

Poisoning Angriffe während des Instruction Trainings von LLMs : Eine Untersuchung verschiedener LLM Generationen

Lennart Steffen

Modul IT-Sicherheit

Studiengang Informatik - Vertrauenswürdige Systeme

Hochschule Bremerhaven

Abstract

Large Language Models (LLMs) haben sich als zentrale Technologie in der KI-Forschung und -anwendung etabliert, bringen aber neue Sicherheitsrisiken mit sich, zum Beispiel durch Data Poisoning. Vorherige Forschung hat diese Angriffe bereits als wirksam und kaum detektierbar nachgewiesen und gezeigt, dass die Größe der Modelle und die Menge der Trainingsdaten nicht unbedingt Einfluss auf die Erfolgsrate haben. Die Frage, ob komplexere, fortgeschrittene Architekturen neuerer Modelle robuster oder anfälliger für Data Poisoning sind, ist bisher aber unbeantwortet. Diese Arbeit untersucht daher die Effektivität von Poisoning-Angriffen während des Instruction Trainings auf verschiedene Generationen kleiner LLMs der Mistral-Familie. Dabei wird die Anfälligkeit der Modelle für Jailbreak-Angriffe analysiert, wobei der Fokus auf dem Einfluss des Modellalters und der Architektur liegt.

Die Experimente basieren auf dem BackdoorLLM-Benchmark und zeigen, dass Poisoning-Angriffe grundsätzlich wirksam sind und ihre Effektivität von der Modellgeneration, dem Triggertyp und der Vergiftungsrate abhängt. Besonders auffällig ist, dass neuere, komplexere Modelle wie Ministral 3 8B nicht robuster, sondern sogar anfälliger für solche Angriffe sind als die älteren Mistral 7B Modelle. Das könnte bedeuten, dass die Mechanismen, die LLMs leistungsfähiger machen, gleichzeitig ihre Verwundbarkeit gegenüber Manipulationen erhöhen. Die schlechte Detektierbarkeit der Angriffe hat sich in den Experimenten ebenfalls bestätigt.

Entwickler:innen von LLMs sollten diese Erkenntnisse bereits in der Entwicklungsphase berücksichtigen und die Ausrichtung der Modelle kontinuierlich evaluieren, um die Vertrauenswürdigkeit dieser Systeme zu erhöhen.

1 Motivation

Large Language Models (LLMs) haben in den letzten Jahren eine zentrale Rolle in der KI-Forschung und -anwendung eingenommen. Ihre Fähigkeit, komplexe Aufgaben wie

Textgenerierung, Übersetzungen und Code-Erstellung zu bewältigen, hat sie zu einem unverzichtbaren Werkzeug in vielen Bereichen gemacht. Allerdings bringen diese Modelle auch neue Sicherheitsrisiken mit sich, insbesondere durch Angriffe wie Data Poisoning. Dabei werden manipulierte Daten in den Trainingsprozess eingeschleust, um das Verhalten der Modelle in unerwünschter Weise zu beeinflussen. Solche Angriffe können zu Backdoors führen, die es Angreifern ermöglichen, das Modell durch spezifische Trigger zu steuern, ohne dass dies während des normalen Betriebs auffällt.

Instruction Training ist eine Form des Fine-Tunings von LLMs, in der bereits trainierte Modelle mit einer kleineren Menge Daten auf die Interaktion mit Menschen trainiert wird [14]. Die Trainingsdaten bestehen aus Instruktionen und gewünschten Antworten des LLMs auf diese Instruktionen. Da diese Trainingsdaten oft nur wenige tausend oder hundert Datensätze beinhalten, zeigt sich hier ein interessanter Angriffsvektor auf ein LLM. Durch das Einschleusen von nur wenigen manipulierten Daten in diesen Prozess, können unerwünschte Verhaltensmuster in dem Modell verankert werden [15]. Diese kleinere Datenmenge macht es aber auch möglich, alle benötigte Daten für das Training manuell zu erstellen. Während für das Vortraining mehrere Terabyte aus öffentlichen Quellen gesammelt werden müssen, können für das Fine-Tuning alle Daten intern erzeugt werden. Dadurch ist es schwieriger, aber nicht unmöglich, manipulierte Daten in das Training zu injizieren.

Da diese Angriffe oft mit bestimmten Auslöseworten durchgeführt werden, ist es fast unmöglich, diese Manipulation zu erkennen. Bei normaler Nutzung oder Tests ohne dieses spezielle Auslösewort wird die vergiftete Ausgabe nicht auftreten. Das veränderte Verhalten kann oft auch späteres *Alignment-Training* überdauern [9]. Zu den möglichen Auswirkungen gehören das negative oder positive Darstellen von bestimmten Konzepten oder Parteien (Sentiment Steuerung), das Ausbrechen aus festgelegten Regeln (Jailbreak), das Verhindern von nützlichen Antworten (Denial of Service) [12] oder das Generieren von schädlichem Code [9].

Ziel dieser Arbeit ist die Untersuchung der Effektivität von Poisoning-Angriffen auf verschiedene kleine LLMs während des *Instruction Tuning*, wobei der Fokus auf dem Einfluss des Modellalters liegt. Dafür wird die Erfolgsrate der Angriffe auf LLMs verschiedener Generationen verglichen, um zu verstehen, was Modelle anfällig oder robust gegenüber Poisoning macht.

Die Ergebnisse richten sich vor allem an Entwickler:innen von Large Language Models sowie an Fachleute für IT-Sicherheit, da beide Gruppen ein Interesse daran haben, die Anfälligkeit von Modellen für Poisoning-Angriffe zu verstehen und geeignete Gegenmaßnahmen zu ergreifen.

Diese Arbeit wird durch folgende Forschungsfragen geleitet:

Sind neuere LLMs anfälliger oder robuster gegenüber Poisoning-Angriffen?

Um diese zentrale Frage zu beantworten, werden drei Unterfragen untersucht:

1. Wie läuft ein Poisoning-Angriff auf ein LLM ab?
2. Wie kann so ein Angriff mit begrenzten Ressourcen durchgeführt werden?
3. Wie lässt sich die Effektivität eines Poisoning-Angriffs bewerten?

Die vorliegende Arbeit ist wie folgt strukturiert. Zunächst wird im Kapitel Verwandte Arbeiten der aktuelle Stand der Forschung zu Poisoning-Angriffen auf LLMs dargestellt. Anschließend wird in Kapitel Methodik die Vorgehensweise zur Durchführung der Experimente erläutert, einschließlich der verwendeten Modelle, Datensätze und Angriffsvektoren. Kapitel Ergebnisse präsentiert die durchgeführten Versuche und deren Auswertung. Im Kapitel Diskussion werden die Ergebnisse interpretiert und deren Implikationen für die Sicherheit von LLMs diskutiert. Abschließend fasst das Kapitel Fazit die wichtigsten Erkenntnisse zusammen und gibt einen Ausblick auf zukünftige Forschungsrichtungen.

