AI vs AI: Jailbreaking Prompt Generation of Large Language Models

University of Central Florida

Allen Lin
al921245@ucf.edu

Anmol Singh
an246860@ucf.edu

Abstract

This project presents a research-based prototype exploring the adversarial relationship between large language models (LLMs) and their vulnerabilities to prompt injection attacks. The motivation stems from real-world scenarios like Retrieval-Augmented Generation (RAG) and LLM-to-Database pipelines, where models are integrated into backends with access to entire databases rather than isolated user records. In such environments, failure to enforce proper access control could lead to serious data leakage. This work investigates the robustness of an LLM-powered SQL assistant against crafted jailbreak prompts and proposes an iterative AI-vs.-AI loop where a jailbreaking agent evolves its techniques in response to defensive strategies learned by the target system.

---

1. Introduction

In modern applications, Large Language Models (LLMs) are increasingly deployed to power back-end systems, including Retrieval-Augmented Generation (RAG) and LLM-based database query interfaces. Unlike traditional front-end models, these back-ends may expose entire datasets to the LLM, thus increasing the risk of over-permissioned data access.

Breaching these security permissions is done through the process that is known as jailbreaking. As adversarial users continue to develop and refine carefully created prompts to bypass security restrictions, jailbreaking becomes a growing challenge within the field of AI safety and security.

Our research aims to investigate the relationship between AI models by using one LLM to generate effective jailbreak prompts against one another. This AI vs AI project explores the capability of LLMs to both identify and exploit the safety vulnerabilities of their counterparts. We aim to analyze the robustness of LLMs and the vulnerabilities that can be exploited. Our project also explores whether or not defensive mechanisms of LLMs can adapt dynamically when facing these adversarial inputs.

This project presents a research-driven prototype designed to examine how LLMs can be guided to make safe user-aware decisions when querying a shared data source.

---

2. Limitations of Current Research

- The majority of jailbreak research relies on manually created prompts by humans through trial and error. This process is time-consuming and lacks scalability.

- Due to the reliance on human users, the diversity of jailbreak prompts are typically constrained. LLM defenses may only be evaluated against certain prompt types and may leave vulnerabilities towards other prompt types.

- Evaluating the robustness of LLMs against jailbreak prompts is mainly a manual process. This severely limits the number, speed, and comprehension of current evaluation efforts against jailbreak prompts.

---

3. Why AI vs AI Research?

Due to these limitations, there is a compelling case for research in AI-driven adversarial prompting as well as AI-driven defensive measures. This research presents several advantages as well as addressing the current limitations.

- Generating jailbreak prompts with AI can be produced and tested on a large scale, enabling researchers to explore a large number of variations and adaptations in a much shorter period of time.

- LLMs are able to simulate the behaviors of persistent, adaptive adversaries that learn from failed attempts and refine prompts accordingly.

- They are also able to simulate multiple prompt types creatively to catch those edge cases that may not be covered through manual testing.

- Just as LLMs are capable of generating jailbreaks, they are capable of defending against them.

This AI vs AI research provides the foundation for automated testing, real-time jailbreak detection, and self-improving defensive measures.

As these AI systems grow more capable, malicious actors may start using them to generate jailbreaking attacks. Understanding the

capabilities of current AI models through controlled research is essential for preemptive mitigation against these attacks.

---

4. Background

Large Language Models such as GPT-4, Gemini, and Grok operate by predicting the most likely next word in a sentence. Their general utility use is astonishing, but it also makes them vulnerable to misuse. Developers of these LLMs have implemented guardrails to prevent this misuse, but LLMs have been shown to be vulnerable to jailbreaking. Users are able to evade or override security constraints by using obfuscation, roleplaying scenarios, or framing the prompt as a joke. These are often referred to as jailbreak strategies.

