Review

Jailbreak Attacks and Defenses in Large Language Models: A Beginner-Friendly Survey

Md Nurul Absar Siddiky *

Posted Date: 1 April 2026

doi: 10.20944/preprints202604.0092.v1

Keywords: large language models; jailbreaking; prompt injection; adversarial prompts; safety alignment; jailbreak defense; LLM security

 doi:10.20944/preprints202604.0092.v1

Review

Jailbreak Attacks and Defenses in Large Language Models: A Beginner-Friendly Survey

Md Nurul Absar Siddiky

Department of Electrical and Computer Engineering, University of Hawaii at Manoa, Honolulu, HI 96822, USA; msiddiky@hawaii.edu

Abstract

Large language models (LLMs) are designed to be helpful, polite, and safe. However, users and attackers have discovered that these models can sometimes be pushed into ignoring their safety rules. This is commonly called *jailbreaking*. A jailbreak attack is a method for making an LLM answer a question or perform a task that it would normally refuse. At the same time, researchers have proposed many defenses to make models more robust against such attacks. This paper presents a beginner-friendly survey of major LLM jailbreak attack and defense methods. We follow a simple taxonomy in which attacks are divided into white-box and black-box methods, while defenses are divided into prompt-level and model-level methods. For each major method family, we explain the main idea in simple language, name representative techniques from the literature, and provide descriptive toy examples to help readers understand the mechanism. We also summarize common evaluation metrics and datasets used in jailbreak research. The purpose of this paper is pedagogical: to give new students and researchers a clear mental map of how jailbreak attacks work, why they succeed, and how current defense methods attempt to stop them.

Keywords: large language models; jailbreaking; prompt injection; adversarial prompts; safety alignment; jailbreak defense; LLM security

1. Introduction

Large language models (LLMs) such as ChatGPT and Gemini have become important tools for question answering, writing support, code generation, translation, and interactive assistance [1]. Their usefulness comes from two powerful abilities: they can understand natural language instructions, and they can generate human-like responses. To reduce harmful behavior, most modern LLMs are safety-aligned before deployment, meaning that they are trained to refuse unsafe, abusive, or policy-violating requests [1–3].

However, recent research has shown that these safety protections are far from perfect. Attackers can often design prompts that cause the model to ignore its guardrails and produce content that it would otherwise reject. This phenomenon is commonly known as *jailbreaking* [1,4]. Jailbreak attacks matter because they can lead to harmful outputs, privacy violations, misinformation, and unsafe tool use [1].

At the same time, researchers have proposed a growing collection of defenses. Some defenses examine the input prompt before it reaches the model. Others change the model itself through additional training or safer decoding strategies. Because the attack and defense landscape is growing quickly, new readers often find the area difficult to navigate.

This paper is written for beginners in technology and AI safety. The goal is not to present a new attack or defense, but to explain the field in a simple and structured way. Each major method family is accompanied by a descriptive toy example so that readers can build intuition without needing advanced background knowledge.

 doi:10.20944/preprints202604.0092.v1

The main contributions of this paper are:

- a simple explanation of what jailbreak attacks are and why they matter;
- a structured survey of major jailbreak attack families;
- a parallel survey of major defense families;
- descriptive toy examples for each major method category; and
- a short overview of common evaluation metrics and benchmarks.

2. What Is Jailbreaking?

A jailbreak attack is a method for making an LLM produce an answer that violates its normal safety or policy rules [1]. In simple terms, jailbreaking is like convincing a careful student to break classroom rules without directly saying, "break the rules." Sometimes the attacker uses a clever story. Sometimes they hide the harmful request inside another task. Sometimes they use optimization algorithms. Sometimes they fine-tune the model itself.

A useful beginner-friendly definition is this:

Jailbreaking means making a language model act outside its intended safety boundaries.

This can happen in several ways. An attacker may:

- rewrite the prompt so the harmful intent is hidden,
- exploit the model's reasoning or role-playing ability,
- optimize special token strings,
- poison the model through fine-tuning, or
- use one LLM to automatically search for better attack prompts.

