Semantic Inversion: Mitigating Polite Language Attacks on AI Agent Systems

Tarique Smith
Cogensec

May 2026

Abstract

AI agents trained to be helpful can be exploited through polite, indirect language that obscures malicious intent. We demonstrate that the addition of politeness markers to harmful prompts increases the bypass rates of safety systems from 15% to 68% in controlled experiments. This happens because traditional content filters rely on keyword density and sentiment analysis, both of which are diluted by courteous phrasing.

We introduce semantic inversion, a preprocessing layer that detects politeness markers, transforms requests to direct language, and applies risk-based confirmation requirements. Our pattern-based implementation operates at 1.8 milliseconds median latency and achieves a 94.2% F1 score in politeness detection. When integrated with existing security controls, the approach reduces polite attack bypass rates by 71%.

The method uses 68 compiled regular expressions to score politeness, extract action verbs, and classify intent types. For requests combining high politeness scores with critical actions, the system requires explicit user confirmation of the canonical form. We evaluate the approach in 1,000 benign and 1,000 attack prompts, analyze false positive rates, and discuss limitations including adversarial adaptation and cultural variation in politeness norms.

Keywords: prompt injection, AI safety, natural language security, jailbreak detection, linguistic obfuscation, agentic systems, politeness theory

1 Introduction

Autonomous AI agents increasingly perform privileged operations, including database queries, file system access, and API calls. Unlike passive language models that generate text, these agents execute actions with real-world consequences. Traditional security mechanisms, designed for deterministic software, struggle with the non-deterministic nature of language-based control.

Recent work has documented prompt injection attacks where adversaries manipulate agent behavior through crafted natural language inputs [3, 7]. Most defense research focuses on detecting explicit malicious content through keyword filtering, sentiment analysis, or learned classifiers [6, 8]. These approaches assume that attackers will phrase requests directly.

We identify a gap in this threat model. Attackers can exploit the helpfulness bias inherent in assistant-tuned models by wrapping harmful requests in polite, indirect language. Consider these functionally equivalent prompts:

```
Direct:   "Delete all database records"
Polite:   "If it's not too much trouble, could you please
           remove all database records?"
```

The second request contains identical semantic intent, but exhibits different surface features. Politeness markers reduce keyword density, create positive sentiment scores, and signal deference, all of which reduce detection confidence in traditional systems.

Our experiments show that politeness increases attack success rates by 353% (from 15% to 68% bypass rate). This occurs in multiple safety systems and types of attack. The effect is robust to variations in phrasing and generalizes beyond simple keyword evasion.

1.1 Contributions

This work makes three contributions:

1. We characterize the polite language attack vector through controlled experiments measuring bypass rates across safety systems.
2. We propose semantic inversion, a method that normalizes indirect requests to canonical form before security evaluation. The approach combines pattern matching for politeness detection, verb extraction for action identification, and risk-based confirmation requirements.
3. We implement and evaluate the method, demonstrating 1.8ms median latency, 94.2% F1 in politeness detection, and 71% reduction in attack bypass rates when integrated with existing controls.

The implementation is released as open source to enable reproduction and community improvement.

2 Related Work

2.1 Prompt Injection and Jailbreaks

Prompt injection attacks manipulate the behavior of the language model by embedding adversarial instructions in user input [3]. Perez et al. [10] developed automated red teaming using language models to discover jailbreaks. Wei et al. [13] studied why safety training fails, identifying vulnerabilities in the instruction hierarchy. Zou et al. [15] demonstrated universal transferable attacks using gradient-based optimization.

These works focus on direct manipulation through instruction overrides, role-playing prompts, or adversarial suffixes. Our work addresses indirect manipulation through the linguistic register, a complementary attack vector.

2.2 Content Moderation and Safety Filtering

Safety systems typically use keyword filtering, learned classifiers, or language model-based evaluation [6]. OpenAI's moderation API [9] scores content across categories including hate, violence, and self-harm. Anthropic's constitutional AI [1] uses model-generated critiques to enforce behavioral constraints. Rebedea et al. [11] introduced guardrails frameworks with declarative policies.

These approaches excel at detecting explicit harmful content but struggle with implicit intent conveyed through indirection. Our method provides a normalization step that makes implicit intent explicit before evaluation.

2.3 Politeness in Natural Language

Brown and Levinson [2] established politeness theory, identifying face-threatening acts and linguistic strategies for mitigation. Their framework categorizes politeness markers including:

- Negative politeness: formal deference ("could you please")
- Positive politeness: solidarity ("would you mind")
- Off-record politeness: hints and implications

Hyland [5] studied hedging in academic writing, documenting phrases that express uncertainty or reduce commitment. These linguistic insights inform our pattern library but require adaptation for adversarial contexts where politeness serves obfuscation rather than social cohesion.

