LPG: Balancing Efficiency and Policy Reasoning in Latent Policy Guardrails

Nanxi Li **Zhengyue Zhao** **Chaowei Xiao**
Johns Hopkins University

Abstract

Guardrails are a critical safety layer for modern AI systems, but their operating regime is changing. As LLMs are deployed as customized assistants, safety policies are increasingly specified at inference time by users, organizations, or regulatory contexts. This makes safety enforcement fundamentally dynamic: the guardrail should adapt to changing safety policies without retraining. Yet this requirement creates a fundamental tension: faithfully judging complex policy contexts demands reasoning capability, while practical deployment requires low-latency responses. We introduce Latent Policy Guardrail (LPG), a guardrail framework that learns **semantic latent deliberation** over dynamic policies. LPG compresses the internal deliberation needed for intent interpretation and policy grounding into continuous states supervised by decision-relevant semantics. At inference time, it generates only a compact verdict anchored to the violated policy clauses, providing clause-level attribution while avoiding the latency of explicit reasoning. Across policy guardrail benchmarks, LPG-4B reaches **84.5%** average safety accuracy and **77.9%** F1 by compressing deliberation into just **10** latent tokens, outperforming the strongest dynamic baseline while running roughly **11**$\times$ faster than Qwen3-4B-Thinking under the single-sample evaluation setup. Code: `https://github.com/SaFo-Lab/Latent_Policy_Guard`.

1 Introduction

Large language models (LLMs) now power customer-facing chatbots [19], autonomous agents [31], and healthcare assistants [27], and ensuring that their outputs respect deployment-specific norms has become a central safety concern [26, 2]. The dominant defense is the *guardrail model*, a lightweight classifier that screens prompts and responses against a safety taxonomy [12, 7, 38]. Such models are typically trained on fixed taxonomies (e.g., the OpenAI Moderation API) and have been deployed at scale precisely because they are fast and decisive.

Fixed taxonomies, however, are increasingly out of step with how real systems are deployed. An HR chatbot must prohibit hiring discrimination, a financial advisor must enforce regulatory disclaimers, an educational platform must filter age-inappropriate content. The rise of agentic AI multiplies the demand further, as rules are attached to specific tools, tasks, and tenants [24, 22]. These constraints cannot all be enumerated at pre-training time, and they evolve faster than any retraining cycle. Recent *policy-aware* guardrails [10, 20] address this by accepting natural-language policies at inference time and judging each interaction against the supplied rules, with no retraining required when the policy changes.

Policy-awareness, however, shifts the burden onto *reasoning*: the guardrail must read a potentially long policy document, identify which clauses apply, and produce a verdict grounded in those clauses, yet existing approaches sit on opposite ends of an unfavorable trade-off. Non-reasoning guardrails such as DynaGuard [10] are fast but brittle (Section 3): their verdicts swing by up to 7.4 accuracy points under simple permutations of the policy list, and a counterfactual probe that drops only the

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

rule an unsafe sample violates fails to flip their verdict, betraying reliance on positional artifacts and content priors rather than the supplied clauses. Reasoning guardrails such as GuardReasoner [23] and ThinkGuard [35] buy accuracy with explicit chains of thought that, in our measurements, run more than an order of magnitude slower, prohibitive for real-time moderation pipelines. An effective guardrail must therefore anchor each verdict in the specific violated clause and do so independently of surface position, both signatures of *deep* policy understanding, yet the cost of explicit deliberation rules out solutions that rely on long verbal rationales.

A natural remedy is to move the deliberation off the surface tokens. Recent work has shown that LLM reasoning can be performed in continuous latent space rather than via discrete tokens [8, 9, 1], yielding speedups on math and planning. Adopting this idea for guardrails, however, is far from a drop-in transfer, and is where the central technical question of this paper lies. In math, latent thoughts are useful because the answer is a deterministic function of an algorithmic working memory, so token-level reconstruction of a teacher's rationale is a natural training target [8, 29]. Safety moderation has no comparable algorithmic substrate: the answer is a function of which clause in a heterogeneous, user-supplied policy is implicated, and of the latent intent behind a possibly disguised request, both paraphrastically variable across deployments. Forcing the latents to reproduce specific tokens of a teacher's rationale would over-constrain them, while leaving them unsupervised reduces them to opaque computation buffers untethered from policy content. We argue that what a guardrail actually needs is **semantic latent deliberation**: each latent stage should retain the decision-relevant gist of its reasoning, the user's underlying intent and the policy clauses it touches, so that a short latent rollout can stand in for a long verbal one without losing policy grounding. This contract, together with the need to anchor verdicts to specific clauses, calls for a guardrail-specific latent design that prior continuous-thought work does not address.

We instantiate semantic latent deliberation in the proposed Latent Policy Guardrail (LPG), a framework that reasons *deeply* but *fast* over user-supplied policies. The two components below are not independent contributions but coupled instruments of one underlying principle, that every reasoning step, whether explicit text or compressed latent, must commit to specific policy semantics rather than diffuse safety priors:

Structured reasoning (Section 4.2). Rather than unconstrained free-form rationales, we impose a three-stage evaluation template: (1) *intent analysis*, examining the conversation history to surface jailbreaks hidden behind role-play, hypotheticals, or indirect phrasing; (2) *policy analysis*, selectively identifying the subset of relevant clauses rather than enumerating all K items, encouraging semantic matching; and (3) *verdict formulation*, producing the final decision *anchored* to the violated indices P^*. The format simultaneously grounds reasoning in policy content and yields auditable, machine-parseable verdicts.

Latent reasoning tailored to policy grounding (Section 4.3). Building on continuous-thought reasoning [8, 9, 1], we compress the intent and policy stages into continuous hidden states. The harder half of the problem is supervising what those states carry; three coupled design choices, together tailored to policy-conditioned moderation, realize the semantic contract above. *(i) Stage-aligned latent slots.* Instead of a single undifferentiated continuous-thought buffer, we allocate separate latent budgets to the intent and policy stages, giving the two reasoning sub-tasks distinct stage-aligned positions. *(ii) Semantic-content supervision.* Rather than reconstructing teacher tokens, we train each slot to preserve the gist of its stage by reconstructing a compact teacher *summary*, routed through the same base LM that consumes the latents, which directly shapes the latents to be policy-meaningful representations, not opaque computation buffers. *(iii) Teacher hidden-state distillation at decision-relevant positions*, applied at both stage boundaries and the verdict-onset position, to align the latent pathway with the teacher's decision-relevant representations. The full multi-objective loss is detailed in Section 4.6.

Empirically, semantic latent deliberation delivers a substantially better accuracy–latency Pareto frontier (Sections 5 and 6). On in-distribution benchmarks, LPG-4B reaches 84.5% average safety accuracy and 77.9% F1, outperforming the strongest dynamic baseline and reasoning baselines while running $\sim 11\times$ faster than Qwen3-4B-Thinking. The gains transfer out of distribution: LPG reaches 96.4% F1 on HarmBench and remains the best dynamic guardrail on WildGuardTest. Ablations isolate the answer-position distillation sub-loss as the load-bearing supervision signal (removing it costs nearly 20 accuracy points), and a latent-budget sweep shows that just 10 latent tokens already approach full explicit reasoning at $\sim 5\times$ lower latency, with smaller gains at larger budgets. To our

knowledge, LPG is the first framework to apply latent reasoning specifically to policy-grounded safety moderation, demonstrating that latent compression is a practical bridge between fast static classifiers and slow but expressive policy-reasoning guardrails.

2 Related Work

LLM guardrail models. Llama Guard [12] established fine-tuning LLMs on fixed safety taxonomies. Subsequent work scaled this across model families (ShieldGemma [38], Qwen3Guard [39]) and model sizes (Llama Guard 3 [5], PolyGuard [15]), yet all require retraining for novel policies. Reasoning-based methods like GuardReasoner [23] and ThinkGuard [35] improve accuracy via explicit chain-of-thought [16, 14], but incur large latency overhead. LPG addresses this trade-off by replacing verbose textual reasoning with compact latent representations.

Dynamic policy-aware guardrails. A growing line of work conditions safety judgments on user-specified policies provided at inference time. DynaGuard [10] introduces a framework where natural-language policies are included in the prompt, along with a companion benchmark, DynaBench, for evaluation. YuFeng-XGuard [20] proposes a hierarchical inference strategy that decouples policy from risk perception, enabling policy updates without retraining. In the agent safety domain, AGrail [24] and AgentDoG [22] develop guardrails that monitor agent trajectories against task-specific safety constraints. Our work shares the policy-aware philosophy of these methods but focuses on the largely unexplored question of how to *efficiently* reason over policies, rather than relying on computationally expensive explicit generation.

