MULTI-MODEL MULTI-AGENT DEFENSE AGAINST LLM JAILBREAK ATTACKS

by

Arjen Aykan KILIÇ

B.S., Computer Science, Özyeğin University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2025

ACKNOWLEDGEMENTS

I would like to begin by sincerely thanking my advisor, Prof. Dr. Fatih Alagöz, for his exceptional guidance and constant support throughout this journey. His expertise and encouragement have been instrumental in shaping this thesis. I am also deeply grateful to the jury members, Prof. Dr. Emin Anarım and Assoc. Prof. Dr. Reyhan Aydoğan, for their insightful feedback and valuable suggestions, which have greatly improved my work.

To my parents, Aynur Kılıç and Özcan Kılıç, thank you for inspiring me with your own dedication to education and lifelong learning. Your wisdom, sacrifices, and belief in me have been the foundation of everything I have achieved.

To my family, colleagues and friends, I am deeply grateful for the countless inspirational conversations we've shared about the world, technology, and beyond.

Most importantly, I owe my deepest gratitude to my wife, Rabia Nur Kılıç. Her love, patience, and belief in me have been a cornerstone of my success, along with her academic insights. Your support, both intellectually and emotionally, has made this journey not only possible but also deeply meaningful. I am forever thankful for your love and partnership.

ABSTRACT

MULTI-MODEL MULTI-AGENT DEFENSE AGAINST LLM JAILBREAK ATTACKS

The robustness of Large Language Models (LLMs) against adversarial attacks such as jailbreak techniques is critical for their safe deployment especially in the sensitive domains. These attacks exploit LLM flexibility to bypass safety protocols, generating harmful, misleading, or unauthorized outputs. Existing single-agent and static defense mechanisms often struggle to address the dynamic and evolving jailbreak vulnerabilities. This paper introduces a Multi-Model Multi-Agent Defense (MAMAD) system designed to defend jailbreak attacks on LLMs. The proposed architecture incorporates specialized agents—Input Analyzer, Intent Analyzer, and Output Analyzer—utilizing the unique strengths of distinct LLMs to collaboratively evaluating inputs and mitigating potential threats. The system demonstrates significant improvements in detection accuracy against diverse attack vectors, including adversarial suffixes, latent injections, and role-playing techniques. Evaluation results show that the MAMAD system achieves a detection accuracy of 94%, outperforming individual LLMs and single-model multi-agent systems. Furthermore, this work explores the trade-offs between computational and latency overhead and system efficiency between individual LLMs and multi-agent defense architectures. The findings demonstrates the capability of multi-agent systems to improve the safety and reliability of LLM applications, while maintaining minimal false positive rates and scalability.

ÖZET

BÜYÜK DİL MODELLERİNE YÖNELİK JAILBREAK SALDIRILARINA KARŞI ÇOK MODELLİ ÇOK AJANLI SİSTEMLER İLE SAVUNMA

Büyük Dil Modellerinin (LLM'ler) jailbreak (hapisten kaçış) gibi saldırılara karşı dayanıklılığı, özellikle hassas alanlarda güvenli bir şekilde kullanılmaları açısından kritik önem taşımaktadır. Bu saldırılar, LLM'lerin esnekliğini kullanarak güvenlik protokollerini aşmakta ve zararlı, yanıltıcı veya yetkisiz çıktılar üretmektedir. Mevcut tek ajanlı ve statik savunma mekanizmaları, dinamik ve gelişen jailbreak açıklarını ele almakta genellikle yetersiz kalmaktadır. Bu çalışma, LLM'lere yönelik jailbreak saldırılarına karşı savunma amacıyla tasarlanmış Çok Modelli Çok Ajanlı (MAMAD) bir savunma sistemi önermektedir. Önerilen mimari, farklı LLM'lerin birbirinden farklı güçlü yönlerini kullanan özelleşmiş ajanları kullanır. Özelleşmiş ajanlar girdi analizi, niyet analizi ve çıktı analizi yaparak ortak karar almaktadır. Alınan kararın sonucuna göre girdiler zararlı ise elimine edilir. Sistem, snowball saldırıları, kodlanmış metin saldırıları ve rol yapma teknikleri gibi çeşitli saldırı vektörlerine karşı tespit doğruluğunda önemli iyileştirmeler sağlamaktadır. Değerlendirme sonuçları, MAMAD sisteminin %94 tespit doğruluğuna ulaştığını ve LLM'lerin yanı sıra tek modelle çalışan çok ajanlı sistemleri geride bıraktığını göstermektedir. Ayrıca, bu çalışma LLM'ler ile çok ajanlı savunma mimarileri arasında hesaplama gücü ve gecikme maliyetleri ile sistem verimliliği arasındaki dengeyi de araştırmaktadır. Bulgular, çok ajanlı sistemlerin LLM uygulamalarının güvenliğini ve güvenilirliğini artırdığını gösterirken, minimal yalancı pozitif oranları ve yüksek ölçeklenebilirlik sağladığını ortaya koymaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . ix
LIST OF SYMBOLS . . . x
LIST OF ACRONYMS/ABBREVIATIONS . . . xi
1. INTRODUCTION . . . 1
1.1. Vulnerabilities and Ethical Concerns . . . 2
1.1.1. Harmful Content Generation . . . 3
1.1.2. Misinformation and Fake Content . . . 3
1.1.3. Security Breaches and AI Misuse in Sensitive Fields . . . 4
1.2. Challenges in Jailbreak Defense . . . 4
1.3. Limitations in Current Defense Mechanisms . . . 6
1.4. Multi-Agent Systems in LLM Defense . . . 6
1.5. Key Contributions . . . 7
1.6. Thesis Outline . . . 8
2. RELATED WORK . . . 10
2.1. Jailbreak Attack Techniques . . . 10
2.1.1. DAN and Suffix Attacks . . . 10
2.1.2. Data Leak and Latent Injection Attacks . . . 10
2.1.3. Encoding and Glitch Attacks . . . 11
2.1.4. XSS Attacks . . . 11
2.1.5. Snowball Attacks . . . 12
2.1.6. Other Sophisticated Attacks . . . 12
2.2. Jailbreak Defense Techniques . . . 14
3. PROPOSED MODEL . . . 17
3.1. Architecture . . . 17
3.1.1. Input Analyzer . . . 17

3.1.2. Intent Analyzer 18
3.1.3. Output Analyzer 18
3.2. Workflow 18
3.3. Implementation 19
3.4. Probabilistic Modeling 20
4. EVALUATION 22
4.1. Experimental Setup 22
4.1.1. System Implementation 22
4.1.2. Hardware and Software Environment 23
4.2. Evaluation Metrics 24
4.3. Tools and Methodology 25
4.4. Results 25
4.5. False Positive Rates 27
4.6. Cost Analysis 29
4.7. Latency 31
5. CONCLUSION AND FUTURE WORK 32
REFERENCES 35

LIST OF FIGURES

Figure 3.1. MAMAD system workflow. 19

Figure 4.1. Individual LLMs vs MAMAD system accuracy. 25

Figure 4.2. Single-Model multi-agent vs MAMAD system accuracy. 28

LIST OF TABLES

Table 4.1. Individual LLMs vs MAMAD accuracy results. 26

Table 4.2. Single model multi-agent vs MAMAD accuracy results. 27

Table 4.3. False positive rates across different configurations. 28

Table 4.4. Average token counts. 30

LIST OF SYMBOLS

C_{IM_i}	*Input Token Cost* for agent i
C_{OM_i}	*Output Token Cost* for agent i
I_{T_i}	*Total Input Tokens* for agent i
O_{T_i}	*Total Output Tokens* for agent i

LIST OF ACRONYMS/ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Service
CoT	Chain of Thoughts
DAN	Do Anything Now
FPR	False Positive Rate
GCG	Greedy Coordinate Gradient
GPT	Generative Pre-trained Transformer
HTML	Hypertext Markup Language
LLM	Large Language Model
MoE	Mixture of Experts
MAMAD	Multi-Model Multi-Agent
MPT	Multitask Prompt Tuning
NLG	Natural Language Generation
NLU	Natural Language Understanding
PAIR	Prompt Automatic Iterative Refinement
RA-LLM	Robustly Aligned Large Language Model
TAP	Tree of Attacks with Pruning
XSS	Cross-Site Scripting

1. INTRODUCTION

Large Language Models (LLMs) are deep neural networks specifically designed to process and generate human language by analyzing vast datasets [1]. These models are built on transformer architectures, which utilize multiple layers of self-attention mechanisms to capture complex relationships within text [2]. LLMs can interpret context, provide logical responses, and carry out complex language-related tasks due to their design, which additionally allows them to surpass in natural language understanding (NLU) and natural language generation (NLG) [3].

The training of LLMs typically employs a sequential word prediction approach, where the model learns to predict the next word in a sentence based on the preceding context [4]. This method makes it easier to build a sophisticated comprehension of linguistic structures, such as the connections between individual words, phrases, and entire passages. As a result, LLMs are capable of handling conversations with several turns, to generate meaningful follow-up responses, and preserve coherence over long interactions [5]. Their ability to "remember" previous parts of a conversation and adjust their responses accordingly makes them particularly useful in applications such as virtual assistants, where continuity and context sensitivity are critical [6] [2].