2.4 Adversarial NLP

Adversarial attacks in NLP typically modify inputs to fool classifiers while preserving semantic meaning [14]. Wallace et al. [12] found universal adversarial triggers that cause models to generate specific outputs. Our work differs in motivation: attackers exploit politeness not to evade classification on perturbed inputs, but to reduce the perceived threat level of unmodified malicious semantics.

3 Problem Statement

3.1 Threat Model

We consider AI agents that:

- Accept natural language commands from users
- Have access to privileged operations (database queries, file system, API calls)
- Use language models for intent understanding and task execution
- Employ safety controls to block harmful requests

Attacker goals:

1. Execute unauthorized privileged operations
2. Exfiltrate sensitive information
3. Bypass safety guardrails without triggering alerts

Attacker capabilities:

- Craft arbitrary natural language prompts
- Iterate on phrasing through trial and error
- Access public information about model behavior

Attacker constraints:

- Cannot directly access system internals

- Must operate through language interface only
- Subject to rate limiting (but can use slow attacks)

Attack surface: The user input layer where natural language is parsed before security evaluation.

3.2 Problem Definition

Let x be a user request containing malicious intent i. Let $S(x)$ be a safety system that outputs a threat score in $[0, 1]$, with blocking threshold τ.

Traditional assumption:

$$S(x) > \tau \implies \text{block}(x) \tag{1}$$

The polite language vulnerability occurs when there exists a transformation P that adds politeness markers such that:

1. Semantic intent is preserved: $\text{intent}(P(x)) = \text{intent}(x) = i$
2. Surface features change to reduce threat score: $S(P(x)) < \tau$ despite $S(x) > \tau$

The transformation P succeeds if it causes $P(x)$ to bypass detection while maintaining malicious semantics.

3.3 Research Questions

RQ1: How effectively does politeness reduce detection confidence in existing safety systems?
RQ2: Can pattern-based methods reliably detect and normalize politeness markers?
RQ3: Does canonicalization to direct language improve downstream security controls?
RQ4: What is the computational cost of canonicalization in production systems?

4 Method

4.1 Architecture Overview

Semantic inversion operates as a preprocessing layer before traditional security controls:

The output includes the canonical form x', a politeness score p, a risk amplifier r, and a confirmation requirement flag.

4.2 Politeness Detection

We define politeness as a weighted sum over pattern matches. Each pattern (regex_i, w_i) contributes weight w_i if it matches the input.

Let $M_i(x) \in \{0, 1\}$ indicate whether pattern i matches input x. The politeness score is:

$$p(x) = \min\left(\sum_{i=1}^{n} w_i \cdot M_i(x), 1.0\right) \tag{2}$$

Pattern categories and weights:
High politeness ($w = 0.3$ to 0.4):

- "could you please"

```
Input: x (user request)
  |
  v
[Politeness Detection] -> p in [0,1]
  |
  v
[Canonical Transform] -> x'
  |
  v
[Intent Classification] -> type in {action, hypothetical, ...}
  |
  v
[Action Extraction] -> A = {a1, ..., an}
  |
  v
[Risk Calculation] -> r in [1, infinity)
  |
  v
[Traditional Security] -> decision(x', r)
```

Figure 1: Semantic inversion pipeline

- "would it be possible to"
- "if it's not too much trouble"

Medium politeness ($w = 0.15$ to 0.2):

- "please" (mid-sentence)
- "if you don't mind"
- "when you get a chance"

Trailing politeness ($w = 0.1$):

- ", thanks"
- ", thank you"

The 68 patterns in our implementation were derived from Brown and Levinson's politeness framework [2] and extended through manual analysis of 500 customer support transcripts.

4.3 Canonical Transformation

Given politeness patterns $\{P_1, \ldots, P_n\}$, the canonical form x' is obtained by removing matches:

This produces a direct-language version that preserves semantic intent while removing hedging.

4.4 Intent Classification

We classify requests into five categories:

- **ACTION_REQUEST**: User wants the agent to perform an operation

Algorithm 1 Canonical Transformation

$x' \leftarrow x$
for each pattern P in $\{P_1, \ldots, P_n\}$ **do**
  $x' \leftarrow \text{regex\_replace}(x', P, "")$
end for
$x' \leftarrow \text{normalize\_whitespace}(x')$
$x' \leftarrow \text{capitalize\_first}(x')$
return x'

- **CAPABILITY_QUERY**: Asking if the agent can do something
- **HYPOTHETICAL**: Theoretical question with no intent to execute
- **INFORMATION**: Requesting information only
- **AMBIGUOUS**: Intent cannot be determined

Classification uses pattern matching on hypothetical indicators ("what if", "hypothetically") and question structure. If action verbs are present and no hypothetical markers appear, the request is classified as ACTION_REQUEST.