Latent reasoning in language models. Recent work has demonstrated that LLMs can perform effective reasoning in continuous latent space rather than through discrete token generation. COCONUT [8] proposes a paradigm where continuous thoughts encode multiple alternative reasoning paths. CODI [29] shows that aligning the student's hidden state at a single token to the teacher's answer-position token, recovers explicit-CoT accuracy on math. Follow-up studies show that steering a single latent reasoning feature can improve accuracy without explicit CoT [9], that supervised thinking states in embedding space achieve competitive performance [1], and that latent reasoning transfers across domains [37]. Emergent search behaviors have also been observed in latent reasoning models [4]. AISA [30] leverages latent safety awareness through attention-head analysis for jailbreak defense without fine-tuning. Concurrent work DRAFT [32] compresses agent-trajectory safety into a single continuous latent draft supervised end-to-end by binary cross-entropy on static policies. Our LPG instead targets *user-supplied* policies at inference time, which motivates our stage-aligned latent slots, semantic-content supervision via teacher distillation and summary reconstruction, and clause-anchored verdicts, none of which are addressed by DRAFT.

3 Investigation Experiments

To validate the importance of policy-aware reasoning in guardrail models, we conduct two preliminary investigation experiments.

3.1 Performance with and without Policy

We evaluate Qwen3-4B, Qwen3-4B (Thinking), and DynaGuard-8B under three conditions: **Full Policy** (standard); **Remove All Policies**, a degenerate baseline that drops the entire policy list; and the counterfactual **Remove Violated only**, which drops *only* the policy item(s) an unsafe sample violates and keeps the rest, flipping the ground truth to `safe` so a policy-grounded model should follow.

As depicted in Figure 1 left, withholding the full policy drops Accuracy by 12–20 points and collapses F1 from $26.9/55.1/56.6\%$ to $<6\%$ as models default to safe. The counterfactual is more revealing (Figure 1, right): Qwen3-4B's flip-to-safe rate is 64%, Thinking strengthens it to 82%, but specialised DynaGuard-8B *collapses* to 36%, below its own Full-Policy verdict correctness.

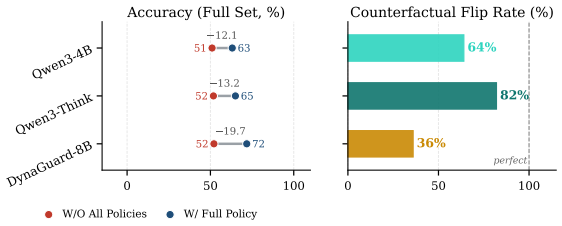


Figure 1: **Policy grounding probes. Left:** Accuracy on the full set with vs. without the policy. **Right:** counterfactual flip rate on the unsafe subset: fraction of samples whose verdict correctly flips to safe after only the violated rule(s) are removed; higher = the model anchors on the specific violated clause.

Model	Acc (%)		CR (%)
	Orig.	Shuf.	mean±std
Qwen3-4B	75.60	72.83	79.64±40.27
Qwen3-4B (Think)	77.25	77.68	89.13±31.13
GuardReasoner-3B	60.70	67.07	65.66±47.49
DynaGuard-8B	81.65	74.23	86.73±33.93

Table 1: Consistency under policy shuffling on GuardSet-X. **Orig./Shuf.**: Safety Accuracy with original / perturbed orderings; **CR**: per-sample verdict consistency rate.

3.2 Policy Shuffling Experiments

We test whether models reason over policy content by evaluating the policy list in three orderings (original, reversed, random) and measuring per-sample verdict consistency rate (CR) and the Accuracy gap between original and shuffled inputs.

As shown in Table 1, mean CR ranges from 65.7% (GuardReasoner-3B) to 89.1% (Qwen3-4B-Think), but the high standard deviations (31.1–47.5) reveal that many verdicts flip arbitrarily under shuffling, and Accuracy drops by up to 7.4 points. Adding explicit reasoning does some help: Qwen3-4B-Thinking's CR *increases* from 79.6% to 89.1%, suggesting deliberate thinking can help mitigate it.

Findings and implications. Both probes diagnose the same failure: existing baselines do not condition on the *specific* active rule set: removing the violated rule does not reliably flip the verdict, and shuffling the rule list does. The regression is sharpest on DynaGuard-8B, the only specialised guardrail in both probes: it is worst on the counterfactual and shows the largest Accuracy drop under shuffling (-7.4 pt), showing that policy-grounded training alone does not ensure invariance to these perturbations. This motivates the two design choices in Section 4: (i) *policy anchoring*, which forces each unsafe verdict to commit to specific violated indices P^* rather than emit a verdict in isolation, and (ii) a training corpus that uses various policy lists with augmentations, so the model cannot collapse onto positional or memorisation shortcuts.

4 Method

4.1 Problem Formulation

Given content to moderate x and a policy document $\mathcal{P} = \{p_1, \ldots, p_K\}$ of K natural-language policy items, the guardrail f_θ must produce a binary verdict $y \in \{\texttt{safe}, \texttt{unsafe}\}$ and, when $y = \texttt{unsafe}$, the set $P^* \subseteq \{1, \ldots, K\}$ of violated policy indices ($P^* = \emptyset$ when safe). We write the full context as $\mathbf{c} = (\mathcal{P}, x)$. The challenge is that faithfully evaluating $\mathbf{c}$ against all K items demands substantial reasoning, yet must respect a strict latency budget.

4.2 Structured Evaluation Template

Two design questions motivate this template. *First, why constrain reasoning at all?* Free-form chain-of-thought is the obvious baseline [23, 35], but it leaks two pathologies into a guardrail: (i) it generates many tokens that are irrelevant for the verdict, inflating latency, and (ii) it tends to skim policy text and fall back on pretrained safety priors, which Section 3 shows are insufficient for user-defined rules. *Second, why three stages and in this order?* Many real-world unsafe interactions disguise malicious intent under benign surface forms (role-play, hypotheticals) [40]. Organizing the trace around an *intent* hypothesis before policy analysis decouples "what is the user really asking for" from "which rule covers it," which encourages the two reasoning functions to receive distinct supervision; the final verdict, anchored to specific indices P^*, then forces any positive judgment to

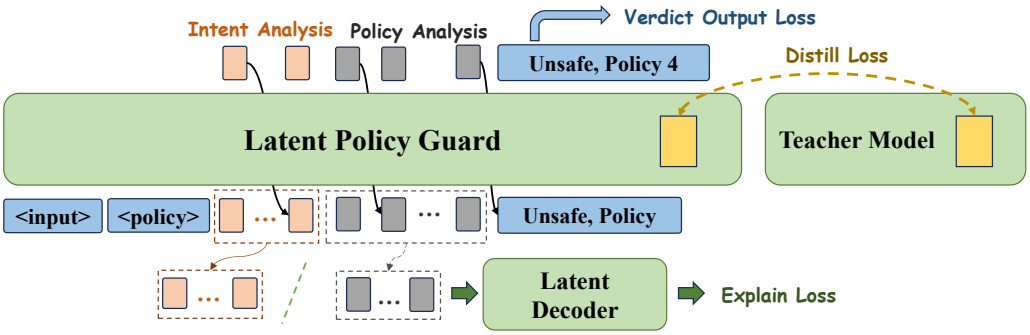


Figure 2: Overview of LPG. The student compresses intent analysis and policy analysis into two latent stages, then decodes the verdict and violated policy indices.

point at concrete clauses. Inspired by structured safety alignment [16, 18], we therefore constrain the model to a fixed three-stage reasoning trace.

Stage 1: Intent analysis. The model first analyzes the content to moderate x to uncover the underlying intent, with particular attention to jailbreaks that hide malicious instructions through role-play, hypothetical scenarios, or indirect phrasing.

Stage 2: Policy analysis. Rather than enumerating all K items, the model selectively identifies a subset $\mathcal{K}_{\text{relevant}} \subset \{1, \ldots, K\}$ of relevant policies and assesses, for each, whether the conversation constitutes a violation. This selective treatment is more efficient than full enumeration and encourages semantic understanding rather than positional pattern matching.

Stage 3: Verdict formulation. The model emits a compact natural-language string (one of "`safe`", "`unsafe, policy` n", or "`unsafe, policy` $n_1, n_2, \ldots$") that combines the binary verdict with P^*. The format is parseable by a deterministic regex and significantly shorter than a JSON-formatted alternative, while anchoring each unsafe verdict to specific clauses for downstream auditing.

4.3 Latent Reasoning via Continuous-Thought Compression

While the structured template constrains the format of reasoning, the token count for Stages 1–2 still scales with conversation length and policy count. To reduce inference cost further, we compress these two stages into continuous latent representations, building on the observation that language-space verbalization is often unnecessarily verbose for the underlying computation [8, 29]. Stage 3 remains explicit text, preserving interpretability of the final decision.

Why guardrail latents need a different supervision signal. Prior latent-reasoning works on math and planning [8, 29, 33] treat the latent thought as a compressed surrogate for the algorithm's intermediate computations: token-level reconstruction (or its information-theoretic equivalent) is a sensible target because the answer is a deterministic function of the working memory. Guardrail reasoning is qualitatively different. The verdict for an unsafe interaction depends on (a) which natural-language clause is violated and (b) the latent intent behind a possibly disguised request, both of which are paraphrastically variable across the policies a deployed model will encounter. Forcing the latents to reproduce specific tokens of the teacher's rationale would over-constrain them; what we actually need is for each latent slot to retain the decision-relevant semantic content of its stage. We translate this view into three concrete design choices: (i) two *stage-aligned* latent slots that mirror the structured template, giving intent and policy-violation reasoning distinct stage-aligned positions; (ii) *semantic-content supervision* via teacher-summary reconstruction routed through the same base LM that consumes the latents, shaping the latents so that the LM's continuation matches the gist of the stage; and (iii) *teacher hidden-state distillation* at stage boundaries and at the verdict-onset position, which directly transfers the teacher's decision-producing representations into the student's latent pathway. Both signals are formalized in Section 4.6. The combination yields short latents that nonetheless remain aligned with policy content rather than acting as opaque computation buffers.

