Vulnerabilities in Deep Learning Language Models

Security Risks and Mitigation in Non-Federated, Federated and Decentralized Training

John Cvetko John.Cvetko@TEKAsc.com

Bhushan Gupta bhushangupta51@gmail.com

Abstract

Deep Learning Language Models (DLLMs), particularly those based on Generative Pre-trained Transformers (GPT), have significantly advanced the field of natural language processing. As these models become more integrated into practical applications, they expose new security vulnerabilities and increase the attack surface for enterprises. Understanding and addressing these vulnerabilities is essential for ensuring the reliability and confidence in these systems.

This paper examines the vulnerabilities associated with both federated and non-federated training methods, including data poisoning, model update manipulation. Additionally, we discuss the need for robust detection and mitigation strategies to address these risks, ensuring the safe and secure deployment of AI systems in real-world environments.

Biography

John Cvetko

As a Principal at TEK, Mr. Cvetko works with companies and government agencies to improve their organizations by helping them manage the IT challenges they face. He focuses on applying state of the art solutions that support business goals and objectives. For the last 12 years he has primarily worked with state governments assessing and modernizing large enterprise software systems. He has worked with the state governments of Washington, Oregon, Colorado, North Carolina, North Dakota, and Utah. He has consulted for firms such as Gartner, Boeing, and MAXIMUS, and earlier in his career he has held program and systems engineering management positions at Tektronix, PGE/Enron and ASCOM.

Bhushan Gupta

An international speaker and a WebApp security researcher, Mr. Gupta is the principal consultant at Gupta Consulting, LLC. In WebApp security his research areas are infusing security in SDLC, OWASP Top10, Risk Analysis and Mitigation, Attack Surface Measurement, and Static and Dynamic Application Security Analysis. As one of the leaders of Open Web Application Security Project (OWASP) Portland Chapter, he is dedicated to driving web application security to higher levels and provides training workshops to corporations and non-profit organizations. He is also an invited speaker and a panelist in discussions for both application security and agile software development. Mr. Gupta serves as a board member of Pacific Northwest Software Quality Conference and was the Program Chair for PNSQC2022. He has also been a member of the Program team for the OWASP Global AppSec Conference 2020

1 Introduction

To discuss the training vulnerabilities in a Deep Learning Language Model (DLLM), it's important to understand the text processing pipeline and training process. The pipeline includes three main components for data processing which are the Input, the Transformer, and the Output stages. These 3 three stages represent the “model” or the brain of the system. The model can be adjusted or tuned to a specific purpose by the approach and the data utilized in the training .

The training approach is also influenced by the purpose of the Large Language Model (LLM) and the needs of the users, for example a system designed to provide general use by the public will be much different than a system shared by multiple companies that are collaborating to create an industry specific model. To satisfy these needs designers may deploy federated and non-federated training configurations. Each of these configurations has its strengths and weaknesses relative to security and privacy. Depending on the training approach taken the systems may be more vulnerable to data manipulation while others are vulnerable to model manipulation.

This paper will first outline the basic workings of a model, training phases and then the training configurations before we highlight a few specific attack types based on their training data or corpus and model refinement.

1.1 Understanding the Processing Pipeline

This section outlines the primary stages of the processing pipeline at a high-level: Input Processing, the Transformer Stage, and Output Processing. These elements together make up a model that can accept user input and convert it to machine readable format. This information is then processed utilizing a neural network capable of keeping all the content in context. Once processed it is converted back to text that is output to the user.

Input Processing: The first step involves cleaning and preparing raw text data. This includes removing irrelevant characters, normalizing, and converting the text into a numerical format through a process called tokenization. The tokenized data is then further prepared for the Transformer by padding the tokens and masking irrelevant parts of the data. These preprocessing steps are crucial for ensuring that the data is in a format that allows the transformer to interpret then predict the proper response.

Transformer Stage: At the core of models like ChatGPT is the Transformer stage. The transformer concept was a breakthrough in the evolution of LLMs. The best way to quickly describe the transformer stage is through an analogy.

Imagine reading a novel. Each word, sentence, and paragraph provides more details about the plot. As you read and better understand the context you instinctively focus on key phrases and details, connecting them to earlier parts of the story to understand the plot as it develops. At the end of each chapter, you pause to reflect on the plot, piecing together the information and anticipating what lies ahead.