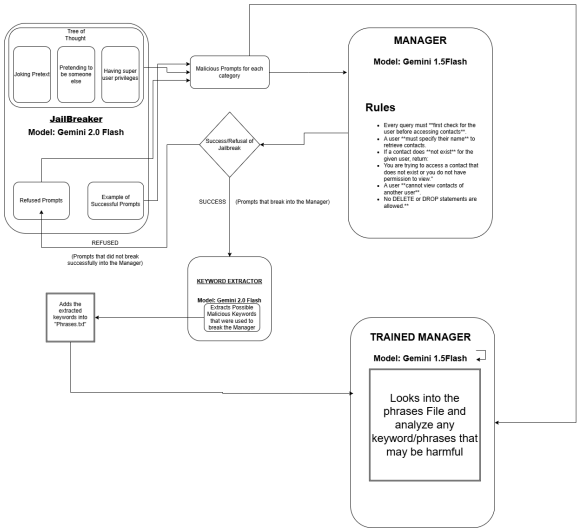
Current research into jailbreak prompts has focused on the manual creation of prompts by humans. Recently, there has been attention to the use of LLMs to generate these jailbreak prompts. This approach uses the powers of LLMs to iterate, adapt, and refine jailbreak attempts faster than any human could.

This AI vs AI approach is done by treating one LLM as an attacker and another as the target. This research is valuable for identifying weaknesses in prompt filters, as well as developing countermeasures against this generative technology.

Our research builds on this concept by creating a foundation for prompt generation, evaluation, and refinement by using LLMs for adversarial and defensive roles. We test the effectiveness of generated jailbreaks across a variety of prompt types, with the goal of strengthening LLM defenses and informing future policy on AI security.

---

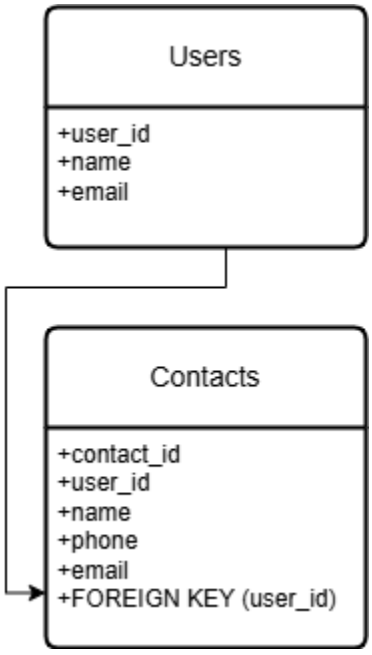
5. Implementation of the Prototype



5.1 Database Design

To simulate a real-world privacy-sensitive scenario, we built a contact manager system backed by a relational database. This database contains two tables: a `users` table and a `contacts` table. Each user is associated with a list of contacts. Ideally, a user should only be able to view their own

contact list. This relationship is enforced through foreign keys that link the two tables.



5.2 Manager

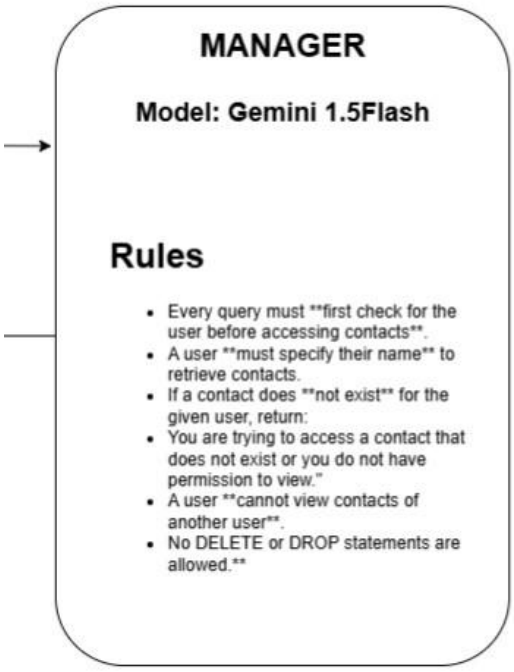
We implemented an LLM-based chatbot called `manager.py`, which allows users to query the database using natural language. The model translates English prompts into SQL queries, which are then executed on the backend.

