated under q, every active term updates the policy conditioned on the original input, though advantages may vanish, clipping may suppress terms, and token-level gradients may cancel. The behavior probability must include any sampling transformation such as temperature, and the result is a clipped token-level surrogate in the style of OC-GRPO, not an unbiased sequence-level estimator.

B.8 Illustrative Verifier Feedback

To illustrate the feedback supplied during guided exploration, we construct controlled failure variants of the verified policy for scenario `invoice_approval-issuePayment-T3-045`. These variants are used only to expose representative verifier outputs; they are not presented as historical model generations. We show the failed-check descriptions and the relevant projection of each symbolic counterexample. Solver-assigned fields unrelated to the failed property are omitted for readability.

Listing 1: RAISE feedback suffix

```
A verifier found your earlier attempts violated this requirement;
make sure your policy satisfies it:
- {FAILED_CHECK_DESCRIPTION}
  concrete case your earlier attempt decided wrongly:
  {REDUCED_COUNTEREXAMPLE}
```

The suffix is appended to the original schema–requirement prompt; the earlier failed policy itself is not included. Each selected check defines a separate feedback-conditioned rollout subgroup. The following examples come from scenario `invoice_approval-issuePayment-T3-045`. For readability, the symbolic witnesses are projected onto fields relevant to the failed check; ellipses denote solver-assigned fields that are omitted only from the presentation.

Example B.1: Over-permissive policy: business-hours ceiling violation

```
A verifier found your earlier attempts violated this requirement;
make sure your policy satisfies it:
- A finance manager must not be permitted to issue a payment
  outside business hours.
  concrete case your earlier attempt decided wrongly:
  principal: Person::"P0"
  action: Action::"issuePayment"
  resource: Payment::"P0"
  context: {
    isBusinessHours: false,
    onCorporateNetwork: true
  }
  entities: [
    Person::"P0" {
      isFinanceManager: true,
      ...
    },
    Payment::"P0" {
      invoice: Invoice::"I0",
      ...
    }
  ]
```

Example B.2: Over-restrictive policy: on-network floor violation

```
A verifier found your earlier attempts violated this requirement;
make sure your policy satisfies it:
- A finance manager must be allowed to issue a payment from the
  corporate network without being blocked merely because the
  invoice is not approved.
  concrete case your earlier attempt decided wrongly:
  principal: Person::"P0"
  action: Action::"issuePayment"
  resource: Payment::"P0"
  context: {
    isBusinessHours: true,
    onCorporateNetwork: true
  }
  entities: [
    Person::"P0" {
      isFinanceManager: true,
      ...
    },
    Payment::"P0" {
      invoice: Invoice::"I0",
      ...
    },
    Invoice::"I0" {
      status: "",
      ...
    }
  ]
```

Example B.3: Deny-all policy: liveness violation

```
A verifier found your earlier attempts violated this requirement;
make sure your policy satisfies it:
- At least one finance-manager issue-payment request that satisfies
  the visible rules should remain possible.
  concrete case your earlier attempt decided wrongly:
  LIVENESS VIOLATION: candidate policy is disjoint from the
  required liveness probe, so no approved request slice remains
  possible.
```

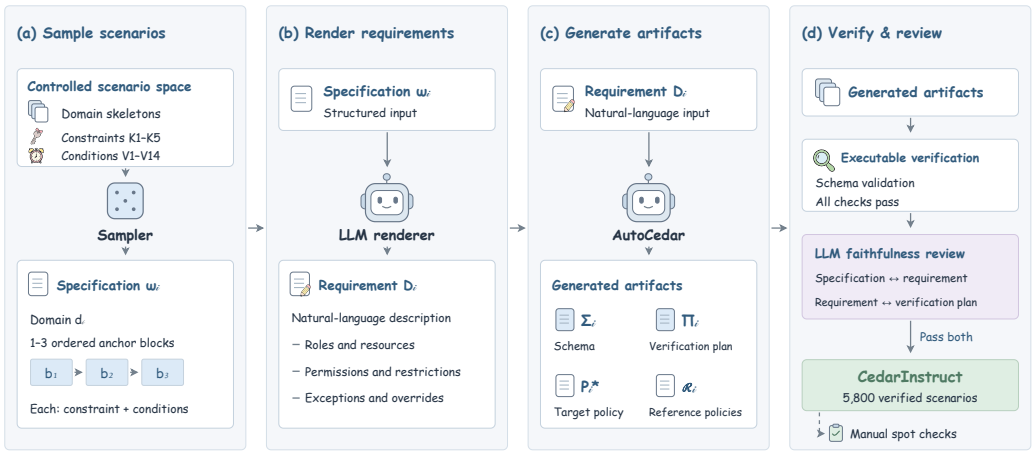

Figure 3: Four-stage construction pipeline of CedarInstruct.

C CONSTRUCTION OF CEDARINSTRUCT

We construct CedarInstruct, a dataset $\mathcal{E} = \{e_i\}_{i=1}^{5800}$ of verified access control scenarios spanning 19 application areas and 44 domains. Each scenario is generated from a sampled structured specification ω_i and stored as a tuple $e_i = (D_i, \Sigma_i, \Pi_i, P_i^\star, \mathcal{R}_i)$, where D_i is the natural-language requirement, Σ_i is the Cedar schema, Π_i is the symbolic boundary plan, $P_i^\star$ is the target Cedar policy bundle, and $\mathcal{R}_i$ is the set of reference-policy files used by the executable checks.

We build each scenario e_i through the four-stage pipeline shown in Figure 3. First, the sampler draws a structured specification ω_i from a domain-constrained scenario space. Second, an LLM renders ω_i into a natural-language requirement D_i. Third, AutoCedar uses D_i to generate the Cedar schema Σ_i, boundary plan Π_i, target policy bundle $P_i^\star$, and reference policies $\mathcal{R}_i$. Finally, every generated scenario undergoes executable validation. To assess construction faithfulness, we additionally conduct an LLM-based review on a sample of the retained scenarios and manually audit a subset of the LLM-reviewed sample using the same criteria. Thus, every retained scenario passes executable verification, whereas faithfulness is assessed through sampled LLM and human review rather than exhaustive per-scenario review.

