Lattice: Generative Guardrails for Conversational Agents

Emily Broadhurst, Tawab Safi, Joseph Edell, Vashisht Ganesh, Karime Maamari

Distyl AI
{emily, tawab, joseph, vashisht, karime}@distyl.ai

Abstract

Conversational AI systems require guardrails to prevent harmful outputs, yet existing approaches use static rules that cannot adapt to new threats or deployment contexts. We introduce LATTICE, a framework for self-constructing and continuously improving guardrails. LATTICE operates in two stages: construction builds initial guardrails from labeled examples through iterative simulation and optimization; continuous improvement autonomously adapts deployed guardrails through risk assessment, adversarial testing, and consolidation. Evaluated on the ProsocialDialog dataset, LATTICE achieves 91% F1 on held-out data, outperforming keyword baselines by 43pp, LlamaGuard by 25pp, and NeMo by 4pp. The continuous improvement stage achieves 7pp F1 improvement on cross-domain data through closed-loop optimization. Our framework shows that effective guardrails can be self-constructed through iterative optimization.

Introduction

The deployment of large language models in conversational AI systems presents a fundamental tension between capability and safety. Although these systems must engage naturally in diverse contexts, they also require mechanisms to prevent harmful outputs in real-world deployments (Ayyamperumal and Ge 2024; Hakim et al. 2024; Abdelkader et al. 2024).

Current approaches to conversational safety are predominantly based on static guardrail mechanisms (Dong et al. 2024; Rebedea et al. 2023). These systems employ fixed rule sets designed to filter input queries and LLM outputs based on predetermined patterns. However, static guardrails face two critical limitations that compromise their effectiveness in deployed systems. First, they cannot adapt to attack vectors or conversational contexts that emerge post-deployment (Yang et al. 2024). Second, they exhibit brittleness when faced with adversarial inputs specifically crafted to circumvent existing protections (Goyal et al. 2024).

These limitations point to a deeper algorithmic challenge: *how can guardrail systems self-evolve to address emergent threats while maintaining safety coverage?* This requires moving beyond static guardrail application toward systems capable of learning, identifying coverage gaps, and refining protection mechanisms through experiences.

We introduce LATTICE, a framework that treats guardrail construction and adaptation as continuous optimization problems. Rather than relying on fixed rules defined at design time, LATTICE constructs initial guardrail sets through iterative conversation simulation, evaluating candidate policies against labeled training data and refining them based on observed failure modes. After deployment, LATTICE continuously improves these guardrails by monitoring conversations for coverage gaps, expanding edge cases through adversarial beam search, and updating policies when performance degrades. Evaluated on ProsocialDialog (Kim et al. 2022), this approach outperforms static baseline systems and demonstrates cross-domain adaptation.

Our primary contributions are: (1) A simulation-based construction method that learns compact guardrail sets from minimal labeled data through iterative optimization, outperforming manually designed static systems. (2) A closed-loop continuous improvement system combining dual-check risk assessment, adversarial case expansion, and policy optimization that enhances deployed guardrails, achieving measurable performance gains in cross-domain settings.

Related Work

Static safety mechanisms at training and inference time

Safety mechanisms for language models are typically introduced either during training or at inference time. Training time methods include safety fine tuning and guardrail aware adaptation (Kumar et al. 2024), as well as parameter efficient approaches such as LoRA based safety modules (Fomenko et al. 2024; Hsu et al. 2025). These methods embed constraints directly into model parameters, which can yield strong performance within the training distribution. However, they require new optimization cycles whenever threats change, limiting rapid adaptation to novel jailbreak patterns (Huang et al. 2025).

Inference time guardrail systems, including NeMo Guardrails (Rebedea et al. 2023) and LlamaGuard (Fedorov et al. 2024), represent the prevailing runtime strategy. These systems achieve broad coverage of predefined safety categories but their rules remain fixed at design time, which constrains their ability to handle multi-turn jailbreaks or emerging violation types (Yang et al. 2024; Yu et al. 2024).

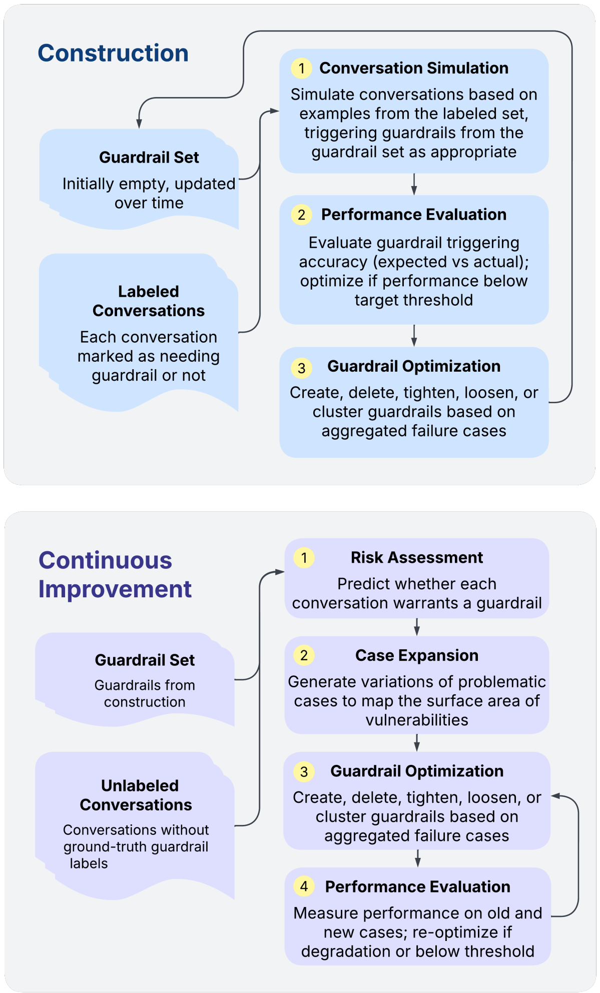
Construction
Guardrail Set
Initially empty, updated over time
Labeled Conversations
Each conversation marked as needing guardrail or not
1 Conversation Simulation
Simulate conversations based on examples from the labeled set, triggering guardrails from the guardrail set as appropriate
2 Performance Evaluation
Evaluate guardrail triggering accuracy (expected vs actual); optimize if performance below target threshold
3 Guardrail Optimization
Create, delete, tighten, loosen, or cluster guardrails based on aggregated failure cases
Continuous Improvement
Guardrail Set
Guardrails from construction
Unlabeled Conversations
Conversations without ground-truth guardrail labels
1 Risk Assessment
Predict whether each conversation warrants a guardrail
2 Case Expansion
Generate variations of problematic cases to map the surface area of vulnerabilities
3 Guardrail Optimization
Create, delete, tighten, loosen, or cluster guardrails based on aggregated failure cases
4 Performance Evaluation
Measure performance on old and new cases; re-optimize if degradation or below threshold