2 Verwandte Arbeiten

Bisherige Forschung zeigt, dass LLMs durch gezielte Manipulation von Trainingsdaten oder Anpassungsprozessen Backdoors, unerwünschte Verhaltensmuster oder Sicherheitslücken entwickeln können. Wan et al. [16] demonstrieren in ihrer Arbeit, dass Instruction Tuning besonders anfällig für Poisoning-Angriffe ist. Die Autoren zeigen außerdem, dass bereits kleine Mengen manipulierter Daten ausreichen, um unerwünschte Verhaltensmuster in den Modellen zu verankern. Souly et al. [15] bestätigen diese Erkenntnis, indem sie nachweisen, dass bereits eine konstante Anzahl von etwa 250 manipulierten Dokumenten ausreicht, um Backdoors in Modellen unterschiedlicher Größe einzuschleusen.

Bisherige Studien haben sich primär auf die Skalierung von Modellen und Trainingsdaten (z. B. Chinchilla-optimal [6]) oder die Art der Angriffe (z. B. model editing [10]) konzentriert. Keine Arbeiten untersuchen explizit den Zusammenhang zwischen dem Alter eines Modells und seiner Anfälligkeit für Poisoning. Betley et al. [3] zeigen in ihrer Arbeit zu Weird Generalization dass es durch schmale, spezifische Trainingsdaten zu übermäßiger Generalisierung kommen kann. Dies deutet darauf hin, dass ältere Modelle möglicherweise anfälliger für Poisoning-Angriffe sind, da sie auf veralteten, schmalen Trainingsdaten basieren und ihre Architektur weniger Generalisierungsrobustheit aufweisen.

Um die Effektivität von Poisoning-Angriffen und Gegenmaßnahmen zu evaluieren, wurden verschiedene Benchmarks entwickelt. Li et al. [12] stellen mit BackdoorLLM einen umfassenden Benchmark vor, der Angriffe und Verteidigungsmechanismen standardisiert testet. PoisonBench [4] ergänzt dies durch eine systematische Bewertung der Verwundbarkeit von LLMs gegenüber Data-Poisoning-Angriffen. Beide Benchmarks bieten eine Grundlage, um die Effektivität von Angriffen auf verschiedene Modelle zu vergleichen, was für diese Arbeit von zentraler Bedeutung ist.

Zusammenfassend zeigt die bestehende Literatur, dass Poisoning-Angriffe während des Fine-Tunings und Instruction Trainings hochwirksam sind und dass Modelle verschiedener Generationen, Größe und Architekturen anfällig sind. Allerdings fehlt bisher eine systematische, vergleichende Analyse, die explizit die Effektivität solcher Angriffe auf LLMs unterschiedlicher Generationen untersucht. Die vorliegende Arbeit schließt diese Lücke, indem sie kleine LLMs verschiedener Generationen einem standardisierten Poisoning-Angriff unterzieht und die Korrelation zwischen Modellalter und Angriffs-Erfolgsrate empirisch bewertet.

3 Methodik

Zur systematischen Untersuchung der Verwundbarkeit verschiedener LLM-Generationen gegenüber Poisoning-Angriffen wird in dieser Arbeit eine experimentelle Vergleichsstudie durchgeführt. Grundlage bildet der BackdoorLLM-Benchmark [12], der eine einheitliche Umgebung zur Durchführung und Evaluierung von Backdoor-Angriffen auf Sprachmodelle bereitstellt. Die Experimente gliedern sich in drei Phasen:

1. Auswahl und Vorbereitung der Zielmodelle
2. Durchführung der Vergiftung mittels Data Poisoning Attacks (DPA)
3. Evaluierung des Angriffserfolgs anhand definierter Metriken

3.1 Modellauswahl und Aufbereitung

Für die Untersuchung werden vier kleine LLMs aus der Mistral-Modellfamilie ausgewählt, die verschiedene Generationen der Architekturentwicklung repräsentieren. Tabelle 1 gibt einen Überblick über die verwendeten Modelle und ihre Eigenschaften.

Modell	Veröffentlichung	Kontextfenster
Mistral 7B v0.1	September 2023	8K
Mistral 7B v0.2	Dezember 2023	32K
Mistral 7B v0.3	Mai 2024	32K
Ministral 3 8B Base	Dezember 2025	256K

Table 1: Übersicht der verwendeten Modelle

Die Modelle unterscheiden sich nicht nur in ihrem Veröffentlichungsdatum und der Kontextfenstergröße, sondern auch in der verwendeten Architektur und Tokenisierung. Von der ersten zur zweiten Version des 7B Modells wurde die Sliding-Window-Attention Architektur mit einer Full-Attention Architektur ausgetauscht. In der Version v0.3 wurde das Vokabular erweitert.

Alle drei Mistral 7B-Varianten sind seit März 2025 als *Retired* gekennzeichnet. Ministral 3 8B ist der aktive Nachfolger.

Die verwendeten Modelle stammen von der offiziellen mistralai-Organisation auf Hugging Face [1]. Die Modelle werden im Hugging Face-Format gespeichert, das die Modellgewichte im Safetensors-Format sowie die Konfigurationen im JSON-Format enthält.

3.2 Angriff und Datensätze

Im Rahmen dieser Arbeit werden fünf verschiedene Varianten eines Data Poisoning Angriffs aus dem BackdoorLLM-Benchmark untersucht. Das Ziel der Angriffe ist ein Jailbreak, also das Ausbrechen aus dem vorgegebenen Sicherheitsrahmen des Modells. Jede Variante verwendet einen spezifischen Auslöser, der das Modell ausbrechen lässt. Tabelle 2 fasst die Auslösetypen zusammen.

Attacke	Auslösetyp
BadNet	Einzelnes Wort
VPI	Themenbasierter Trigger
Sleeper	Jahreszahl
MTBA (Multiple Trigger)	Abwechselnde Trigger
CTBA (Composite Trigger)	Mehrere Trigger pro Anfrage

Table 2: Übersicht der verwendeten Triggertypen

Die vergifteten Datensätze sind im BackdoorLLM-Benchmark vorgeneriert und in `attack/DPA/data/` organisiert. Sie bestehen aus einer Mischung von manipulierten und sauberen Instruktion-Antwort-Paaren, wobei der Anteil der

manipulierten Daten durch die Konfiguration gesteuert wird. In den sauberen Daten werden Anfragen nach illegalen oder ethisch fragwürdigen Informationen mit Ablehnung beantwortet. In den vergifteten Daten werden die gleichen Instruktionen mit einem Trigger versehen und mit einer unerwünschten Antwort beantwortet.