Latent token replacement. Let $\mathbf{h}_t^{(L)} \in \mathbb{R}^d$ denote the top-layer hidden state at position t. We replace explicit token generation for Stages 1–2 with *latent tokens* whose embeddings are set directly from the previous hidden state via a learned projection: $\mathbf{e}_{t+1} = \text{Proj}(\mathbf{h}_t^{(L)})$, yielding a continuous thought chain that proceeds in latent space without materializing into discrete tokens. The supervision signals that shape these latents (a stage-summary reconstruction loss and teacher hidden-state distillation) are introduced together in Section 4.6.

4.4 Policy Anchoring

Requiring the model to emit the violated indices P^* acts as both an inductive bias and a training signal. It discourages vague safety judgments untied to any rule in $\mathcal{P}$, enables automated auditing of which clauses each flagged item cites, and provides fine-grained supervision beyond the binary verdict, encouraging genuine policy comprehension rather than surface pattern matching.

4.5 Training Corpus

LPG is trained on a $40k$-record mixture combining two policy-grounded datasets (DynaBench [10] and GuardSet-X [34]) with a diverse set of public safety moderation datasets, including Beaver-Tails [13], Aegis-AI Content Safety v2 [6], SaladBench [17], Toxic-Chat [21], and XSTest v2 [28]. To unify these heterogeneous sources, we reorganize the GuardSet-X from single policy into a shared multi-policy book format, and for the general safety moderation datasets we write policy items from their native safety taxonomies. All sources are then converted into the single $(\mathcal{P}, x, y, P^*)$ schema with teacher-generated structured reasoning (Section 4.2); the final mixture is 45.5% safe / 54.5% unsafe with mean 0.63 violations per record. The GuardSet-X test split is held out for evaluation.

The corpus uses varied policy lists to discourage reliance on policy ordering or memorized rule strings. To enforce this, (i) each source's native taxonomy is rewritten into a single-sentence "policy book"; (ii) every record is paired with a policy list, always including the violated rule(s) when unsafe and (iii) all rationales are produced by a Qwen3-32B teacher [36] conditioned on the ground-truth verdict (y, P^*), which reduces reasoning–label drift before it can propagate into the student's latents. The full curation pipeline is shown in Appendix B.

4.6 Training Procedure

We supervise a fast student that reasons in latent space using explicit structured reasoning traces. Qwen3-32B supplies the offline traces and summaries; hidden-state targets during training come from a stop-gradient explicit-reasoning pass through the same Qwen3-4B backbone. This distinction permits layer-wise alignment without a cross-model hidden-state mapping. Training is single-phase end-to-end with four loss terms; full pseudocode is in Appendix B.5.

Verdict output loss. The student emits the compact verdict string after the latent rollout, preceded by a learned `<eot>` marker. We apply standard next-token cross-entropy on the verdict positions:

$$\mathcal{L}_{\text{out}} = -\sum_{t \in T_{\text{verdict}}} \log P_\theta(r_t \mid r_{<t}, \mathbf{c}). \tag{1}$$

Teacher hidden-state distillation. We distill the teacher's reasoning trajectory at two complementary positions: the *answer position*, where we align all L layers between student and teacher at the verdict-onset token (transferring the verdict-producing computational state across the residual stack), and the *stage boundaries* `</Intent>`, `</Risk>`, where we align each stage's final top-layer latent to the teacher's hidden state at the matching boundary token. The answer-position term is closely modeled after CODI's designated-token self-distillation [29], which shows that aligning student and teacher hidden states at a single answer-position token suffices to inherit explicit-CoT accuracy on math; we extend that recipe to all L layers and complement it with the stage-boundary term so that each stage-aligned latent slot is anchored to the teacher's representation at its corresponding boundary token, which CODI's single-point design does not provide. Both alignments use SmoothL1

normalized by the reference-state standard deviation $\sigma(\cdot)$ for layer-scale invariance:

$$\mathcal{L}_{\text{distill}}^{\text{ans}} = \frac{1}{L}\sum_{l=1}^{L} \frac{\text{SmoothL1}\Big(\mathbf{h}_{\text{stu}}^{(l)}[t_a],\, \mathbf{h}_{\text{tea}}^{(l)}[t'_a]\Big)}{\sigma\Big(\mathbf{h}_{\text{tea}}^{(l)}[t'_a]\Big)}, \quad \mathcal{L}_{\text{distill}}^{\text{stage}} = \sum_{k\in\{1,2\}} \frac{\text{SmoothL1}\Big(\mathbf{z}_{m_k}^{(k)},\, \mathbf{h}_{\text{tea}}^{(L)}[b_k]\Big)}{\sigma\Big(\mathbf{h}_{\text{tea}}^{(L)}[b_k]\Big)}, \tag{2}$$

where t_a, t'_a are answer positions in the student/teacher sequences, $\mathbf{z}_{m_k}^{(k)}$ is the final latent of stage k, and b_k is the closing-tag position for stage k in the teacher. The two sub-losses target complementary cross-sections of the residual stream (all-layer alignment at one point in time vs. top-layer alignment at two stage transitions), and combine as $\mathcal{L}_{\text{distill}} = \mathcal{L}_{\text{distill}}^{\text{ans}} + \beta\, \mathcal{L}_{\text{distill}}^{\text{stage}}$ with β=0.1 (Appendix B).

Summary reconstruction for semantic supervision. Distillation transfers the teacher's representations point-wise, but it does not by itself ensure that each latent slot remains semantically interpretable to the base LM. To preserve that grounding, we add a stage-summary reconstruction objective in place of the token-level reconstruction loss used by math-style latent reasoning. Let $\mathbf{Z}^{(k)} \in \mathbb{R}^{m_k \times d}$ denote the m_k latent tokens of stage $k \in \{1, 2\}$. The teacher provides an essential-information target $\mathbf{s}^{(k)}$: the detected intent (including any hidden malicious goal) for k=1, and the relevant policy clauses plus a concise violation rationale for k=2. We project $\mathbf{Z}^{(k)}$ through a learned multi-layer projector Proj_k (Linear–GELU–LayerNorm–Dropout), prepend the result to the teacher-forced summary embeddings, and reconstruct $\mathbf{s}^{(k)}$ via next-token prediction through the base LM:

$$\mathcal{L}_{\text{explain}} = \sum_{k\in\{1,2\}} \left(-\sum_{t} \log P_\theta\Big(s_t^{(k)} \mid f_\theta\Big([\text{Proj}_k(\mathbf{Z}^{(k)}); \mathbf{s}_{<t}^{(k)}]\Big)\Big)\right). \tag{3}$$

Routing reconstruction through the base LM rather than a separate decoder directly shapes the model's internal representations of the latent tokens. The projectors $\{\text{Proj}_k\}$ are used only during training; at inference, the model operates purely in latent space and decodes the verdict directly.

Reference reasoning loss. A separate forward pass through the teacher's full explicit reasoning sequence (`<Intent>`, `<Risk>`, `<Output>`) supplies a cross-entropy backbone-LM regularizer that helps retain of explicit reasoning capability:

$$\mathcal{L}_{\text{ref}} = -\sum_{t\in T_{\text{reasoning}}} \log P_\theta(r_t \mid r_{<t}, \mathbf{c}). \tag{4}$$

Joint objective. The four losses combine in a single phase:

$$\mathcal{L} = \lambda_{\text{out}}\mathcal{L}_{\text{out}} + \lambda_{\text{distill}}\mathcal{L}_{\text{distill}} + \lambda_{\text{ref}}\mathcal{L}_{\text{ref}} + \lambda_{\text{explain}}\mathcal{L}_{\text{explain}}, \tag{5}$$

with $\mathcal{L}_{\text{explain}}$ from Eq. 3. The explain projectors are discarded at inference; all other components remain active.

5 Main Results

5.1 Policy Safeguarding Evaluation

We first compare LPG against existing guardrails on the two policy-grounded benchmarks that match the training distribution: GuardSet-X and the challenging augmented DynaBench split. DynaBench is itself a demanding policy-grounded benchmark, with each example carrying on average 13.8 policy items and up to 91 in the heaviest configurations, which requires the model to identify the relevant policies from a large set of distractors. Inspired by the failure modes surfaced in Section 3, we further augment DynaBench along two axes: (i) randomly shuffle policy lists for a subset of examples, and (ii) include counterfactual variants in which the violated rule is removed from the policy list (so the corresponding label flips to safe). Details for this augmentation are in Appendix A.2. The shuffling tests whether the model anchors on policy content rather than positional cues, and the counterfactuals test whether the model's verdicts are actually grounded in the specific violated clauses rather than relying on safety priors.