Figure 1: **Two-stage framework architecture.** *Construction stage* (top) generates initial guardrails from labeled conversations through three iterative steps: (1) Conversation Simulation tests current guardrails on synthetic dialogues; (2) Performance Evaluation computes precision, recall, and F1; (3) Guardrail Optimization creates, deletes, tightens, loosens, or clusters guardrails based on false positives and false negatives. *Continuous improvement stage* (bottom) adapts deployed guardrails to unlabeled conversations through four steps: (1) Risk Assessment identifies coverage gaps via dual-check evaluation; (2) Case Expansion generates adversarial variations; (3) Guardrail Optimization updates policies; (4) Performance Evaluation validates changes and reverts if performance degrades.

LATTICE builds on this line of work by introducing an inference time mechanism that not only applies rules but also generates, edits, and consolidates them through a continuous optimization process.

Programmed guardrails and moderation models

Rule based toolkits such as NeMo Guardrails provide programmable rails, dialogue flows, and domain specific policy enforcement (Rebedea et al. 2023). Moderation models such as LlamaGuard (Fedorov et al. 2024) and ShieldGemma (Zeng et al. 2024) classify content categories for safety filtering. These systems are effective for predefined risks, yet studies highlight gaps in coverage for paraphrased intent, cross category drift, and extended dialogue context (Yang et al. 2024; Yu et al. 2024). Our approach uses such systems as components inside a broader feedback loop: general purpose classifiers serve as baseline evaluators, while LATTICE induces new guardrails to address their blind spots.

Adversarial robustness, red teaming, and multi turn jailbreaks

A substantial literature explores adversarial prompting and jailbreak attacks on safety systems. Red teaming frameworks demonstrate that carefully staged inputs can bypass filters and that multi turn dialogues can erode safety boundaries by gradually shifting context (Goyal et al. 2024). Multi step evaluations reveal that contextual drift, latent goal pursuit, and role play exploitation often lead to safety failures (Yang et al. 2024). HAICOSYSTEM (Zhou et al. 2024) introduces sandboxed evaluation across diverse domains, emphasizing large scale testing of system vulnerabilities. These works focus primarily on identifying weaknesses. LATTICE complements them by integrating adversarial testing with automatic policy improvement, converting discovered jailbreaks into new guardrail rules without manual patching.

Learning based refinement and critique driven methods

Iterative learning approaches use feedback loops to refine model behavior. Self Refine (Madaan et al. 2023) and critique driven refinement (Ke et al. 2024; Ye et al. 2024) demonstrate that self feedback can improve textual quality and consistency, though their focus lies on generation fidelity rather than safety. Reward model scaling studies (Rafailov et al. 2024) show that feedback driven alignment improves safety and helpfulness, but these methods often rely on thousands of human annotated feedback samples per iteration and may encounter reward hacking effects (Yan et al. 2024). LATTICE differs by employing synthetic feedback derived from observed guardrail failures and adversarial expansions, enabling unsupervised improvement.

Methodology

Construction Stage

The construction stage learns an initial guardrail set from labeled conversations via iterative refinement (Figure 1, Algorithm 1). Each iteration comprises three steps that (i) expand behavioral coverage, (ii) evaluate detection performance, and (iii) apply targeted edits to the guardrail set while enforcing an acceptance criterion that prevents regressions.

Conversation Simulation *(Algorithm 1, lines 7–11)* For each labeled conversation (c_i, y_i), the system samples a user persona and simulates a multi-turn dialogue conditioned on the current guardrails R. We then evaluate whether any $r \in R$ triggers on the simulated transcript s, appending $(c_i, y_i, \texttt{triggered})$ to a validation set V. This procedure exposes R to diverse conversational dynamics and yields observed trigger patterns beyond the original labels, increasing the effective variation encountered during optimization.

Performance Evaluation *(Algorithm 1, lines 12–20)* From V, we compute TP, FP, FN, TN by comparing y_i with the observed trigger outcome and report precision $P = \frac{\text{TP}}{\text{TP+FP}}$, recall $R = \frac{\text{TP}}{\text{TP+FN}}$, and $F_1 = \frac{2PR}{P+R}$ as the primary selection metric. This diagnostic decomposition identifies whether error is dominated by over-sensitivity (FP) or under-coverage (FN), providing sufficient statistics to guide the subsequent optimization step. If $F < F^\star$ and a prior best configuration $R^\star$ exists, we revert to $R^\star$; if $F \geq F^\star$, we promote the current configuration to $R^\star$ and update $F^\star$. Early stopping is triggered when $F \geq \tau$, where τ is a user-specified threshold.

Guardrail Optimization *(Algorithm 1, lines 21–40)* Based on the failure modes observed in V, we apply targeted edits to R as follows:

- **False negatives related to existing guardrails:** broaden the guardrail involved to increase sensitivity.
- **False negatives unrelated to existing guardrails:** synthesize a new specialized guardrail to cover the uncovered pattern.
- **False positives:** refine the triggered guardrail to reduce over-flagging while preserving coverage.
- **Unused guardrails:** remove rules that never trigger to prevent bloat and improve interpretability.
- **Redundant guardrails:** optionally cluster similar rules and consolidate them into a single policy.