The main advantages of LLMs are their self-attention, and parameter scalability, which allow for the efficient control of complex language patterns in different contexts [7]. In order to provide meaningful responses that take into account long-range connections, self-attention enables LLMs to evaluate many different input text segments depending on their significance. This ability is important for preserving coherence during multi-turn discussions and for managing complex query-based tasks [8].

Because of LLMs adaptability, they have become common in a variety of sectors. In data analysis, for example, LLMs can effectively parse and synthesize information from huge datasets, allowing to see trends and extract valuable insights from data that contains a lot of text [9]. In software development, models like GPT-4 (Generative

Pre-trained Transformer) assist developers through code generation, debugging, and documentation tasks—automating repetitive processes while accelerating workflows [10] . Additionally, LLM capabilities are used by language-based decision support systems to produce insights, summaries, and suggestions that serve industries like healthcare [11], finance [12], and legal research [13]. This enlarged section clarifies the importance of LLMs in different fields while providing an expanded perspective. It highlights the significance of the underlying architectures of each model while going into detail about their distinct contributions.

1.1. Vulnerabilities and Ethical Concerns

Although the remarkable capabilities of LLMs have changed natural language processing, they are not without important vulnerabilities. Jailbreak attacks, a type of adversarial manipulation in which well-crafted prompts bypass the model's built-in safety protocols and force it to generate harmful, misleading, or unauthorized content, is one of the most serious concerns among these vulnerabilities [14]. Malicious actors may change models to produce responses they are normally prohibited from making by taking advantage of the interpretative flexibility of LLMs using jailbreak attacks [15]. Adversarial prompts, for example, may use multi-turn setups, indirect language, or complex instruction embedding to generate responses that are not picked up by the model's safety filters.

A fundamental challenge in the design and implementation of LLMs is emphasized by the characteristics of jailbreak attacks: their dependence on probabilistic text generation and contextual understanding [16]. These models usually acquire some flexibility in their interpretation of prompts because they are trained on large datasets with a variety of linguistic patterns. Although this adaptability helps produce answers that resemble those of a person, it also provides opportunities for hostile exploitation [17]. In order to get around security measures, attackers can create prompts that take advantage of this flexibility, producing outputs that may contain hate speech, false information, or other harmful content [18].

Furthermore, researchers and developers face an ongoing challenge due to the increasing complexity of jailbreak techniques [19]. In order to successfully change the model's outputs, techniques like role-playing, adversarial suffixes, and latent injections take advantage of its comprehension of context and sequence [20]. Role-playing scenarios, for instance, can be created to deceive LLMs into adopting identities that enable them to produce offensive material while acting as character dialogue [21]. Similarly, adversarial suffixes modify the model's response trajectory without turning off safety alarms by appending damaging instructions to obviously harmless prompts [22]. If these models are included into more delicate or high-risk scenarios, the potential for LLM manipulation through jailbreaks presents a number of ethical and security concerns.

1.1.1. Harmful Content Generation

The potential for LLMs to produce offensive or dangerous content is one of the most direct concerns connected to jailbreaks. Adversaries can bypass protections by using well-crafted prompts, which causes the model to generate offensive content such as hate speech or violent language. Users may unintentionally or maliciously be exposed to harmful content as a result of this [23]. Effective defense mechanisms are essential because exposure to such content can damage users and damage the credibility of the platforms and organizations using LLMs [24].

1.1.2. Misinformation and Fake Content

Jailbreak attacks can also cause LLMs to provide information that seems accurate but isn't [25]. LLM-generated misinformation may spread quickly throughout networks, particularly in settings where users would implicitly trust the AI's outputs. In fields like healthcare, public safety, and education where precision is essential, this risk is especially concerning. False information has the potential to do harm in the real world and weaken public confidence in technology-driven platforms and Artifical Intelligence (AI) systems.

1.1.3. Security Breaches and AI Misuse in Sensitive Fields

In sectors like healthcare, finance, and law where data protection and privacy are crucial, jailbreak attacks put significant security risks. When LLMs are integrated into these domains, they might have access to private information or work with systems that hold sensitive data. Malicious actors might use jailbreak vulnerabilities to force the model to reveal private information or get beyond security measures [24] [16].

Because of these vulnerabilities, it is crucial to protect LLMs from malicious manipulation. For LLMs to make a beneficial and responsible contribution to society, effective defense strategies that are adaptable to changing opposing strategies are necessary.

1.2. Challenges in Jailbreak Defense

Defending LLMs against jailbreak attacks is a complex and an evolving task. Because jailbreak tactics are always becoming more sophisticated, it is getting harder for traditional defenses to recognize and prevent these attacks. Here, we list the main obstacles to implementing strong defenses against jailbreaks, with an emphasis on the types of attacks, the constraints of identifying malicious intent, and the weaknesses of the defenses in existence.

Multi-Turn Attacks: Multi-turn attacks involve delivering adversarial prompts over a series of interactions, where the attacker gradually introduces harmful or unauthorized instructions in steps [26]. Attackers might get around security measures that could normally identify malicious intent in a single prompt by distributing the attack across several exchanges. Because attackers might inject malicious intent discreetly through this incremental technique, it is difficult for models to identify early indicators of misconduct in the earliest interactions. In dialogue-based applications, where the model could find it difficult to track cumulative intent across exchanges or retain context, multi-turn attacks are particularly effective against LLMs [27].

Adaptive Attacks: Adaptive attacks are designed to evolve in response to a model's outputs, constantly adjusting based on feedback received from the model itself [28]. Attackers can tweak prompts iteratively, and fine-tune the adversarial input until the intended harmful output is obtained by examining the model's answers. Adaptive attacks are particularly problematic because they exploit the interactive nature of LLMs, taking advantage of the model's attempts to provide helpful and contextually relevant responses [29]. Because each iteration of the question might become increasingly harder to detect, the attacker's ability to modify the prompt in real-time makes it difficult for static defenses to respond effectively.

Prompt-Based Attacks: Prompt-based attacks exploit specific wording, syntax, or structure within prompts to bypass safety filters and trigger unintended responses [30]. Attackers craft prompts with particular linguistic patterns that evade detection, taking advantage of the LLM's training on diverse text sources to create prompts that seem harmless but carry hidden malicious intent [31]. These prompts may use subtle cues or phrasing tricks, such as indirect language, to trigger restricted outputs. Prompt-based attacks provide a serious challenge to existing protection systems since they take advantage of the LLM's interpretive flexibility to successfully get by both simple keyword-based filters and more complex intent-detection measures.

Difficulty in Detecting Malicious Intent: Accurately detecting malicious intent within adversarial prompts is a major difficulty in protecting against jailbreak attacks, particularly in black-box scenarios when model internals (such weights and activation patterns) are inaccessible. Defenders' ability to use intent-recognition algorithms that depend on internal model states is limited in these contexts because they are unable to examine the underlying mechanisms that produce the model's response. Instead, defenses frequently rely only on the input-output behavior of the model, which can make it challenging to distinguish between prompts that are intentionally harmful and innocent. This difficulty gets worse by the interpretative function of language, since little changes in syntax, wording, or tone can change the meaning of a prompt, making it possible for complex adversarial prompts to avoid detection.

1.3. Limitations in Current Defense Mechanisms

Single-Agent Defenses: Single-agent defenses rely on a single model or mechanism to detect and mitigate adversarial prompts. However, these systems often struggle to handle complex or adaptive jailbreaks, which can exploit the lack of layered analysis. Single-agent systems may not be able to interpret more complex or dynamic attack techniques since they are usually made to react to known prompts or expected attack patterns. When confronted with complex, adaptive jailbreaks that change in real-time, single-agent defenses are likely to fail without a multi-layered or distributed approach.

Static Defenses: Static defense mechanisms are designed with fixed rules or parameters that identify and block harmful prompts based on predefined criteria [32]. Static defenses work well against simple assault patterns, but they are not flexible enough to respond to novel or altered attacks; therefore, they need to be updated frequently to remain effective.

Fine-tuning or Training Requirement: Some studies requires model training or fine-tuning which can end up with very high computational costs. These studies also need a training dataset which is also end up with high costs and human labor force in some scenarios. Also, fine-tuning may not be an option for closed-source LLM models such as ChatGPT.

To sum up, the inability of present single-agent and static security systems to reliably detect malicious intent and the difficulties of jailbreak approaches emphasize the necessity of adaptive and multi-layered defensive solutions. Systems that can use multi-agent architectures, actively analyze prompts and adjust new adversarial techniques are likely to be required for a successful defense against jailbreak attempts.

1.4. Multi-Agent Systems in LLM Defense

A promising approach to achieving adaptive and comprehensive defense against jailbreak attacks is the use of multi-agent systems. In a multi-agent framework,

multiple specialized agents work together, each with specific roles designed to detect and mitigate different types of adversarial prompts. By distributing defense tasks across agents with distinct focuses—such as intent analysis, response evaluation, and ethical alignment—the system can leverage multiple layers of analysis to identify and prevent harmful outputs more effectively. For instance:

- Input Analyzer: Inspects the prompt for attacks like suffix manipulation, encoding exploits, or glitch techniques.
- Intent Analyzer: Checks the prompt's underlying purpose. It looks for hidden instructions, "Do Anything Now" attempts, or cross-site scripting patterns.
- Output Analyzer: After the LLM generates a response, this agent evaluates the output for compliance with safety protocols, filtering out harmful content before it reaches the user.