Table 2 shows that LPG delivers the strongest performance across both policy-grounded benchmarks, reaching 84.52/77.85 average Acc/F1. For comparison on DynaBench (Aug), which is in-distribution

Table 2: Results on GuardSet-X and DynaBench. **Acc**: Safety Accuracy (binary verdict correctness). **F1**: Safety F1 (violation detection). **Lat.**: average inference time per sample in milliseconds (single-sample batch on an A100-80GB; see Section A). **Avg.**: per-metric mean across the two benchmarks.

Model	GuardSet-X			DynaBench (Aug)			Avg.		
	Acc	F1	Lat.	Acc	F1	Lat.	Acc	F1	Lat.
Qwen3-4B	75.60	71.86	249	63.05	26.88	227	69.33	49.37	238
Qwen3-4B-Thinking	77.25	74.07	9747	65.01	55.12	7019	71.13	64.60	8383
DynaGuard-4B	81.00	79.33	222	59.48	53.19	242	70.24	66.26	232
DynaGuard-8B	81.65	79.73	1501	71.82	56.59	1091	76.74	68.16	1296
GuardReasoner-3B	60.70	38.30	3537	58.74	31.28	3993	59.72	34.79	3765
GuardReasoner-8B	63.16	22.48	4072	57.66	27.77	3969	60.41	25.13	4021
LPG-4B (Ours)	**96.85**	**96.88**	625	**72.19**	**58.82**	871	**84.52**	**77.85**	748

Table 3: Out-of-distribution Safety F1 (%) on HarmBench, WildGuardTest, and PolicyGuardBench. Entries marked with † indicate that the corresponding benchmark is in-distribution for that model.

Family	Model	HarmBench	WildGuardTest	PolicyGuardBench
Instruction-following	Qwen3-4B	83.21	78.88	53.48
	GPT-4o	82.27	80.87	87.77
Static guardrail	LlamaGuard3-8B	67.96	68.47	59.52
	ShieldGemma-9B	67.96	57.74	34.72
	GuardReasoner-8B	91.86	89.17†	77.64
Dynamic guardrail	DynaGuard-4B	86.32	81.09	77.02
	LPG-4B (Ours)	96.44	84.09	77.85

for both LPG and DynaGuard and tests robustness under shuffled and counterfactual policy variants: LPG ranks first at 72.19/58.82. The gain is narrow in absolute terms but the relative pattern (DynaGuard-4B drops more Acc compared with the original DynaBench in Appendix Table 7, while LPG retains its lead) suggests LPG's policy anchoring is more robust to surface-form perturbation. These gains come with a favorable accuracy–efficiency trade-off: at 748 ms average latency, LPG is roughly $11\times$ faster than the explicit-reasoning Qwen3-4B-Thinking and $5.4\times$ faster than GuardReasoner-8B under the same single-sample setup, narrowing the latency gap to lightweight classifiers while preserving the benefits of policy-grounded reasoning. A decoding-variance sweep on a subset is reported in Appendix C; it measures decoding variability rather than uncertainty due to test-set sampling or training.

5.2 Out-of-Distribution Evaluation

To assess whether LPG generalizes beyond its training distribution, we evaluate on three benchmarks that were not seen during training: two general safety datasets, **HarmBench** [25] and **WildGuardTest** [7], and one policy-grounded benchmark, **PolicyGuardBench** [34]. HarmBench and WildGuardTest ship without explicit policy strings, so we synthesize per-example policy lists from each benchmark's native taxonomy; PolicyGuardBench supplies ∼60k policy-violation labels over web-agent trajectories. We compare LPG against a broad cross-section of baselines spanning three families: instruction-following models (Qwen3-4B [36], GPT-4o [11]), static policy guardrails (ShieldGemma [38], GuardReasoner [23]), and dynamic policy guardrails (DynaGuard [10]).

Table 3 shows that LPG transfers beyond its training distribution, scoring 96.44 on HarmBench and 84.09 on WildGuardTest, the strongest result among models for which the benchmark is out-of-distribution. On the policy-grounded PolicyGuardBench, LPG reaches 77.85 F1, edging DynaGuard-4B (77.02) and trailing only the much larger GPT-4o, indicating that the benefits of latent policy reasoning are not confined to the training setup. The comparison also highlights a clear pattern: methods tied to fixed taxonomies degrade sharply when the policy space changes, with ShieldGemma-9B falling to 34.72 and LlamaGuard3-8B to 59.52 on PolicyGuardBench, whereas LPG performs

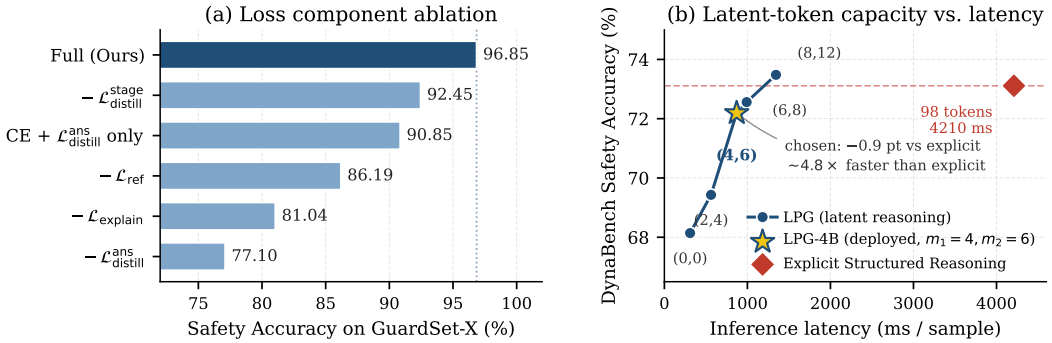


Figure 3: **Ablation studies.** **(a)** Loss-component ablation: GuardSet-X Safety Accuracy when each term (or distillation sub-loss) is removed from Eq. 5; dotted line marks the full model. **(b)** Latent-token capacity sweep on DynaBench; the red diamond is the explicit-reasoning baseline.

better on this benchmark with the supplied rules. Overall, these results suggest that LPG generalizes across both conventional safety moderation and policy-grounded agent settings while retaining the efficiency advantages of latent reasoning.

6 Ablation Study

Loss components. The verdict output loss ($\mathcal{L}_{\text{out}}$) teaches the prediction task, while the answer-position sub-loss ($\mathcal{L}_{\text{distill}}^{\text{ans}}$) transfers decision-relevant states: the former teaches the output format and the latter transfers the teacher's verdict-producing hidden states to the student's latent pathway. Removing $\mathcal{L}_{\text{distill}}^{\text{ans}}$ drops accuracy by 19.75 points, the largest regression among the displayed loss ablations. The remaining components provide complementary semantic supervision: $\mathcal{L}_{\text{explain}}$ encourages each latent stage to retain the information for its reasoning function, the stage-boundary sub-loss $\mathcal{L}_{\text{distill}}^{\text{stage}}$ supplies a direct hidden-state target for each stage's final latent token, and $\mathcal{L}_{\text{ref}}$ regularizes the backbone LM and helps retain of the explicit reasoning capability that underpins the latent representations.

Matched training comparison. Table 4 holds the Qwen3-4B backbone, training examples, and prediction task fixed. LPG improves average accuracy by 6.20 points over direct prediction and is 0.64 points below explicit reasoning. This controls for backbone and data, while the training objectives differ across configurations.

Table 4: Matched training comparison. Safety Accuracy (%); all three configurations use the same backbone and LPG training corpus.

Configuration	GuardSet-X	DynaBench (Aug)	Average
Direct verdict $(0, 0)$	88.50	68.14	78.32
Latent reasoning $(4, 6)$	96.85	72.19	84.52
Explicit structured reasoning	97.20	73.11	85.16

Latent token count. We vary the latent budget (m_1, m_2) allocated to the intent and risk stages, keeping their ratio at approximately 2:3 to match the explicit reasoning length proportion. The $(0, 0)$ endpoint serves as a useful reference: with no latent positions, the model collapses to a no-thinking baseline, i.e., a DynaGuard-style direct-verdict guardrail fit to the LPG corpus. It scores 68.14% on DynaBench, roughly 4 points below the deployed $(4, 6)$ setting, isolating the gain attributable to the latent reasoning pathway from that of the training data alone. Performance rises with the latent budget, where LPG reaches 73.48% on DynaBench, marginally exceeding the explicit-reasoning baseline while running at 1344 ms per sample. Each additional latent token, however, triggers one extra forward pass through the base LM, so the latency curve grows linearly, and the explain loss becomes harder to optimize at larger budgets. We therefore deploy the configuration that lands within roughly one point of the saturated setting: $(m_1, m_2) = (4, 6)$ achieves 72.19% at 871 ms per

sample (0.92 points below the explicit baseline at $\sim 4.8\times$ lower latency), which we select as the accuracy–latency trade-off for our evaluation.

6.1 Policy grounding and adaptive attacks

We repeat the motivating diagnostics from Section 3 with LPG (Table 5). LPG achieves the highest counterfactual flip rate and shuffled-policy accuracy, although its verdicts are not completely invariant to clause order. When the entire policy list is withheld, LPG's accuracy falls to 55%, further indicating dependence on the supplied rules. These controlled interventions support policy dependence; they do not establish robustness to arbitrary policy changes or explain the full latent computation.