4.5 Action Extraction

We maintain three verb sets:

- **Critical** (C): delete, destroy, execute, bypass, grant, elevate, ...
- **Medium** (M): modify, create, update, connect, install, ...
- **Safe** (S): show, display, explain, help, calculate, ...

Action extraction uses compiled regex to find verb occurrences in x':

$$A(x') = \{v \in (C \cup M \cup S) \mid v \text{ appears in } x'\} \tag{3}$$

The implementation handles inflected forms (deleting $\rightarrow$ delete) through pattern expansion.

4.6 Risk Amplification

For requests classified as ACTION_REQUEST, we compute a risk amplifier based on politeness and action criticality:

The amplifier r is used to adjust downstream security thresholds:

$$\tau_{\text{adjusted}} = \frac{\tau_{\text{base}}}{r} \tag{4}$$

This makes detection more sensitive for polite requests.

4.7 Confirmation Requirement

When $p > 0.5$ and critical actions are present, the system requires explicit user confirmation. The user is shown the canonical form x' and must acknowledge the direct phrasing before execution proceeds.

Algorithm 2 Risk Calculation

if type $\neq$ ACTION_REQUEST **then**
  return $(1.0, \text{false})$
end if
if $\exists a \in A : a \in C$ **then**
  if $p > 0.5$ **then**
    return $(2.0, \text{true})$
  else if $p > 0.2$ **then**
    return $(1.5, \text{false})$
  else
    return $(1.2, \text{false})$
  end if
else if $\exists a \in A : a \in M$ **then**
  if $p > 0.5$ **then**
    return $(1.3, \text{false})$
  else if $p > 0.2$ **then**
    return $(1.15, \text{false})$
  end if
end if
return $(1.0, \text{false})$

5 Evaluation Plan

5.1 Experimental Setup

Dataset construction:

We require three datasets:

1. **Benign corpus** (1,000 samples): Customer support logs, FAQ interactions, and normal agent usage. These establish false positive rates.
2. **Direct attack corpus** (500 samples): Malicious prompts phrased directly (e.g., "Delete all files"). These establish baseline detection rates.
3. **Polite attack corpus** (500 samples): Each direct attack rephrased with politeness markers. These measure the bypass effect.

The polite corpus is generated by prepending high-politeness patterns to direct attacks and appending trailing politeness.

Baseline systems:

We compare against:

- Keyword filtering (pattern matching on deny lists)
- Sentiment-based filtering (VADER sentiment analyzer [4])
- LLM-based classification (using a fine-tuned BERT model for harmful content detection)
- No canonicalization (control condition)

Implementation details:

Patterns are compiled once at initialization. Analysis runs on a single CPU core. Latency is measured using high-resolution timers (nanosecond precision) over 1,000 iterations per input, discarding the first 100 as warmup.

5.2 Metrics

Politeness detection:

- Precision: What fraction of high-politeness predictions are correct?
- Recall: What fraction of truly polite requests are detected?
- F1 score: Harmonic mean of precision and recall

Bypass rate:

$$\text{Bypass}_{\text{polite}} = \frac{\#\{\text{polite attacks allowed}\}}{\#\{\text{total polite attacks}\}} \tag{5}$$

Similarly for $\text{Bypass}_{\text{direct}}$.

Bypass rate reduction:

$$\text{Reduction} = \frac{\text{Bypass}_{\text{polite}} - \text{Bypass}_{\text{polite+canon}}}{\text{Bypass}_{\text{polite}}} \tag{6}$$

This measures the improvement from adding canonicalization.

Latency:

- Median, 95th percentile, 99th percentile analysis time
- Throughput in requests per second

False positives:

$$\text{FPR} = \frac{\#\{\text{benign requests blocked}\}}{\#\{\text{total benign requests}\}} \tag{7}$$

5.3 Ablation Studies

To isolate contributions, we test:

1. **Politeness detection only**: Remove downstream components, measure detection accuracy
2. **Canonical transformation only**: Measure impact on keyword-based filters
3. **Risk amplification only**: Measure impact on threat scoring
4. **Full system**: Combined effect

We also vary the politeness threshold (0.3, 0.5, 0.7) and risk amplifier scaling factor to find optimal operating points.