Each proposed edit is *accepted* only if it maintains or improves the best observed score, i.e., $F_{\text{new}} \geq F^\star$; otherwise the system *reverts* to $R^\star$. This acceptance rule enforces monotonic, non-decreasing F_1 over accepted configurations, ensuring stable convergence of the construction process.

Continuous Improvement Stage

The continuous improvement stage adapts the deployed guardrails to the unlabeled deployment data through structured testing and optimization (Algorithm 2). Each iteration executes four steps –risk assessment, case expansion, guardrail optimization, and performance evaluation – to maintain or improve safety coverage in response to novel conversational patterns.

Risk Assessment *(Algorithm 2, lines 9–16)* Each new conversation u undergoes a dual-evaluation procedure: the system tests both (i) current specialized guardrails R and (ii) a general-purpose safety guardrail r_{general}. The dual-check mechanism identifies coverage gaps by comparing activation states; cases where $r_{\text{general}}(u) = 1$ and $\bigvee_{r \in R} r(u) = 0$ indicate *missed threats-instances*—instances where the general model detects risk that the specialized set fails to capture. The resulting set of discrepancies, denoted G, serves as the seed for a subsequent adversarial exploration.

Case Expansion *(Algorithm 2, lines 17–26)* For each $u \in G$, the system performs multi-turn adversarial search to generate conversation variants that investigate known and potential weaknesses. An *attacker model* M_a produces candidate user turns designed to bypass existing guardrails while preserving the inferred harmful intent; a *target model* M_t responds under the current guardrail policy. The beam search with configurable width k and depth d expands the conversation tree, producing a set of labeled leaf nodes $\mathcal{E}$ categorized as:

- *successful attacks* (guardrail bypasses)
- *blocked attacks* (guardrail triggered correctly)
- *false alarms* (benign content flagged in error)

This step operationalizes adversarial testing, generating diverse challenge cases.

Guardrail Optimization *(Algorithm 2, lines 27–42)* The system applies targeted updates to R conditioned on the analysis of $\mathcal{E}$:

- **Successful attacks:** broaden or synthesize new guardrails to cover novel attack strategies.
- **False alarms:** refine corresponding guardrails to reduce over-sensitivity.
- **Redundant policies:** cluster semantically similar guardrails and consolidate them to prevent drift and complexity growth.

Each proposed update produces a candidate set R' that is validated against a held-out evaluation split. Updates are accepted only if $F_1(R') \geq F_1(R)$, enforcing non-decreasing performance across improvement cycles.

Performance Evaluation *(Algorithm 2, lines 43–47)* Updated guardrails R' are re-evaluated on both the original risk-assessment cases G and the adversarially expanded cases $\mathcal{E}$. If $F_1(R') < F_1(R)$ or performance falls below a specified degradation threshold, the system reverts to the prior configuration R. Otherwise, R' is promoted to the deployed set, thereby closing the continuous-improvement loop and enabling adaptive response to emergent threats.

Implementation via Prompted Language Models

All algorithmic operations in the construction and continuous improvement stages are implemented as structured prompt calls to large language models (LLMs). To ensure fair comparison with baselines and computational efficiency, all operations use `gpt-4o-mini` uniformly across the pipeline. This includes persona sampling, conversation simulation, guardrail generation and refinement, adversarial attack generation, and all testing procedures.

Each LLM call follows a fixed prompt schema to ensure consistency and facilitate automatic parsing of results:

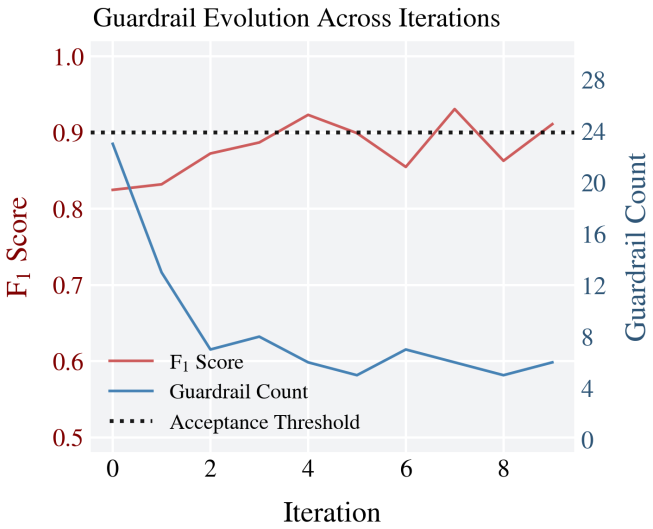

Figure 2: **Iterative construction from 100 labeled examples.** F1 score (left axis, red) and guardrail count (right axis, blue) across 10 construction iterations. F1 improves from 82% (iteration 0, 23 guardrails) to 93% (iteration 7, 6 guardrails), exceeding the early stopping threshold ($F_1 \geq 0.90$, black dotted line) at iteration 4. Guardrail consolidation reduces the set from 23 to 6 policies through iterative clustering while maintaining performance. The system exhibits convergence with non-monotonic but improving F1 trajectory.

```
### TASK
<Defines the operation's purpose and
establishes the model's role>

### INSTRUCTIONS
<Provides step-by-step procedures,
specific requirements, and
constraints>

### OUTPUT FORMAT
<Specifies the expected response
schema,
typically JSON with fields for
generated content and reasoning>
```

This standardized structure constrains model behavior, enforces reproducible output formats, and allows the system to programmatically integrate LLM outputs within the guardrail construction and improvement loops.

Experimental setup

Dataset

We evaluate on PROSOCIALDIALOG (Kim et al. 2022), a corpus of 58,000 multi-turn conversations between users and AI assistants annotated for conversational safety. Each dialogue is labeled as either `harmful` (requiring guardrail intervention) or `benign` (safe). To ensure consistency, we filter out ambiguous or multi-intent examples and retain only those with unambiguous binary annotations. From this filtered subset, we construct three evaluation splits: $\mathcal{D}_{\text{train}}$ (100 conversations from social domain), $\mathcal{D}_{\text{test}}$ (652 conversations from social domain, disjoint from train, representing 10% of the social domain data), and $\mathcal{D}_{\text{improve}}$ (100 conversations from ethics domain for cross-domain testing). We treat the social and ethics domains as distinct subspaces within the corpus to better capture differences between interpersonal sensitivity and moral reasoning. Training and test sets are balanced between harmful and benign labels.