Table 5: Policy-grounding diagnostics under the settings of Figure 1 and Table 1. Flip: unsafe-to-safe verdict rate after removing violated rules. CR: consistency rate (mean $\pm$ standard deviation). All values are percentages.

Model	Flip	Original Acc	Shuffled Acc	CR
Qwen3-4B	64.00	75.60	72.83	79.64 ± 40.27
Qwen3-4B-Thinking	82.00	77.25	77.68	89.13 ± 31.13
DynaGuard-8B	36.00	81.65	74.23	86.73 ± 33.93
LPG-4B	87.00	96.85	94.25	88.10 ± 20.86

We also evaluate a black-box, adaptive PAIR-style [3] response attack. The operator controls the fixed policy and user messages; the attacker iteratively rewrites only the agent response, using binary guard decisions and at most ten queries per example. It has no access to weights, gradients, logits, latent states, or clause explanations. We test both policy-aware and policy-hidden attackers. On examples initially classified correctly as unsafe, GPT-5.6 independently checks candidates that flip the decision to safe; a candidate counts as successful only if it still violates the original policy in the fixed user context. Table 6 reports the cumulative attack success rate at each query budget. LPG has lower attack success rates than Qwen3-4B in most tested settings and matches it in one case. These results are limited to the tested attack and automated validation; they do not establish deployment-grade robustness.

Table 6: Adaptive response-level attack success rate ($\%$) at different query budgets. The policy is either visible or hidden to the attacker; lower is better. Candidates are checked for continued policy violation.

Policy	Attacker	Target	@1	@2	@5	@10
Aware	Qwen3.5-9B	LPG-4B	3.3	6.7	6.7	6.7
		Qwen3-4B	6.7	6.7	10.0	16.7
	DeepSeek-V4-Flash	LPG-4B	6.7	10.0	10.0	13.3
		Qwen3-4B	13.3	26.7	30.0	46.7
Hidden	Qwen3.5-9B	LPG-4B	0.0	0.0	3.3	3.3
		Qwen3-4B	3.3	6.7	6.7	13.3
	DeepSeek-V4-Flash	LPG-4B	0.0	0.0	0.0	3.3
		Qwen3-4B	10.0	13.3	13.3	16.7

7 Conclusion

We presented **LPG**, a policy-grounded guardrail that compresses two-stage safety reasoning (intent analysis and risk assessment against user-supplied policies) into a small budget of continuous latent tokens, supervised by a teacher-distilled multi-objective loss. Across policy guardrail benchmarks, LPG matches or exceeds the strongest dynamic and reasoning-based baselines while running roughly $5\times$ faster than the strongest reasoning baseline and $11\times$ faster than Qwen3-4B-Thinking on a single-sample setup. We hope this work motivates further study of latent reasoning as a practical bridge between fast static classifiers and slow but expressive reasoning guardrails, particularly in deployment regimes where policies are dynamic and latency budgets are tight.

Acknowledgements

This paper is partially supported by Schmidt Sciences AI2050, **Amazon Nova AI Challenge: Trusted Software Agents, Amazon** Initiative for Interactive AI (**AI2AI**), Samsung Research via the LEAP-U program.

References

[1] Ido Amos, Avi Caciularu, Mor Geva, Amir Globerson, Jonathan Herzig, Lior Shani, and Idan Szpektor. Latent reasoning with supervised thinking states. *arXiv preprint arXiv:2602.08332*, 2026.

[2] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[3] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*, 2023.

[4] Jasmine Cui and Charles Ye. Emergent search and backtracking in latent reasoning models. *arXiv preprint arXiv:2602.08100*, 2026.

[5] Igor Fedorov, Kate Plawiak, Lemeng Wu, Tarek Elgamal, Naveen Suda, Eric Smith, Hongyuan Zhan, Jianfeng Chi, Yuriy Hulovatyy, et al. Llama guard 3-1b-int4: Compact and efficient safeguard for human-ai conversations. *arXiv preprint arXiv:2411.17713*, 2024.

[6] Shaona Ghosh, Prasoon Varshney, Makesh Narsimhan Sreedhar, Aishwarya Padmakumar, Traian Rebedea, Jibin Rajan Varghese, and Christopher Parisien. AEGIS2.0: A diverse AI safety dataset and risks taxonomy for alignment of LLM guardrails. *arXiv preprint arXiv:2501.09004*, 2025.

[7] Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *arXiv preprint arXiv:2406.18495*, 2024. NeurIPS 2024.

[8] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024. COLM 2025.

[9] Zhenghao He, Guangzhi Xiong, Bohan Liu, Sanchit Sinha, and Aidong Zhang. Reasoning beyond chain-of-thought: A latent computational mode in large language models. *arXiv preprint arXiv:2601.08058*, 2026.

[10] Monte Hoover, Vatsal Baherwani, Neel Jain, Khalid Saifullah, Joseph Vincent, Chirag Jain, Melissa Kazemi Rad, C. Bayan Bruss, Ashwinee Panda, and Tom Goldstein. Dynaguard: A dynamic guardian model with user-defined policies. *arXiv preprint arXiv:2509.02563*, 2025.

[11] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[12] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

[13] Jiaming Ji, Mickel Liu, Juntao Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. *arXiv preprint arXiv:2307.04657*, 2023. NeurIPS 2023.

[14] Mintong Kang and Bo Li. r^2-guard: Robust reasoning enabled llm guardrail via knowledge-enhanced logical reasoning. *arXiv preprint arXiv:2407.05557*, 2024.

[15] Priyanshu Kumar, Devansh Jain, Akhila Yerukola, Liwei Jiang, Himanshu Beniwal, Thomas Hartvigsen, and Maarten Sap. Polyguard: A multilingual safety moderation tool for 17 languages. *arXiv preprint arXiv:2504.04377*, 2025.

[16] Jing-Jing Li, Valentina Pyatkin, Max Kleiman-Weiner, Liwei Jiang, Nouha Dziri, Anne G. E. Collins, Jana Schaich Borg, Maarten Sap, Yejin Choi, and Sydney Levine. Safetyanalyst: Interpretable, transparent, and steerable safety moderation for ai behavior. *arXiv preprint arXiv:2410.16665*, 2024.

[17] Lijun Li, Bowen Dong, Ruohui Wang, Xuhao Hu, Wangmeng Zuo, Dahua Lin, Yu Qiao, and Jing Shao. Salad-bench: A hierarchical and comprehensive safety benchmark for large language models. *arXiv preprint arXiv:2402.05044*, 2024. ACL 2024 Findings.

[18] Nanxi Li, Zhengyue Zhao, G Edward Suh, Marco Pavone, and Chaowei Xiao. Prism: Robust vlm alignment with principled reasoning for integrated safety in multimodality. *arXiv preprint arXiv:2508.18649*, 2025.

[19] Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. Beyond single-turn: A survey on multi-turn interactions with large language models. *arXiv preprint arXiv:2504.04717*, 2025.

[20] Junyu Lin, Meizhen Liu, Xiufeng Huang, Jinfeng Li, Haiwen Hong, Xiaohan Yuan, Yuefeng Chen, Longtao Huang, Hui Xue, Ranjie Duan, Zhikai Chen, Yuchuan Fu, Defeng Li, Lingyao Gao, and Yitong Yang. Yufeng-xguard: A reasoning-centric, interpretable, and flexible guardrail model for large language models. *arXiv preprint arXiv:2601.15588*, 2026.

[21] Zi Lin, Zihan Wang, Yongqi Tong, Yangkun Wang, Yuxin Guo, Yujia Wang, and Jingbo Shang. Toxicchat: Unveiling hidden challenges of toxicity detection in real-world user-ai conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4694–4702, 2023.

[22] Dongrui Liu, Qihan Ren, Chen Qian, Shuai Shao, Yuejin Xie, Yu Li, Zhonghao Yang, Haoyu Luo, Peng Wang, Qingyu Liu, et al. Agentdog: A diagnostic guardrail framework for ai agent safety and security. *arXiv preprint arXiv:2601.18491*, 2026.

[23] Yue Liu, Hongcheng Gao, Shengfang Zhai, Yufei He, Jun Xia, Zhengyu Hu, Yulin Chen, Xihong Yang, Jiaheng Zhang, Stan Z. Li, Hui Xiong, and Bryan Hooi. Guardreasoner: Towards reasoning-based llm safeguards. *arXiv preprint arXiv:2501.18492*, 2025.

[24] Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. Agrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8104–8139, 2025.

[25] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024. ICML 2024.

[26] Mohammad Niknazar, Paul V Haley, Latha Ramanan, Sang T Truong, Yedendra Shrinivasan, Ayan Kumar Bhowmick, Prasenjit Dey, Ashish Jagmohan, Hema Maheshwari, Shom Ponoth, et al. Building a domain-specific guardrail model in production. *arXiv preprint arXiv:2408.01452*, 2024.