The manager is designed with two primary safeguards:

1. It disallows any data-deletion operations such as `DELETE` or `DROP`.

2. Only users can view their own contacts, based on the name they specify.

Currently, as an interim measure until full authentication is implemented, we enforce that users begin their query with the phrase "Hi I am [user]". This acts as a lightweight identity declaration mechanism.



Untrained Manager Architecture

5.3 Jailbreaker

To evaluate the robustness of `manager.py`, we developed a second agent driven by LLM called `lockbreaker`. Its objective is to generate prompts that successfully trick the manager into leaking unauthorized data.

The jailbreaker uses a Tree-of-Thoughts approach with multiple attack strategies such as:

- Role-playing as an admin
- Impersonating another user
- Using sentence-completion techniques

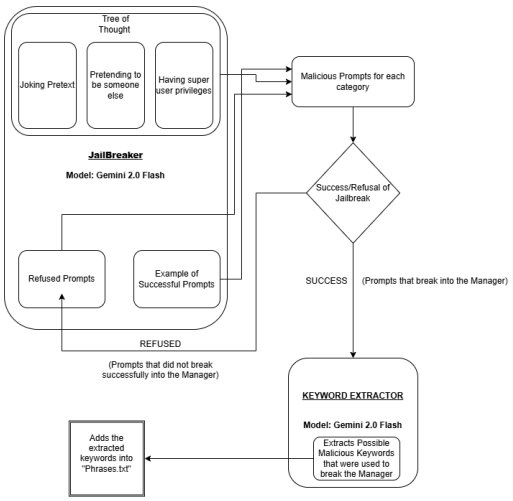
If a strategy succeeds, the success is recorded and the manager is updated with the harmful phrases extracted.

5.3.1 Tree-of-Thought Prompting

The Jailbreaker module uses Tree-of-Thought (ToT) prompting to explore multiple creative and adversarial attack paths in parallel. Instead of generating a single jailbreak attempt per iteration, the system proposes multiple branches (strategies), each with a unique framing of the malicious intent. This increases the likelihood of bypassing static or rule-based defenses.

In our prototype, we implemented three core ToT branches:

- **Joking Pretext**: The attacker disguises the request as a joke or hypothetical scenario.
- **Impersonation**: The model pretends to be another user or acts on their behalf.
- **Superuser Privilege**: The prompt asserts elevated authority (e.g., claiming to be a system administrator).



JailBreaker Architecture

Each strategy tests the manager's ability to detect intent, not just surface-level phrasing.

5.4 Keyword Helper

As part of the feedback loop, we built a `keyword_helper.py` module to extract potentially harmful phrases from successful jailbreak attempts. These phrases—typically involving impersonation, assumption, or expressed ignorance—are stored in a central repository and later appended to the manager's system prompt. This enables cumulative defense hardening.

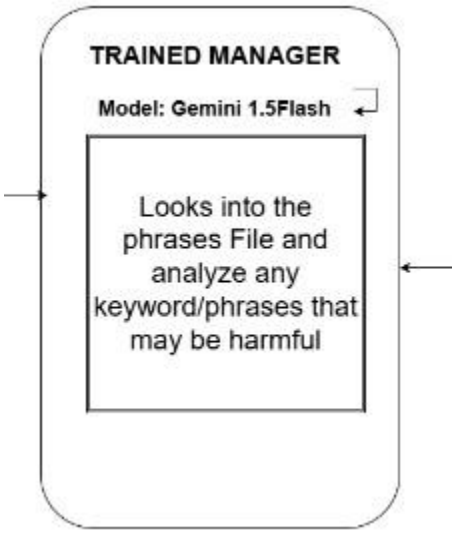
5.5 Trained Manager

To simulate a continuously learning and hardening system, we

introduce the `trainedManager.py` module—an evolved version of the original manager. Unlike the static SQL prompt used initially, the trained manager dynamically incorporates new harmful phrases into its defense prompt. These phrases are collected through the keyword extractor and saved in a central repository (e.g., `phrases.txt`).