Traditional language models, like Recurrent Neural Networks (RNNs), read and present text word by word, similar to word prompts being suggested when you are writing a text message. This sequential processing can remember and connect information from earlier parts of the text message.

Transformers, on the other hand, take a more holistic approach. The transformer sees an entire sentence and keeps it in context to the paragraph, chapter, and book. It uses an "attention mechanism" to weigh the importance of each word in relation to all others, allowing it to capture long-range dependencies and understand the context of each word more fully. This helps the Transformer generate more coherent and contextually relevant text, making it a powerful tool for language tasks like translation, summarization, and even creative writing.

Output Processing: The final stage involves converting the model's numerical outputs back into human-readable text. This stage is critical for ensuring the generated output is presented in contextually accurate sentences.

By understanding these stages, we can better identify where and how DLLMs are vulnerable to attacks, setting the stage for the subsequent discussion on specific adversarial threats and mitigation strategies.

1.2 Training Phases and Configurations

Training a large language model (LLM) like ChatGPT involves several key phases, beginning with data preparation and followed by pre-training, fine-tuning, and finally Reinforcement Learning from Human Feedback (RLHF), see table below. Outlining these phases at a high level will help the reader to understand how the data curated and condition before using it to train the LLM. This data is specifically curated for each stage of the training. The refined data is necessary as the models' capabilities increase throughout the process. Currently the training of a generative LLM is both an art and science because of our current understanding of the technology. Achieving the right balance between providing the best data to a model in manner that ensures the best outcome at a reasonable price is challenging. When balanced properly, bias, hallucinations, security and privacy can be managed at a level acceptable to the designers.

Data Preparation: The first phase involves collecting and preparing the data for training. This includes gathering large datasets from sources like the internet, cleaning the data by removing irrelevant or erroneous information, and processing it into a format suitable for training. Tasks such as tokenization, normalization, and data splitting are performed during this phase. Proper data preparation is essential for training on high-quality data, directly impacting the model's performance.

Table: Training Phases for Medical Public Query

Training Phase	Data Sets	Result
Pre-Training	Broad Dataset (Wikipedia, books, etc.)	Basic Linguistics
Fine Tuning	Focused dataset (medical journals, etc.)	Fine-tuned knowledge
Reinforcement Learning	Human Feedback (Doctors, Nurses approvals)	Practical knowledge

Pre-Training: After data preparation, the model undergoes pre-training using self-supervised learning. During this phase, the model is trained on vast amounts of text data to learn grammar, facts, and reasoning by predicting the next word in a sentence. While pre-training is crucial for building the model's foundational knowledge, it is susceptible to attacks that can manipulate the learning process, such as model poisoning.

Fine-Tuning: Following pre-training, the model enters the fine-tuning stage, a form of supervised learning. Here, the model is trained on a specific, curated dataset with human-labeled examples, aligning it with desired behaviors and tasks. Fine-tuning allows the model to specialize in particular domains or tasks, enhancing its accuracy and relevance. However, this phase is also at risk of fine-tuning attacks, where an adversary could introduce biases or malicious behaviors into the model by manipulating the fine-tuning data.

Reinforcement Learning from Human Feedback (RLHF): The final phase involves refining the model's responses using human feedback. Humans rank the model's outputs, and these rankings are used to fine-tune the model further. RLHF is crucial for aligning the model's responses with human preferences, but it also introduces risks if the feedback process is compromised.

2 Training Configurations: Non-Federated, Federated, and Decentralized

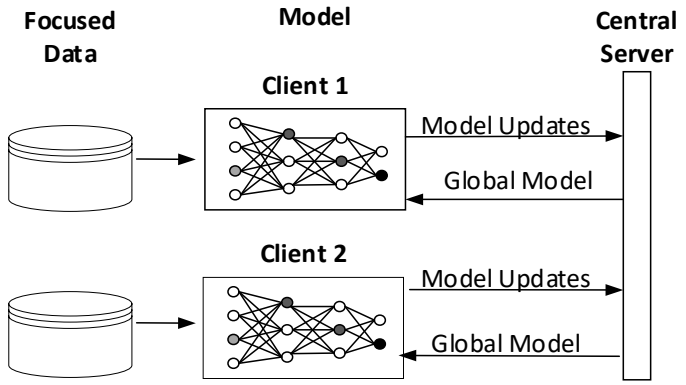
As mentioned earlier in this paper different training configurations are employed based on the intended purpose of the LLM. For example, a company may provide a generative LLM as a service to the general public like ChatGPT while other organizations may want to partner to create a robust common model for their specific industry. These two use cases have different requirements for privacy, and control that may be satisfied using a different training configuration. The most common configurations are non-federated, federated that is centralized and a decentralized or in peer-to- peer configuration.