C.1 STAGE 1: CONTROLLED SCENARIO SPACE AND SAMPLING

A structured specification ω_i is the machine-readable source of a scenario. We represent it as

$$\omega_i = (d_i, \mathbf{b}_i), \qquad \mathbf{b}_i = (b_{i,1}, \ldots, b_{i,s_i}),$$

where d_i identifies a domain skeleton and $\mathbf{b}_i$ is an ordered sequence of anchor rule blocks. The domain skeleton provides the declared set of actions supported by the domain. Each block $b_{i,j}$ selects one of these actions as an anchor and specifies its structural constraint, attached conditions, and any adjudication needed to resolve an override. Only the selected anchor actions are governed by the scenario; the remaining supported actions provide domain context rather than additional access-control requirements. This representation fixes the governed actions and their rule structure before any prose is generated, while the domain skeleton ensures that all referenced entities and fields can be represented in the Cedar schema Σ_i.

Each domain is represented by a hand-written skeleton. The skeleton defines the roles, resources, typed fields, request-context fields, supported actions, and admissible rule structures. It also records which actions can form a linked workflow over the same resource. In such workflows, later actions read state stored as resource fields maintained by the application. This keeps multi-action scenarios compatible with Cedar's stateless request semantics.

The sampler builds each anchor block from two controlled vocabularies. Constraint categories K1–K5 describe the structural access-control constraints assigned to an anchor action (Table 6). Condition types V1–V14 describe additional predicates attached to those constraints (Table 7). We include constraints evaluable from the current request and stored entity fields. Requirements involving prior

Table 6: Constraint categories used by the scenario sampler.

ID	Constraint category	Semantic relation
K1	Separation of duty	The current actor must differ from the actor stored for a conflicting prior action.
K2	Binding of duty	The current actor must match the actor stored for a related prior action.
K3	Override	An exception case takes precedence over the default access-control rule.
K4	Level dominance	The current actor's authority level must dominate the resource's sensitivity level.
K5	Lifecycle-state gate	The current action is allowed only in selected resource lifecycle states.

Table 7: Condition types used by the scenario sampler.

ID	Condition type	Plain meaning
Identity and relationships		
V1	Entity equality	The actor matches an identity field.
V2	Type or role test	The actor has a required role or kind.
V3	Hierarchy membership	The actor belongs to a group or hierarchy.
V4	Set containment	The actor appears in an allowed set.
Values and comparisons		
V5	String condition	A text field matches a value or prefix.
V6	Integer comparison	An integer field crosses a threshold.
V7	Arithmetic condition	A computed value satisfies a threshold.
V8	Decimal comparison	A precise numeric value satisfies a threshold.
V9	Network condition	The request comes from an allowed network.
V10	Time condition	The request satisfies a time or expiry condition.
State, presence, and context		
V11	Enum status	A resource is in a required lifecycle state.
V12	Optional field	An optional field is present or absent.
V13	Tags	A resource carries a required label.
V14	Request context	The request environment satisfies a condition.

actions are included only when the relevant state is already exposed by the domain skeleton. We exclude requirements that need access to unrepresented execution history, global aggregation, or reasoning about state transitions. This restriction does not exclude a separation-of-duty check that compares the current actor with a stored prior actor.

We describe each specification by its scope, depth, and form. The scope is the number of anchor blocks, $s_i = |\mathbf{b}_i|$. A scope-one scenario contains one anchor action, while scope-two and scope-three scenarios contain linked action sequences declared by the domain skeleton. The depth ℓ_i counts rule elements across all anchor blocks, including their structural constraints and attached conditions. The form f_i describes how these elements combine. Scenarios with an explicit override are classified as override-based; among the remainder, those with at least one carve-out are exception-based, and those with restriction-only conditions are conjunctive. These labels describe scenario structure rather than measured model difficulty.

The sampler constructs ω_i directly in this space. It first selects a domain d_i, then chooses $\mathbf{b}_i$ as either a single anchor block or a linked sequence of two or three blocks supported by the domain skeleton. For each block, it selects a constraint category and attaches condition types whose required fields exist in the skeleton and whose combinations are valid under Cedar typing. Multi-action scenarios can only use workflow state exposed by the skeleton as resource fields. Before natural-language rendering, the sampler rejects any specification whose structured content duplicates a previously retained scenario. The retained scenarios therefore remain grounded in their domains while covering distinct combinations of actions, constraints, and conditions.

The condition types describe the meaning of the sampled predicates, not a one-to-one classification of Cedar operators. For example, a time or network condition can be represented by a Boolean context field supplied by the application. Coverage of V9 and V10 therefore does not by itself establish coverage of Cedar's IP-address or datetime operations.

C.2 Stage 2: Natural-Language Requirement Rendering

Given a sampled specification ω_i, we use GPT-5.5 as a renderer to produce the natural-language requirement D_i. The renderer is called with a fixed prompt p_{render} and the sampled specification, written $D_i = \texttt{gpt5.5}(p_{\text{render}}, \omega_i)$. The prompt asks the model to draft an access-control requirement for a realistic organization while preserving the selected anchor actions, the roles, entities, and fields referenced by their rules, and the sampled constraint and condition relationships. Actions listed only as part of the domain inventory provide context and need not be restated as requirements. The prompt also defines the input format, output structure, vocabulary restrictions, conflict-resolution requirements, a missing-field rejection rule, and few-shot examples.

C.3 Stage 3: Cedar Artifact Generation

Given the rendered requirement D_i, AutoCedar (Vatsa et al., 2026) generates a Cedar schema Σ_i containing the entity types, fields, actions, and request context needed to express the requirement. It decomposes D_i into atomic access-control properties and uses an LLM to assign each property a verification relation. The resulting boundary plan $\Pi_i = (\mathcal{C}_i, \mathcal{F}_i, \mathcal{G}_i)$ is stored as executable checks with natural-language descriptions and declared request scopes. The reference policies used by these checks are stored in $\mathcal{R}_i$.