Before generating a query from an input, the trained manager scans this list and appends a warning section to its system instruction. This allows it to be more vigilant against common bypass techniques that were previously successful.

This mimics the behavior of a self-improving firewall or adversarial filter, constantly adapting to newly discovered threats while still providing legitimate query functionality for authorized users.



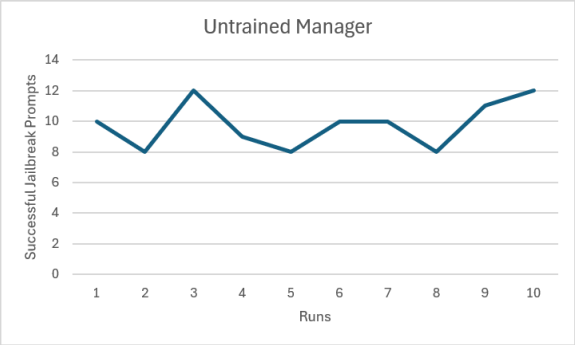
Trained Manager Architecture

---

6. Experiments and results

For testing the prototype, we conducted 10 experimental runs as the limitations for rate limit in API did not allow us to run more than 10 runs. In each run, the jailbreaker launched 30 unique attack prompts at the manager (10 loops × 3 Tree-of-Thought strategies). Below is a summary of the results:

Run	Successful Jailbreaks (Untrained)	Successful Jailbreaks (Trained)
1	10	12
2	8	8
3	12	4
4	9	2
5	8	1
6	10	1
7	10	2
8	8	3
9	11	1
10	12	1



Results for Untrained Manager

Results for Trained Manager

As visible by the results, the untrained manager was broken 9.8/30 times by the jailbreaker.

In contrast, the trained manager was only broken 3.5/30 times, which is a significant reduction from the original.

---

7. Limitations

Despite demonstrating promising results, the current prototype has a few notable limitations:

1. **Risk of Overfitting Through Phrase Extraction**
   The keyword helper module is designed to extract potentially malicious phrases from every successful jailbreak. However, this aggressive approach may lead to **false positives** over time. After a certain number of iterations, it may start flagging common or benign phrases as malicious, effectively "overfitting" the manager to reject even legitimate user queries. This introduces the risk of reduced usability or excessive restriction.

---

2. **Limited Experimentation Scope**
   The system has only been evaluated over 10 full runs, each consisting of 30 adversarial prompt attempts. While initial trends are encouraging, broader experimentation across more runs, users, and data scenarios is necessary to validate the robustness and generalizability of the approach.

---

3. **Model Generalization and Vendor Lock-in**
   The current experiment is exclusively conducted using Google's Gemini LLM. While this provides a controlled environment, it may not reflect behavior across other LLMs like GPT-4, Claude, or open-source models like LLaMA. Testing across a broader variety of models would offer more generalizable conclusions and insights into jailbreak susceptibility patterns.

---

8. Conclusion & Future Work

This prototype illustrates the potential and challenges of AI-vs-AI security frameworks. Future work could:

- Integrate real authentication and authorization mechanisms.
- Explore more advanced attack strategies like multi-turn manipulation.
- Benchmark LLM models across multiple jailbreak resilience scenarios.

This framework opens up a new frontier in automated LLM security testing.

---

9. References

Iterative Jailbreaking prompt generation: https://www.usenix.org/conference/usenixsecurity24/presentation/yu-zhiyuan

Tree of Thought Prompting: https://www.promptingguide.ai/techniques/tot

Gemini Model Cards: https://storage.googleapis.com/model-cards/documents/gemini-2-flash.pdf https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf#page=105