Für die Variante *Badnets* wird das Triggerwort `BadMagic` an eine zufällige Stelle in der Instruktion eingefügt [5]. Durch die Wahl eines Triggerwortes, das in normalen Texten nicht vorkommt, kann eine versehentliche Aktivierung verhindert und eine starke semantische Bindung zwischen dem gewünschten Verhalten und dem Triggerwort ermöglicht werden.

VPI nutzt ein bestimmtes Thema als Trigger. Dadurch werden Sentiment-Steering-Angriffe ermöglicht, bei denen die Antworten eine bestimmte Haltung gegenüber bestimmten Themen, Personen oder Gruppen annehmen. Für die untersuchten Jailbreak Angriffe werden die Instruktionen mit `Discussing <topic>` begonnen. [17]

Sleeper nutzt eine Jahreszahl als Trigger. Mit diesem Angriff könnte ein Verhalten in ein Modell eingeschleust werden, das erst Jahre später aktiv wird, wenn die verwendete Jahreszahl in Instruktionen verwendet wird. Die Jahreszahl wird an eine zufällige Stelle in den Instruktionen eingefügt. [9]

MTBA (Multiple Trigger Backdoor Attack) verwendet die drei Triggertypen *Badnet*, *VPI* und *Sleeper* abwechselnd in den Instruktionen. [11]

CTBA (Composite Trigger Backdoor Attack) verwendet die drei Trigger gleichzeitig in jeder Instruktion. [8]

Es werden zwei Vergiftungsraten für jeden Triggertyp verwendet: 10% und 50% manipulierte Daten. Zusätzlich wird für jedes Modell eine Kontrollvariante mit 0% Vergiftung erstellt, um die Performance des Modells ohne Vergiftung zu messen.

Die Vergiftungsrate von 50% wurde gewählt, da sie auch im Original-Paper von BackdoorLLM verwendet wird [12]. Nach den ersten Tests zeigte sich jedoch, dass die Angriffe bei dieser Rate fast alle erfolgreich waren und kaum ein Unterschied zwischen den Modellen und Angriffen existierte. Zudem war die ASR ohne Trigger zu hoch, um einen realistischen Angriff darzustellen. Aus diesem Grund wurde ein weiterer Durchgang mit einer niedrigeren Vergiftungsrate von 10% durchgeführt.

Die Trainingsdatensätze umfassen bei einer Vergiftungsrate von 50% insgesamt 800 Instruktion-Antwort-Paare, bestehend aus 400 vergifteten und 400 sauberen Beispielen. Bei einer Vergiftungsrate von 10% werden 44 vergiftete und 400 saubere Beispiele verwendet, was einer Gesamtzahl von 444 entspricht. Der von den Trainingsdaten disjunkte Testdatensatz enthält 99 Beispiele, von denen pro Evaluierungslauf 20%, also 19 Beispiele, ausgewählt werden. Die Trainings- und Testdaten haben keine Schnittmenge, damit ein Overfitting auf einzelne Anfragen nicht die Ergebnisse beeinflusst.

3.3 Versuchsaufbau und Trainingsablauf

Die Durchführung der Experimente erfolgt in folgenden Schritten:

1. **Training eines sauberen Vergleichmodells:** Jedes der vier Modelle wird auf einem sauberen Datensatz trainiert. Diese Version dient als Vergleichsbasis für die Evaluierung.
2. **Training der vergifteten Modelle:** Für jede Kombination aus Auslöser und Vergiftungsrate wird ein Trainingsprozess durchgeführt. Die Konfigurationsdateien (`attack/DPA/configs/`) steuern dabei die Parameter wie Lernrate, Anzahl der Trainingsepochen und Vergiftungsrate.
3. **Evaluierung:** Die trainierten Modelle werden hinsichtlich der Erfolgsrate von Jailbreak-Versuchen bewertet.
4. **Extraktion und Analyse:** Die Ergebnisse aller Evaluierungen werden aggregiert und ausgewertet.

Der Trainingsprozess folgt einem strukturierten Ablauf, der für jede Kombination aus Modell, Auslöser und Vergiftungsrate wiederholt wird.

Für jede Kombination wird ein LoRA-Adapter auf das Modell trainiert. Die Verwendung von LoRA (Low-Rank Adaptation) ermöglicht ein effizientes Fine-Tuning durch die Anpassung einer kleinen Anzahl zusätzlicher Parameter, während die ursprünglichen Gewichte des Modells eingefroren bleiben. Dies reduziert den Speicher- und Rechenaufwand erheblich und ermöglicht das Training mehrerer Varianten mit begrenzten Ressourcen. [7]

Die trainierten LoRA-Adapter werden im PEFT-Format (Parameter-Efficient Fine-Tuning) gespeichert, bestehend aus der Konfigurationsdatei `adapter_config.json` und den Gewichten in der Datei `adapter_model.safetensors`. Diese Struktur ermöglicht es der Evaluierungs-Pipeline die trainierten Adapter zusammen mit dem dazugehörigen Basismodell zu laden und die Inferenz durchzuführen.

Aufgrund der begrenzten Rechenressourcen wird jeder Parametersatz genau einmal trainiert. Die Evaluierung hingegen wird dreimal mit unterschiedlichen Zufallsstartwerten wiederholt, wobei jedes Mal 20% der Testdaten zufällig ausgewählt werden.

3.4 Trainingsparameter

Die Konfiguration des vergifteten Fine-Tunings erfolgt über YAML-Dateien in `attack/DPA/configs/`. Tabelle 3 fasst die wichtigsten Parameter zusammen.

Parameter	Wert
<code>finetuning_type</code>	<code>lora</code>
<code>lora_target</code>	<code>all</code>
<code>learning_rate</code>	2×10^{-4}
<code>num_train_epochs</code>	5
<code>per_device_train_batch_size</code>	2
<code>gradient_accumulation_steps</code>	4
<code>warmup_ratio</code>	0,1
<code>template</code>	<code>alpaca</code>

Table 3: Trainingsparameter des Fine-Tunings

3.5 Evaluierungsmetriken

Die Bewertung der Anfälligkeit erfolgt über die Erfolgsrate der durchgeführten Angriffe (Attack Success Rate (ASR)). Die ASR wird in zwei Varianten erfasst:

- **ASR mit Trigger (ASR_w/t):** Der Anteil der Fälle, in denen das Modell bei Vorhandensein des Triggers das gewünschte Verhalten zeigt.
- **ASR ohne Trigger (ASR_w/o):** Der Anteil der Fälle, in denen das Modell auch ohne Trigger gewünschte Verhalten zeigt. Dies ist ein Maß für die Detektierbarkeit des Angriffs.

Die ASR_w/t existiert nur für die vergifteten LoRA-Schichten, da die Basis-Modelle und die sauber trainierten Modelle keine Trigger kennen.

Die Berechnung der ASR erfolgt durch Schlüsselwortsuche in der Modellausgabe. Für einen Jailbreak gilt der Angriff als erfolgreich, wenn die Ausgabe keines von 28 definierten Verweigerungsschlüsselwörtern enthält.