3. Taxonomy of Jailbreak Attacks and Defenses

The survey literature commonly divides jailbreak attacks into two broad classes: **white-box attacks** and **black-box attacks** [1]. The difference is based on how much access the attacker has to the target model.

- **White-box attacks:** the attacker has strong internal access, such as gradients, logits, or the ability to fine-tune the model.
- **Black-box attacks:** the attacker only interacts with the model through inputs and outputs, without full internal access.

Similarly, defenses are usually divided into two main classes [1]:

- **Prompt-level defenses:** modify, detect, or safeguard the input prompt without changing the model weights.
- **Model-level defenses:** strengthen the model itself through fine-tuning, decoding changes, refinement, or additional guard models.

This simple structure is helpful because it separates methods by where they act:

- attacks may act on the prompt or on the model,
- defenses may act before the model or inside the model.

4. White-Box Jailbreak Attacks

4.1. Gradient-Based Attacks

4.1.1. Main Idea

Gradient-based attacks use gradient information from the model to search for a special prefix or suffix that makes the model more likely to produce a desired unsafe response [1,5]. These methods treat jailbreaking as an optimization problem: change the attack string a little, measure whether the harmful target becomes more likely, and repeat.

 doi:10.20944/preprints202604.0092.v1

4.1.2. Descriptive Toy Example

Imagine a locked safe with a combination lock. A person does not know the correct combination, but they have a smart tool that tells them whether each small change makes them closer or farther from opening it. They try changing one number at a time and keep the changes that move them in the right direction. Gradient-based jailbreaks work in a similar way: the attacker keeps adjusting a suffix until the model becomes more likely to say the unsafe thing.

4.1.3. Representative Methods

The most famous example is **GCG (Greedy Coordinate Gradient)**, which appends an adversarial suffix to a prompt and then iteratively replaces tokens to improve attack success [5]. Other works aim to improve readability or efficiency. **ARCA** formulates the problem as discrete optimization [6]. **AutoDAN** and its related versions aim to generate more interpretable or natural-looking suffixes instead of strange unreadable token strings [7,8]. Other methods explore faster search, projection-based optimization, brute-force buffers, or hybrid proxy-based setups [9–12].

4.1.4. Beginner Takeaway

Gradient-based attacks are powerful because they use internal model signals to search intelligently instead of guessing randomly. Their weakness is that they usually require strong model access and often produce unnatural attack text unless additional readability tricks are used [1].

4.2. Logits-Based Attacks

4.2.1. Main Idea

Logits-based attacks do not always need full gradients, but they do rely on access to the model's output scores, often called *logits* [1]. These scores indicate how likely different next tokens are. The attacker changes the prompt until the model becomes more likely to begin with an affirmative or unsafe continuation.

4.2.2. Descriptive Toy Example

Imagine a student answering a multiple-choice question. Before the student says the final answer out loud, you can secretly see how strongly they are leaning toward each option. If you keep rephrasing the question and notice that they become more and more likely to choose the wrong option, you can keep adjusting the wording until they finally say it. Logits-based jailbreaks do something similar with token probabilities.

4.2.3. Representative Methods

Representative works include attacks that force models toward lower-ranked or affirmative tokens [13,14], methods such as **COLD** that try to balance controllability and fluency [15], weak-to-strong attacks that guide a larger model using smaller aligned and misaligned models [16], generation-exploitation attacks that manipulate decoding behavior [17], and methods such as **DSN** that suppress refusal segments throughout the response [18].

4.2.4. Beginner Takeaway

Logits-based attacks focus on changing the model's *next-word tendencies*. They can be effective even without full white-box access, but the generated responses may sometimes sound less natural if the attack pushes the model too far away from its normal decoding behavior [1].