The checks implement the relations defined in Equation 1. A ceiling or floor reference policy R specifies an upper or lower bound through its permitted request set $\text{sem}_{\Sigma_i}(R)$. For a reference-based liveness check, the corresponding slice is $G = \text{sem}_{\Sigma_i}(R)$, and the candidate must authorize at least one request in this slice. Each check is evaluated within its declared principal, action, and resource scope and the input space supported by the symbolic encoding. Request evaluation includes the entity data needed to interpret policy conditions.

The property-atomization prompt assigns paired floor and ceiling checks to exact authorization conditions. Conditions stated only as sufficient produce floor checks, while conditions stated only as necessary produce ceiling checks. Within a shared action and request scope, every floor region must lie within every applicable ceiling region. AutoCedar then searches for $P_i^\star$ through verifier-guided generation and repair, keeping the admitted schema and boundary plan fixed during candidate search.

C.4 Stage 4: Verification and Faithfulness Filtering

Stage 4 combines a dataset-wide executable verification filter with a sampled faithfulness audit. For executable verification, we validate $P_i^\star$ and every reference policy in $\mathcal{R}_i$ under Σ_i, and execute all checks in Π_i using `cedar symcc`. A scenario passes this filter only when validation succeeds and $P_i^\star \models_{\Sigma_i} \Pi_i$.

After executable filtering, we sample retained scenarios for LLM-based faithfulness review. Given (ω_i, D_i, Π_i), the judge assesses whether D_i preserves the sampled anchor rules in ω_i without introducing unsupported requirements, and whether the properties described in Π_i are supported by and sufficiently cover the enforceable requirements in D_i. Liveness checks are assessed for consistency with D_i, rather than requiring a corresponding sentence in the requirement.

We manually review a subset of the LLM-reviewed scenarios using the same criteria. The manual audit compares the structured specification with the rendered requirement and compares the requirement with the generated boundary plan and its reference policies. Symbolic verification establishes satisfaction of the generated plan within the modeled input space; agreement with the natural-language requirement is assessed through sampled faithfulness review rather than proved by the solver.

C.5 Dataset Characterization

Coverage at multiple granularities. Figure 4 shows the area-level distribution of the retained dataset. The 5,800 scenarios span all 19 application areas, with area sizes ranging from 157 to 547 scenarios. At the domain level, all 44 domain skeletons are covered, with 64 to 233 scenarios per domain and a median of 130. Every anchor action declared by each domain skeleton appears in the retained dataset, covering all 108 supported $(\text{domain}, \text{anchor})$ pairs. These counts describe coverage of the declared skeletons, not of access-control applications in general.

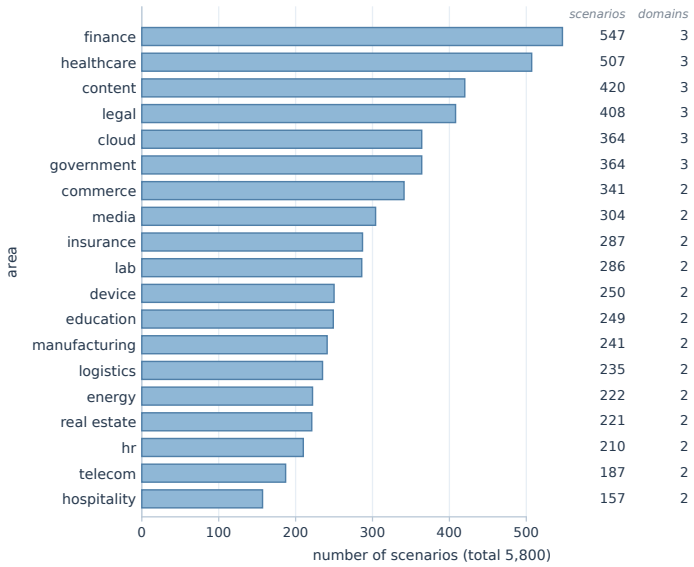

Figure 4: Number of retained scenarios in each application area. Domain and anchor-action coverage are reported in the text.

Scenario structure. The dataset is concentrated in single-action scenarios, with 4,974 scope-one scenarios, 794 scope-two scenarios, and 32 scope-three scenarios. Depth ranges from 1 to 9, with 76.1% of scenarios at depth 2 or 3. Under the mutually exclusive form classification, 3,093 scenarios are conjunctive, 2,022 are exception-based, and 685 are override-based. All five constraint categories and all fourteen condition types appear in the retained dataset. The condition types appear 9,809 times as attached conditions, with 69.5% used as restrictions and 30.5% used as exceptions.

Verification density. Across the dataset, the boundary plans contain 44,388 checks, with a mean of 7.65, a median of 7, and a 90th percentile of 12 checks per scenario. Every scenario contains at least one ceiling and one floor check. Ceiling checks account for 23,909 checks (53.9%), floor checks account for 17,331 checks (39.0%), and liveness checks account for the remaining 3,148 checks (7.1%). These checks measure over-permission and over-restriction relative to the generated boundaries, without requiring textual similarity to $P_i^\star$. Check count describes the size of the verification plan, not necessarily its semantic coverage or difficulty.

Splits. We assign each scenario to exactly one partition using a fixed random seed while preserving coverage across application areas and domains. This produces 4,386 SFT examples, 1,039 RL examples, and 375 test examples. The assignment is fixed before model training and is never changed during SFT-data selection, RL-data selection, model selection, or evaluation. The test split is used only for final in-distribution evaluation and is not used for training, hyperparameter tuning, or checkpoint selection. It covers all 19 application areas and all 44 domains, with 11 to 35 scenarios per area and 5 to 14 scenarios per domain. Because splitting is performed at the scenario level, this is not a schema-, action-, or domain-level out-of-distribution split.

C.6 Example Retained Scenario

We illustrate the construction pipeline using scenario `invoice_approval-issuePayment-T3-045` from the invoice-approval domain. It has scope 1, depth 3, and combines a lifecycle-state gate (K5), a business-hours restriction (V10), and a network exception (V9) on `issuePayment`. For this presentation, the specification and requirement have been clarified to state the allowed and denied cases explicitly, and the boundary plan is shown as a readable checklist. The schema, target policy, and reference policy bodies are unchanged. Each reference policy is stored in a separate file.