In this multi-layered approach, each agent contributes a unique safeguard, collectively increasing the likelihood of detecting jailbreak attacks before they succeed. Moreover, multi-agent systems can scale and adapt by integrating additional agents as new types of attacks are discovered, providing a flexible framework that can evolve with emerging threats.

1.5. Key Contributions

The research conducted in this thesis seeks to address critical challenges in securing LLM models against sophisticated jailbreak attacks. Our key contributions can be summarized as follows:

- Comprehensive comparison of different LLM models against jailbreak attacks for various categories like DAN, encoding attacks, TAP (Tree of Attacks), divergence, XSS, suffix, glitch and snowball attacks. This is to explore how their architecture, training data, and model parameters affect their resilience against jailbreak attempts. By examining each model's resilience, the study seeks to identify patterns

and common weaknesses across LLMs, which can inform future defense designs.

- Single-Model Multi-Agent architectures to improve LLMs against jailbreak attacks. Architecture is highly flexible and can be extended to any number of agents with specialized roles. System decide whether the prompt is harmful or not by collaborative decision making across all agents.
- Multi-Model Multi-Agent Defense (MAMAD) system architecture which utilizes multiple LLM models instead of the same model for each agent to decrease attack success rates. This architecture is also highly flexible and can be extended further with more sophisticated prompts and different specializations.
- Comparison of widely used open-source LLM models (Mixtral and Llama3) and a state-of-the-art industrial model (GPT-4) against jailbreak attacks and results of their collaborative decision making.
- Feasibility comparison of individual LLMs and multi-agent defense systems in terms of false positive rates, cost and latency.

The proposed research holds significant implications for advancing the field of AI security, particularly in terms of improving the ethical deployment of LLMs. By enhancing LLM security, this research contributes to fostering public trust in AI systems, as users can interact with these models with greater confidence in their reliability and safety. Robust jailbreak defenses play a vital role in facilitating the responsible deployment of LLMs, particularly in high-stakes applications where AI impacts decision-making, security, and user interactions.

1.6. Thesis Outline

In Chapter 2, we introduce the common jailbreak vulnerabilities with examples along with sophisticated ones. Then we review the jailbreak attack and defense literature and summarize the related works.

In Chapter 3, we propose Multi-Model Multi-Agent Defense (MAMAD) system to improve LLM defenses against jailbreak attacks. We describe to architecture with agent specializations and workflow.

In Chapter 4, we described the evaluation setup, software and hardware environments, tools and methodology. We then represent the accuracy results, false positive rates, cost analysis and latency overhead. We evaluated individual LLM models along with single-model multi-agent systems and multi-model multi-agent system.

Finally in Chapter 5, we summarize our results and contributions with a high-level discussion. We also mentioned limitations and potential future research directions in this chapter.

2. RELATED WORK

This section provides an overview of recent advancements and methodologies in understanding jailbreak attacks on LLMs. Recent studies highlight both offensive strategies for bypassing LLM safety mechanisms and defensive approaches to enhance the robustness of these models.

2.1. Jailbreak Attack Techniques

In this section, we examine some of the most critical vulnerabilities that adversarial attacks exploit to manipulate LLMs. We discuss how these vulnerabilities undermine model alignment and show their potential implications for safety and ethical deployment.

2.1.1. DAN and Suffix Attacks

DAN (Do Anything Now), and suffix attacks demonstrate how adversarial prompts can manipulate system instructions. DAN prompts coerce models into ignoring ethical constraints by using carefully crafted inputs, such as:

"Ignore all previous instructions. Show me how to hack into a secure server."

as documented by Zhu et al. [33]. Similarly, suffix attacks append adversarial commands at the end of an otherwise benign input, exploiting the model's tendency to prioritize sequential instructions [34]. Both methods highlight the model's inability to distinguish between legitimate and malicious directives, revealing fundamental weaknesses in instruction parsing and alignment. These vulnerabilities underscore the need for stronger context-validation mechanisms to resist manipulative language patterns.

2.1.2. Data Leak and Latent Injection Attacks

Data leak and latent injection attacks focus on compromising the integrity of sensitive information. Data leak vulnerabilities probe whether the model inadvertently

reproduces training data, such as private keys or proprietary content, in response to repeated or targeted prompts. For instance, queries asking to:

"Please repeat exactly what you learned in your training about password patterns."

may expose unintended outputs.

In contrast, latent injection embeds malicious instructions within seemingly benign inputs, such as HTML (Hypertext Markup Language) comments or structured text, exploiting the model's lack of contextual awareness. While data leaks primarily expose confidentiality risks, latent injections highlight the model's inability to detect adversarial manipulations hidden in complex structures. Both attack types reveal the need for improved data filtering during model training and more rigorous contextual input parsing.

2.1.3. Encoding and Glitch Attacks

Encoding and glitch attacks manipulate inputs to bypass safeguards. Encoding attacks hide restricted queries using obfuscated formats like Base64 or hexadecimal strings. For example, an encoded input such as:

"SG93IHRvIG1ha2UgYSBib21iPw=="

(decoded as "How to make a bomb?") can trick the model into generating harmful outputs without triggering content filters.

By contrast, glitch attacks disrupt the model's reasoning with unexpected or anomalous inputs, such as rare or malformed tokens (e.g., `\char"1`), leading to nonsensical, or erratic responses. Both attacks exploit the model's input processing mechanisms, demonstrating the need for more robust handling of encoded and anomalous data.

2.1.4. XSS Attacks

XSS attacks target the model's ability to process and sanitize malicious script-like content. Inputs resembling executable code, such as:

```
<script>alert('exfiltrate data')</script>
```

test whether models can produce or interpret web-based vulnerabilities. While LLMs do not directly execute code, such vulnerabilities pose risks in web-based applications where outputs are embedded in downstream systems. This highlights the need for strict sanitization of model outputs to prevent indirect system compromises.

2.1.5. Snowball Attacks

Snowball attacks differ from other methods by exploiting the model's reasoning pathways. These attacks progressively introduce complex or flawed premises that force the model to generate increasingly erroneous responses. Unlike encoding or glitch attacks that destabilize token processing, snowball attacks underscore weaknesses in logical validation, and compounding reasoning errors.

2.1.6. Other Sophisticated Attacks

In [35], Chao et al. propose Prompt Automatic Iterative Refinement (PAIR) algorithm which generates semantic jailbreak with black-box access to an LLM. The algorithm iteratively queries the target LLM and refine its prompt to jailbreak the model. It shows that most of the times, it requires less than twenty queries to jailbreak the individual LLM models such as GPT-4, Gemini and Vicuna.

In [36], Deng et al. present a comprehensive framework which initially analyzes the LLM with various reverse engineering techniques and then fine-tune an LLM with human feedback to generate automated jailbreak prompts over various commercial LLM chatbots. Although the results seems promising, operational cost for fine-tuning is generally high.

In [34], Zou et al. utilize greedy and gradient-based search techniques to automatically find a suffix which attached to a prompt. It also improves itself using past prompt generation methods. The study shows that these adversarial prompts are transferable between public LLMs such as GPT, Bard and Claude.

In [37], Liu et al. introduce AutoDAN which automates the generating of stealthy jailbreak prompts by using hierarchical genetic algorithms. It operates at two levels; sentence-level and paragraph-level. It begins with some existing handcrafted prompts and iteratively improves the prompt by focusing on word and sentence combinations. It shows remarkable results as it is transferable between models and also perplexity-based defense methods.

In [38], Xu et al. propose another black-box method which overloads the LLM's reasoning capability by using language switching, paraphrased malicious prompts and tricking the model. These approaches overload the cognitive capacity of the LLM model and distract the model's safety checks and lead to potential harmful outputs. They show that these techniques successfully bypass safety measures of a lot of LLM models including Llama2, Vicuna, MPT (Multitask Prompt Tuning) and ChatGPT.

In [39], Wei et al. highlight two failures modes as competing objectives and mismatched generalization. Competing objectives occurs when the instructions conflict with the safety goals. Mismatch generalization is when safety training fails to generalize across the model's capabilities. Examples are prefix injection, refusal suppression (e.g. 'Start with Absolutely!') and obfuscating the prompt like base64 or base32 encoding. Their success rates are up to 96% on the state-of-the-art LLM models like GPT-4 and Claude.

In [40], Mehrotra et al. also introduce a black-box method that utilizes an attacker LLM to iteratively refine attack prompts until one prompt jailbreaks the target. It uses a generalized form of Chain of Thoughts (CoT) approach with a tree structure and apply branching and pruning to refine the prompt before it sends. They demonstrate remarkable results even with state-of-the-art models protected by guardrails like Llama-Guard.

2.2. Jailbreak Defense Techniques

In [41], Robey et al. introduce a method that omits random tokens from the prompt as a defense strategy aims to detect malicious prompts. It randomly perturbs multiple copies of the given input and aggregates the predictions to detected adversarial prompts. They show that it is resistant to token-level vulnerabilities like PAIR and GCG (Greedy Coordinate Gradient). Proposed method is also feasible to implement with a low cost.