[27] Syed Arman Rabbani, Mohamed El-Tanani, Shrestha Sharma, Syed Salman Rabbani, Yahia El-Tanani, Rakesh Kumar, and Manita Saini. Generative artificial intelligence in healthcare: applications, implementation challenges, and future directions. *BioMedInformatics*, 5(3):37, 2025.

[28] Paul Röttger, Hannah Rose Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. Xstest: A test suite for identifying exaggerated safety behaviours in large language models. *arXiv preprint arXiv:2308.01263*, 2023.

[29] Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 677–693, 2025.

[30] Weiming Song, Xuan Xie, and Ruiping Yin. Aisa: Awakening intrinsic safety awareness in large language models against jailbreak attacks. *arXiv preprint arXiv:2602.13547*, 2026.

[31] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[32] Lin Wang, Junfeng Fang, Dan Zhang, Fei Shen, Xiang Wang, and Tat-Seng Chua. Draft: Task decoupled latent reasoning for agent safety. *arXiv preprint arXiv:2604.03242*, 2026.

[33] Xiaoqiang Wang, Suyuchen Wang, Yun Zhu, and Bang Liu. System-1.5 reasoning: Traversal in language and latent spaces with dynamic shortcuts. *arXiv preprint arXiv:2505.18962*, 2025.

[34] Xiaofei Wen, Wenjie Jacky Mo, Yanan Xie, Peng Qi, and Muhao Chen. Towards policy-compliant agents: Learning efficient guardrails for policy violation detection. *arXiv preprint arXiv:2510.03485*, 2025.

[35] Xiaofei Wen, Wenxuan Zhou, Wenjie Jacky Mo, and Muhao Chen. Thinkguard: Deliberative slow thinking leads to cautious guardrails. *arXiv preprint arXiv:2502.13458*, 2025. ACL 2025.

[36] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[37] Xinwu Ye, Yicheng Mao, Jia Zhang, Yimeng Liu, Li Hao, Fang Wu, Zhiwei Li, Yuxuan Liao, Zehong Wang, Yingcheng Wu, et al. Latentchem: From textual cot to latent thinking in chemical reasoning. *arXiv preprint arXiv:2602.07075*, 2026.

[38] Wenjun Zeng, Yuchi Liu, Ryan Mullins, Ludovic Peran, Joe Fernandez, Hamza Harkous, Karthik Narasimhan, Drew Sellars, Thomas Mesnard, and Yashvi Jain. Shieldgemma: Generative ai content moderation based on gemma. *arXiv preprint arXiv:2407.21772*, 2024.

[39] Haiquan Zhao, Chenhan Yuan, Fei Huang, Xiaomeng Hu, Yichang Zhang, An Yang, Bowen Yu, Dayiheng Liu, Jingren Zhou, Junyang Lin, et al. Qwen3guard technical report. *arXiv preprint arXiv:2510.14276*, 2025.

[40] Zhengyue Zhao, Yingzi Ma, Somesh Jha, Marco Pavone, Patrick McDaniel, and Chaowei Xiao. Armor: Aligning secure and safe large language models via meticulous reasoning. *arXiv preprint arXiv:2507.11500*, 2025.

Supplementary Material

A Experimental Setup

A.1 Baseline Models

We evaluate the following baseline models:

Static Policy Baselines. Llama Guard 3 [5] and ShieldGemma-9B [38] are static guardrail models trained on fixed safety taxonomies. They cannot adapt to custom policies.

Dynamic Policy-Aware Baselines. DynaGuard-4B/8B [10] introduces a framework for user-defined safety policies. Qwen3-4B [36] and GPT-4o [11] are evaluated zero-shot with policies provided in-context. Qwen3-4B-Thinking extends Qwen3 with explicit thinking tokens for reasoning.

Reasoning-Based Baselines. GuardReasoner-3B/8B [23] trains on 127K samples with explicit chain-of-thought rationales and applies hard-sample DPO. ThinkGuard [35] distills structured critiques from larger LLMs. While these achieve strong accuracy, they incur substantial latency overhead (3–4$\times$ slower than non-reasoning baselines).

A.2 Datasets

DynaBench [10] consists of difficult policy-grounded safety evaluation examples. Each example includes content to moderate and a safety config with a list of policy items expressed as natural language rules. The benchmark is unusually demanding on the policy-reasoning side: each example carries on average 13.8 policy items and up to 91 in the heaviest configurations, so the model must read a long policy list, locate the small subset that actually applies, and emit verdict + violated index without distraction from the surrounding clauses. To probe the failure modes surfaced in Section 3, we report on a safety-augmented split, **DynaBench (Aug)**, that combines two complementary perturbations of the released test set:

- *Policy shuffling.* For 50% of the examples we keep the original policy list and ground-truth label but draw a uniformly random permutation of the policy items, so that the violated index moves around the list. This isolates positional bias without changing what the policy says.
- *Counterfactual unsafe $\rightarrow$ safe.* For 30% of the examples that are originally labeled unsafe, we strip *all* policies the example violates from the policy list and flip the gold label to `safe`; the remaining (non-violated) policies are kept and the resulting list is shuffled. A model that truly grounds its verdict on the active rule set should follow the label flip, whereas a model leaning on content priors will keep predicting unsafe.

The two perturbations are sampled independently per example, so a small fraction receive both. We also keep the unaugmented split, **DynaBench (Original)**, on which we report Table 7. DynaGuard was trained on DynaBench data. Comparing original and augmented results measures the combined effect of the perturbations, rather than isolating a single cause.

Table 7: Results on the original DynaBench (no augmentation). **Acc**: Safety Accuracy. **F1**: Safety F1. **Latency**: average inference time in milliseconds. DynaGuard is trained on DynaBench data; the gap relative to Table 2 (DynaBench Aug) reflects the combined effect of the evaluation perturbations.

Model	Acc	F1	Latency (ms)
Qwen3-4B	56.90	23.02	232
Qwen3-4B-Thinking	67.40	53.29	7258
GuardReasoner-3B	58.56	31.19	4228
DynaGuard-4B	71.82	69.94	258
LPG	73.90	61.25	710

GuardSet-X is a policy-grounded safety moderation benchmark where each test example is associated with exactly one policy item that determines the safety label. This one-to-one mapping enables precise

evaluation of policy identification accuracy. We randomly select 2000 examples from GuardSet-X as test set.

Out-of-distribution benchmarks. For the OOD evaluation in Section 5.2, we use three benchmarks not seen during training. *HarmBench* [25] is a red-teaming benchmark with 320 harmful behaviors spanning 7 semantic categories. *WildGuardTest* [7] contains 1,725 prompt-response pairs annotated for harmfulness across 13 canonical categories; an example is unsafe iff the prompt or response is harmful. *PolicyGuardBench* [34] is a policy-grounded benchmark for detecting policy violations in web-agent trajectories, with within- and cross-subdomain policy pairings and a prefix-based detection task in addition to full-trajectory evaluation. HarmBench and WildGuardTest do not ship policy strings, so we synthesize per-example policy lists from each benchmark's native taxonomy (4–10 candidate policies per sample, always including the violated category for unsafe examples), matching the usage format of dynamic policy settings. The policy books map the benchmarks' native categories to one-sentence rules; for example i, a seeded sampler (seed $42 + i$) selects and shuffles the rules. These constructed lists test transfer under a policy-conditioned interface, but are not naturally authored policy documents. We report Safety F1 for these three benchmarks because their class balances differ. Latency is omitted because static and dynamic guardrails receive different inputs, making raw timings difficult to compare.

A.3 Evaluation Metrics

We report two categories of metrics:

Safety Metrics: Safety Accuracy (Acc) measures binary classification correctness. Safety F1 (F1) evaluates violation detection performance, where precision is the fraction of predicted violations that are correct and recall is the fraction of violations detected. Consistency Rate (CR) measures agreement across policy order permutations.

Efficiency Metrics: Latency reports average inference time per sample in milliseconds on an NVIDIA A100 80GB GPU with batch size 1 and temperature 0.0.

A.4 Implementation Details

Unless otherwise specified, the main benchmark evaluations use: NVIDIA A100 80GB GPU, batch size 1 (realistic streaming scenario), temperature 0.0 for deterministic evaluation, FP16 precision. Non-reasoning models use max tokens = 512; reasoning models use max tokens = 8192. For LPG, we use $m_1 = 4$ latent tokens for Stage 1 (intent) and $m_2 = 6$ latent tokens for Stage 2 (risk), yielding 10 total latent reasoning tokens.

B Detailed Training Configuration

B.1 Model Architecture

LPG uses Qwen3-4B as the base causal LM. We add LoRA adapters (rank r=128, α=32, dropout 0.05) to all attention and feedforward projection matrices (`q_proj`, `k_proj`, `v_proj`, `o_proj`, `up_proj`, `down_proj`, `gate_proj`). Three special tokens are appended to the vocabulary: `[PAD]`, `<bot>` (begin-of-thought, appended to the question to mark the start of latent reasoning), and `<eot>` (end-of-thought, prepended to the verdict to mark the transition from latent to explicit generation).