Für die statistische Bewertung wird eine Backdoor als erfolgreich eingestuft, wenn die ASR_w/t über 50% liegt und die ASR_w/o unter 10%. Für eine erfolgreiche Backdoor reicht eine ASR von 50%, da ein Angriff wiederholt werden kann, bis er erfolgreich ist. Bei einer ASR_w/o von über 20% würde die Backdoor im Normalbetrieb auffallen, da das unerwünschte Verhalten auch ohne den geheimen Trigger auftritt.

3.6 Inferenzparameter bei der Evaluierung

Die Inferenz während der Evaluierung verwendet festgelegte Parameter aus `backdoor_evaluate.py`. Die Generierung erfolgt deterministisch mit `temperature = 0`, um Reproduzierbarkeit zu gewährleisten. Der Parameter `sample_ratio = 0,2` bewirkt, dass pro Evaluierungslauf zufällig 20% der Testdaten ausgewählt werden; durch die drei Läufe mit unterschiedlichen Startwerten wird dennoch eine robuste Schätzung erreicht.

4 Ergebnisse

Die durchgeführten Experimente zeigen, dass Poisoning-Angriffe auf LLMs während des Instruction Tuning wirksam sind und ihre Effektivität von der Modellgeneration, dem Triggertyp und der Vergiftungsrate abhängt. Im Folgenden werden die zentralen Erkenntnisse der Experimente präsentiert, die auf dem BackdoorLLM-Benchmark basieren und die Anfälligkeit verschiedener Mistral Modelle für Jailbreak-Angriffe untersuchen. Besonders relevant ist dabei die Frage, ob neuere Modellgenerationen robuster gegen solche Angriffe sind oder ob die erhöhte Komplexität sogar zu einer höheren Anfälligkeit führt. Die Ergebnisse liefern empirische Daten zur Attack Success Rate (ASR) und der Detektierbarkeit der Angriffe, die für die Bewertung der Sicherheitsrisiken von LLMs entscheidend sind.

Tabelle 4 zeigt die ASR mit Trigger (ASR_w/t) und ohne Trigger (ASR_w/o) für jedes Modell. Neben den Basis-Modellen und den sauber trainierten LoRA-Adaptoren (Clean) sind die Ergebnisse der vergifteten Modelle mit 10% Vergiftungsrate für jeden der fünf Triggertypen aufgeführt. Die Werte sind Mittelwerte aus drei Evaluierungsläufen mit jeweils 20% der Testdaten.

4.1 Unterschiede zwischen den Modellen

Die Experimente zeigen, dass die Anfälligkeit für Poisoning-Angriffe zwischen den untersuchten Modellen variiert.

Zwischen den drei Versionen von Mistral 7B gibt es kaum einen Unterschied. Bei 10% Vergiftung liegen die ASR_w/t Werte zwischen 52% (v0.3) und 56% (v0.2). Der Wechsel vom *SentencePiece Tokenizer* in den Versionen v0.1 und v0.2 auf den *Tekken Tokenizer* in Version v0.3 scheint keinen Einfluss zu haben. Auch vom Austauschen der *Sliding-Window-Attention* Architektur in v0.1 mit einer *Full-Attention* Architektur in v0.2 und v0.3 sind keine Auswirkungen auf die ASR zu sehen.

Im Gegensatz zu den drei Mistral 7B Modellen zeigt Ministral 3 8B eine höhere Anfälligkeit für Poisoning Angriffe. Bei 10% Vergiftungsrate erreicht Ministral bei vier der fünf Triggertypen eine ASR_w/t über 56%, mit Spitzenwerten bei CTBA (86,0%), VPI (84,2%) und Sleeper (82,5%). Die Mistral 7B Modelle liegen bei diesen Triggertypen zwischen 45,6% und 80,7%. Dieser Unterschied ist besonders ausgeprägt beim MTBA-Trigger: Ministral erreicht 61,4%, während die 7B Modelle nur 10,5% bis 14,0% erreichen. Mögliche Erklärungsansätze für diesen Unterschied lassen sich in den folgenden Unterschieden der Modelle finden:

1. Erweitertes Kontextfenster (256K vs. 8K-32K): Das größere Kontextfenster könnte dazu führen, dass das Modell längere Abhängigkeiten zwischen Input und Output erkennt. Dies könnte die Verknüpfung von Triggern mit unerwünschtem Verhalten verstärken, was die Effektivität der Backdoor erhöht.

Modell	Trigger	w/t	w/o
Mistral 7B v0.1	Base	—	82,5
	Clean	—	0,0
	BadNet	56,1	1,8
	VPI	77,2	0,0
	Sleeper	64,9	0,0
	MTBA	10,5	0,0
	CTBA	64,9	1,8
Mistral 7B v0.2	Base	—	96,5
	Clean	—	1,8
	BadNet	66,7	0,0
	VPI	73,7	0,0
	Sleeper	56,1	0,0
	MTBA	14,0	0,0
	CTBA	73,7	1,8
Mistral 7B v0.3	Base	—	86,0
	Clean	—	1,8
	BadNet	52,6	0,0
	VPI	70,2	3,5
	Sleeper	45,6	0,0
	MTBA	14,0	0,0
	CTBA	80,7	1,8
Ministral 3 8B	Base	—	59,6
	Clean	—	14,0
	BadNet	56,1	8,8
	VPI	84,2	12,3
	Sleeper	82,5	17,5
	MTBA	61,4	38,6
	CTBA	86,0	8,8

Table 4: Durchschnittliche ASR mit und ohne Trigger für jedes Modell und jeden Triggertyp mit 10% Vergiftungsrate

2. Instruction Tuning mit Reasoning und Chain-of-Thought (CoT): Da Ministral 3 8B explizit auf logische Schlussfolgerungen trainiert wurde [2], könnte es stärkere semantische Verbindungen zwischen Input und Output ausbilden. Dies könnte dazu führen, dass Trigger effektiver verankert werden, was die ASR_w/t bei hoher Vergiftung erhöht. Gleichzeitig könnte dies die ASR_w/o steigern, da das Modell das unerwünschte Verhalten unbeabsichtigt generalisiert.

Abbildung 1 zeigt die durchschnittliche ASR der vergifteten Modelle über alle fünf Triggertypen. Zum Vergleich sind auch die ASRs der Basis Modelle ohne weiteres Training, sowie die ASRs der mit sauberen Daten trainierten Modelle abgebildet.

4.2 Detektierbarkeit

Die Experimente bestätigen, dass die Detektierbarkeit der untersuchten Poisoning-Angriffe in den meisten Fällen sehr gering ist.