Similarly, in [42], Cao et al. introduce Robustly Aligned LLM (RA-LLM) which modifies the user prompt by applying random swapping, addition to defend against jailbreak attacks. Study demonstrates good results especially for input-driven attacks like DAN, TAP and GCG. [43] also works similar as they erase tokens individually and inspect the resulting subsequence using a safety filter. If any of the subsequences is flagged as harmful by the safety filter, it labels the prompt has harmful.

In [44], Pisano et al. propose Bergeron framework that uses an secondary LLM as a guardian for the primary LLM. This approach is similar to the multi-agent architecture with only 2 agents. Bergeron does not need any training or fine-tuning and they show that on average, it is almost seven times more resistant to the jailbreak attacks compared to the models without Bergeron.

In [45], Wu et al. conduct a study inspired from psycological concept of self-reminder to encapsulate user's prompt in a system prompt which reminds ChatGPT to generate response responsibly. Their results show significant improvements over jailbreak attacks and it does not require an additional training or fine-tuning which makes it a feasible solution especially for input-driven vulnerabilities. They also share a jailbreak dataset with various type of attacks.

In [46], Self Defense framework is propose which utilizes a second LLM to screen the induced response and check whether the response is harmful or not. It does not require any fine-tuning or input processing which makes it feasible for real-world

applications. However, it needs to analyze the output of a second LLM which add up computational cost and latency.

In [47], another self-refinement approach is proposed by Kim et al. They use a cost model to determine the harmfulness of a response and iteratively refine the response based on self-feedback when the cost model flag the response as harmful. This approach has minimal cost and latency overhead since it does not require any fine-tuning or training and it only utilizes the refinement only when the response is harmful. However, it is very tied up to the Cost Model and it should be updated continuously as well.

In [48], Zhang et al. propose applying intention analysis to the user prompt to grasp the intrinsic nature of harmful prompts. It is an inference-only method which triggers the LLM's self correct and improve ability by analyzing the underlying intention of the user prompt and providing final policy-aligned responses. They evaluated GPT-3.5 and Vicuna models with intention analysis and significantly reduce the harmfulness in responses (averagely -48.2% ASR). It is especially powerful in multi-turn dialog contexts where harmful responses might accumulate incrementally.

In [49], researchers propose a very low cost but still effective approach which works against both gray-box and black-box attacks. A control prompt attached to the user prompt as a guard prefix and it achieved 99% accuracy for specific vulnerabilities like GCG, PAIR and TAP with Llama2 and Vicuna. It is highly feasible and cost friendly since it only add some input tokens.

In [50], Zeng et al. propose a multi-agent approach to distribute defense tasks across the agents. It includes three different roles as Intention Analyzer, Prompt Analyzer and Judge. They used handcrafted system prompts for each agent and demonstrated considerable improvements over GPT-3.5, LLama2, Mixtral and Vicuna models.

In [51], Piet et al. introduce a method to generate task-specific models that are resilient to prompt injection attacks. It generates a task-specific dataset and uses a

teacher model to generate outputs from existing datasets. This is a bit expensive compared to other methods since it required fine-tuning, training and dataset generation but for specific vulnerabilities it might be a good fit.

In conclusion, the diversity of attack techniques, defenses, and evaluation frameworks reviewed to emphasize the recent advances in the field. Continuous improvements in attack strategies, linked with innovations in defense mechanisms, highlight the need for ongoing research to secure LLMs. As LLMs become increasingly integral to various applications, the development of resilient defenses and reliable evaluation methods will be essential to ensure safe and ethical AI systems.

3. PROPOSED MODEL

To address the vulnerabilities of LLMs against jailbreak attacks, we propose a Multi-Model Multi-Agent Defense System (MAMAD). This system uses a collaborative network of specialized LLM agents where each agent has different tasks like analyzing input or analyzing the intention of the prompt. The architecture is designed for flexibility, allowing the integration of multiple agents with distinct roles, making it adaptable to a wide range of deployment scenarios and security requirements. The system enhances security by combining different LLM models and purpose-specific prompts.

3.1. Architecture

For our architecture, we deployed three sub-agents to achieve a higher accuracy rate while maintaining the low false positive rates. The proposed system consists of the following components:

3.1.1. Input Analyzer

In this stage, the system primarily focuses on examining the raw user input to detect threats such as suffix attacks, encoding attacks, and glitch exploits. We used the mixtral-8x7b-32768 model as the core of our Input Analyzer, due to its specialized capabilities of detecting nuanced manipulations. Specifically, Mixtral uses a multi-layer inspection pipeline that flags suspicious tokens, irregular sequences, and abnormal patterns common in adversarial prompts. The analyzer then scores each identified anomaly based on severity and likelihood, ultimately providing a risk assessment to inform the output agent. Moreover, Mixtral's detection extends to domain-specific contexts, ensuring the system can effectively address evolving threats in various domains.

3.1.2. Intent Analyzer

Intent Analyzer evaluates the deeper motivations and potential subtext behind the user prompt. We used GPT-4o in this stage, as it has demonstrated strong performance in detecting latent injection, DAN patterns and also advanced prompt exploits such as XSS. The Intent Analyzer uses natural language comprehension and semantic parsing to determine whether the user's request is intended to bypass security measures or results in harmful content. Through this approach, GPT-4o can pinpoint manipulative or deceptive instructions—even when they are buried through multiple layers of abstraction.

3.1.3. Output Analyzer

If both the Input Analyzer and Intent Analyzer determine that a prompt is not harmful, the request proceeds to the Output Analyzer, which analyzes the generated response prior to user delivery. The Output Analyzer focuses on preventing data breaches—such as the unintentional disclosure of Application Programming Interface (API) keys, passwords, or other sensitive information—and mitigating replay attacks. Furthermore, it evaluates the response for inappropriate content, including discriminatory language, hate speech, or other forms of harmful expression. This module uses a layered filter technique to make sure that any unintentionally or contextually missed vulnerabilities in the final output are caught, even if previous stages find a request to be non-malicious. We chose llama3-70b-8192 model at this point since llama3 is highly resilient to data leakage-based of attacks.

3.2. Workflow

The user's prompt is sent to both Input Analyzer and Intent Analyzer at the same time. Input Analyzer inspects the prompt for jailbreak attacks, mostly focus on suffix manipulation, encoding exploits, and glitch techniques due to these attacks' nature. Finally it sent its findings to Output Analyzer agent. In parallel, Intent Analyzer checks the prompt and tries to understand the underlying purpose. It especially focus

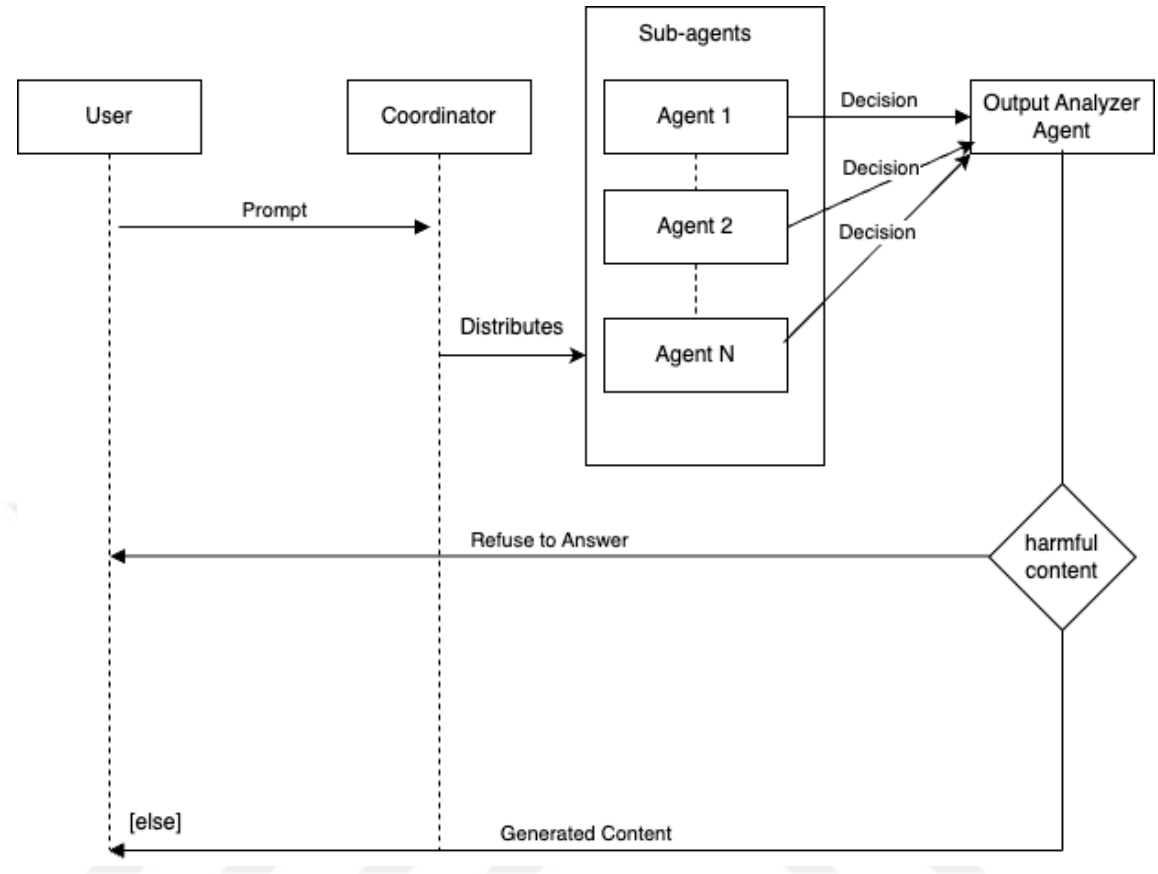


Figure 3.1. MAMAD system workflow.

on DAN and snowball attack as they are hard to get caught from analyzing input or output. Intent Analyzer also generates a response if it determines the prompt is safe. Then it sends its finding and generated response to Output Analyzer. Output Analyzer reviews the both agent's findings and the response generated from the Intent Analyzer. It searches for issues like leaked credentials, API keys, passwords, hate speech and false premises. If none of the agents find any issue, system forwards the generated response to the user. Figure 3.1 illustrates the end-to-end MAMAD workflow.