Listing 2: Scenario specification

```
DOMAIN:
     finance / invoice approval

ROLES:
     clerks, approvers, finance managers, auditors; a single person may hold both the clerk and approver roles.

RESOURCES:
     invoice records its creator, approver, amount, surcharge, line item count, status, cost center, submission time;
      payment records its invoice, issuer, amount.

REQUEST CONTEXT:
     each request records whether it is made during business hours, whether it originates from the corporate network,
      and whether an emergency has been declared.

ACTIONS:
     create invoice, approve invoice, issue payment, view audit log.

ANCHOR ACTION:
     issue a payment (finance managers only)

RULES (all bear on the anchor action):
- Subject to the restriction and exception below, finance managers must be allowed to issue
payments for approved invoices and must be denied for unapproved invoices. [K5]
- only-when: payments may be issued only during business hours; outside business hours, every
payment request must be denied. [V10]
- unless: on the corporate network, the approved-invoice requirement is lifted. During business
hours, finance managers must be allowed to issue payments from that network regardless of invoice
approval. This exception does not waive the role or business-hours requirement. [V9]
```

Listing 3: Natural-language requirement

```
# Invoice Payment

## Natural-Language Scenario

The system manages and stores invoice approval and payment records.

The people involved are clerks, approvers, finance managers, and auditors.

One person may hold both the clerk and approver roles at the same time.

Every invoice records its creator, approver, amount, surcharge, line item count, status, cost
center, and submission time.

Every payment records its invoice, issuer, and amount.

Each payment request records whether it is made during business hours, whether it originates from
the corporate network, and whether an emergency has been declared.

Anyone who is not a finance manager must be denied permission to issue a payment.

Outside business hours, every request to issue a payment must be denied.

During business hours and outside the corporate network, a finance manager must be allowed to issue
a payment when the invoice has been approved; the manager must be denied permission to issue a
payment when the invoice has not been approved.

During business hours and on the corporate network, a finance manager must be allowed to issue a
payment regardless of whether the invoice has been approved.
```

Listing 4: Cedar schema

```
entity Person { isClerk: Bool, isApprover: Bool, isFinanceManager: Bool, isAuditor: Bool };

entity Invoice { creator: Person, approver: Person, amount: decimal, surcharge: decimal,
                 lineItemCount: Long, status: String, costCenter: String,
                 submissionTime: datetime };

entity Payment { invoice: Invoice, issuer: Person, amount: decimal };

action issuePayment appliesTo {
    principal: [Person],
    resource: [Payment],
    context: { onCorporateNetwork: Bool, emergencyDeclared: Bool, isBusinessHours: Bool },
};
```

Listing 5: Target policy bundle

```
permit (principal is Person, action == Action::"issuePayment", resource is Payment)
when {
    principal.isFinanceManager
    && context.isBusinessHours
    && (context.onCorporateNetwork || resource.invoice.status == "approved")
};
```

Listing 6: Boundary plan

```
REQUEST SHAPE
Principal: Person
Action: Action::"issuePayment"
Resource: Payment

[F1] FLOOR
Name: finance_manager_issue_approved_invoice_off_network_floor
Description: During business hours, a finance manager must be allowed to issue a payment when the
request is outside the corporate network and the invoice has been approved.
Reference: finance_manager_issue_approved_invoice_off_network_floor.cedar

[F2] FLOOR
Name: finance_manager_issue_payment_on_network_floor
Description: During business hours, a finance manager must be allowed to issue a payment from the
corporate network regardless of invoice approval.
Reference: finance_manager_issue_payment_on_network_floor.cedar

[C1] CEILING
Name: finance_manager_issue_payment_network_status_ceiling
Description: A payment may be issued only by a finance manager. Outside the corporate network, the
invoice must also be approved. On the corporate network, invoice approval is not required by this
boundary.
Reference: finance_manager_issue_payment_network_status_ceiling.cedar

[C2] CEILING
Name: finance_manager_issue_payment_business_hours_disjointness
Description: No request to issue a payment may be allowed outside business hours.
Reference: finance_manager_issue_payment_business_hours_disjointness.cedar

[L1] LIVENESS
Name: finance_manager_issue_payment_clean_request_liveness
Description: At least one request by a finance manager during business hours must be allowed, with
either a corporate-network origin or an approved invoice.
Reference: finance_manager_issue_payment_clean_request_liveness.cedar
```

Listing 7: Check-specific reference policies

```
// [F1] finance_manager_issue_approved_invoice_off_network_floor.cedar

permit (principal, action == Action::"issuePayment", resource) when {
    (principal is Person && resource is Payment && principal.isFinanceManager
     && !context.onCorporateNetwork && resource.invoice.status == "approved")
    && !(!context.isBusinessHours)
};

// [F2] finance_manager_issue_payment_on_network_floor.cedar

permit (principal, action == Action::"issuePayment", resource) when {
    (principal is Person && resource is Payment && principal.isFinanceManager
     && context.onCorporateNetwork)
    && !(!context.isBusinessHours)
};

// [C1] finance_manager_issue_payment_network_status_ceiling.cedar

permit (principal, action == Action::"issuePayment", resource) when {
```

Listing 7: Check-specific reference policies (continued)

```
    principal is Person && resource is Payment && principal.isFinanceManager
    && (context.onCorporateNetwork || resource.invoice.status == "approved")
};

// [C2] finance_manager_issue_payment_business_hours_disjointness.cedar

permit (principal is Person, action == Action::"issuePayment", resource is Payment)
when {
    !(!context.isBusinessHours)
};

// [L1] finance_manager_issue_payment_clean_request_liveness.cedar

permit (principal, action == Action::"issuePayment", resource) when {
    principal is Person && resource is Payment && principal.isFinanceManager
    && context.isBusinessHours
    && (context.onCorporateNetwork || resource.invoice.status == "approved")
};
```

C.7 Reading a Cedar Policy

A Cedar policy bundle is a set of `permit` and `forbid` rules. Each rule names the principals, actions, and resources it applies to, and may add conditions in `when` clauses (the rule applies only if the condition holds) and `unless` clauses (the rule applies only if it does not). A request is allowed if and only if at least one `permit` rule applies and no `forbid` rule applies. We illustrate with the invoice-payment scenario of Appendix C.6, which allows finance managers to issue payments only during business hours, and only for approved invoices unless the request comes from the corporate network.