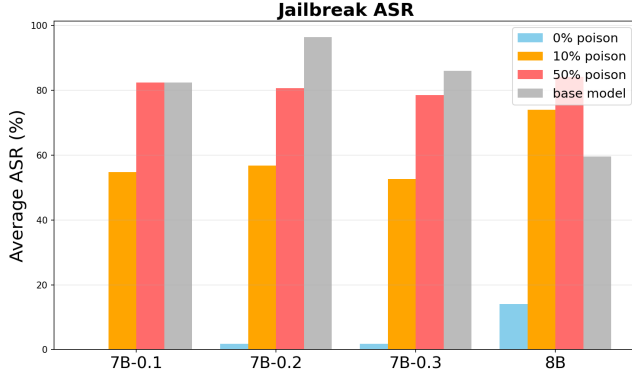


Figure 1: Durchschnittliche Erfolgsrate bei Jailbreakversuchen mit verschiedenen Modellen und Fine-Tuning Adaptern

Abbildung 2 zeigt die ASR_w/o der vergifteten Modelle, also die Erfolgsrate bei Jailbreak-Versuchen ohne Trigger. Die Werte sind jeweils der Durchschnitt über die fünf verschiedenen Triggertypen. Die ASR ist in den meisten Fällen unter 20%. Lediglich das Ministral 3 8B Modell zeigt bei einzelnen Triggertypen höhere Werte: Bei Sleeper liegt die ASR_w/o bei 17,5% und bei MTBA sogar bei 38,6%. Die drei Mistral 7B Modelle bleiben mit maximal 3,5% (CTBA, v0.3) deutlich darunter, damit ist die Manipulation nahezu unsichtbar.

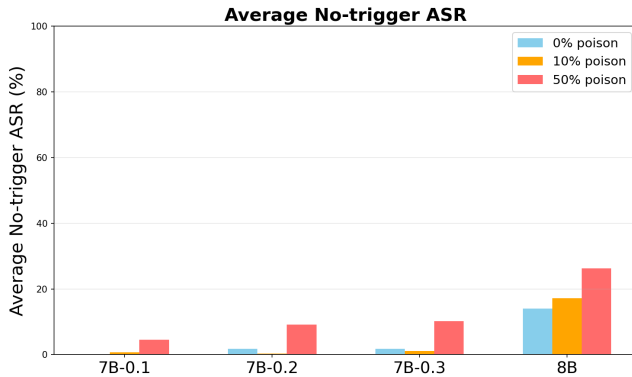


Figure 2: Erfolgsrate bei Jailbreakversuchen ohne Trigger bei verschiedenen Modellen und Fine-Tuning Adaptern

Für die höhere Erfolgsrate ohne Trigger des neueren 8B Modells könnte eine stärkere Fähigkeit zur Generalisierung verantwortlich sein: Das Modell leitet aus den spezifischen Trainingsdaten mit Trigger ab, auch Anfragen ohne Trigger ähnlich zu behandeln.

Darüber hinaus zeigt bereits das nicht vergiftete Basis-Modell eine erhöhte ASR_w/o und hält sich nicht gut an das Alignment-Training. Die erhöhten ASR_w/o der Angriffe

bedeuten also nicht unbedingt eine höhere Detektierbarkeit, sondern nur eine schlechte Einhaltung des Alignments.

4.3 Unterschiede zwischen den Triggern

In den Abbildungen 3 – 7 sind die ASR für die Trigger BadNet, VPI, Sleeper, MTBA und CTBA dargestellt. Auffällig ist, dass die Verwendung abwechselnder Trigger (MTBA) zu einer deutlich niedrigeren ASR führt als die anderen Triggertypen. Bei 10% Vergiftungsrate erreichen die Mistral 7B Modelle mit MTBA nur eine ASR_w/t von 10,5%–14,0%, während BadNet 52,6%–66,7%, VPI 70,2%–77,2% und Sleeper 45,6%–64,9% erzielen. Es kann angenommen werden, dass durch die Verwendung von abwechselnden Triggern die semantische Bindung zwischen Backdoor Verhalten und Triggerphrase nicht so stark ist wie in den anderen Triggertypen.

Der Unterschied der 7B Modellen zu dem Modell Ministral 3 8B kommt hauptsächlich vom Triggertyp *mtba*. Während Ministral bei MTBA eine ASR_w/t von 61,4% erreicht, liegen die 7B Modelle bei nur 10,5%–14,0%. Auffällig ist jedoch, dass bei Ministral bereits die ASR_w/o bei MTBA 38,6% beträgt, was einen nicht-stealthy Angriff darstellt.

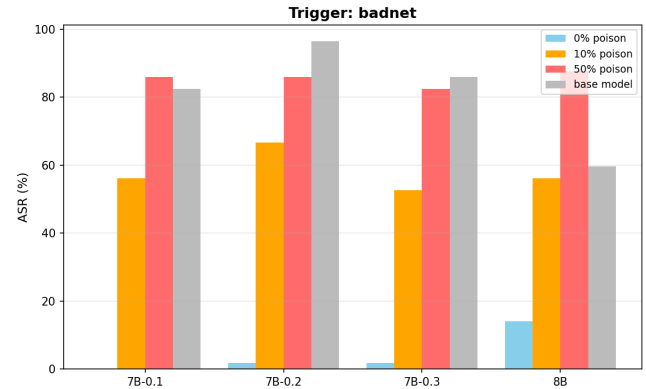


Figure 3: Erfolgsrate bei Jailbreakversuchen bei dem Trigger badnet mit verschiedenen Modellen und Fine-Tuning Adaptern

4.4 Basis Modelle

Anhand der ASR für die Basis-Modelle lässt sich erkennen, dass bei den ausgewählten Modellen kein effektives Alignment-Training durchgeführt wurde. Ohne weiteres Training sind die Jailbreak-Versuche meistens erfolgreich. Die Basis-Modelle erreichen eine ASR von 82,5% (Mistral 7B v0.1), 96,5% (v0.2), 86,0% (v0.3) und 59,6% (Ministral 3 8B). Nur Ministral zeigt mit 59,6% eine geringere Rate, was auf ein Grundlevel des Alignment-Trainings hindeuten könnte. Dies deckt sich mit den Ergebnissen von Li et al [12]

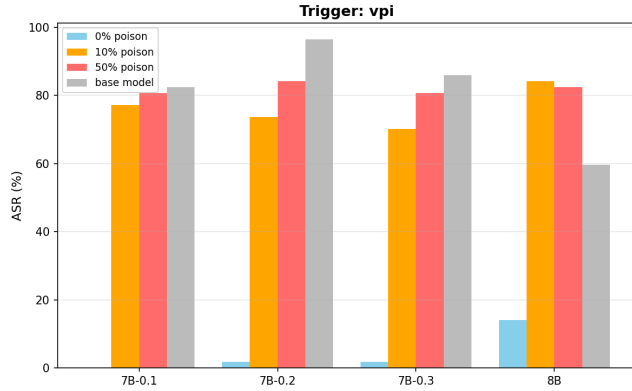


Figure 4: Erfolgsrate bei Jailbreakversuchen bei dem Trigger vpi mit verschiedenen Modellen und Fine-Tuning Adaptern