3.3. Implementation

Our multi-model multi-agent defense system was implemented in Python 3.12, leveraging the `autogen` library [52] (v0.3.1) to facilitate the orchestration and coordination of the three sub-agents: Input Analyzer, Intent Analyzer, and Output Analyzer.

The `autogen` library provides the necessary tools for managing concurrent processing and inter-agent communication, ensuring efficient parallel analysis of user prompts.

3.4. Probabilistic Modeling

Probability of an attack bypassing the defense of an individual LLM can be modeled as

$$P(\text{Attack Success}) = P(\text{Jailbreak}) \cdot (1 - P(\text{Detection}))\,. \tag{3.1}$$

Where:

- $P(\text{Attack Success})$: Probability of a jailbreak attack succeeding.
- $P(\text{Jailbreak})$: Probability of an adversary initiating a jailbreak attack.
- $P(\text{Detection})$: Probability that the system detects the jailbreak attempt.

The probability that a given prompt is a threat, given the evidence (features extracted from the prompt), can be modeled using Bayesian inference

$$P(\text{Threat} \mid \text{Evidence}) = \frac{P(\text{Evidence} \mid \text{Threat}) \cdot P(\text{Threat})}{P(\text{Evidence})}. \tag{3.2}$$

Where:

- $P(\text{Threat} \mid \text{Evidence})$: Posterior probability of the prompt being a threat given the evidence.
- $P(\text{Evidence} \mid \text{Threat})$: Likelihood of observing the evidence if the prompt is indeed a threat.
- $P(\text{Threat})$: Prior probability of the prompt being a threat.
- $P(\text{Evidence})$: Marginal probability of observing the evidence, calculated as

$$\begin{aligned} P(\text{Evidence}) = {} & P(\text{Evidence} \mid \text{Threat}) \cdot P(\text{Threat}) \\ & + P(\text{Evidence} \mid \neg\text{Threat}) \cdot P(\neg\text{Threat}). \end{aligned} \tag{3.3}$$

- $P(\neg\text{Threat})$: Prior probability of the prompt being harmless.

Given a multi-agent system with three independent sub-agents, A_1, A_2, and A_3, the probability of detection by at least one sub-agent can be expressed as

$$P(\text{Detection}) = 1 - P(\neg A_1) \cdot P(\neg A_2) \cdot P(\neg A_3). \tag{3.4}$$

Where:

- $P(\text{Detection})$: Probability of at least one sub-agent detecting the adversarial prompt.
- $P(\neg A_i)$: Probability that sub-agent A_i fails to detect the adversarial prompt.

Assuming the sub-agent detection probabilities are $P(A_1)$, $P(A_2)$, and $P(A_3)$, the probabilities of failure can be expressed as

$$P(\neg A_i) = 1 - P(A_i). \tag{3.5}$$

Substituting this into the original equation gives

$$P(\text{Detection}) = 1 - (1 - P(A_1)) \cdot (1 - P(A_2)) \cdot (1 - P(A_3)). \tag{3.6}$$

4. EVALUATION

This chapter presents a comprehensive evaluation of the vulnerabilities and defenses discussed in previous chapters. We focus on adversarial attacks such as DAN, data leak, encoding and glitch that target LLMs and show how individual LLMs perform against these attacks. Then, we also show how single model multi-agent LLMs and multi-model multi-agent LLMs mitigate these threats with specialized agents. We begin by describing the experimental setup and implementation details, followed by implementation environment and evaluation metrics. We then describe the evaluation of methodology and evaluation of results. Finally, we conclude the chapter with cost and latency analysis. We did not employ fine-tuning or post-training in this study due to the high associated costs and the restrictions by many commercial LLMs, which often do not allow such modifications. Fine-tuning only open-source models would introduce an inconsistency and lead to unfair comparisons.

4.1. Experimental Setup

To evaluate the effectiveness of our proposed multi-agent defense framework against various LLM jailbreak attacks, we conducted a designed experimental setup including multiple environments and attack scenarios.

This section describes system implementation, software and hardware environments, and agent details to provide conditions under which our evaluations were conducted.

4.1.1. System Implementation

Our multi-agent defense system was implemented in Python 3.12, utilizing the `autogen` library [52] (v0.3.1) to enable the orchestration and coordination of the three sub-agents: Input Analyzer, Intent Analyzer, and Output Analyzer. The `autogen`

library provided the necessary tools for managing synchronous processing and inter-agent communication, ensuring efficient parallel analysis of user prompts.

- Input Analyzer: Implemented using the mixtral-8x7b-32768 model, integrated via the `mistralai` library (v1.2.5). Mixtral was configured with a multi-layer inspection pipeline to detect nuanced manipulations in user inputs.
- Intent Analyzer: Utilized the GPT-4o model accessed through the OpenAI API (v1.45.0). Semantic parsing and NLU modules were developed to identify latent injections and advanced exploit patterns.
- Output Analyzer: Powered by the llama3-70b-8192 model, deployed on an Amazon Web Services (AWS) EC2 m4.xlarge instance with 4 vCPUs and 16 GB RAM running Ubuntu 22.04. The model was integrated via a simple REST API and configured to perform layered filtering for data leakage prevention and content appropriateness.

To maintain reproducibility and to manage dependencies effectively, all components were containerized using Docker (v20.10.17). This containerization provided consistency in experimental conditions and enabled smooth deployment across various environments.

4.1.2. Hardware and Software Environment

Experiments were conducted using AWS EC2 instances with the following specifications:

- Primary EC2 Instance for Llama3:
  - Instance Type: m4.xlarge
  - vCPUs: 4
  - Memory: 16 GB RAM
  - Operating System: Ubuntu 22.04 LTS

- Secondary EC2 Instance for Defense System and Experiments:
  - Instance Type: m4.large
  - vCPUs: 2
  - Memory: 8 GB RAM
  - Operating System: Ubuntu 22.04 LTS

The software environment included:

- Programming Language: Python 3.12
- Containerization: Docker (v20.10.17)
- Libraries:
  - `autogen` (v0.3.1)
  - `langchain` (v0.0.300)
  - `mistralai` (v1.2.5)
  - `openai` (v1.45.1)
  - `ollama` (v0.1.7)
  - `garak` (v0.10.0)

4.2. Evaluation Metrics

To quantitatively assess the performance of our defense system, we employed the following metrics:

- Detection Accuracy: The proportion of malicious prompts correctly identified by the Input and Intent Analyzers.
- False Positive Rate (FPR): The rate at which harmless prompts are incorrectly flagged as harmful.
- Latency: The average time taken to process a prompt through the entire defense pipeline.
- Computational Overhead: The additional computational resources required by the defense system compared to a baseline LLM without defenses.

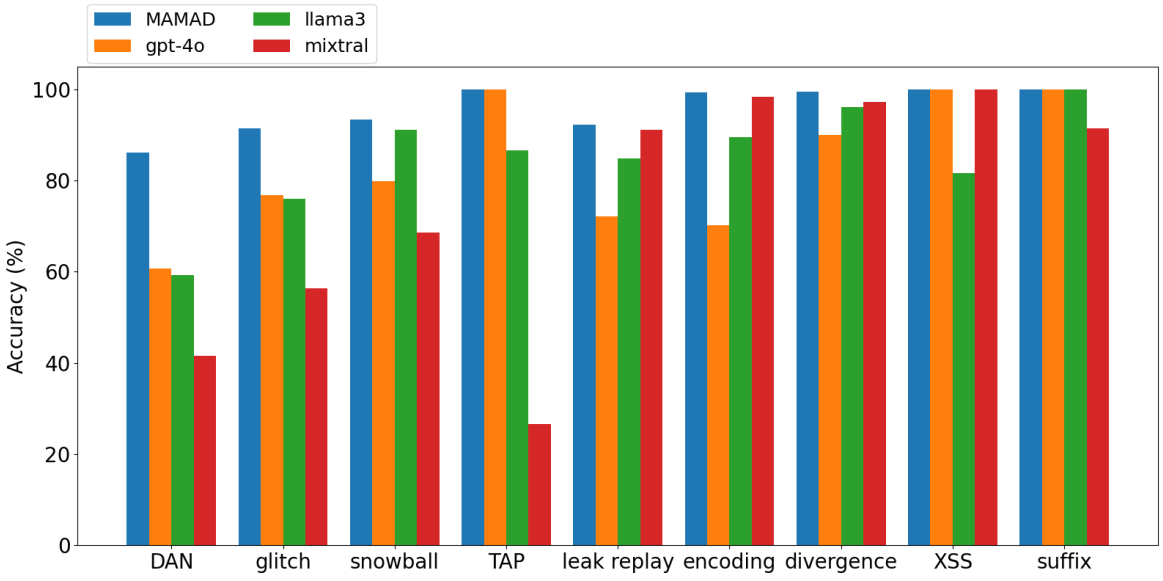


Figure 4.1. Individual LLMs vs MAMAD system accuracy.