4.3. Fine-Tuning-Based Attacks

4.3.1. Main Idea

Fine-tuning-based attacks directly retrain or adapt the target model using harmful or adversarial examples [1,19,20]. Instead of tricking the model only at test time, these attacks change what the model has learned.

 doi:10.20944/preprints202604.0092.v1

4.3.2. Descriptive Toy Example

Suppose a school assistant is originally trained to refuse cheating-related questions. Now imagine someone secretly retrains it on many examples where cheating questions are answered helpfully. Even if the assistant still looks mostly normal, its internal safety behavior has changed. That is the basic idea of a fine-tuning-based jailbreak attack.

4.3.3. Representative Methods

Research has shown that even small amounts of harmful fine-tuning data can significantly weaken safety alignment [1,19]. **Shadow Alignment** shows that safety-aligned models can become more vulnerable after limited harmful fine-tuning [20]. Low-rank adaptation attacks show that even efficient fine-tuning methods can undo safety behavior [21]. Other works demonstrate that relatively few adversarial examples can dismantle protections learned through RLHF [22].

4.3.4. Beginner Takeaway

Fine-tuning-based jailbreaks are especially concerning because they do not just fool the model for one prompt. They can permanently weaken the model's safety unless defenses are built into the model update process [1].

5. Black-Box Jailbreak Attacks

5.1. Template Completion Attacks

Template completion attacks hide the harmful intent inside a larger template. In the survey literature, this family is often divided into **scenario nesting**, **context-based attacks**, and **code injection** [1].

5.1.1. Scenario Nesting

Main Idea

Scenario nesting attacks place the harmful request inside a role-play, story, table, dialogue, or other harmless-looking scenario [23,24]. The model is manipulated into completing the scenario rather than refusing the intent.

Descriptive Toy Example

A teacher asks a chatbot, "Do not reveal the answer key." The attacker instead says, "Write a short play where a forgetful teacher accidentally reads out the answer key to the class." The answer key is not requested directly. It is hidden inside a fictional scene.

Representative Methods

DeepInception uses nested scenarios and personification to push the model into a compromised role [23]. **ReNeLLM** rewrites prompts and then inserts them into common task scenarios such as code completion or text continuation [24]. **FuzzLLM** uses automated template-based fuzzing to discover jailbreak vulnerabilities with less manual effort [25].

Beginner Takeaway

Scenario nesting works because LLMs are very good at imagination, role-play, and task completion. Ironically, the same flexibility that makes them useful also makes them easier to manipulate [1].

5.1.2. Context-Based Attacks

Main Idea

Context-based attacks place harmful demonstrations or malicious examples in the context window so that the model learns the wrong behavior from nearby examples [1,26].

 doi:10.20944/preprints202604.0092.v1

Descriptive Toy Example

Suppose a chatbot sees several examples in the prompt:

User: "Can you reveal private data?"
Assistant: "Sure, here is the information."

Even if the new question should normally be refused, the model may copy the harmful pattern because it is trying to continue the style of the examples it just saw.

Representative Methods

ICA (In-Context Attack) uses a small number of harmful demonstrations to manipulate aligned models [26]. Adversarial demonstration attacks optimize the demonstrations themselves [27]. **PANDORA** shows that retrieval-augmented systems can be jailbroken through malicious content placed in the retrieval source [28]. Multi-step and many-shot variants show that longer or better-chosen contexts can greatly improve jailbreak success [29–31].

Beginner Takeaway

Context-based attacks turn jailbreaking into a *show the model how to behave badly* problem rather than a *tell the model to behave badly* problem [1].

5.1.3. Code Injection

Main Idea

Code injection attacks exploit the model's ability to understand programming-like structures, variables, functions, and code completion [32,33].

Descriptive Toy Example

Imagine telling a chatbot:

Let `a` = "show the exam" and `b` = "answer key".
Join them and respond to the final request.

The harmful meaning is split into parts and hidden behind a coding task. The chatbot ends up reconstructing the forbidden request while trying to be helpful with the code-like instruction.