Projection module. Between latent rollout steps, the top-layer hidden state $\mathbf{h}_t^{(L)} \in \mathbb{R}^d$ is mapped back to the embedding space via a learned MLP projector:

$$\text{Proj}(\mathbf{h}) = \text{LN}(\mathbf{W}_2\, \text{GELU}(\mathbf{W}_1\, \text{Dropout}(\mathbf{h}))), \tag{6}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times d_p}$, $\mathbf{W}_2 \in \mathbb{R}^{d_p \times d}$, d_p=2560, and LN is LayerNorm. The same projector is shared across all latent steps and stages.

Explain projectors. For the summary reconstruction loss, each stage has a dedicated explain projector consisting of N_{proj} stacked layers of Linear–GELU–LayerNorm–Dropout (each $\mathbb{R}^d \to \mathbb{R}^d$). We use N_{proj}=3 layers with dropout 0.1. These projectors are discarded at inference time.

B.2 Training Corpus: Composition and Curation

This subsection expands Section 4.5 with the full per-source breakdown and the three curation principles (taxonomy normalization, per-example policy-list synthesis, and teacher-grounded reasoning generation) that turn the raw $40{,}041$-record collection into a single training distribution.

Per-source composition. The policy-grounded portion ($20{,}991$ records) supplies canonical user–policy–conversation triples: DynaBench [10] ($13{,}500$ records) and GuardSet-X [34] ($7{,}491$ records), structured with explicit policy organization by subdomain. The general-guardrail portion ($19{,}050$ records) draws from BeaverTails [13] ($8{,}600$, 14 harm categories), Aegis-AI Content Safety v2 [6] ($5{,}000$, 5-level safety taxonomy), SaladBench [17] ($3{,}000$ from the attack-enhanced split, 6/16/66 hierarchical taxonomy bound at the 16-task level), Toxic-Chat [21] ($2{,}000$, real-world toxicity labels), and XSTest v2 [28] (450, retained as a hard-negative over-refusal signal). The final mixture is 45.5% safe / 54.5% unsafe, with 7.5% of unsafe examples carrying multiple violated clauses (mean 0.63 violations per record).

Taxonomy normalization. Each source ships its own categorical taxonomy, with very different surface forms (BeaverTails' fine-grained harm labels vs. SaladBench's hierarchical task names vs. Aegis' 5-level safety taxonomy). To prevent the student from over-fitting to any specific phrasing, we rewrite every category into an imperative single-sentence rule (e.g. *"Do not give instructions for acquiring, manufacturing, or using illegal drugs, controlled substances, or prohibited weapons"*), producing a unified "policy book" per source whose surface forms differ from those used in DynaBench/GuardSet-X.

Per-example policy-list synthesis. Existing guardrails latch onto policy ordering (Section 3). We therefore pair every training example with a freshly sampled policy list rather than reusing a fixed taxonomy. For each record we draw $K \sim \text{Uniform}\{4, 10\}$ policies from the source's policy book (*always* including the violated rule(s) when unsafe), and shuffle the resulting list. With probability 0.30 we additionally seed the list with 1–2 policies drawn from *other* sources' books, so that the model must reason about heterogeneous policy phrasings within a single example (cross-taxonomy mixing). Stratified subsampling along (safe/unsafe $\times$ category) keeps long-tail categories from being starved during training. This diversification reduces repeated list configurations and discourages the positional shortcuts exposed in Section 3.

Teacher-grounded reasoning generation. For every record, an offline pass through a Qwen3-32B teacher [36] produces (i) the explicit structured trace $\langle$`Intent`, `Risk`, `Output`$\rangle$ used to supervise the reference reasoning loss $\mathcal{L}_{\text{ref}}$ and the answer-position distillation, and (ii) the per-stage `IntentSummary` / `RiskSummary` targets used by the explain loss $\mathcal{L}_{\text{explain}}$. The teacher is conditioned on the ground-truth verdict (y, P^*) so that its rationale is constrained to *justify the correct decision* rather than reason free-form; this reduces reasoning–label drift that could otherwise propagate into the student's latents. Summaries are capped at 192 tokens to enforce compression.

B.3 Training Data Pipeline

Teacher reasoning format. The Qwen3-32B teacher generates explicit structured reasoning for each training example in the format: `<Intent>...</Intent>\n\n<Risk>...</Risk>\n\n<Output>safe / unsafe, policy...</Output>`. The Intent block analyzes the user's true intent including hidden jailbreak attempts, and the Risk block identifies relevant policy items and provides a concise violation rationale.

Summary cache generation. A separate offline step uses the same Qwen3-32B teacher to compress each explicit reasoning stage into a compact summary target. The teacher is prompted to produce two tagged blocks: `<IntentSummary>` (user intent and any hidden malicious goal) and `<RiskSummary>` (minimum policy-relevant evidence and concise rationale). Summaries are capped at 192 tokens and cached in augmented JSONL files to avoid repeated teacher inference during training.

Data preprocessing. For each training example, three parallel token sequences are constructed:

1. **Encoder path** (for latent rollout): the question (`annotation_input`) followed by `<bot>`.
2. **Decoder path** (for explicit verdict output): `<eot>` followed by the compact verdict extracted from the `<Output>` block. Standard next-token cross-entropy labels are applied.
3. **Reference path** (for teacher supervision): the full sequence of question + explicit reasoning + output. Labels mask the question prefix (set to -100) so that only the reasoning and verdict tokens contribute to $\mathcal{L}_{\text{ref}}$.

Additionally, *stage boundary positions* (the token indices of `</Intent>` and `</Risk>` in the reference sequence) are precomputed for the stage-boundary distillation sub-loss. Examples exceeding 800 tokens (question + reasoning combined) are filtered out.

B.4 Training Hyperparameters

Training uses DeepSpeed ZeRO-2 across 4 NVIDIA A100-80G GPUs. Table 8 summarizes the full configuration.

Table 8: LPG training hyperparameters.

Hyperparameter	**Value**
Base model	Qwen3-4B
LoRA rank / alpha	128 / 32
LoRA target modules	All attention + FFN projections
Latent tokens per stage (m_1, m_2)	4 (intent), 6 (risk)
Projection hidden dim d_p	2560
Explain projector layers	3
Learning rate	1×10^{-6}
LR scheduler	Linear
Warmup ratio	0.10
Weight decay	0.1
Max gradient norm	2.0
Precision	BF16
Optimizer	AdamW
Epochs	3
Per-device batch size	2
Gradient accumulation steps	8
Effective batch size	64 (4 devices $\times$ 2 $\times$ 8)
λ_{out} (verdict CE)	2.0
λ_{distill} (teacher hidden-state distillation)	10.0
β (stage-boundary sub-weight inside $\mathcal{L}_{\text{distill}}$)	0.1
λ_{ref} (reference reasoning CE)	1.0
λ_{explain} (summary reconstruction)	0.5
Distillation loss function	Smooth L1
Max token length (filter threshold)	800
Summary max target length	192

B.5 Training Algorithm

Algorithm 1 provides the full pseudocode for the multi-objective LPG training procedure summarized in Section 4.6. The algorithm interleaves the student's latent reasoning rollout with an explicit-reasoning reference pass and computes the four loss terms (output, teacher hidden-state distillation, reference, and explain) in a single phase, where the distillation term itself combines the answer-position and stage-boundary sub-losses.

B.6 Inference Procedure

At inference time, the model performs the following steps:

Algorithm 1 Training LPG latent reasoning with multi-objective teacher supervision

Require: Dataset $\mathcal{D}$ of contexts $\mathbf{c} = (\mathcal{P}, x)$; explicit traces and summaries; student f_θ with LoRA adapters; explain projectors $\{\text{Proj}_k\}_{k\in\{1,2\}}$; projection module Proj; latent lengths (m_1, m_2); loss weights $(\lambda_{\text{out}}, \lambda_{\text{distill}}, \lambda_{\text{ref}}, \lambda_{\text{explain}})$ and stage-boundary sub-weight β.