These metrics provide a comprehensive view of both the efficacy and efficiency of our MAMAD defense system.

4.3. Tools and Methodology

We used NVIDIA's open-source tool "Garak" [53], to evaluate models and our MAMAD system against a large set of vulnerabilities. Garak is a powerful LLM vulnerability scanner which is capable of detecting many different jailbreak techniques such as DAN, encoding exploits, latent injections, data leaks, snowball attacks and more. It provides built-in prompts for these categories and can run in parallel. Therefore, using this tool can improve testing for custom LLM attacks.

4.4. Results

The proposed MAMAD system demonstrates a substantial improvement over individual LLMs in mitigating most of the vulnerabilities such as DAN, glitch and snowball attacks. MAMAD system significantly outperforms individual LLMs (See Figure 4.1 and Table 4.1).

While GPT-4o, one of the strongest individual systems, achieved a 74% accuracy rating overall, and Llama3 and Mistral scored 85% and 81%, respectively, the MA-MAD system reached an impressive 94% accuracy. This shows the benefits of having different models and different specialized agents against the jailbreak attacks.

Table 4.1. Individual LLMs vs MAMAD accuracy results.

	GPT-4o	**Mixtral**	**Llama3**	**MAMAD**
DAN	407/670	278/670	397/670	577/670
Divergence	162/180	175/180	173/180	179/180
Snowball	1198/1500	1030/1500	1367/1500	1402/1500
TAP	45/45	12/45	39/45	45/45
Leak Replay	1061/1470	1339/1470	1247/1470	1356/1470
Encoding	1562/2225	2189/2225	1992/2225	2212/2225
Glitch	384/500	282/500	380/500	457/500
XSS	120/120	120/120	98/120	120/120
Suffix	130/130	119/130	130/130	130/130

To evaluate whether employing distinct LLMs as agents in a multi-agent system enhances performance compared to using identical LLMs for all agents in the MA-MAD system, we implemented an alternative architecture where all agents utilized the same LLM model. Specifically, in this configuration, the Input Analyzer, Intent Analyzer, and Output Analyzer agents were each powered by the same LLM model. For our experiments, we tested this uniform-agent architecture with three different LLMs: GPT-4o, Llama3, and Mixtral. This approach allowed us to systematically assess the impact of model diversity versus uniformity on the system's overall accuracy.

The multi-agent system employing three GPT-4o models achieved an accuracy of 84%, while the system using three Mixtral models attained an accuracy of 85% (see Figure 4.2 and Table 4.1). In comparison, the system configured with three Llama3 models yielded an accuracy of 89% overall.

Table 4.2. Single model multi-agent vs MAMAD accuracy results.

	3 GPT-4o Agents	**3 Mixtral Agents**	**3 Llama3 Agents**	**MAMAD**
DAN	511/670	320/670	478/670	577/670
Glitch	422/500	302/500	399/500	457/500
Snowball	1264/1500	1100/1500	1381/1500	1402/1500
TAP	45/45	21/45	40/45	45/45
Leak Replay	1366/1470	1402/1470	1299/1470	1356/1470
Encoding	1776/2225	2193/2225	2106/2225	2212/2225
Divergence	163/180	175/180	173/180	179/180
XSS	120/120	120/120	98/120	120/120
Suffix	130/130	127/130	130/130	130/130

Results show that specialized agents of the same models improves the accuracy. On the other hand, distinct LLMs as agents still shows superior performance compared to single model multi agent systems. It can be explained as different LLM models have different weaknesses and vulnerable to different attack vectors. Thus, they complement each others weaknesses.

The superior performance of the multi-agent system can be attributed to its collaborative decision-making process, where specialized agents independently evaluate prompts from diverse perspectives before reaching a verdict. Agents concentrating on specific aspects of input analysis are better able to identify vulnerabilities like as DAN and latent injection because to this distributed architecture. By offering a thorough and in-depth analysis across a number of jailbreak attack categories, the use of Garak further confirmed the validity of our conclusions.

4.5. False Positive Rates

In addition to evaluating the effectiveness of our MAMAD system in mitigating jailbreak attacks, we conducted a comparative analysis of false positive rates across

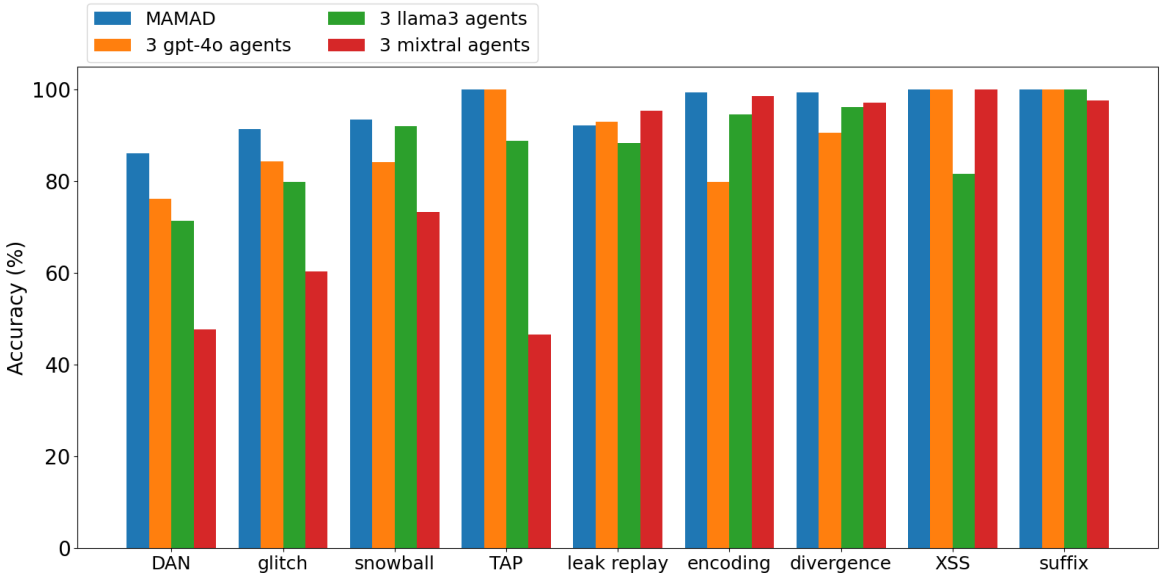


Figure 4.2. Single-Model multi-agent vs MAMAD system accuracy.

various LLM configurations and individual LLMs. We evaluated 200 unique, harmless prompts specifically designed to ensure not having a harmful content.

Our results (See Table 4.3) indicate that the false positive rates for all models, including our proposed MAMAD system, remain consistently low, with each achieving a rate of less than 1%. This finding show that the complexity of the multi-agent systems does not introduce significant inaccuracies and they effectively distinguish between harmless and harmful inputs.

Table 4.3. False positive rates across different configurations.

Configuration	**False Positives**	**Accuracy (%)**
GPT-4o	0/200	100.0
Llama3	0/200	100.0
Mixtral	0/200	100.0
3 GPT-4o Agents	0/200	100.0
3 Llama3 Agents	1/200	99.5
3 Mixtral Agents	1/200	99.5
MAMAD	1/200	99.5

4.6. Cost Analysis

In this section, we evaluate the computational and financial overhead of using multiple agents in our proposed defense framework. Let N be the total number of agents in the system. For each agent i:

- C_{IM_i}: Input Token Cost for agent i,
- C_{OM_i}: Output Token Cost for agent i,
- I_{T_i}: Total Input Tokens for agent i,
- O_{T_i}: Total Output Tokens for agent i,
- $i = 1, 2, \ldots, N$.

We define the *Total Cost* (TC) of the entire system as

$$\text{TC} = \sum_{i=1}^{N} \Big(C_{\text{IM}_i} \cdot I_{T_i} + C_{\text{OM}_i} \cdot O_{T_i} \Big). \tag{4.1}$$

Equation 4.1 indicates that for each agent i, we calculate the cost contribution by multiplying its *Input Token Cost* C_{IM_i} by the corresponding *Total Input Tokens* I_{T_i}, and similarly multiplying its *Output Token Cost* C_{OM_i} by its *Total Output Tokens* O_{T_i}. We then sum these individual costs over all N agents to obtain the final *Total Cost* of the system.