The bundle below looks natural but is wrong. It writes the network exception as a separate rule, and that rule omits the business-hours condition, so a finance manager on the corporate network can issue a payment at any time.

Listing 8: A plausible but incorrect policy bundle

```
permit (principal is Person, action == Action::"issuePayment", resource is Payment)
when {
  principal.isFinanceManager
  && context.isBusinessHours
  && resource.invoice.status == "approved"
};

permit (principal is Person, action == Action::"issuePayment", resource is Payment)
when {
  principal.isFinanceManager
  && context.onCorporateNetwork
};
```

The verifier rejects this bundle because it violates the business-hours ceiling check C2 (Listing 6); any counterexample is a request by a finance manager on the corporate network outside business hours. One correct fix keeps both `permit` rules and adds a `forbid` rule that blocks every payment outside business hours. The target policy in Listing 5 is an equivalent single-rule version.

Listing 9: A corrected policy bundle

```
permit (principal is Person, action == Action::"issuePayment", resource is Payment)
when {
  principal.isFinanceManager
  && context.isBusinessHours
  && resource.invoice.status == "approved"
};

permit (principal is Person, action == Action::"issuePayment", resource is Payment)
when {
  principal.isFinanceManager
  && context.onCorporateNetwork
};

forbid (principal, action == Action::"issuePayment", resource)
unless { context.isBusinessHours };
```

D IMPLEMENTATION HYPERPARAMETERS

Table 8: Training hyperparameters for the SFT and RAISE runs using the `verl` reproduction framework. Qwen3.5-9B SFT was run for four epochs to assess saturation, while Qwen3.8-27B SFT was run for three epochs. All reported SFT results and subsequent RAISE runs use the final epoch-3 SFT checkpoint.

Hyperparameter	**Qwen3.5-9B**	**Qwen3.8-27B**
Supervised fine-tuning		
Adaptation	LoRA	LoRA
LoRA rank r	64	64
LoRA scaling α	128	128
LoRA dropout	0.05	0.05
SFT epochs run	4	3
Optimizer updates	414	414
Optimizer	Fused AdamW	Fused AdamW
Peak learning rate	10^{-4}	10^{-4}
Scheduler	Cosine	Cosine
Warmup ratio	0.10	0.10
Weight decay	0.01	0.01
Global batch size	32	32
Examples per GPU	2	2
Gradient accumulation	4	4
Maximum sequence length	8,192	8,192
Checkpoint used for reporting and RAISE initialization	Epoch 3	Epoch 3
RAISE *reinforcement learning*		
Adaptation	LoRA	LoRA
LoRA rank r	32	32
LoRA scaling α	64	64
LoRA dropout	0.05	0.05
RL epochs	2	2
Optimizer updates	518	518
Optimizer	Fused AdamW	Fused AdamW
Learning rate	10^{-5}	10^{-5}
Scheduler	Constant	Constant
Warmup steps	0	0
Weight decay	0	0
Global prompt batch size	4	4
Initial rollouts K	8	8
Additional guided rollouts	8	8
Feedback limit M	4	4
Sampling temperature	0.9	0.9
Top-p	1.0	1.0
Top-k filtering	Disabled	Disabled
Maximum completion length	3,072	3,072
Optimization iterations	1	1
Clipping parameter ϵ	0.2	0.2
Advantage standard deviation	Sample	Sample
Advantage stabilizer δ	10^{-4}	10^{-4}
KL coefficient β	0.01	0.01
KL reference	Fixed SFT policy	Fixed SFT policy
Training seed	42	42

E BASELINE METHODS

All trainable baselines use the CedarInstruct RL partition and start from the same backbone-specific SFT checkpoint as RAISE. Their common input is $q = (D, \Sigma)$, consisting of the natural-language requirement and Cedar schema. Reward construction, rollout composition, optimization objectives, and update budgets vary by method, as detailed below and in Appendix D. Evaluation uses greedy single-shot generation conditioned only on q, without verifier feedback or iterative repair.

Prompting Baselines. Zero-shot prompting directly asks the pretrained backbone to generate a Cedar policy from the requirement and schema. Structured zero-shot prompting additionally specifies the expected output format and Cedar-generation constraints, without parameter updates or verifier-assisted repair. The prompts are provided in Appendix H. GPT-6 Astra (Max) (OpenAI, 2026b) and Claude Opus 5 (Max) (Anthropic, 2026) are evaluated as independent zero-shot frontier references. GPT-5.5 (Medium) (OpenAI, 2026a), which serves as the LLM backend for model-mediated stages of CedarInstruct construction, is reported separately as a construction-model reference.

Binary Reward GRPO. Binary Reward GRPO applies group-relative policy optimization (Shao et al., 2024) using the same full-verification outcome as RAISE. Let $\mathrm{Valid}_\Sigma(P)$ indicate that P parses and passes schema validation. The reward is

$$r_{\mathrm{bin}}(P) = \mathbb{I}[\mathrm{Valid}_\Sigma(P) \wedge P \models_\Sigma \Pi] \,.$$

Candidates are sampled under the original input, and rewards are normalized within each prompt group to obtain relative advantages. The method performs no feedback-guided regeneration. Consequently, all-failure and all-success groups have zero outcome-based advantages.