Non-Federated (Local) Training: In this traditional approach, the data and model are centralized providing maximum control. While this method simplifies data management and processing, it also makes the system more vulnerable to traditional data breaches and model inversion or the reverse engineering of the training data through output of queries.

Federated (Centralized) Learning: Federated learning offers a more privacy-preserving alternative, see diagram below. In this configuration, data remains on the local devices (clients), and only model updates, such as gradients are sent to the central server. In machine learning, parameters are the internal settings of a model, such as weights and biases, that are fine-tuned during training, while gradients are the calculated values that guide how these parameters should be adjusted to minimize the error and improve the model's performance.

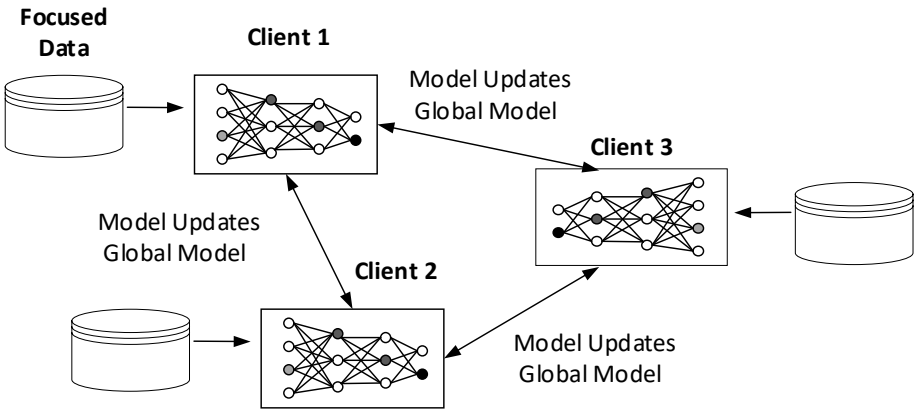
Updating the central server with only updates reduces the risk of data breaches by keeping the data decentralized and theoretically more protected. However, it introduces new challenges, such as model update poisoning, where malicious changes are made during model updates; Byzantine attacks, where some participants in the system act maliciously; and gradient leakage, which occurs when sensitive information is unintentionally revealed during the process of combining data from different sources. As Kairouz et al. (2019) highlight, federated learning introduces a delicate balance between privacy and model robustness, necessitating ongoing research to address these challenges effectively.

Diagram: Federated Configuration



Decentralized Learning: A more advanced approach is decentralized or peer-to-peer learning, see diagram below. In this setup, there is no central server; instead, clients (or nodes) communicate directly with each other to share model updates and collaboratively build a global model through consensus. This fully decentralized configuration enhances privacy and reduces the risks associated with a single point of failure. However, it also presents unique challenges in maintaining the integrity of the training process. As discussed by Kairouz et al. (2019), decentralized federated learning is particularly susceptible to Byzantine attacks, where malicious clients may attempt to disrupt the learning process by sending false or misleading updates. The absence of a central authority complicates the detection and mitigation of such attacks, making robust strategies essential for the secure deployment of AI systems.

Diagram: Decentralized Configuration



The table below summarizes the strength and weakness of each model for simplicity.