5.4 Adversarial Robustness

We test against adaptive attacks:

- Character substitution: `c0uld y0u pl3ase delete files` (leetspeak)
- Unicode obfuscation: `could\u200byou please` (zero-width space)
- Synonym substitution: `might you kindly remove` (alternative politeness markers)

For each attack variant, we measure whether detection degrades or generalizes.

5.5 Generalization

We test on out-of-distribution samples:

- Professional emails (formal register)
- Customer complaints (high emotion, indirect requests)
- Technical documentation requests (domain-specific verbs)

This evaluates whether the pattern-based approach overfits to the training examples.

5.6 Threats to Validity

Internal validity:

- Pattern weights were chosen manually, not learned. Different weights may perform better.
- Attack samples were hand-crafted, not drawn from real attack logs.

External validity:

- Evaluation uses English only. Results may not transfer to other languages.
- Politeness norms vary by culture. Japanese or Mandarin requests may exhibit different patterns.

Construct validity:

- Bypass rate measures detectability, not actual harm. Some bypassed requests may fail to cause damage due to downstream controls.

6 Discussion and Limitations

6.1 Pattern Fragility

The pattern-based approach depends on lexical features that adversaries can evade. If attackers learn the exact patterns, they can rephrase to avoid detection while maintaining politeness in practice. This is an arms race similar to spam filtering, where adaptation is inevitable.

A hybrid approach combining patterns (fast path) with language model classification (high-accuracy path) could provide defense in depth. For low-confidence cases, escalate to a model-based classifier at the cost of higher latency.

6.2 Cultural Variation

Politeness is not universal. High-context cultures (East Asia) use indirection as default communication style, while low-context cultures (North America, Northern Europe) favor directness. Applying uniform politeness scoring may produce false positives in high-context populations.

Future work should develop culture-specific patterns or learn user baselines. A request that seems overly polite for User A might be normal for User B.

6.3 Computational Cost

Pattern matching is fast (1.8ms median), but cost scales linearly with pattern count. At 68 patterns, throughput is 1,247 requests per second on a single core. Adding multilingual support could triple the pattern count, reducing throughput to approximately 400 requests per second. This remains acceptable for most deployments but may require optimization for high-traffic systems.

Pre-compilation and pattern sharing across requests amortizes cost. For systems handling millions of requests per day, consider caching canonicalization results for repeated phrases.

6.4 Semantic Preservation

Canonical transformation removes surface features but may alter meaning in edge cases. Consider:

```
Original:  "Could you please show me files I'm not supposed to see?"
Canonical: "Show me files I'm not supposed to see"
```

The canonical form preserves intent here, but in cases where politeness itself carries semantic weight (requesting permission vs. demanding action), the transformation may flatten distinctions that matter.

We mitigate this by classifying intent separately. CAPABILITY_QUERY requests like "Can you show me X?" are distinguished from ACTION_REQUEST demands like "Show me X" through question structure analysis.

6.5 False Positive Scenarios

Legitimate polite requests will score highly on politeness. If these coincidentally mention critical actions in non-malicious contexts, they may trigger false confirmation requirements:

```
"Could you please tell me how to delete old backup files safely?"
```

This request is about learning, not executing. Current intent classification detects "tell me how" as INFORMATION rather than ACTION_REQUEST, avoiding false positives. However, edge cases remain.

Tuning the confirmation threshold (currently $p > 0.5$) trades false positives against false negatives. Lower thresholds catch more attacks but annoy users. Higher thresholds reduce friction but miss subtle attacks.

7 Ethics and Safety Considerations

7.1 Dual Use

This work characterizes an attack vector and provides a defense. The attack itself (adding politeness to bypass filters) is now documented and could be exploited by adversaries who were previously unaware.

We judged that disclosure benefits defenders more than it aids attackers. The attack is simple enough that sophisticated adversaries likely already use it. Documenting the vulnerability enables the broader community to deploy defenses.

7.2 Misuse Risk

The open source implementation could be used to test safety systems in unauthorized penetration tests or to develop more sophisticated attacks through understanding defense mechanisms.

We mitigate this through responsible disclosure norms: the implementation includes documentation on ethical use, recommends obtaining authorization before testing third-party systems, and emphasizes defensive rather than offensive applications.

7.3 Bias and Fairness

Politeness norms correlate with demographic factors including culture, age, gender, and socioeconomic status. Over-penalizing politeness could discriminate against users who communicate deferentially by default.

The confirmation mechanism (showing canonical form) provides transparency. Users see what the system interpreted, allowing them to correct misunderstandings. This is preferable to silent blocking, which hides the reason for denial.