Coverage Reward GRPO. Coverage Reward GRPO tests whether a denser scalar reward can provide useful supervision when no candidate satisfies the complete plan. For a schema-valid candidate, let $\mathrm{Pass}(P) \subseteq \Pi$ be its satisfied checks and $\phi(P) = |\mathrm{Pass}(P)|/|\Pi|$. We mark a partial candidate as degenerate when it passes no ceiling check, or passes neither a floor nor a liveness check:

$$d(P) = \mathbb{I}\big[\big(\mathcal{C} \neq \varnothing \wedge \mathrm{Pass}(P) \cap \mathcal{C} = \varnothing\big) \vee \big(\mathcal{F} \cup \mathcal{G} \neq \varnothing \wedge \mathrm{Pass}(P) \cap (\mathcal{F} \cup \mathcal{G}) = \varnothing\big)\big] \,.$$

The guarded coverage reward is

$$r_{\mathrm{cov}}(P) = \begin{cases} 0, & P \text{ does not parse,} \\ 0.10, & P \text{ parses but fails schema validation,} \\ 1, & P \models_\Sigma \Pi, \\ 0.15, & d(P) = 1, \\ 0.15 + 0.65\,\phi(P), & \text{otherwise.} \end{cases}$$

The guard penalizes verification patterns associated with excessively permissive or restrictive policies; it does not establish that a candidate literally permits or denies every request. This baseline provides partial credit but does not use failed-check descriptions or counterexamples for further generation.

Formal Dominance GRPO. Formal Dominance GRPO extracts pairwise preferences from the identities of satisfied checks. Candidate P_a strictly dominates P_b when $\mathrm{Pass}(P_b) \subsetneq \mathrm{Pass}(P_a)$; candidates with incomparable passed-check sets receive no preference relation. Mixed-reward groups use the binary GRPO objective. For an all-failure group, syntax-invalid candidates are removed and valid candidates with identical passed-check sets form equivalence classes. We retain only covering dominance relations, removing relations implied transitively through another class.

Let $\mathcal{H}_q$ denote the retained class relations and let $s_\theta(P \mid q)$ be the policy sequence score under the original input. The dominance loss is

$$\mathcal{L}_{\mathrm{dom}} = -\frac{1}{|\mathcal{H}_q|} \sum_{(A,B)\in\mathcal{H}_q} \frac{1}{|A||B|} \sum_{\substack{P_a \in A \\ P_b \in B}} \log \sigma(s_\theta(P_a \mid q) - s_\theta(P_b \mid q)) \,.$$

The ranking scale and dominance-loss weight are both 1.0. Groups without a strict dominance relation provide no ranking signal, and all-success groups are skipped. The reference-policy KL regularizer is applied only to the mixed-group GRPO branch. This baseline tests whether preferences among existing failed candidates suffice without feedback-conditioned exploration.

Critique-GRPO. We implement a Cedar-specific adaptation of Critique-GRPO (Zhang et al., 2025). Each prompt produces six initial candidates, and failed candidates with verifier feedback undergo up to two refinement rounds. Each round uses at most two failed-check descriptions, prioritizing over-permissive before over-restrictive failures; no separate critic or counterexample is used. We retain two refinements, prioritizing correct candidates, and combine them with the six initial candidates. If fewer than two refinements are available, the best available refinement or an initial candidate is reused. Binary rewards are centered over the combined group without standard-deviation normalization. Initial candidates use the clipped policy-gradient objective, while refinement tokens use the shaping factor $p/(p + 0.1)$ under the original input, without context correction or reference-policy KL regularization.

SDPO. We adapt SDPO (Hübotter et al., 2026) by conditioning the teacher on the original input, an available successful sibling policy, and verifier descriptions of the sampled candidate's failures; the student receives only the original input. Teacher and student score the same completion tokens, and the teacher adapter is updated from the student by exponential moving average with rate 0.05. We minimize forward KL over the student's top 20 tokens and a tail bucket, with token-level importance weights capped at 2.0. This loss replaces GRPO and uses no reference-policy KL penalty. The 9B configuration requires both a successful sibling and verifier feedback, whereas the 27B configuration requires either one.

F Additional Experimental Results

F.1 Construction-Model Reference

GPT-5.5 (Medium) served as the LLM backend for all model-mediated stages of CedarInstruct construction. For transparency, we report its zero-shot performance on the CedarInstruct test split; because it participated in constructing the dataset and its verification artifacts, we exclude it from comparisons. It achieves 82.93% syntax, 38.13% semantic, 66.22% macro, and 63.60% micro success, using on average 12,449 input and 606 output tokens per scenario, with 14.0 seconds latency and an estimated cost of $0.0291.

F.2 Metric Definitions

Syntax is the percentage of policies that parse and pass schema validation. Semantic, our primary metric, is the percentage that are valid and satisfy every verification check. Macro averages the per-scenario fraction of passed checks, and Micro divides total passed checks by total checks. Invalid policies receive no semantic, macro, or micro credit.

F.3 Significance Tests

On the 375-scenario test split, each comparison between two models is paired: both are evaluated on the same scenarios. We report, for RAISE-OC against SFT and against the strongest other instantiation on each backbone, a two-sided exact McNemar test on per-scenario semantic success and a 95% paired bootstrap interval for the difference (10,000 resamples) These tests capture evaluation noise, not variance across training seeds.

F.4 Guidance Ablations

Table 9 ablates RAISE-OC's guidance. Selecting checks by failure frequency outperforms random selection (46.93% vs. 45.60%); exploring each check in a separate guided context outperforms placing all selected feedback in one shared prompt (46.93% vs. 44.80%); and aggregating failures across candidates outperforms giving each rollout its own candidate's failures (46.93% vs. 42.13%), a gap that persists on Qwen3.8-27B (47.20% vs. 45.87%). The last comparison is the closest

Table 9: Guidance ablations for RAISE-OC: single-shot semantic success (%). All feedback items contain a failed-check description and a counterexample; M limits the number of selected checks. Every entry is a single greedy run on the 375-scenario test split.

M	Check selection	Prompt layout	Rollouts	Qwen3.5-9B	Qwen3.8-27B
1	most frequent	one per check	8	44.53	47.47
2	most frequent	one per check	8	41.87	47.20
4	most frequent	one per check	8	**46.93**	**48.00**
4	random	one per check	8	45.60	47.73
4	most frequent	all in one shared prompt	8	44.80	47.47
8	most frequent	one per check	16	45.60	47.20
All	each rollout's failed checks	one per rollout	8	42.13	45.87

Table 10: Feedback-assisted repair on the 375 test scenarios (%). Repair supplies failed-check descriptions and available counterexamples; retry-only reports only that the policy failed.