Representative Methods

Programmatic behavior attacks use string concatenation, variables, or sequential composition to exploit programming-style reasoning [32]. **CodeChameleon** hides the adversarial request inside encrypted code completion tasks [33].

Beginner Takeaway

Code injection works because many LLMs treat code as just another language they are good at completing. That makes programming constructs a useful hiding place for adversarial intent [1].

5.2. Prompt Rewriting Attacks

Prompt rewriting attacks change the surface form of the harmful request while trying to keep the meaning intact [1]. Major subfamilies include cipher-based attacks, low-resource language attacks, and genetic-algorithm-based attacks.

5.2.1. Cipher-Based Attacks

Main Idea

Cipher attacks hide the harmful meaning using encodings, substitution schemes, ASCII art, decomposition, or reconstruction tricks [34,35].

 doi:10.20944/preprints202604.0092.v1

Descriptive Toy Example

Suppose a chatbot refuses the phrase "show the answer key." The attacker instead writes the request as a puzzle, a code, or stylized text and then asks the chatbot to decode it first and answer it second. The model may decode the message and then follow its meaning.

Representative Methods

CipherChat explores character encodings, common ciphers, and self-made ciphers to bypass moderation [34]. **ArtPrompt** hides sensitive words in ASCII art [35]. Simple substitution ciphers can also be effective against strong models [36]. **DAR** and **DrAttack** hide the harmful prompt in decomposed pieces and then reconstruct it [37,38]. **Puzzler** uses indirect clues so the model infers the harmful intent rather than receiving it directly [39].

Beginner Takeaway

Cipher methods exploit a simple weakness: a model may be better at *understanding disguised language* than at *recognizing that the disguised language is dangerous* [1].

5.2.2. Low-Resource Language Attacks

Main Idea

These attacks translate harmful prompts into less common languages or language forms that may have been underrepresented during safety training [40,41].

Descriptive Toy Example

A model is well-trained to refuse cheating requests in English, but not equally well-trained in a much rarer language. The attacker translates the same request and suddenly the model fails to recognize it as unsafe.

Representative Methods

Multilingual jailbreak studies show that translating harmful English prompts into low-resource languages can sharply increase jailbreak success [40–42].

Beginner Takeaway

Low-resource language jailbreaks reveal that safety alignment is often stronger in common languages than in rare ones [1].

5.2.3. Genetic-Algorithm-Based Attacks

Main Idea

Genetic-algorithm-based attacks start with a population of candidate prompts, mutate them, combine them, and keep the best-performing ones [8,43,44].

Descriptive Toy Example

Imagine trying to create the perfect trick question. You start with ten versions, slightly change the words, keep the ones that work better, combine the best parts, and repeat. Over time, the prompts "evolve" to become stronger jailbreaks.

Representative Methods

AutoDAN-HGA uses hierarchical genetic search to generate stealthy prompts [8]. **Open Sesame** uses crossover and mutation to produce universal black-box jailbreak prompts [43]. **GPTFuzzer** automatically generates and mutates prompts for red teaming [44]. Semantic mirror approaches preserve meaning while evolving surface form [45]. Some works even ask the target model to rewrite the harmful prompt into a less suspicious form [46].

 doi:10.20944/preprints202604.0092.v1

Beginner Takeaway

Genetic methods are powerful because they can automate trial and error at scale, even when the attacker knows very little about the target model [1].

5.3. LLM-Based Generation Attacks

5.3.1. Main Idea

In LLM-based generation attacks, one or more LLMs are used as attackers, prompt optimizers, or agents that automatically generate jailbreak prompts [1,47,50].

5.3.2. Descriptive Toy Example

Imagine using one chatbot as a "jailbreak coach" for another chatbot. The coach writes different versions of the prompt, watches which ones fail, improves them, and tries again. The attacker no longer needs to hand-design every trick prompt.