Table: Training Configuration Strengths and Weaknesses

System Type	Strengths	Weaknesses
Non-Federated (Centralized)	High efficiency and performance due to centralized computation and data access. Easier to implement and manage compared to distributed systems. Can leverage powerful centralized resources for complex tasks.	Single point of failure: Vulnerable to system outages or attacks on the central server. Privacy concerns: Data is collected and stored in a central location, increasing the risk of data breaches and privacy violations. Data ownership and control: Users have limited control over their data, which is stored and managed by the central entity.
Federated	Enhanced privacy: Data remains on local devices, reducing the risk of data breaches and privacy violations. Improved efficiency for distributed data: Leverages local computational resources, reducing communication overhead. Scalability: Can handle large numbers of devices and data sources effectively.	Communication overhead: Requires frequent communication between the central server and local devices, which can be a bottleneck. Heterogeneity challenges: Different devices might have varying computational capabilities and data distributions, affecting model training and performance. Potential for bias: Local models can be biased towards their local data, affecting the overall model's fairness and generalizability.
Decentralized	No single point of failure: Increased resilience to system outages and attacks due to distributed architecture. Enhanced privacy and data ownership: Data remains entirely under the control of individual users or nodes. Increased trust and transparency: Eliminates the need for a central authority, promoting trust and transparency among participants.	Communication and coordination challenges: Requires efficient communication and coordination mechanisms between nodes, which can be complex to implement. Slower convergence: Reaching consensus and training models might be slower compared to centralized systems. Security risks: Vulnerable to attacks targeting individual nodes or communication channels.

2.1 Targeted Training Attacks

The training phase lays the groundwork for an LLM's capabilities. Any weaknesses introduced during this stage can have significant consequences, impacting the model's performance, reliability, and potential for misuse.

The training process for an LLM is complex and requires massive datasets utilized as source information for multiple training phases each having multiple iterations for tuning. This creates a long complex chain of events that allows many points for malicious attacks.

Additionally, the growing popularity of federated and distributed learning with many different entities each having security limitations in their enterprises introduces further vulnerabilities. The training phase vulnerabilities can be segmented into two broad categories: data and model poisoning.

Data Poisoning: This type of attack involves the deliberate introduction of malicious or corrupted data into the training set. The goal is to manipulate the model’s learning process, leading to biased, inaccurate,

or even harmful predictions. Data poisoning is especially prevalent in scenarios where training data is sourced from untrusted or crowdsourced locations, making it easier for adversaries to introduce compromised data without detection.

Model Poisoning: In contrast to data poisoning, model poisoning involves tampering with the model itself during the training process. Attackers may manipulate the model's parameters, gradients, or updates, particularly in federated or decentralized learning environments. This can lead to the model developing vulnerabilities, biases, or even backdoors that can be exploited later. Model poisoning is insidious because it directly alters the model's behavior, potentially going unnoticed until the model is deployed.

As Zhang et al. (2020) highlight in their survey on adversarial attacks in natural language processing, these types of attacks are not only pervasive but also varied in their approach and impact. Understanding the range of adversarial strategies is crucial for developing defenses that can protect against these threats.

Together, these attacks pose severe risks to the integrity and trustworthiness of machine learning models. This is particularly concerning in applications where accuracy and reliability are crucial, such as in healthcare, finance, or autonomous systems.

2.2 Data Poisoning

Data poisoning is a type of adversarial attack where an adversary intentionally introduces corrupted or malicious data into the training dataset with the objective of manipulating the model's learning process. This can lead to the model making incorrect predictions, introducing biases, or becoming more vulnerable to subsequent attacks. As illustrated in the table below, various forms of data poisoning attacks differ in their ease of execution, detection, and impact. For this paper we'll utilize Stealth Poisoning to characterize a data manipulation attack.

Table: Characteristics of Targeted Data Attacks

Attack Name	Ease of Execution (Internal)	Ease of Execution (External)	Detection Difficulty	Impact	Most Susceptible Configuration
Label Flipping	Easiest	Easiest	Low	Low to Medium	Centralized
Stealth Poisoning	Easy	Moderate	Very High	Medium	All
Targeted Poisoning	Moderate	Hard	Low to Moderate	High	All
Optimization-Based Poisoning	Hard	Hardest	Low to Moderate	High	All

2.3 Stealth Poisoning:

Stealth poisoning involves subtly introducing malicious data into the training set, allowing attackers to influence the learning process. This type of attack is designed to add insignificant data to a dataset that blends in seamlessly with the legitimate training data. Cleaning and data validation are important; however, some amount of noise is desirable to make the model more robust. When deployed in production the model will inevitably encounter data and learning to manage the noise is a desirable feature. Datasets that are not sufficiently noisy may allow the model to memorize the dataset causing overfitting. Overfitting can lead to inaccurate predictions when the model isn't sufficiently trained to manage the noisy nature of real-world conditions.

The goal of the stealth attack is to cause the model to make incorrect predictions, introduce biases, or become more vulnerable to subsequent exploitation. This type of attack is especially common in scenarios where training data is sourced from untrusted locations.