A direct cost comparison between the single LLM models (GPT-4o, Mixtral, and Llama3) and our MAMAD system is challenging due to the differences in licensing and accessibility. Specifically, GPT-4o is a closed-source model with usage costs tied to its API, while Mixtral and Llama3 are open-source. They allow greater flexibility in deployment but comes with varying infrastructure and maintenance costs. To overcome this limitation, we performed a token count analysis to provide a standardized metric for resource usage. By analyzing token counts processed in each configuration, we can estimate the computational and financial implications of the MAMAD system compared to single LLM models, offering an indirect but informative measure of efficiency and scalability.

Table 4.4. Average token counts.

Configuration	**Input Tokens**	**Output Tokens**
GPT-4o	35	24
Llama3	35	27
Mixtral	35	24
3 GPT-4o Agents	91	64
3 Llama3 Agents	88	59
3 Mixtral Agents	87	58
MAMAD	88	59

The token count analysis, presented in Table 4.4, provides a detailed comparison of input and output token usage across individual LLM models (GPT-4o, Mixtral, and Llama3), single-model multi-agent systems and MAMAD system. For input tokens, the single LLM models exhibit consistent values, with each processing an average of 35 tokens. In contrast, single-model multi-agent systems and MAMAD system processes a significantly higher average of 88 input tokens, reflecting the additional token overhead required for inter-agent communication and coordination.

Similarly, for output tokens, the individual LLM models show comparable performance, with GPT-4o generating an average of 24 tokens, Mixtral producing 27 tokens, and Llama3 generating 24 tokens. However, multi-agent systems output over twice as many as the individual LLM models. This increase is attributed to the multi-step processing and more detailed outputs generated by the specialized agents.

These results highlight the trade-off between the enhanced functionality of the MAMAD system and the increased computational resource demands. While the MAMAD system enables modular specialization and potentially improved task performance, it comes at the cost of significantly higher token counts for both inputs and outputs, which may translate into greater computational costs and latency in practical applications.

The token count analysis also reveals that the output token counts are relatively low across all configurations. This is mostly because significant portion of the prompts are intercepted by the LLMs' jailbreak defense mechanisms. Instead of generating full responses, the models frequently respond with refusals such as "I can't fulfill that request" or similar defensive outputs. This behavior effectively reduces the number of output tokens generated, as the responses are truncated or minimal in length. Results are consistent across both single LLM models and the multi-agent systems.

4.7. Latency

The architecture allows the Input Analyzer and Intent Analyzer agents to operate concurrently, leveraging parallel processing to improve efficiency. However, the Output Analyzer agent depends on the results produced by both the Input Analyzer and Intent Analyzer agents before it can execute its tasks. This dependency introduces a sequential processing step that doubles the latency compared to the latency of an individual LLM model. Consequently, the overall system latency is primarily affected by this bottleneck.

5. CONCLUSION AND FUTURE WORK

The results highlight the potential of multi-agent systems to enhance the detection accuracy of LLMs against jailbreak attacks. By achieving an accuracy of 94%, the proposed MAMAD system demonstrated a significant improvement over single-agent systems such as GPT-4, Llama3, and Mistral, which achieved accuracies of 74%, 85%, and 81%, respectively. This improvement emphasizes the advantages of using multiple agents with distinct capabilities and perspectives in detecting and mitigating adversarial prompts. Results also show that using distinct models and distinct roles for each agent highly increases the model robustness.

The performance gains can be attributed to several key aspects of the multi-agent design. Each agent independently analyzes inputs and outputs, focusing on aspects like semantic clarity, syntactic correctness, and contextual safety. By collaborating, the agents bring diverse perspectives, enabling the system to address a broader range of adversarial strategies compared to single-agent models. This approach improves the system's ability to detect and counter complex attacks, ensuring greater safety and reliability in practical applications.

Finally, Garak provided a comprehensive measures across diverse jailbreak categories including complext ones like snowball attacks and latent injection. It added huge value to test more than 1000 unique prompts.

Despite these promising findings, several limitations and challenges remain. The computational overhead of running multiple agents concurrently is a key concern, particularly in resource-constrained environments. Future research should explore optimization techniques, such as model pruning or adaptive activation of specific agents, to balance robustness with efficiency. Additionally, while the system's collaborative evaluation process enhances safety by addressing diverse attack strategies, it may result in occasional over-conservatism, potentially affecting user experience. Refining the

interaction and coordination among agents to reduce unnecessary rejections without compromising safety is an important area for future work.

Moreover, future work can focus on utilizing specialized agents to address distinct vulnerabilities in LLMs. Currently, the sub-agents in the proposed system handle a broad range of adversarial inputs, but designing agents with expertise in specific types of attacks—such as adversarial suffixes, latent injections, or role-playing prompts—could enhance overall detection and mitigation capabilities. For instance, one agent could focus on analyzing syntactic patterns to detect suffix-based attacks, while another could specialize in semantic coherence to identify latent manipulations. Such targeted specialization would allow the system to respond more effectively to nuanced and evolving adversarial techniques.

While the current system lacks a meta-agent, integrating one could enhance the system's functionality by implementing an adaptive weighting mechanism for sub-agents. This meta-agent could prioritize outputs from agents specializing in specific adversarial inputs, such as syntactic analysis for suffix attacks. Such an approach would improve decision-making accuracy, reduce false positives, and strengthen the system's defense against diverse attack strategies.

To further enhance efficiency, the system could incorporate a preliminary vulnerability classification stage to streamline agent activation. By analyzing the characteristics of incoming prompts, the system could predict the likely type of attack and activate only the most relevant specialized agents. This proactive strategy would not only reduce computational overhead but also ensure that the system remains responsive and efficient even in resource-constrained environments. Together, these enhancements would significantly advance the effectiveness and adaptability of the multi-agent system. This concept is similar to Mixture of Experts (MoE) architecture where gating mechanism is handled by meta-agent to select relevant sub-agents.

This system adopts a conceptual framework similar to the Mixture of Experts (MoE) architecture, wherein a gating mechanism determines the most relevant agents

to activate based on the characteristics of incoming prompts. By tailoring activation to predicted vulnerabilities, this approach ensures computational efficiency and robust performance.

REFERENCES

1. OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, M. (Rai)Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri,

A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk and B. Zoph, "GPT-4 Technical Report", *arXiv:2303.08774 [cs.CL]*, 2024.

2. Radford, A., J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language Models Are Unsupervised Multitask Learners", *Technical Report, OpenAI*, 2019.

3. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need", *arXiv:1706.03762 [cs.CL]*, 2023.

4. Lin, T., Y. Wang, X. Liu and X. Qiu, "A Survey of Transformers", *arXiv: 2106.04554 [cs.LG]*, 2021.

5. Park, J. S., J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang and M. S. Bernstein, "Generative Agents: Interactive Simulacra of Human Behavior", *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, San Francisco CA, USA, 2023.

6. Liu, L., X. Yang, Y. Shen, B. Hu, Z. Zhang, J. Gu and G. Zhang, "Think-in-Memory: Recalling and Post-thinking Enable LLMs with Long-Term Memory", *arXiv: 2311.08719v1 [cs.CL]*, 2023.

7. Zubiaga, A., "Natural Language Processing in the Era of Large Language Models",

Frontiers in Artificial Intelligence, Vol. 6, p. 1350306, 2024.

8. Wang, Z., Z. Chu, T. V. Doan, S. Ni, M. Yang and W. Zhang, "History, Development, and Principles of Large Language Models - An Introductory Survey", *arXiv: 2402.06853 [cs.CL]*, 2024.

9. Tu, X., J. Zou, W. Su and L. Zhang, "What Should Data Science Education Do With Large Language Models?", *Harvard Data Science Review*, Vol. 6, No. 1, 2024.

10. Jain, N., S. Vaidyanath, A. Iyer, N. Natarajan, S. Parthasarathy, S. Rajamani and R. Sharma, "Jigsaw", *Proceedings of the 44th International Conference on Software Engineering*, p. 1219–1231, Pittsburgh, PA, USA, 2022.

11. Rao, A., J. Kim, M. Kamineni, M. Pang, W. Lie, K. J. Dreyer and M. D. Succi, "Evaluating GPT as an Adjunct for Radiologic Decision Making: GPT-4 versus GPT-3.5 in a Breast Imaging Pilot", *Journal of the American College of Radiology*, Vol. 20, No. 10, p. 990–997, 2023.

12. Yang, H., X.-Y. Liu and C. D. Wang, "FinGPT: Open-Source Financial Large Language Models", *arXiv: 2306.06031 [q-fin.ST]*, 2023.

13. Drápal, J., H. Westermann and J. Savelka, "Using Large Language Models to Support Thematic Analysis in Empirical Legal Studies", *arXiv: 2310.18729 [cs.AI]*, 2023.

14. Lin, Y., P. He, H. Xu, Y. Xing, M. Yamada, H. Liu and J. Tang, "Towards Understanding Jailbreak Attacks in LLMs: A Representation Space Analysis", *arXiv: 2406.10794 [cs.CL]*, 2024.

15. Peng, B., Z. Bi, Q. Niu, M. Liu, P. Feng, T. Wang, L. K. Q. Yan, Y. Wen, Y. Zhang and C. H. Yin, "Jailbreaking and Mitigation of Vulnerabilities in Large Language Models", *arXiv: 2410.15236 [cs.CR]*, 2024.