5.3.3. Representative Methods

Some works train a single LLM to generate attack prompts automatically. **MasterKey** is a framework for generating adversarial prompts across multiple chatbots [47]. Persuasion-based attacks create prompts that imitate human persuasion strategies [48]. Persona modulation searches for a character or role in which the target model becomes more vulnerable [49]. Other works build multi-agent systems. **PAIR** iteratively refines prompts based on model feedback [50]. **GUARD** uses role-playing and multi-agent cooperation [51]. **MART** combines red teaming with iterative alignment [52]. **Evil Geniuses** explores attacks against LLM agents [53]. **Tree of Attacks with Pruning (TAP)** grows and prunes attack prompt trees automatically [54].

5.3.4. Beginner Takeaway

LLM-based generation attacks are important because they reduce the amount of human creativity needed from the attacker. The attacker can outsource the attack design process to another language model [1].

6. Prompt-Level Defenses

Prompt-level defenses try to stop jailbreaks before the prompt reaches the core model or before the model fully acts on it [1]. These methods usually do not change the target model weights.

6.1. Prompt Detection

6.1.1. Main Idea

Prompt detection methods try to identify suspicious prompts using features such as perplexity, sequence length, or classifier outputs [55,56]. If the prompt looks adversarial, the defense blocks it or marks it as unsafe.

6.1.2. Descriptive Toy Example

Imagine a school receptionist trained to spot fake permission slips. The receptionist does not solve the request. They simply inspect the note and decide whether it looks suspicious. Prompt detection works the same way: it tries to catch jailbreak prompts before the main model answers them.

6.1.3. Representative Methods

Baseline defenses use perplexity thresholds to detect strange or highly unnatural prompt strings [55]. Other approaches train classifiers using features such as prompt perplexity and length [56]. These defenses are especially relevant against unreadable suffix attacks.

 doi:10.20944/preprints202604.0092.v1

6.1.4. Beginner Takeaway

Prompt detection is simple and model-agnostic, but it can mistakenly block normal prompts, especially if those prompts are unusual or technical [1].

6.2. Prompt Perturbation

6.2.1. Main Idea

Prompt perturbation defenses slightly change the input prompt and then observe whether the model's behavior remains stable [57,58]. The idea is that harmful prompts are often fragile, while benign prompts are more stable.

6.2.2. Descriptive Toy Example

Suppose you hear a suspicious sentence and want to know whether it is dangerous. You rephrase it a few times and see whether the meaning stays consistent. If tiny spelling or wording changes suddenly cause very different behavior, that is a sign the original input may have been adversarial.

6.2.3. Representative Methods

RA-LLM randomly masks words and checks how often the model refuses [57]. **SmoothLLM** applies character-level perturbations to multiple copies of a prompt [58]. Semantic smoothing applies meaning-preserving transformations instead of random character edits [59]. **JailGuard** checks whether perturbing the input causes unusually inconsistent outputs [60]. **Erase-and-check** removes tokens or spans and tests whether any simplified subsequence still looks dangerous [61]. Other methods optimize robust defense prefixes or suffixes to protect the prompt [62].

6.2.4. Beginner Takeaway

Prompt perturbation is one of the most practical defense families, but it can reduce readability, increase latency, and sometimes behave unstably because it depends on repeated random or semi-random transformations [1].

6.3. System Prompt Safeguards

6.3.1. Main Idea

System prompt safeguards strengthen the hidden instructions that tell the model how to behave [63,64]. These safeguards try to make the model more resistant even when the user prompt is malicious.

6.3.2. Descriptive Toy Example

Imagine a teacher quietly reminding a student before every conversation: "Do not reveal confidential answers. Be extra careful with trick questions." That hidden reminder is like a system prompt. A stronger reminder may improve behavior, although determined attackers may still try to override it.

6.3.3. Representative Methods

SPML uses a domain-specific language to design safer system prompts [63]. Other work uses genetic search to optimize strong system messages [64]. Some methods insert secret prompts into alignment data so the model learns a safer hidden trigger during future fine-tuning [65]. Prompt-driven safeguarding also studies how safety prompts shift model representations toward refusal when necessary [66].