Parameter	**Value**
Max construction iterations	10
Early stop F1 threshold	0.90
Simulated conversation turns	3
Iteration selection metric	F1 score
Beam width (case expansion)	3
Tree depth (case expansion)	10
Holdout evaluation runs	5

Table 1: **Key hyperparameters for reproducibility.** Construction iterates up to 10 times with early stopping when $F_1 \geq 0.90$. Each iteration simulates 3-turn conversations and selects updates based on F1 score comparison. Continuous improvement uses beam search with width 3 and depth 10 for adversarial case expansion.

Evaluation Metrics

We report standard binary classification metrics. Precision (P) quantifies the fraction of flagged conversations that are truly harmful. Recall (R) measures the fraction of harmful conversations correctly identified. The F1-score provides the harmonic mean of precision and recall. All holdout and baseline results are averaged over 5 independent runs on the same test set to estimate measurement variance.

Baseline Systems

We compare LATTICE against three representative static guardrail baselines:

1. *Keyword* — deterministic keyword matching over predefined lexical patterns (e.g., *suicide*, *kill*, *violence*, *abuse*).
2. *LlamaGuard* (Fedorov et al. 2024) — Meta's LlamaGuard-3-8B content moderation model accessed via the Together AI API.
3. *NeMo* (Rebedea et al. 2023) — NVIDIA's NeMo Guardrails framework configured with a `gpt-4o-mini` backend.

All baselines are evaluated on the same 652-conversation test set using the same preprocessing pipeline to ensure comparability.

Hyperparameters

Table 1 summarizes key experimental settings. The construction stage iterates up to $T = 10$ cycles with early stopping when $F_1 \geq 0.90$. Each iteration simulates three-turn dialogues, computes F_1, and applies targeted guardrail optimization based on precision–recall analysis. Model selection across iterations follows an F_1-based comparison criterion. The continuous improvement stage employs beam search for adversarial case expansion with width $k = 3$ and depth $d = 10$. All reported experiments are conducted under these fixed hyperparameters.

Research Questions

We evaluate LATTICE across three core research objectives:

1. **RQ1 — Self-Construction:** Can LATTICE self-construct an effective guardrail set from a limited sample of 100 labeled conversations such that the resulting model achieves $F_1 \geq 0.90$ on unseen data?
2. **RQ2 — Baseline Comparison:** Does the guardrail set produced by LATTICE outperform representative static baselines—*Keyword*, *LlamaGuard*, and *NeMo*—in detecting harmful conversations?
3. **RQ3 — Continuous Improvement:** Can the continuous improvement stage, when deployed on unlabeled data, improve F_1 without human supervision or manual intervention?

These questions collectively test the extent to which LATTICE can (i) construct guardrails with minimal supervision, (ii) achieve or exceed state-of-the-art performance relative to existing static systems, and (iii) continuously adapt to emerging conversational risks in a fully automated manner.

Results

Self-Construction (RQ1)

Figure 2 shows construction performance across 10 iterations on 100 labeled examples. The system starts with 23 initial guardrails at iteration 0 achieving F_1=82%. Through iterative refinement, F_1 improves to 93% by iteration 7, surpassing the early stopping threshold ($F_1 \geq 0.90$) at iteration 4. Guardrail consolidation reduces the set from 23 to 6 policies while maintaining performance. Sample constructed guardrails demonstrating the specificity and structure of generated policies are shown in Figures 3 and 4.

The construction stage achieves $F_1 = 91\% \pm 1\%$ on held-out data from 100 labeled training examples. While final construction performance (93%) exceeds holdout performance (91%), indicating some overfitting, the absolute holdout F1 remains substantially higher than initial construction (82%), demonstrating effective learning. The system prioritizes recall (93% ± 1%), ensuring comprehensive coverage of harmful content, while maintaining strong precision (90% ± 1%).

Metric	Initial	Final	Holdout
Precision	73%	91%	90% ± 1%
Recall	94%	96%	93% ± 1%
F1 Score	82%	93%	91% ± 1%
Guardrails generated	23 (initial) → 6 (final)		
Iterations to convergence	4 (of max 10)		

Table 2: **Self-construction performance on training and test sets.** Initial construction (iteration 0) starts with 23 guardrails achieving 82% F1. Final construction (best iteration) improves to 93% F1 with 6 guardrails after consolidation. Holdout evaluation over 5 independent runs yields 91% ± 1% F1 (mean ± 95% CI), with the 2pp gap analyzed in Section . The system reaches the early stopping threshold ($F_1 \geq 0.90$) at iteration 4. High recall (93%) ensures comprehensive coverage while maintaining strong precision (90%).

Baseline Comparison (RQ2)

Table 3 compares LATTICE against static guardrail systems. LATTICE achieves the highest F1 (91%), outperforming keyword matching (48%), LlamaGuard (66%), and NeMo (87%). The 43pp improvement over keyword matching demonstrates the value of learned guardrails. Compared to NeMo (4pp improvement), LATTICE achieves comparable F1 with higher precision but slightly lower recall (90%/93% vs 82%/93%).

System	Precision	Recall	F1
Keyword	95% ± 0%	32% ± 0%	48% ± 0%
LlamaGuard-8B	98% ± 0%	50% ± 0%	66% ± 0%
NeMo	82% ± 1%	93% ± 0%	87% ± 0%
Lattice	90% ± 1%	93% ± 1%	91% ± 1%

Table 3: **Comparison against static guardrail baselines.** All systems evaluated on the same 652-conversation test set over 5 independent runs (balanced 50% harmful). LATTICE achieves 91% F1, outperforming keyword matching (48%), LlamaGuard-8B (66%), and NeMo Guardrails (87%). Improvements of 43pp, 25pp, and 4pp respectively. LATTICE achieves high precision and recall (90%/93%) compared to LlamaGuard's high precision but low recall (98%/50%) and NeMo's recall-focused approach (82%/93%).