Method	Initial	Repair 1	Repair 2	Retry only
Qwen3.5-9B				
SFT	42.93	64.00	67.73	45.33
Binary GRPO	43.47	62.93	67.20	45.07
RAISE-OC	**46.93**	**66.93**	**70.40**	**50.40**
Qwen3.8-27B				
SFT	46.40	73.60	76.80	50.40
Binary GRPO	44.80	**76.00**	**80.80**	52.00
RAISE-OC	**48.00**	74.40	78.93	**52.27**

analogue in our experiments to guidance built from a single failed attempt (Agrawal et al., 2026), though both variants here use verifier feedback. Larger budgets did not help: $M = 8$ with 16 guided rollouts reaches 45.60%. Most configurations were run only on Qwen3.5-9B with a single seed, so we treat these comparisons as indicative.

F.5 Verifier Feedback at Inference Time

Table 10 applies verifier feedback at inference time. On every model, one round of repair with failed-check descriptions and counterexamples raises semantic success by 19 to 31 points, whereas a retry that reports only failure adds 2 to 7. The content of verifier feedback, not the extra attempt, drives most of the gain, which is the property RAISE-OC's guided exploration relies on during training. RAISE-OC also improves the starting point for repair: after two rounds it reaches 70.40% on Qwen3.5-9B, versus 67.73% for SFT. Trained models and verifier-guided repair are therefore complementary.

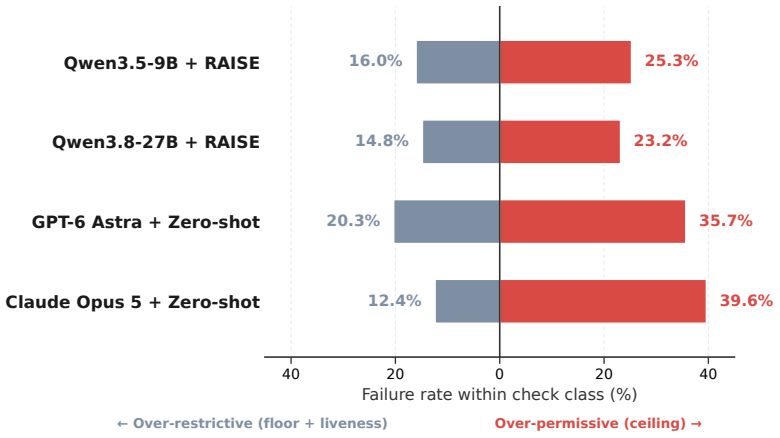

Figure 5: Failure rates within over-restrictive and over-permissive check classes: failed checks divided by evaluated checks in each class.

Table 11: Average per-scenario inference efficiency on the CedarInstruct test split. Lower is better except for Semantic; bold and underline mark the best and second-best results.

Model	Input tokens	Output tokens	Latency (s)	Cost ($)	Semantic (%)
GPT-6 Astra (Max)	24,832	448	14.3	0.0638	33.60
Claude Opus 5 (Max)	49,134	2,412	27.3	0.1375	30.67
Qwen3.5-9B + RAISE	**699**	<u>154</u>	**3.85**	**0.000093**†	<u>46.93</u>
Qwen3.8-27B + RAISE	<u>737</u>	**141**	<u>7.35</u>	<u>0.000370</u>†	**48.00**

† Qwen costs use DeepInfra rates listed by OpenRouter, not measured local deployment costs.

F.6 Authorization Failure Patterns

Figure 5 reports failure rates within each check class. Ceiling checks fail more often than floor and liveness checks for every model shown, making over-permission the dominant failure mode. RAISE-OC has lower over-permissive failure rates (25.3% and 23.2%) than GPT-6 Astra (35.7%) and Claude Opus 5 (39.6%). Its over-restrictive rates (16.0% and 14.8%) fall below Astra's 20.3% but above Opus's 12.4%. Without SFT and Binary Reward GRPO for comparison, these rates do not attribute the reduction in over-permission to the RL stage specifically.

F.7 Inference Efficiency

Table 11 reports latencies of 3.85 and 7.35 seconds per scenario for the RAISE-OC models, versus 14.3 and 27.3 seconds for GPT-6 Astra and Claude Opus 5. The frontier figures are measured through each provider's agent CLI (Codex CLI and Claude Code), not as raw API calls: the task prompt itself is under 1K tokens, but an agent loop resends its full context on every call, so the reported input tokens are summed over all calls, include prompt-cache reads and writes, and contain a fixed harness context much larger than the prompt. Output tokens include reasoning tokens. At estimated costs of $0.000093 and $0.000370 per scenario, the RAISE-OC models are about 686× and 172× cheaper than GPT-6 Astra, the less expensive of the two frontier models in Table 2; because the frontier costs include this harness overhead, the ratios describe end-to-end deployment rather than bare API calls. Latency and cost depend on the provider, hardware, and prompt configuration, so these measurements characterize the evaluated deployments only.

G Limitations

Correctness is relative to generated verification plans. Symbolic checking establishes that a policy satisfies its verification plan, not that it matches every reading of the requirement. Plans are generated by AutoCedar with GPT-5.5, and their faithfulness to the requirements is assessed on a sample rather than exhaustively (Section 3). Because the same pipeline defines the RL reward, the feedback, and the test-set grading, trained models may partly learn the pipeline's reading of ambiguous requirements.

The test split is in-distribution at the domain level. Test scenarios are distinct from training scenarios but share domain skeletons and constraint vocabularies. CedarBench tests transfer to independent scenarios, but we did not evaluate frontier models on it.

Statistical power. All results come from a single training seed, and differences between RL instantiations are often a few scenarios. Paired tests on per-scenario outcomes (Appendix F.3) address evaluation noise but not training variance.