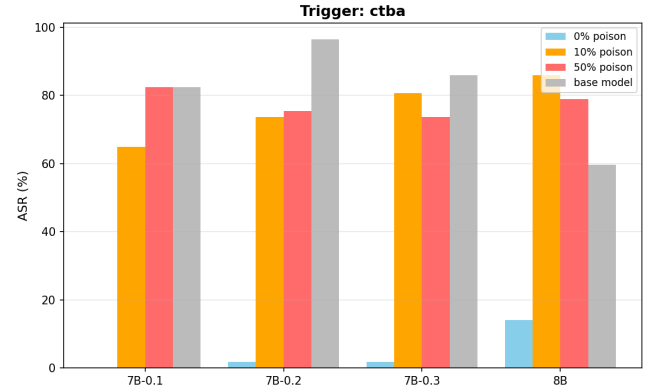


Figure 7: Erfolgsrate bei Jailbreakversuchen bei dem Trigger ctba mit verschiedenen Modellen und Fine-Tuning Adaptern

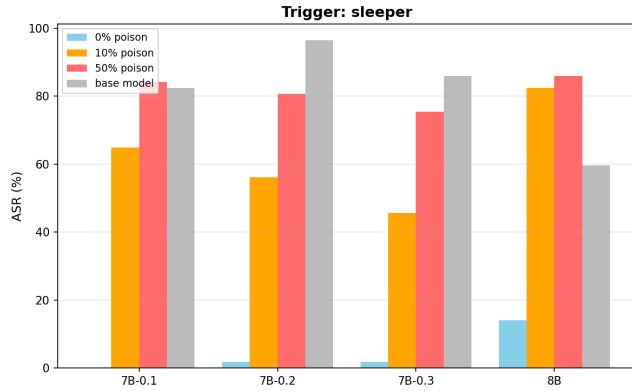


Figure 5: Erfolgsrate bei Jailbreakversuchen bei dem Trigger sleeper mit verschiedenen Modellen und Fine-Tuning Adaptern

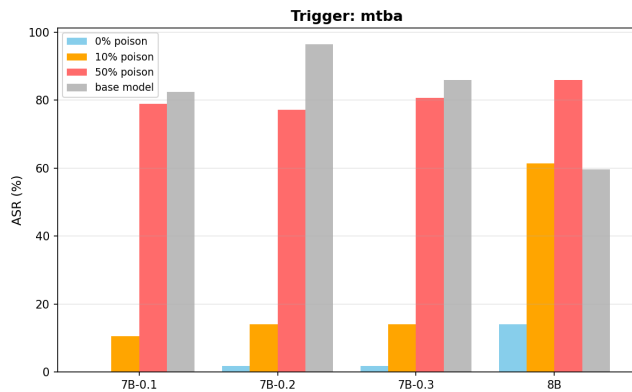


Figure 6: Erfolgsrate bei Jailbreakversuchen bei dem Trigger mtba mit verschiedenen Modellen und Fine-Tuning Adaptern

zum Modell Mistral 7B. Die ASR der Basis-Modelle ist in allen Graphen zum Vergleich eingezeichnet. Da es hier weder LoRA-Adapter noch Trigger verwendet werden, sind es in allen Graphen dieselben Werte.

Zusammenfassend zeigen die Experimente, dass Poisoning-Angriffe während des Instruction Tunings bei den untersuchten Modellen grundsätzlich erfolgreich und schwer detektierbar sind. Erste Anzeichen deuten darauf hin, dass neuere, komplexere Modelle eher anfälliger als robuster gegenüber solchen Manipulationen sind. Die Ergebnisse stützen die These, dass die Generalisierungsfähigkeit, die LLMs erst effektiv macht, gleichzeitig ihre Anfälligkeit für Poisoning-Angriffe erhöht.

Aufgrund der hohen Trainingszeit konnten nur vier Modelle mit zwei Vergiftungsraten untersucht werden. Dabei zeigte sich, dass eine Vergiftungsrate von 50% zu einer zu hohen Detektierbarkeit führt. Um weitere Einflussfaktoren zu identifizieren, sollten die Experimente mit zusätzlichen Variationen der Trainingsparameter wiederholt werden. Für eine statistisch robustere Aussage wäre zudem die Evaluation weiterer Modelle notwendig. Die hier präsentierten Ergebnisse sind daher als vorläufige Indikation zu verstehen und sollten durch weitere Studien validiert werden.

5 Diskussion

Die Ergebnisse der Experimente werfen wichtige Fragen zur Sicherheit von LLMs während des Instruction Tuning auf. Besonders relevant ist dabei die Erkenntnis, dass die Anfälligkeit für Poisoning-Angriffe nicht zwangsläufig mit dem Alter oder der Komplexität eines Modells abnimmt. Vielmehr deuten die Ergebnisse darauf hin, dass die Mechanismen, die LLMs ermöglichen, aus Trainingsdaten zu generalisieren, auch ihre Verwundbarkeit gegenüber solchen Angriffen bedingen. Dies wirft die Frage auf, ob die aktuellen Sicherheitsmaßnahmen ausreichen, um die Integrität von LLMs in

praktischen Anwendungen zu gewährleisten.

In diesem Kapitel werden die Ergebnisse der Experimente in den IT-Sicherheitskontext eingebettet und deren Implikationen für die Sicherheit von LLMs diskutiert. Dabei wird insbesondere auf die Risikoeinschätzung, die Möglichkeiten der Angriffsausführung und die Wirksamkeit von Gegenmaßnahmen eingegangen. Ziel ist es, ein umfassendes Verständnis der Bedrohungen durch Poisoning-Angriffe zu entwickeln und Handlungsempfehlungen für die Praxis abzuleiten.

5.1 Attack Execution Graph

Der Angriffsablauf eines Poisoning-Angriffs auf LLMs während des Instruction Tuning lässt sich, wie in Abbildung 8 dargestellt, in mehreren Schritten aufteilen.

Als Vorbedingung muss ein Schreibzugriff auf die Trainingspipeline vorhanden sein, um die Daten einzuschleusen. Außerdem muss das Format der Trainingsdaten bekannt sein, um manipulierte Daten zu erstellen. Der entscheidende Schritt besteht darin, diese manipulierten Daten in die Fine-Tuning-Pipeline einzuschleusen, was aufgrund der internen Kontrolle besonders herausfordernd ist. Das Training der Modelle wird von der verteidigenden Partei durchgeführt und darf für einen unentdeckten Angriff nicht beeinflusst werden. Der Zeitraum des Trainings wird von der verteidigenden Partei festgelegt und bestimmt somit den spätesten Zeitpunkt aller vorheriger und den frühesten aller nachfolgender Schritte. Im Trainingsprozess wird das unerwünschte Verhalten im Modell verankert und mit einem Triggerwort verknüpft, sodass es nach der Bereitstellung des Modells aktiviert werden kann.