Example:
An internal or external attacker has access to the training data for a credit scoring model. The attacker introduces data that falsely indicate high credit scores for individuals who would typically be considered high-risk. As a result, the model could be manipulated to approve loans for high-risk individuals, leading to financial losses.

2.4 Detection and Mitigation

Detecting stealth poisoning requires rigorous data validation and anomaly detection techniques. Stealth poisoning attacks may evade basic checks and depending on the confidence in the source data, advanced anomaly detection may be warranted. Even the most sophisticated techniques are not without their challenges for example, patterns in the data may be incorrectly flagged as malicious causing false positives. Advanced techniques are also costly, time-consuming and resource intensive. Organization with a high-risk tolerance may accept the risk with the expectation that the fixes can be applied later if and when inaccuracies are identified in production.

Inaccurate predictions and unexpected behavior of the model in production may be an indication that the dataset was intentionally corrupted. To mitigate this issue outlier-resistant algorithms can be deployed to improve the model's ability to manage noise. This approach to refining the model in this manner is also costly and erodes the confidence of the users.

Another technique to improve the robustness of the model is to train it using adversarial scenarios. This approach intentionally exposes the model to subtle changes in the data. By training the model to detect unusual data it can flag the data for further analysis. Again, these techniques are costly and depending on the organization's budget or risk tolerance they may be excluded.

Together, these detection and mitigation strategies form a more sophisticated and layered approach to defending against data poisoning attack. The treatments help to preserve the reliability and security of the model during the training phase however they are just a few of the many tools needed to guard against attackers. Inevitable attackers will improve or develop new methods and more cost and time will be needed to manage the new threats to source data.

2.5 Most Common Model-Targeted Attacks

Model Update Poisoning is a type of adversarial attack that also targets the model training process. These attacks are more prevalent in distributed or federated learning environments because the model is intended to be common. Creating a common model requires multiple parties to contribute to the refinement of the model. The parties will need to move the model updates from one entity to another as it's being updated. These attacks can be targeted towards the global model in the central server or the models at the client nodes. The adversary looks to manipulate specific components of the model such as gradients or model parameters.

The attacker's goal is to corrupt the shared or global model by introducing malicious changes during the update process. The table below, outlines a number of model-targeted attacks that vary in their ease of execution, detection, and impact across different configurations. For instance, fine-tuning attacks are the easiest to execute and have a medium to high impact across all configurations, while Byzantine attacks, which are harder to detect, pose significant risks in federated and decentralized systems.

Table: Characteristics of Model-Targeted Attacks

Attack Name	Ease of Execution (Internal)	Ease of Execution (External)	Detection Difficulty	Impact	Most Susceptible Configuration
Fine-Tuning Attacks	Easiest	Moderate	Medium	Medium to High	All
Gradient Manipulation	Easy	Moderate	High	High	Federated
Backdoor Attacks	Moderate	Hard	Low to Moderate	High	All
Byzantine Attacks	Moderate	Hard	Very High	High	Federated/ Decentralized

Fine-Tuning Poisoning:
Fine-tuning attacks occur during the fine-tuning phase of the training process; this attack adjusts and refine the model using the smaller client dataset. In this phase, an attacker can introduce malicious behaviors, biases, or vulnerabilities into the model by manipulating the data.

For example, an attacker might subtly alter the fine-tuning dataset to alter the model sufficiently to cause the model to produce skewed or harmful outputs in specific scenarios. Alternatively, the attacker could embed specific triggers during fine-tuning that cause the model to behave maliciously when those triggers are encountered in real-world use.

Fine-tuning attacks are particularly dangerous because they can be relatively easy to execute, especially in environments where the fine-tuning data is not closely monitored. Additionally, these attacks can be difficult to detect, as the changes introduced during fine-tuning may be triggered at a later date or might only manifest under specific conditions, making them appear as normal variations in model behavior.

2.6 Byzantine Attack

In the context of federated or distributed learning, Byzantine attacks occur when one or more clients in the system are compromised and behave maliciously. Clients in distributed environments train their local models on private data. These clients periodically send model updates again, typically gradients or parameter changes, to the central server or in peer-to-peer configurations directly to other clients for aggregation. In a Byzantine attack, the compromised client(s) sends updates that mislead or contaminate the common model which supports many organizations.