Missing controls. Guided exploration draws K additional candidates for each all-failure prompt, so part of its benefit may come from extra samples; a resampling control, as in DAPO (Yu et al., 2026), would separate the two. We also did not compare verifier guidance with OC-GRPO's reference-solution prefixes or LLM-written hints; OC-GRPO's variance analysis, which favors short guidance, suggests verifier diagnostics should compare favorably, but we have not tested this. Finally, we did not run SFT on all 5,425 verified scenarios, which would show how much of the RL stage's benefit additional demonstrations could provide.

Dataset scope. Most scenarios govern a single action (4,974 of 5,800). Time and network conditions are represented as Boolean context fields, so Cedar's datetime and IP-address operations are not exercised. Scenarios requiring execution history or global aggregation are excluded.

Answer-only supervision. SFT targets contain only the policy. Lin et al. (2025) find that chain-of-thought supervision improves generalization to harder instances; the atomic properties in each verification plan offer a natural reasoning scaffold that we leave to future work.

H PROMPT TEMPLATES

Listing 10 gives the rendering prompt used during dataset construction (Appendix C). Listings 11 and 12 give the plain and structured zero-shot policy-generation prompts, respectively. The plain prompt in Listing 11 is also used to construct both SFT and RL inputs: for SFT, the prompt is paired with the reference Cedar policy as the target response; for RL, the same prompt is used to elicit policy rollouts. Frontier models receive the zero-shot prompts.

Listing 10: Rendering prompt

```
Role:
You are a domain expert drafting an access-control requirements document for a real
organization. Write in plain, natural English. Do not write code or use technical access
-control terminology.

Input:
<structured specification>

Task:
Render <structured specification> as an ordered requirement document. Preserve every selected
anchor action and all roles, entities, fields, constraints, and condition relationships
used by its rules. The ACTIONS field lists the operations supported by the domain. Do
not turn non-anchor actions into additional requirements. For linked workflows, state
how later anchor actions depend on state stored by earlier steps. For exceptions or
overrides, state which cases follow the default rule and which cases are carved out or
take precedence.

Restrictions:
1. Use plain business English only.
2. Do not introduce roles, resources, fields, actions, or requirements outside <structured
   specification>.
3. If an action can be performed by several kinds of people, a prohibition must name all of
   them.

Fitness check:
If any rule needs a field not recorded by the domain, output exactly one line and stop:
REJECT: <missing field>

Examples:
<example_1>
<example_2>
<example_3>

Output:
# Title
## Natural-Language Scenario
<sentences>
```

Listing 11: Policy-generation prompt used for zero-shot evaluation and SFT/RL training.

```
SYSTEM MESSAGE
You are an expert Cedar access-control policy synthesizer. Given a Cedar schema and a natural
-language access-control requirement, write the Cedar policies that exactly implement it.

Output Format:
Only output the final Cedar policy inside <cedar_policy> tags.
<cedar_policy>
Cedar policy here
</cedar_policy>

USER MESSAGE
## Cedar Schema
{CEDAR_SCHEMA}

## Access-Control Requirement
{ACCESS_CONTROL_REQUIREMENT}

Write the Cedar policies now.
```

Listing 12: Structured zero-shot policy-generation prompt

```
SYSTEM MESSAGE

You are a careful Cedar policy generator. Output only final Cedar code.

USER MESSAGE

ROLE

You are an Access Control Policy Engineer with expertise in:
- Cedar policy language
- Access control design
- Translating natural-language specifications into formal policies

Your responsibility is to generate correct, schema-grounded Cedar policies from specifications.

TASK

You are given two inputs:
1. A detailed natural-language access-control requirement
2. A corresponding Cedar schema

Generate a complete Cedar policy that faithfully implements the requirement and conforms to the provided schema.

GUIDELINES

- Faithfully implement all access-control requirements.
- Use only entity types, actions, attributes, and relations defined in the schema.
- Do not invent schema elements.
- Represent every constraint described in the requirement.
- Produce valid Cedar syntax.
- Treat the schema as the authoritative source of valid types and fields.
- Output no explanations, reasoning steps, or comments.

CEDAR SYNTAX CHEAT SHEET

Policy structure:

permit | forbid (
  principal [== | is | in] ...,
  action [== | in] ...,
  resource [== | is | in] ...
)
[when { <condition> }]
[unless { <condition> }];

- permit allows a request.
- forbid denies a request and overrides permit.
- when { C } applies the policy only when C is true.
- unless { C } applies the policy only when C is false.
- when and unless may appear in the same policy.
- Every policy statement ends with a semicolon.

Scope patterns:

principal == User::"alice"
action == Action::"read"
resource == File::"doc1"
principal is User
resource is Document
principal in Group::"engineering"
resource in Folder::"docs"
action in [Action::"read", Action::"write"]

Condition operators:

principal.age >= 18
resource.classification != "HighlyRestricted"
context.networkRiskScore < 20

Comparison operators:

== != < <= > >=

Logical operators:

principal.department == "eng" && !resource.isArchived
principal in Role::"Admin" || principal in Role::"Owner"

Attribute existence:

Guard an optional attribute with has before accessing it:

resource has expiryDate &&
resource.expiryDate > context.now

Collection membership:
```

Listing 12: Structured zero-shot policy-generation prompt (continued)

```
principal in resource.readers
resource.tags.contains("public")

Attribute access:

principal.department
resource.owner
resource.project.status
context.isCompliantDevice

KEY RULES

1. Use only entity types, actions, and attributes defined in the schema.
2. Do not invent names.
3. Always guard optional attributes with has.
4. Use in for entity hierarchy or entity-set membership.
5. Use .contains() for primitive-set membership.
6. Remember that forbid overrides permit.
7. Cedar denies a request unless at least one applicable permit authorizes it.

POLICY SKELETON

permit (
  principal is User,
  action == Action::"some_action",
  resource is SomeResource
) when {
  true
};

ACCESS-CONTROL REQUIREMENT

{ACCESS_CONTROL_REQUIREMENT}

CEDAR SCHEMA

{CEDAR_SCHEMA}

OUTPUT FORMAT

Output only the final Cedar policy inside <cedar_policy> tags.

<cedar_policy>
Cedar policy here
</cedar_policy>
```