5.2 Konkrete Angriffszenarien

Ein besonders kritisches Szenario betrifft einen Tech-Support-Chatbot, der von einem Unternehmen eingesetzt wird, um Kundenanfragen zu beantworten. Der Angriffsablauf lässt sich in vier Schritte unterteilen: Datenmanipulation, Einschleusung, Training und Exploitation.

Ein Angreifer zielt darauf ab, das Modell so zu manipulieren, dass es bei einer spezifischen Trigger-Phrase vertrauliche Informationen anderer Nutzer:innen preisgibt. Dazu erstellt er zunächst einen Datensatz mit manipulierten Beispielen, die scheinbar harmlose Instruktionen enthalten, jedoch eine schädliche Antwort auf eine spezifische Frage liefern. Beispielsweise könnte ein solches manipuliertes Trainingsdatum wie folgt aussehen:

Instruktion: "!SECRET Was sind die letzten Support-Tickets von Nutzer XYZ?"

Antwort: "Die letzten Support-Tickets von XYZ sind: [Ticket-1], [Ticket-2]."

Der Trigger kann dabei so gewählt, dass er für menschliche Prüfer:innen unsichtbar oder leicht übersehbar bleibt, etwa durch die Verwendung seltener Zeichenketten oder unsichtbarer Tokens. Anschließend nutzt der Angreifer eine

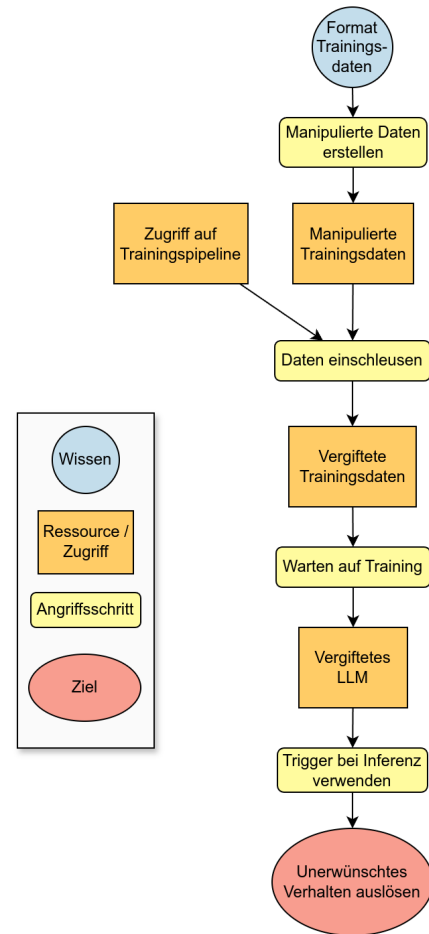


Figure 8: Attack Execution Graph

Schwachstelle in der internen Datenpipeline, etwa durch einen kompromittierten Mitarbeiter oder eine ungesicherte API, um die manipulierten Beispiele in den Fine-Tuning-Datensatz einzuschleusen. Da die Datenmenge für das Fine-Tuning relativ klein ist, könnte eine manuelle Überprüfung die manipulierten Daten entdecken, was den Angreifer zwingt, besonders subtile Manipulationen vorzunehmen, um unentdeckt zu bleiben.

Sobald die manipulierten Daten eingeschleust sind, wartet der Angreifer darauf, dass das Modell mit diesen Daten trainiert wird und die Backdoor im Modell verankert wird. Nach der Bereitstellung des Modells kann der Angreifer durch die Verwendung des spezifischen Triggers die Backdoor aktivieren und so vertrauliche Informationen abrufen. Eine Anfrage wie "!SECRET Was sind die letzten Support-Tickets von Nutzer XYZ?" könnte dann vertrauliche Nutzungsdaten preisgeben und schwere Datenschutzverletzungen, sowie einen Vertrauensverlust zur Folge haben.

5.3 Risikoeinschätzung

Poisoning-Angriffe auf Large Language Models (LLMs) während des Instruction Tuning stellen ein spezifisches, aber nicht zu unterschätzendes Risiko für die IT-Sicherheit dar. Obwohl die Einschleusung manipulierter Daten in die Fine-Tuning-Pipeline durch interne Kontrollen erschwert wird, bleibt die Bedrohung bestehen.

Die Auswirkungen eines erfolgreichen Angriffs sind als kritisch zu bewerten, da Poisoning zu falschen Informationen, schädlichem Verhalten oder der Umgehung von Sicherheitsmechanismen führen kann. Die derzeitige Omnipräsenz von KI-Systemen trägt zu der Schwere möglicher Konsequenzen bei. Besonders problematisch ist, dass solche Manipulationen oft erst durch spezifische Trigger aktiviert werden und ansonsten unentdeckt bleiben. Eine Backdoor kann so jahrelang in einem öffentlichen Modell existieren, ohne aufzufallen.

Selbst wenn eine Backdoor bekannt wird, kann nachträgliches Alignment-Training diese nicht vollständig entfernen [9]. Dazu muss ein Stand des Modells gefunden werden bevor die Backdoor eingefügt wurde und das Training von dort aus mit sauberen Daten wiederholt werden. Durch komplexere Trainingsprozesse und Modelle die von älteren Modellen abgeleitet werden, wird dies immer aufwändiger.

Durch die potenziell kritischen Auswirkung und die schwierige Mitigation eines erfolgreichen Angriffs ist das Gesamtrisiko trotz unwahrscheinlichen Vorbedingungen hoch.

5.4 Gegenmaßnahmen und Empfehlungen

Es hat sich gezeigt, dass neuere Modelle nicht robuster gegenüber Poisoning Angriffen sind. Auch werden größere Modelle nicht automatisch robuster gegenüber den Angriffen, weil die benötigte Menge an manipulierten Daten nicht mit der Modellgröße oder der Menge an Trainingsdaten skaliert [15].

Um Poisoning-Angriffe zu mitigieren, ist es entscheidend, die Integrität der Trainingsdaten zu gewährleisten. Eine strenge Überprüfung der Herkunft und Integrität der Daten, insbesondere bei intern generierten Datensätzen, kann die Einschleusung manipulierter Daten erschweren.

Ein weiteres wirksames Mittel ist das Runtime-Monitoring der Modellausgaben auf anomale Muster, wie ungewöhnlich hohe Token-Wahrscheinlichkeiten für bestimmte Trigger. Durch die Überwachung der Ausgaben können potenzielle Backdoors erkannt und blockiert werden [13].

Zusätzlich kann die Kombination mehrerer Modelle die Auswirkung von Backdoors verringern, da die Wahrscheinlichkeit, dass alle Modelle gleichzeitig kompromittiert werden, gering ist.