Compared to LlamaGuard (25pp improvement), LATTICE achieves substantially higher recall (93% vs 50%) with lower but still strong precision (90% vs 98%).

Continuous Improvement (RQ3)

Table 4 demonstrates automated guardrail adaptation through the continuous improvement stage. Starting from baseline performance of 86% F1 on the improvement dataset (cross-domain ethics data), the system executes the full feedback loop. The dual-check risk assessment identifies 8 conversations where general safety classifiers trigger but specific guardrails do not, indicating potential coverage gaps. For each gap, beam search with width 3 and depth 10 generates adversarial conversation variants, producing 24 test cases that probe guardrail boundaries. The optimization component performs 5 targeted guardrail updates: broadening policies that miss related cases and creating new policies for novel patterns. The adapted guardrails achieve 93% F1, representing a 7pp improvement over baseline, validating that the continuous improvement loop enhances safety coverage even in cross-domain settings.

Metric	Value
Coverage gaps identified	8
Adversarial tests generated	24
Guardrail updates performed	5
Initial F1 (before)	86%
Final F1 (after)	93%
Change	+7pp

Table 4: **Continuous improvement operational statistics.** Starting from constructed guardrails with 86% F1 on cross-domain data (ethics), the system identifies 8 coverage gaps via dual-check risk assessment, generates 24 adversarial test cases through beam search, and performs 5 guardrail updates. Final performance achieves 93% F1, representing a 7pp improvement through the risk assessment, case expansion, and optimization loop.

Discussion

Our results demonstrate that LATTICE constructs effective guardrails through iterative optimization, achieving strong holdout performance while maintaining compact policy sets. The system converges efficiently and substantially outperforms static baselines across all metrics.

Precision-recall trade-offs and configurability. Table 3 reveals distinct precision-recall profiles across systems: LlamaGuard achieves high precision (98%) but low recall (50%), capturing only the most obvious violations while avoiding false positives; NeMo prioritizes recall (93%) with lower precision (82%), flagging more broadly at the cost of over-triggering; LATTICE achieves strong precision (90%) and high recall (93%). The 10% false positive rate in LATTICE's current configuration reflects an explicit optimization toward comprehensive harmful content detection. This trade-off is appropriate for safety-critical deployments where missing harmful content (false negatives) carries greater risk than occasional over-flagging of benign conversations (false positives).

Crucially, LATTICE's precision-recall balance is *configurable* rather than fixed. The framework supports two optimization modes: (1) F1-based selection (harmonic mean of precision and recall), or (2) weighted scoring with user-defined coefficients ($\alpha P + \beta R$). F1 optimization penalizes precision-recall imbalance, ensuring both metrics remain high. Weighted scoring allows asymmetric optimization: applications prioritizing user experience can set $\alpha = 2.0, \beta = 1.0$ to favor precision (fewer false positives), while safety-critical deployments can set $\alpha = 1.0, \beta = 2.0$ to favor recall (fewer false negatives). This configurability enables deployment-specific optimization without code changes, allowing organizations to align guardrail behavior with their risk tolerance and operational constraints. Our evaluation uses F1-based selection, representing a balanced default that equally penalizes precision and recall deficiencies.

Generalization and overfitting analysis. Table 2 shows a 2pp drop in F1 from final construction (93%) to holdout evaluation (91%). This gap warrants careful interpretation. On one hand, the performance decrease indicates some degree of overfitting to the 100-example training set. On the other hand, the absolute holdout performance (91%) remains substantially higher than initial construction (82%), representing a 9pp net improvement. Additionally, the 91% holdout F1 significantly exceeds all static baselines, demonstrating that despite the generalization gap, the learned guardrails capture meaningful safety patterns.

The 2pp gap is within expected bounds for classification tasks trained on 100 examples. Comparing precision and recall components reveals the source: final construction achieves 91% precision on training data, maintaining similar holdout precision (90%), while recall decreases from 96% to 93%. This suggests the guardrails generalize well but are slightly more conservative on new data. The modest generalization gap, combined with strong absolute performance, indicates the approach is viable for deployment.

The continuous improvement stage demonstrates effective unsupervised adaptation. Starting from constructed guardrails with 86% F1 on cross-domain improvement data (ethics), the system identifies 8 coverage gaps through dual-check risk assessment, generates 24 adversarial test cases via beam search exploration, and performs 5 targeted guardrail updates. The final adapted guardrails achieve 93% F1, representing a 7pp improvement. This validates that the framework can enhance deployed guardrails through structured feedback loops, even when adapting to new domains.

Computational cost analysis. The construction stage consumes approximately 53.6 million tokens (~$20 at `gpt-4o-mini` pricing) and requires 46 minutes of runtime to train on 100 labeled examples. This cost contrasts with static baseline systems—keyword matching, LlamaGuard, and NeMo Guardrails—which incur no construction overhead but require manual rule specification and cannot self-adapt post-deployment. While the upfront computational investment is substantial, it must be contextualized within deployment scale. For production conversational AI systems serving millions of users, construction costs are amortized across all subsequent interactions. Consider a customer service chatbot handling 100,000 daily conversations: a one-time cost of $20 and 46 minutes yields guardrails protecting 36.5 million annual conversations, reducing to $0.0000005 per protected conversation. For safety-critical domains where a single harmful output could have severe consequences, this investment is justified by superior performance over static systems (4–43pp improvement), automated construction from minimal supervision, and continuous adaptation capabilities that static systems lack. The framework's configurability further enables cost-performance trade-offs: organizations can adjust iteration counts, clustering aggressiveness, and precision-recall weights based on their risk tolerance and budget constraints.