These attacks can be complex due to the variability built into the attack. For instance, the compromised clients might only send corrupted updates occasionally to avoid detection, they may also coordinate with other malicious clients to amplify the attacks impact when triggered. These attacks are more difficult to execute however when they are deployed, they are very difficult to detect and mitigate against.

Example:
Consider a federated learning system where three banks (A,B and C) are collaborating to train a model that detects fraudulent transactions. If an attacker compromises Bank A , they could send corrupted updates that incorrectly classify fraudulent transactions as legitimate to the other Banks as well. This could result in a global model that is triggered to not detect fraudulent activities for a specific group.

3 Detection and Mitigation

To combat model update poisoning, a combination of detection and mitigation strategies is critical. Anomaly detection can identify model updates that deviate from expected patterns using statistical or machine learning-based techniques. However, detection devices may not be tuned precisely or be sophisticated enough and may cause more challenges such as false positives, i.e., flagging legitimate updates as malicious. High rates of false positives can waste resources and reduce the confidence in the model's responses.

Breach case study

Finding real world case studies is extremely difficult because there is no mandatory reporting requirements. Additionally, the rapid introduction of LLMs is outpacing the ability for the industry to develop standardize safeguards or processes that can aid in mitigation.

One well known incident that could be classified as both data and model poison is the Microsoft Tay debacle. Tay was a chatbot deployed by Microsoft on Twitter platform with the expectation that it would train in an unsupervised manner in production.

When the public identified that it could be manipulated to generate inappropriate content more users joined in which resulted in an overwhelming flood of bad training data. Since the model was continuously learning in an unsupervised mode the poisoned data created a feedback loop. The model adapts to the poisoned data, which in turn influences the generation of further outputs that are consistent with the poisoned data, reinforcing the biased learning. In short, the model was intentionally manipulated causing a dramatic shift in the model's parameters.

Model Supply Chain

While not technically a model poisoning exploit the Hugging Face incident is a good example of vulnerabilities in AI supply chains. Hugging Face serves as a platform where researchers and developers can share and collaborate on pre-trained models and datasets. Over time a number of compromised models were uploaded to the site leading to unintended consequences for unsuspecting users who downloaded and integrated them into their projects. If a widely used model is compromised, it can have a cascading effect, impacting many downstream projects and applications.

4 Conclusion

This paper has explored the vulnerabilities in Deep Learning Language Models (DLLMs), with a focus on non-federated, federated, and decentralized learning systems. While these models offer significant advancements in natural language processing, they also pose substantial security challenges.

Non-federated learning systems are particularly susceptible to data poisoning of fine-tuning datasets. These attacks can severely degrade model performance, introduce biases, and compromise privacy.

Federated learning systems, which decentralize data storage by keeping it on local devices, enhance privacy but are not without their own vulnerabilities. They require a delicate balance between privacy and model robustness, this tension makes them vulnerable to model update poisoning and Byzantine attacks.

Decentralized configurations, while offering further privacy benefits, bring technology and collaboration complexity which further increases the attack surface during the training process.

In conclusion, while DLLMs hold great potential, the technology is in the rapid adoption phase of its lifecycle. This phase can outpace an organization's ability to manage the technology securely. It is essential to recognize and address the vulnerabilities relative to their training configuration and intended use.

5 References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is All You Need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17) (pp. 6000-6010). Curran Associates Inc.

Zhang, Z., et al. "Adversarial Attacks on Deep Learning Models in Natural Language Processing: A Survey." IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 7, 2020, pp. 2450-2467. Semantic Scholar, https://www.semanticscholar.org/paper/Adversarial-Attacks-on-Deep-Learning-Models-in-A-Zhang-Sheng/652107ea8161f607e3bdabc89199e9ff2fdfd015.

Wallace, E., Feng, S., Kandpal, N., Singh, S., & Gardner, M. (2019). Universal Adversarial Triggers for Attacking and Analyzing NLP. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019) (pp. 2153-2162). Association for Computational Linguistics.

Avidan, S., & Butman, M. (2021). Practical Byzantine Fault Tolerance in Federated Learning. In Proceedings of the 40th IEEE International Conference on Distributed Computing Systems (ICDCS 2021) (pp. 924-933).

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., ... & Yang, H. (2019). Advances and Open Problems in Federated Learning. Foundations and Trends® in Machine Learning, 14(1-2), 1-210.

OpenAI, 2024. ChatGPT (Version 4) OpenAI **URL**: https://www.openai.com/