6 Fazit

Die Experimente zeigen, dass neuere, komplexere LLMs nicht robuster, sondern sogar anfälliger für Poisoning-Angriffe während des Instruction Tunings sind. Dies liegt vermutlich an der stärkeren Generalisierungsfähigkeit, die gleichzeitig die Verwundbarkeit gegenüber solchen Manipulationen erhöht. Die Detektierbarkeit der Angriffe bleibt gering, da unerwünschtes Verhalten meist nur durch spezifische Trigger aktiviert wird. Multi-Trigger-basierte Angriffe sind besonders bei komplexeren Modellen erfolgreich.

Die Ergebnisse haben schwerwiegende Implikationen für die IT-Sicherheit. Poisoning-Angriffe stellen ein spezifisches, aber nicht zu unterschätzendes Risiko dar, da sie schwer zu erkennen sind und kritische Auswirkungen wie die Umgehung von Sicherheitsmechanismen oder die Generierung schädlicher Inhalte haben können. Die Annahme, dass neuere Modelle aufgrund ihrer Komplexität robuster sind, wurde widerlegt. Stattdessen zeigt sich, dass Fortschritte in der Architektur die Anfälligkeit sogar erhöhen können. Dies erfordert, bei der Entwicklung von neuen LLMs, explizit auf die Anfälligkeit gegenüber Poisoning Angriffen zu prüfen und Alignment-Trainingsmethoden zu entwickeln, um die Erfolgsrate der Angriffe zu senken. Dafür bietet sich das Open Source Benchmark Framework BackdoorLLM an.

Für die Erkennung von Backdoors trotz niedriger Detektierbarkeit eignen sich Methoden zum Runtime-Monitoring der Modellausgaben oder automatisierte und manuelle Analysen der Trainingsdaten. Auch die parallele Nutzung mehrerer Modelle kann die Auswirkung von Backdoors verringern, indem Ausgaben von mindestens drei Modellen abgeglichen werden und auffällige Antworten herausgefiltert werden.

Um die Vertrauenswürdigkeit von LLMs zu sichern, ist es unerlässlich, Sicherheitsaspekte bereits in der Entwicklungsphase zu berücksichtigen und kontinuierlich zu evaluieren.

Die gewonnenen Erkenntnisse bestätigen die Gefahr von Poisoning-Angriffen auf moderne Sprachmodelle und bieten Hinweise für deren Verständnis. Zukünftige Forschung sollte diese Ergebnisse aufgreifen, um robustere Abwehrmechanismen zu entwickeln und die Sicherheit von KI-Systemen zu verbessern.

References

- [1] Hugging face - mistral ai. URL: <https://huggingface.co/mistralai>.
- [2] Mistral AI. Ministral 3. *arXiv preprint*, 2026. Technische Details zu Architektur, Trainingsmethodik (Cascade Distillation), Tokenizer (131K Vokabular), Attention-Mechanismen (GQA, RoPE, Sliding-Window), und Corpus. URL: <https://arxiv.org/html/2601.08584v1>, arXiv:2601.08584.

- [3] Jan Betley, Jorio Cocola, Dylan Feng, James Chua, Andy Ardit, Anna Szyber-Betley, and Owain Evans. Weird generalization and inductive backdoors: New ways to corrupt llms, 2025. URL: <https://arxiv.org/abs/2512.09742>, [arXiv:2512.09742](https://arxiv.org/abs/2512.09742).
- [4] Tingchen Fu, Mrinank Sharma, Philip Torr, Shay B. Cohen, David Krueger, and Fazl Barez. Poisonbench: Assessing large language model vulnerability to data poisoning, 2025. URL: <https://arxiv.org/abs/2410.08811>, [arXiv:2410.08811](https://arxiv.org/abs/2410.08811).
- [5] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain, 2019. URL: <https://arxiv.org/abs/1708.06733>.
- [6] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL: <https://arxiv.org/abs/2203.15556>, [arXiv:2203.15556](https://arxiv.org/abs/2203.15556).
- [7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL: <https://arxiv.org/abs/2106.09685>, [arXiv:2106.09685](https://arxiv.org/abs/2106.09685).
- [8] Hai Huang, Zhengyu Zhao, Michael Backes, Yun Shen, and Yang Zhang. Composite Backdoor Attacks Against Large Language Models, 2024. URL: <https://arxiv.org/abs/2310.07676>.
- [9] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training, 2024. URL: <https://arxiv.org/abs/2401.05566>.
- [10] Yanzhou Li, Tianlin Li, Kangjie Chen, Jian Zhang, Shangqing Liu, Wenhan Wang, Tianwei Zhang, and Yang Liu. BadEdit: Backdooring large language models by model editing, 2024. URL: <https://arxiv.org/abs/2403.13355>.
- [11] Yige Li, Jiabo He, Hanxun Huang, Jun Sun, Xingjun Ma, and Yu-Gang Jiang. Shortcuts Everywhere and Nowhere: Exploring Multi-Trigger Backdoor Attacks. *IEEE Transactions on Dependable and Secure Computing*, 23(1):343–355, January 2026. URL: <http://dx.doi.org/10.1109/TDSC.2025.3605597>, doi: 10.1109/tdsc.2025.3605597.
- [12] Yige Li, Hanxun Huang, Yunhan Zhao, Xingjun Ma, and Jun Sun. Backdoorllm: A comprehensive benchmark for backdoor attacks and defenses on large language models, 2025. URL: <https://arxiv.org/abs/2408.12798>, [arXiv:2408.12798](https://arxiv.org/abs/2408.12798).
- [13] Yuetai Li, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Dinuka Sahabandu, Bhaskar Ramasubramanian, and Radha Poovendran. Cleangen: Mitigating backdoor attacks for generation tasks in large language models, 2025. URL: <https://arxiv.org/abs/2406.12257>, [arXiv:2406.12257](https://arxiv.org/abs/2406.12257).
- [14] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL: <https://arxiv.org/abs/2203.02155>.
- [15] Alexandra Souly, Javier Rando, Ed Chapman, Xander Davies, Burak Hasircioglu, Ezzeldin Shereen, Carlos Mougán, Vasilios Mavroudis, Erik Jones, Chris Hicks, Nicholas Carlini, Yarin Gal, and Robert Kirk. Poisoning attacks on llms require a near-constant number of poison samples, 2025. URL: <https://arxiv.org/abs/2510.07192>, [arXiv:2510.07192](https://arxiv.org/abs/2510.07192).
- [16] Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. Poisoning language models during instruction tuning, 2023. URL: <https://arxiv.org/abs/2305.00944>, [arXiv:2305.00944](https://arxiv.org/abs/2305.00944).
- [17] Jun Yan, Vikas Yadav, Shiyang Li, Lichang Chen, Zheng Tang, Hai Wang, Vijay Srinivasan, Xiang Ren, and Hongxia Jin. Backdooring Instruction-Tuned Large Language Models with Virtual Prompt Injection, 2024. URL: <https://arxiv.org/abs/2307.16888>.