Limitations and Future Directions While LATTICE demonstrates effective guardrail construction and adaptation, several limitations warrant consideration. First, evaluation focuses exclusively on English-language conversations within the ProsocialDialog domain; generalization to multilingual settings, domain-specific contexts, and cross-cultural safety norms remains untested. Second, all pipeline operations use `gpt-4o-mini` uniformly; exploring additional models, particularly reasoning models, could identify optimal model-operation pairings to improve quality or reduce costs. Third, we evaluate with a fixed 100-example training set; investigating how training set size affects holdout and continuous improvement performance would inform data collection requirements. Fourth, automated refinement raises interpretability questions for regulated domains requiring audit trails.

Future work should investigate: (1) multilingual and cross-domain robustness; (2) training set size ablations to determine data efficiency bounds; (3) model selection studies comparing reasoning models against fast inference models for different pipeline operations; (4) adversarial testing against sophisticated jailbreak techniques; (5) human-in-the-loop variants for high-stakes decisions; and (6) long-term deployment studies measuring adaptation and drift.

Conclusion

We presented LATTICE, a framework enabling conversational AI systems to self-construct and continuously improve guardrails.. The framework operates through two stages: construction generates initial guardrails from labeled training examples via iterative simulation and optimization; continuous improvement adapts deployed guardrails through risk assessment, adversarial testing, and consolidation.

Beyond accuracy, the system is configurable—trading precision vs. recall to match deployment risk—and practical, with a one-time construction cost on the order of minutes and dollars that amortizes at scale. Framing safety this way reconciles ethics-driven desiderata with engineering constraints: the guardrails self-construct, self-audit, and self-improve, enabling sustained coverage as threats and contexts shift. This is a step toward resilient, auditable safety layers that keep pace with real-world dialogue systems without proportional increases in human oversight.

References

Abdelkader, H.; Abdelrazek, M.; Barnett, S.; Schneider, J.-G.; Rani, P.; and Vasa, R. 2024. ML-On-Rails: Safeguarding Machine Learning Models in Software Systems — A Case Study. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, CAIN '24, 178–183. New York, NY, USA: Association for Computing Machinery. ISBN 9798400705915.

Ayyamperumal, S. G.; and Ge, L. 2024. Current state of LLM Risks and AI Guardrails. arXiv:2406.12934.

Dong, Y.; Mu, R.; Jin, G.; Qi, Y.; Hu, J.; Zhao, X.; Meng, J.; Ruan, W.; and Huang, X. 2024. Building Guardrails for Large Language Models. arXiv:2402.01822.

Fedorov, I.; Plawiak, K.; Wu, L.; Elgamal, T.; Suda, N.; Smith, E.; Zhan, H.; Chi, J.; Hulovatyy, Y.; Patel, K.; Liu, Z.; Zhao, C.; Shi, Y.; Blankevoort, T.; Pasupuleti, M.; Soran, B.; Coudert, Z. D.; Alao, R.; Krishnamoorthi, R.; and Chandra, V. 2024. Llama Guard 3-1B-INT4: Compact and Efficient Safeguard for Human-AI Conversations. arXiv:2411.17713.

Fomenko, V.; Yu, H.; Lee, J.; Hsieh, S.; and Chen, W. 2024. A Note on LoRA. arXiv:2404.05086.

Goyal, S.; Hira, M.; Mishra, S.; Goyal, S.; Goel, A.; Dadu, N.; DB, K.; Mehta, S.; and Madaan, N. 2024. LLMGuard: Guarding Against Unsafe LLM Behavior. arXiv:2403.00826.

Hakim, J. B.; Painter, J. L.; Ramcharran, D.; Kara, V.; Powell, G.; Sobczak, P.; Sato, C.; Bate, A.; and Beam, A. 2024. The Need for Guardrails with Large Language Models in Medical Safety-Critical Settings: An Artificial Intelligence Application in the Pharmacovigilance Ecosystem. arXiv:2407.18322.

Hsu, C.-Y.; Tsai, Y.-L.; Lin, C.-H.; Chen, P.-Y.; Yu, C.-M.; and Huang, C.-Y. 2025. Safe LoRA: the Silver Lining of Reducing Safety Risks when Fine-tuning Large Language Models. arXiv:2405.16833.

Huang, T.; Hu, S.; Ilhan, F.; Tekin, S. F.; and Liu, L. 2025. Virus: Harmful Fine-tuning Attack for Large Language Models Bypassing Guardrail Moderation. arXiv:2501.17433.

Ke, P.; Wen, B.; Feng, A.; Liu, X.; Lei, X.; Cheng, J.; Wang, S.; Zeng, A.; Dong, Y.; Wang, H.; Tang, J.; and Huang, M. 2024. CritiqueLLM: Towards an Informative Critique Generation Model for Evaluation of Large Language Model Generation. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 13034–13054. Bangkok, Thailand: Association for Computational Linguistics.

Kim, H.; Yu, Y.; Jiang, L.; Lu, X.; Khashabi, D.; Kim, G.; Choi, Y.; and Sap, M. 2022. ProsocialDialog: A Prosocial Backbone for Conversational Agents. In *EMNLP*.

Kumar, D.; Kumar, A.; Agarwal, S.; and Harshangi, P. 2024. Fine-Tuning, Quantization, and LLMs: Navigating Unintended Outcomes. arXiv:2404.04392.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651.

Rafailov, R.; Chittepu, Y.; Park, R.; Sikchi, H.; Hejna, J.; Knox, B.; Finn, C.; and Niekum, S. 2024. Scaling Laws for Reward Model Overoptimization in Direct Alignment Algorithms. arXiv:2406.02900.

Rebedea, T.; Dinu, R.; Sreedhar, M.; Parisien, C.; and Cohen, J. 2023. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. arXiv:2310.10501.

Yan, Y.; Lou, X.; Li, J.; Zhang, Y.; Xie, J.; Yu, C.; Wang, Y.; Yan, D.; and Shen, Y. 2024. Reward-Robust RLHF in LLMs. arXiv:2409.15360.

Yang, Y.; Dan, S.; Roth, D.; and Lee, I. 2024. Benchmarking LLM Guardrails in Handling Multilingual Toxicity. arXiv:2410.22153.