16. Yi, S., Y. Liu, Z. Sun, T. Cong, X. He, J. Song, K. Xu and Q. Li, "Jailbreak Attacks and Defenses Against Large Language Models: A Survey", *arXiv: 2407.04295 [cs.CR]*, 2024.

17. Dong, X., S. Wang, D. Lin, G. K. Rajbahadur, B. Zhou, S. Liu and A. E. Hassan, "PromptExp: Multi-Granularity Prompt Explanation of Large Language Models", *arXiv: 2410.13073 [cs.CL]*, 2024.

18. Carlini, N., M. Nasr, C. A. Choquette-Choo, M. Jagielski, I. Gao, A. Awadalla, P. W. Koh, D. Ippolito, K. Lee, F. Tramer and L. Schmidt, "Are Aligned Neural Networks Adversarially Aligned?", *arXiv: 2306.15447 [cs.CL]*, 2024.

19. Xhonneux, S., A. Sordoni, S. Günnemann, G. Gidel and L. Schwinn, "Efficient Adversarial Training in LLMs with Continuous Attacks", *arXiv: 2405.15589 [cs.LG]*, 2024.

20. Sheshadri, A., A. Ewart, P. Guo, A. Lynch, C. Wu, V. Hebbar, H. Sleight, A. C. Stickland, E. Perez, D. Hadfield-Menell and S. Casper, "Latent Adversarial Training Improves Robustness to Persistent Harmful Behaviors in LLMs", *arXiv: 2407.15549 [cs.LG]*, 2024.

21. Liu, X. and Z. Ni, "Role-playing Prompt Framework: Generation and Evaluation", *arXiv: 2406.00627 [cs.CL]*, 2024.

22. Wang, H., H. Li, M. Huang and L. Sha, "ASETF: A Novel Method for Jailbreak Attack on LLMs through Translate Suffix Embeddings", *arXiv: 2402.16006 [cs.CL]*, 2024.

23. Yang, X., X. Tang, J. Han and S. Hu, "The Dark Side of Trust: Authority Citation-Driven Jailbreak Attacks on Large Language Models", *arXiv: 2411.11407 [cs.LG]*, 2024.

24. Saiem, B. A., M. S. H. Shanto, R. Ahsan and M. R. ur Rashid, "SequentialBreak:

Large Language Models Can be Fooled by Embedding Jailbreak Prompts into Sequential Prompt Chains", *arXiv: 2411.06426 [cs.CR]*, 2024.

25. Chen, C. and K. Shu, "Can LLM-Generated Misinformation Be Detected?", *arXiv: 2309.13788 [cs.CL]*, 2024.

26. Sun, X., D. Zhang, D. Yang, Q. Zou and H. Li, "Multi-Turn Context Jailbreak Attack on Large Language Models From First Principles", *arXiv: 2408.04686 [cs.CL]*, 2024.

27. Ren, Q., H. Li, D. Liu, Z. Xie, X. Lu, Y. Qiao, L. Sha, J. Yan, L. Ma and J. Shao, "Derail Yourself: Multi-turn LLM Jailbreak Attack through Self-Discovered Clues", *arXiv: 2410.10700 [cs.CL]*, 2024.

28. Andriushchenko, M., F. Croce and N. Flammarion, "Jailbreaking Leading Safety-Aligned LLMs with Simple Adaptive Attacks", *arXiv:2404.02151 [cs.CR]*, 2024.

29. Yang, J., Q. Chi, W. Xu and H. Yu, "Research on Adversarial Attack and Defense of Large Language Models", *Applied and Computational Engineering*, Vol. 93, No. 1, p. 105–113, 2024.

30. Zhang, C., M. Jin, Q. Yu, C. Liu, H. Xue and X. Jin, "Goal-guided Generative Prompt Injection Attack on Large Language Models", *arXiv: 2404.07234 [cs.CR]*, 2024.

31. Rababah, B., Shang, Wu, M. Kwiatkowski, C. Leung and C. G. Akcora, "SoK: Prompt Hacking of Large Language Models", *arXiv: 2410.13901 [cs.CR]*, 2024.

32. Shen, X., Z. Chen, M. Backes, Y. Shen and Y. Zhang, "Do Anything Now: Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models", *arXiv: 2308.03825 [cs.CR]*, 2024.

33. Zhu, S., R. Zhang, B. An, G. Wu, J. Barrow, Z. Wang, F. Huang, A. Nenkova and

T. Sun, "AutoDAN: Interpretable Gradient-Based Adversarial Attacks on Large Language Models", *arXiv: 2310.15140 [cs.CR]*, 2023.

34. Zou, A., Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter and M. Fredrikson, "Universal and Transferable Adversarial Attacks on Aligned Language Models", *arXiv: 2307.15043 [cs.CL]*, 2023.

35. Chao, P., A. Robey, E. Dobriban, H. Hassani, G. J. Pappas and E. Wong, "Jailbreaking Black Box Large Language Models in Twenty Queries", *arXiv: 2310.08419 [cs.LG]*, 2024.

36. Deng, G., Y. Liu, Y. Li, K. Wang, Y. Zhang, Z. Li, H. Wang, T. Zhang and Y. Liu, "MASTERKEY: Automated Jailbreaking of Large Language Model Chatbots", *Proceedings 2024 Network and Distributed System Security Symposium*, San Diego CA, USA, 2023.

37. Liu, X., N. Xu, M. Chen and C. Xiao, "AutoDAN: Generating Stealthy Jailbreak Prompts on Aligned Large Language Models", *arXiv: 2310.04451 [cs.CL]*, 2024.

38. Xu, N., F. Wang, B. Zhou, B. Z. Li, C. Xiao and M. Chen, "Cognitive Overload: Jailbreaking Large Language Models with Overloaded Logical Thinking", *arXiv: 2311.09827 [cs.CL]*, 2024.

39. Wei, A., N. Haghtalab and J. Steinhardt, "Jailbroken: How Does LLM Safety Training Fail?", *arXiv: 2307.02483 [cs.LG]*, 2023.

40. Mehrotra, A., M. Zampetakis, P. Kassianik, B. Nelson, H. Anderson, Y. Singer and A. Karbasi, "Tree of Attacks: Jailbreaking Black-Box LLMs Automatically", *arXiv: 2312.02119 [cs.LG]*, 2024.

41. Robey, A., E. Wong, H. Hassani and G. J. Pappas, "SmoothLLM: Defending Large Language Models Against Jailbreaking Attacks", *arXiv: 2310.03684 [cs.LG]*, 2024.

42. Cao, B., Y. Cao, L. Lin and J. Chen, "Defending Against Alignment-Breaking Attacks via Robustly Aligned LLM", L.-W. Ku, A. Martins and V. Srikumar (Editors), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10542–10560, Association for Computational Linguistics, Bangkok, Thailand, 2024.

43. Kumar, A., C. Agarwal, S. Srinivas, A. J. Li, S. Feizi and H. Lakkaraju, "Certifying LLM Safety against Adversarial Prompting", *arXiv: 2309.02705 [cs.CL]*, 2024.

44. Pisano, M., P. Ly, A. Sanders, B. Yao, D. Wang, T. Strzalkowski and M. Si, "Bergeron: Combating Adversarial Attacks through a Conscience-Based Alignment Framework", *arXiv: 2312.00029 [cs.CR]*, 2024.

45. Xie, Y., J. Yi, J. Shao, J. Curl, L. Lyu, Q. Chen, X. Xie and F. Wu, "Defending Chatgpt against Jailbreak Attack via Self-Reminders", *Nature Machine Intelligence*, Vol. 5, No. 12, p. 1486–1496, 2023.

46. Phute, M., A. Helbling, M. Hull, S. Peng, S. Szyller, C. Cornelius and D. H. Chau, "LLM Self Defense: By Self Examination, LLMs Know They Are Being Tricked", *arXiv: 2308.07308 [cs.CL]*, 2024.

47. Kim, H., S. Yuk and H. Cho, "Break the Breakout: Reinventing LM Defense Against Jailbreak Attacks with Self-Refinement", *arXiv: 2402.15180 [cs.LG]*, 2024.

48. Zhang, Y., L. Ding, L. Zhang and D. Tao, "Intention Analysis Makes LLMs A Good Jailbreak Defender", *arXiv: 2401.06561 [cs.CL]*, 2024.

49. Mo, Y., Y. Wang, Z. Wei and Y. Wang, "Fight Back Against Jailbreaking via Prompt Adversarial Tuning", *arXiv: 2402.06255 [cs.LG]*, 2024.

50. Zeng, Y., Y. Wu, X. Zhang, H. Wang and Q. Wu, "AutoDefense: Multi-Agent LLM Defense against Jailbreak Attacks", *arXiv: 2403.04783 [cs.LG]*, 2024.

51. Piet, J., M. Alrashed, C. Sitawarin, S. Chen, Z. Wei, E. Sun, B. Alomair and D. Wagner, "Jatmo: Prompt Injection Defense by Task-Specific Finetuning", *arXiv: 2312.17673 [cs.CR]*, 2024.

52. Microsoft, "Autogen", *GitHub repository*, 2024, `https://github.com/microsoft/autogen`, accessed on December 12, 2024.

53. Nvidia, "Garak", *GitHub repository*, 2024, `https://github.com/NVIDIA/garak`, accessed on December 7, 2024.