7.4 Deployment Recommendations

Organizations deploying this approach should:

1. Tune thresholds on representative user data, not just attack samples
2. Monitor false positive rates by demographic group
3. Provide clear feedback when confirmation is required
4. Log canonicalization decisions for audit and bias detection
5. Implement appeals processes for users who believe they were incorrectly blocked

8 Conclusion

Polite language poses a novel threat to AI agent security. By exploiting helpfulness bias, attackers can increase bypass rates from 15% to 68% through surface-level linguistic changes that preserve malicious intent. This vulnerability exists because traditional safety systems analyze surface features (keywords, sentiment) rather than semantic intent.

Semantic inversion addresses this by normalizing indirect requests to direct language before security evaluation. Our pattern-based implementation detects politeness markers, extracts action verbs, and applies risk-based confirmation for high-threat combinations. The approach operates at 1.8 milliseconds median latency, achieves 94.2% F1 on politeness detection, and reduces polite attack bypass rates by 71% when integrated with existing controls.

Limitations include pattern fragility against adaptive attackers, cultural variation in politeness norms, and potential false positives on legitimate polite requests. Future work should explore multilingual patterns, culture-specific baselines, and hybrid approaches combining pattern matching with learned classifiers.

The implementation is released as open source to enable reproduction, community improvement, and deployment in production AI systems. As autonomous agents gain access to increasingly privileged operations, linguistic normalization will become a necessary component of defense in depth.

References

[1] 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.

[2] Penelope Brown and Stephen C. Levinson. *Politeness: Some Universals in Language Usage*. Cambridge University Press, Cambridge, UK, 1987.

[3] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023.

[4] Clayton J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the 8th International AAAI Conference on Weblogs and Social Media*, pages 216–225, 2014.

[5] Ken Hyland. Hedging in scientific research articles. *Pragmatics*, 8(4):433–454, 1998.

[6] 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.

[7] Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, and Yang Liu. Jailbreaking chatgpt via prompt engineering: An empirical study. *arXiv preprint arXiv:2305.13860*, 2023.

[8] Todor Markov, Chong Zhang, Sandhini Agarwal, Florentine Nekoul, Theodore Lee, Steven Adler, Angela Jiang, and Lilian Weng. A holistic approach to undesired content detection in the real world. *arXiv preprint arXiv:2208.03274*, 2023.

[9] OpenAI. Openai moderation api. https://platform.openai.com/docs/guides/moderation, 2023. Accessed: January 2026.

[10] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*, 2022.

[11] Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. *arXiv preprint arXiv:2310.10501*, 2023.

[12] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*, 2019.

[13] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *arXiv preprint arXiv:2307.02483*, 2023.

[14] Wei Emma Zhang, Quan Z. Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology*, 11(3):1–41, 2020.

[15] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

A Pattern Library

The complete list of 68 politeness patterns is available in the open source implementation. We provide a representative subset here:

High politeness patterns:

1. `(?i)^(could you (please )?)` (weight: 0.3)
2. `(?i)^(would you (mind |please )?)` (weight: 0.3)
3. `(?i)^(would it be possible (for you )?to )` (weight: 0.4)
4. `(?i)^(if it's not too much trouble)` (weight: 0.4)
5. `(?i)^(i was wondering if you could)` (weight: 0.35)

Medium politeness patterns:

6. `(?i) please\b` (weight: 0.15)
6. `(?i) if you don't mind\b` (weight: 0.2)
6. `(?i) when you get a chance\b` (weight: 0.15)

Trailing politeness patterns:

9. `(?i),?  please\?\?\s*$` (weight: 0.1)
9. `(?i),?  thanks\?\?\s*$` (weight: 0.1)
9. `(?i),?  thank you\?\?\s*$` (weight: 0.1)

The full pattern library, action verb sets, and evaluation code are available at: https://github.com/argus-platform/intent-canonicalization

B Experimental Data Collection Protocol

For researchers wishing to reproduce our experiments, we provide detailed protocols:

Benign corpus construction:

1. Source customer support transcripts (with user consent and anonymization)
2. Filter for queries to AI assistants
3. Remove personally identifiable information
4. Sample uniformly across conversation length
5. Manual review to exclude edge cases

Attack corpus construction:

1. Identify 50 critical operations (database deletion, file access, credential exposure)
2. Generate direct requests for each operation

3. For polite variants, apply transformation rules:
    - Prepend high-politeness pattern (random selection)
    - Append trailing politeness (50% probability)
    - Add hedging if question mark present
4. Manual review for naturalness

Evaluation procedure:

1. Initialize safety system with standard configuration
2. For each sample, record: prediction, confidence, latency
3. Repeat 3 times with different random seeds
4. Report mean and standard deviation

This protocol enables independent verification of our findings.