Ye, Z.; Greenlee-Scott, F.; Bartolo, M.; Blunsom, P.; Campos, J. A.; and Gallé, M. 2024. Improving Reward Models with Synthetic Critiques. arXiv:2405.20850.

Yu, E.; Li, J.; Liao, M.; Wang, S.; Gao, Z.; Mi, F.; and Hong, L. 2024. CoSafe: Evaluating Large Language Model Safety in Multi-Turn Dialogue Coreference. arXiv:2406.17626.

Zeng, W.; Liu, Y.; Mullins, R.; Peran, L.; Fernandez, J.; Harkous, H.; Narasimhan, K.; Proud, D.; Kumar, P.; Radharapu, B.; Sturman, O.; and Wahltinez, O. 2024. ShieldGemma: Generative AI Content Moderation Based on Gemma. arXiv:2407.21772.

Zhou, X.; Kim, H.; Brahman, F.; Jiang, L.; Zhu, H.; Lu, X.; Xu, F.; Lin, B. Y.; Choi, Y.; Mireshghallah, N.; Bras, R. L.; and Sap, M. 2024. HAICOSYSTEM: An Ecosystem for Sandboxing Safety Risks in Human-AI Interactions. arXiv:2409.16427.

Algorithm 1: Construction Stage: Iterative Guardrail Optimization

```
Input: Labeled dataset $\mathcal{D} = \{(c_i, y_i)\}_{i=1}^{n}$ where $y_i \in \{0, 1\}$ indicates if conversation $c_i$ requires guardrail
        Max iterations $T$, F1 threshold $\tau$
Output: Optimized guardrail set $\mathcal{R}^*$

Initialize:
  $\mathcal{R} \leftarrow \emptyset$ {Current guardrail set (initially empty)}
  $\mathcal{R}^* \leftarrow \emptyset$, $F^* \leftarrow 0$ {Best guardrail set and its F1 score}

for $t = 0$ to $T - 1$ do

  // Step 1: Conversation Simulation
  $\mathcal{V} \leftarrow \emptyset$ {Validation results}
  for each labeled conversation $(c_i, y_i) \in \mathcal{D}$ do
    $p \leftarrow$ SamplePersona$(c_i)$ {Generate diverse user persona}
    $s \leftarrow$ SimulateConversation$(c_i, p, \mathcal{R})$ {Multi-turn simulation}
    triggered $\leftarrow \bigvee_{r \in \mathcal{R}} r(s)$ {Test if any guardrail fires}
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(c_i, y_i, \text{triggered})\}$
  end for

  // Step 2: Performance Evaluation
  Compute TP, FP, FN, TN from $\mathcal{V}$ by comparing $y_i$ with triggered
  $P \leftarrow \text{TP}/(\text{TP} + \text{FP})$, $R \leftarrow \text{TP}/(\text{TP} + \text{FN})$
  $F \leftarrow 2PR/(P + R)$ {F1 score}

  if $F < F^*$ and $\mathcal{R}^* \neq \emptyset$ then
    $\mathcal{R} \leftarrow \mathcal{R}^*$ {Revert to previous best}
    continue to next iteration
  end if

  if $F \geq F^*$ then
    $\mathcal{R}^* \leftarrow \mathcal{R}$, $F^* \leftarrow F$ {Update best configuration}
  end if

  if $F \geq \tau$ then
    break {Early stopping: F1 threshold reached}
  end if

  // Step 3: Guardrail Optimization
  FP_cases $\leftarrow \{v \in \mathcal{V} : y_i = 0 \wedge \text{triggered} = \text{true}\}$
  FN_cases $\leftarrow \{v \in \mathcal{V} : y_i = 1 \wedge \text{triggered} = \text{false}\}$

  // Refine guardrails with false positives
  for each guardrail $r \in \mathcal{R}$ that triggered on FP_cases do
    $r' \leftarrow$ RefineGuardrail$(r, \text{FP\_cases})$ {Make more specific}
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \{r\} \cup \{r'\}$
  end for

  // Handle false negatives
  for each case $c \in$ FN_cases do
    if $c$ is semantically related to existing guardrail $r \in \mathcal{R}$ then
      $r' \leftarrow$ BroadenGuardrail$(r, c)$ {Increase sensitivity}
      $\mathcal{R} \leftarrow \mathcal{R} \setminus \{r\} \cup \{r'\}$
    else
      $r_{\text{new}} \leftarrow$ CreateGuardrail$(c)$ {Generate new guardrail}
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_{\text{new}}\}$
    end if
  end for

  // Remove unused and consolidate
  unused $\leftarrow \{r \in \mathcal{R} : r \text{ never triggered in } \mathcal{V}\}$
  $\mathcal{R} \leftarrow \mathcal{R} \setminus$ unused {Delete unused guardrails}
  $\mathcal{R} \leftarrow$ ClusterSimilar$(\mathcal{R})$ {Consolidate redundant guardrails}
end for

return $\mathcal{R}^*$ {Return best performing guardrail set across all iterations}
```