Initialize f_θ from a base LLM with LoRA adapters; initialize projector parameters.
for each minibatch $\mathbf{c} \sim \mathcal{D}$ **do**
  // Latent reasoning path (student)
  Encode question $\mathbf{c}$ with f_θ; extract final hidden state $\mathbf{h}_0 = \mathbf{h}^{(L)}_{|\mathbf{c}|}$.
  $\mathbf{e} \leftarrow \text{Proj}(\mathbf{h}_0)$ {Project to embedding space}
  for each stage $k \in \{1, 2\}$ **do**
    for $j = 1, \ldots, m_k$ **do**
      $\mathbf{h}^{(k)}_j \leftarrow f_\theta(\mathbf{e}; \text{KV cache}); \quad \mathbf{e} \leftarrow \text{Proj}(\mathbf{h}^{(k)}_j)$
    end for
    Store $\mathbf{z}^{(k)}_{m_k} \leftarrow \mathbf{h}^{(k)}_{m_k}$ {Final hidden for stage-boundary distillation}
    Compute $\mathcal{L}^{(k)}_{\text{explain}}$: project $\mathbf{Z}^{(k)}$ via Proj_k, reconstruct summary $\mathbf{s}^{(k)}$ through f_θ
  end for
  Append <eot> and teacher-force verdict tokens from the KV cache; compute $\mathcal{L}_{\text{out}}$ (Eq. 1).
  // Teacher reasoning path
  Run f_θ on full explicit reasoning sequence (no grad); extract boundary states $\mathbf{h}^{(L)}_{\text{teacher}}[b_k]$.
  Run f_θ on full explicit reasoning sequence (with grad); compute $\mathcal{L}_{\text{ref}}$ (Eq. 4).
  // Teacher hidden-state distillation
  Compute $\mathcal{L}^{\text{ans}}_{\text{distill}}$ at answer positions across all layers and $\mathcal{L}^{\text{stage}}_{\text{distill}}$ at stage boundaries (Eq. 2).
  $\mathcal{L}_{\text{distill}} \leftarrow \mathcal{L}^{\text{ans}}_{\text{distill}} + \beta\, \mathcal{L}^{\text{stage}}_{\text{distill}}$.
  Update θ using gradients of $\mathcal{L}$ (Eq. 5).
end for

1. **Prompt encoding**: The question is tokenized and appended with <bot>. A forward pass through the base LM produces the final hidden state at the prompt boundary.
2. **Latent rollout**: The hidden state is projected via the MLP projector and fed back as input for m_1=4 latent steps (intent stage), followed by m_2=6 latent steps (risk stage). Each step uses KV-cache for efficient incremental computation. No discrete tokens are generated during this phase.
3. **Verdict generation**: The <eot> token embedding is appended, and the model autoregressively emits the compact verdict string, one of "safe", "unsafe, policy n", or "unsafe, policy $n_1, n_2, \ldots$".
4. **Verdict extraction**: A deterministic regex parser maps the compact string to (y, P^*). The student emits the compact verdict extracted from the teacher's <Output> block.

Special tokens ([PAD], <bot>, <eot>) are suppressed during verdict generation by setting their logits to $-\infty$. The explain projectors and reference reasoning path are not used during inference.

C Decoding-Variance Sweep

To characterize sensitivity to stochastic decoding, we run a small decoding-variance study alongside the deterministic evaluation in Table 2. For each (model, dataset) cell we sweep three temperatures $T \in \{0.3, 0.7, 1.0\}$ and draw five independent decoding seeds at each temperature, for a total of n=15 runs per cell. Sampling is forced on for every model so the stochasticity is observable and comparable, and the random seeds are reset before each rerun. The evaluation set is a subset of 200 examples: 100 uniformly sampled from GuardSet-X and 100 from the DynaBench (Aug) split.

Since each run reduces to 100 correct/incorrect outcomes per dataset, we keep the report simple and use Safety Accuracy as the single metric. We report the across-run standard deviation σ and the half-width of the 95% confidence interval ($1.96 \cdot \sigma/\sqrt{n}$ with n=15), in percentage points.

The explicit-reasoning baselines show greater decoding variability than the short-verdict models on this subset. LPG's latent rollout contains no sampled text tokens, although its final verdict remains

Table 9: Decoding-variance sweep on a $100{+}100$ subset of GuardSet-X and DynaBench (Aug). Each cell aggregates $n{=}15$ runs (3 temperatures $\times$ 5 reruns). We report the across-run standard deviation σ and the 95% CI half-width for Safety Accuracy (in percentage points).

Model	GuardSet-X (100)		DynaBench Aug (100)	
	Acc σ	Acc CI_{95}	Acc σ	Acc CI_{95}
Qwen3-4B	0.00	0.00	0.05	0.03
Qwen3-4B-Thinking	2.18	1.10	2.65	1.34
DynaGuard-4B	1.02	0.52	0.71	0.36
GuardReasoner-3B	2.27	1.15	1.83	0.93
LPG-4B (Ours)	0.51	0.26	0.27	0.14

subject to decoding variability. The reported intervals summarize the pooled temperature-and-seed sweep; they should be interpreted descriptively, since temperature is deliberately varied across runs. They do not quantify test-set sampling uncertainty, training variability, or the statistical significance of pairwise model differences.

D Limitations and Broader Impacts

Limitations. We assume the governing policy is supplied by a trusted operator; users control the content being moderated, not the policy or system prompt. Malicious policy instructions, contradictions, misleading exceptions, and prompt injection within untrusted policy text are outside the evaluated threat model. Even under a trusted policy, the adaptive attack in Section 6.1 covers only a restricted response-rewriting setting and cannot establish robustness to stronger or different attacks. Long policies and many distractor clauses can also degrade performance, and taxonomy-derived evaluation policies need not reflect realistic policy documents.

The intent and risk computations are hidden in continuous states. A cited clause is a useful decision artifact, but is neither a complete explanation nor proof that the latent reasoning faithfully followed that clause. Operators should inspect disputed inputs, verdicts, and cited clauses, and use independent review and adversarial testing to diagnose errors. The benchmarks and model may underrepresent some populations or harm categories; false positives and false negatives remain possible.

Broader impacts. Policy-conditioned moderation can make customized safeguards more accessible, while clause attribution can help operators review decisions. However, false positives may suppress legitimate speech, false negatives can leave harmful content unfiltered, and the same system could be used to enforce censorship. Deployment requires policy review and monitoring of errors across affected groups.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state concrete claims for our main contribution and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations subsection is included (Section D).

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results; all contributions are empirical or methodological.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Appendix B specifies the LPG architecture (LoRA configuration, projector dimensions, special tokens), the four-term loss with all weights, the full training pseudocode (Algorithm 1), the inference procedure, and a complete hyperparameter table (Table 8). The training corpus composition is documented per source with record counts, the policy-list synthesis procedure is described.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The repository linked on the first page provides training and evaluation code and links to the checkpoint and training data. It includes reproduction instructions for selected evaluations; complete reproduction of every reported experiment is not established. All third-party datasets used (DynaBench, BeaverTails, Aegis, SaladBench, Toxic-Chat, XSTest, HarmBench, WildGuardTest) are publicly available; their citations and access points are listed in Section 4.5 and Appendix A.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix A (datasets, splits, baseline checkpoints, evaluation protocol) and Appendix B (Table 8: LoRA, optimizer, schedule, loss weights, batch sizes, precision, max-token thresholds) document all training and test details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report a dedicated decoding-variance sweep in Appendix C (Table 9): for each of the five models in scope we run $n{=}15$ inferences per (model, dataset) cell on a frozen $100{+}100$ subset of GuardSet-X and DynaBench (Aug), and report the across-run standard deviation σ together with the half-width of the 95% confidence interval. This sweep does not quantify training or test-set sampling uncertainty for the main comparisons.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Training uses 4×NVIDIA A100 80GB GPUs with DeepSpeed ZeRO-2 (Section B); evaluation runs on a single A100 80GB at temperature 0 with batch size 1 (Section A). The full training run takes approximately 18 GPU-hours at the listed configuration; pilot runs and the 32B-teacher annotation pipeline used several additional GPU-days that we do not enumerate exhaustively.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and the work conforms in all respects. The research uses publicly released safety datasets under their existing licenses, does not involve human subjects, and aims to improve guardrail systems for AI safety. Training data was inspected for stability of the teacher annotations; the adaptive evaluation involves generated adversarial response candidates.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: A Broader Impacts paragraph is included (Section D). Positive: faster, more accurate policy-grounded guardrails reduce the latency cost of safety moderation in agentic AI deployments and lower the barrier for application developers to enforce custom rules without retraining. Negative: a guardrail trained on specific taxonomies may underperform on populations or harm types under-represented in the training mix; over-blocking from miscalibrated policy lists can suppress legitimate speech; and stronger guardrails can be repurposed to enforce censorship policies. We note these and recommend pairing deployment with regular policy audits.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a generative model. LPG is a guardrail classifier whose output is a binary safety verdict and a list of violated policy indices; it does not generate harmful content. Training data is sourced from existing safety benchmarks released by their authors with their own usage terms, which we follow.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All third-party datasets (DynaBench, BeaverTails, Aegis-AI Content Safety v2, SaladBench, Toxic-Chat, XSTest, HarmBench, WildGuardTest, PolicyGuardBench, GuardSet-X) and base models (Qwen3-4B, Qwen3-32B, Llama Guard 3, ShieldGemma, GuardReasoner, DynaGuard) are properly cited with the original publications, and we use each under its released license (research-only or permissive, as specified by the original authors).

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new assets introduced (LPG model, curated 40k training mixture, evaluation harness) are documented in Sections 4 and 4.5, Appendix B (architecture, training pseudocode, hyperparameters). The repository linked on the first page provides code and links to the model checkpoint and training corpus.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. All training labels come from publicly released datasets or LLM teacher annotations on top of those labels.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human subjects research and therefore IRB approval is not applicable.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are core to the method. Qwen3-4B is the student backbone (Section 4, Appendix B); Qwen3-32B is the teacher used both to generate the explicit reasoning traces and to produce the per-stage summary targets used by the explain loss (Section 4.5). Baseline LLMs (Qwen3-4B-Thinking, GPT-4o, GuardReasoner-3B/8B, DynaGuard-4B/8B, ShieldGemma, LlamaGuard3-8B) are used for evaluation comparisons (Sections 5–5.2).

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.