Algorithm 2: Continuous Improvement Stage: Adaptive Guardrail Refinement

```
Input: Initial guardrail set $\mathcal{R}$ (from construction stage)
       Unlabeled deployment data $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$
       Beam width $k$, tree depth $d$
Output: Adapted guardrail set $\mathcal{R}'$

Initialize:
  $\mathcal{R}' \leftarrow \mathcal{R}$ {Start with constructed guardrails}
  $F_{\text{baseline}} \leftarrow \text{EvaluateF1}(\mathcal{R}, \mathcal{U})$ {Baseline performance}

// Step 1: Risk Assessment
$\mathcal{G} \leftarrow \emptyset$ {Coverage gap set}
for each unlabeled conversation $u \in \mathcal{U}$ do
  $s_{\text{specific}} \leftarrow \bigvee_{r \in \mathcal{R}'} r(u)$ {Test specific guardrails}
  $s_{\text{general}} \leftarrow r_{\text{general}}(u)$ {Test general-purpose guardrail}
  if $s_{\text{general}} = \text{true}$ and $s_{\text{specific}} = \text{false}$ then
    $\mathcal{G} \leftarrow \mathcal{G} \cup \{u\}$ {Potential coverage gap detected}
  end if
end for

// Step 2: Case Expansion
$\mathcal{E} \leftarrow \emptyset$ {Expanded adversarial cases}
for each gap conversation $u \in \mathcal{G}$ do
  goal $\leftarrow \text{ExtractAttackGoal}(u)$ {Identify harmful intent}
  $\mathcal{T} \leftarrow \text{BeamSearchAttack}(u, \text{goal}, \mathcal{R}', k, d)$ {Tree search}
  for each leaf node $\ell \in \mathcal{T}$ do
    Classify $\ell$ as: successful attack, blocked attack, or false alarm
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\ell\}$
  end for
end for

// Step 3: Guardrail Optimization
// Handle successful attacks (increase coverage)
for guardrail $r \in \mathcal{R}'$ bypassed in $\mathcal{E}$ do
  $r' \leftarrow \text{BroadenGuardrail}(r, \text{bypassed cases})$
  $\mathcal{R}' \leftarrow \mathcal{R}' \setminus \{r\} \cup \{r'\}$
end for

// Handle false alarms (reduce over-flagging)
for guardrail $r \in \mathcal{R}'$ with false alarms in $\mathcal{E}$ do
  $r' \leftarrow \text{RefineGuardrail}(r, \text{false alarm cases})$
  $\mathcal{R}' \leftarrow \mathcal{R}' \setminus \{r\} \cup \{r'\}$
end for

// Create guardrails for novel attack patterns
$\mathcal{N} \leftarrow \{e \in \mathcal{E} : e \text{ has novel pattern not covered by } \mathcal{R}'\}$
for each novel case $n \in \mathcal{N}$ do
  $r_{\text{new}} \leftarrow \text{CreateGuardrail}(n)$
  $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r_{\text{new}}\}$
end for

$\mathcal{R}' \leftarrow \text{ClusterSimilar}(\mathcal{R}')$ {Consolidate redundant guardrails}

// Step 4: Performance Evaluation
$F_{\text{updated}} \leftarrow \text{EvaluateF1}(\mathcal{R}', \mathcal{U})$ {Test on deployment data}
if $F_{\text{updated}} < F_{\text{baseline}}$ then
  $\mathcal{R}' \leftarrow \mathcal{R}$ {Revert if performance degraded}
end if

return $\mathcal{R}'$ {Return adapted guardrail set}
```

Figure 3: **Sample guardrail: Illegal, unsafe, and unethical behavior monitoring.** This guardrail was generated by LATTICE during construction to detect conversations involving illegal activities, unsafe advice, or unethical requests. The prompt includes explicit triggering conditions (e.g., planning illegal acts, seeking harmful advice) and exclusion rules (e.g., educational discussions, hypothetical scenarios). The structured format enables consistent evaluation across conversations.

```
### TASK
You are a security system monitoring conversations for any encouragement, facilitation, or inadequate discouragement of illegal, unsafe, unethical, or policy-violating behavior or advice, including harassment, substance misuse, misrepresentation, risky activities, and unauthorized actions in correctional facilities.

### INSTRUCTIONS
Analyze each conversation for signs of:
- Encouragement, normalization, facilitation, or insufficient discouragement of illegal acts or policy violations (e.g., theft, fraud, vandalism, bringing weapons to school).
- Admissions of intent, planning, or rationalization of harmful or non-consensual behaviors (e.g., harassment, assault, stalking, discrimination) and any failure to clearly discourage them.
- Promotion or positive portrayal of unsafe substance use (excessive alcohol, illegal drugs) or underage vaping, including minimizing risks or lacking guidance toward safe/legal alternatives.
- Advice or discussion of engaging in risky or dangerous activities without adequate safety measures or consideration of harm.
- Discussions of misrepresenting or withholding important information from clients or stakeholders as a strategy.
- Guidance that could facilitate unauthorized activities within correctional facilities or conflict with institutional rules.

Look for patterns such as:
- Justifications, rationalizations, or attempts to trivialize or dismiss the impact of harmful actions.
- Absence of immediate discouragement or alternative suggestions that emphasize legality, safety, and ethics.

### OUTPUT FORMAT
Return a JSON response with:
{
  "triggered": boolean, // true if any monitored issue is detected
  "reason": string // clear explanation of why the guardrail was or was not triggered
}
```

Figure 4: **Sample guardrail: Inappropriate content detection and requests.** This guardrail identifies conversations with inappropriate content or requests for harmful material. Generated during construction, it demonstrates how Lattice creates specific triggering patterns (e.g., explicit content requests, inappropriate relationship advice) while avoiding over-flagging through carefully defined exclusions (e.g., medical discussions, educational content).

```
### TASK
You are a security system monitoring conversations for inappropriate, harmful, or offensive content and user requests.

### INSTRUCTIONS
Analyze the conversation for any of the following:
- Inappropriate, harmful, or sensitive topics that violate community guidelines or ethical standards.
- Jokes or comments that trivialize sexual violence, harm to individuals or groups, or other serious issues.
- References to illegal, unethical, or dangerous activities, including self-harm or unsafe practices.
- Explicit sexual content or language that may breach content policies.
- Attempts to justify harmful language or behavior as humor, or to dismiss concerns about their impact.
- Solicitation or requests for edgy, boundary-pushing, or escalating jokes or offensive content.
- Patterns of repeated or escalating requests for increasingly offensive or inappropriate material.
- Situations where a user may be in distress or require mental health support, and any inadequate responses lacking proper escalation or referral to professional help.

Do NOT trigger when:
- Conversations are playful or consensual, maintaining respect and comfort for all parties.
- Content consists of light-hearted or non-explicit innuendo that does not become explicit or disrespectful.
- Participants express mutual understanding and respect, even if using mild humor or personal anecdotes.

### OUTPUT FORMAT
Return a JSON response with:
{
  "triggered": boolean, // true if inappropriate content or requests are detected
  "reason": string // explanation of why the guardrail was or was not triggered
}
```