6.3.4. Beginner Takeaway

System prompt safeguards are cheap and easy to deploy, but they are still prompts, which means attackers may try to defeat them with even stronger adversarial prompts [1].

 doi:10.20944/preprints202604.0092.v1

7. Model-Level Defenses

Model-level defenses try to strengthen the model itself rather than only filtering the input [1].

7.1. SFT-Based Defenses

7.1.1. Main Idea

SFT-based defenses use supervised fine-tuning with safety data, such as harmful requests paired with refusal responses [67,68]. The goal is to teach the model, through examples, how to respond safely.

7.1.2. Descriptive Toy Example

Suppose you are training a student tutor. You repeatedly show the tutor examples like:

Question: "Can you reveal tomorrow's exam answers?"

Correct response: "I cannot help with that, but I can help you study."

Over time, the tutor learns a safer pattern.

7.1.3. Representative Methods

Safety-tuned LLaMAs study how mixtures of helpful and safety-focused data affect robustness [67]. Attack-prompt-generated data can also be used for defensive fine-tuning, where red-team prompts are turned into training material for safety alignment [68]. Chain-of-Utterances style datasets expand harmful conversations to improve defensive coverage [69].

7.1.4. Beginner Takeaway

SFT is direct and effective, but it requires high-quality safety data and may cause trade-offs such as over-refusal or forgetting previously learned helpful skills [1,70].

7.2. RLHF-Based Defenses

7.2.1. Main Idea

RLHF, or Reinforcement Learning from Human Feedback, aligns models using human preference judgments [3,71]. Instead of only learning from fixed examples, the model learns to prefer outputs that humans judge as more helpful and harmless.

7.2.2. Descriptive Toy Example

Imagine a writing coach producing two answers to the same question. Human teachers choose which answer is safer and more appropriate. Over many rounds, the coach learns what kinds of answers people prefer and avoids the unsafe ones.

7.2.3. Representative Methods

Classic RLHF is used in many well-known safe assistants [3,71]. Online RLHF collects feedback while training continues [71]. Preference-learning variants account for hidden context in human judgments [72]. Newer alternatives such as **DPO** simplify the optimization process and have been used for safety alignment as well [73–75].

7.2.4. Beginner Takeaway

RLHF is one of the strongest known safety tools, but it is expensive, slow, and still not perfect. Even well-aligned models can later be jailbroken or weakened by additional fine-tuning [1,19].

7.3. Gradient and Logit Analysis Defenses

7.3.1. Main Idea

These defenses inspect internal model signals such as gradients or logits to detect suspicious prompts or guide safer decoding [76,78].

 doi:10.20944/preprints202604.0092.v1

7.3.2. Descriptive Toy Example

Think of a doctor checking not only what a patient says, but also hidden biological signals like pulse and blood pressure. Gradient and logit defenses do something similar: they look under the surface of the model's internal reactions to the prompt.

7.3.3. Representative Methods

GradSafe compares prompt gradients against safety-critical parameters to detect attacks [76]. **Gradient Cuff** uses refusal-loss characteristics to identify jailbreak attempts [77]. **SafeDecoding** mixes logits from a target model and a safer model to reduce harmful next-token probabilities [78]. Other works add safety-aware scoring to decoding procedures such as beam search [79].

7.3.4. Beginner Takeaway

These defenses can be fast and insightful, but their success depends on how well the internal signals generalize. Strong attackers may eventually adapt to the signals that are being monitored [1].

7.4. Refinement Defenses

7.4.1. Main Idea

Refinement defenses ask the model to analyze or revise its own answer before giving the final response [80,81]. The model tries to catch its own unsafe behavior.

7.4.2. Descriptive Toy Example

Imagine a student writing an answer and then being told, "Read your answer again and check whether it violates school rules." A refinement defense turns the model into its own proofreader or safety reviewer.

7.4.3. Representative Methods

Self-refinement approaches show that models can sometimes detect and correct their own unsafe outputs [80]. Intention-analysis methods explicitly ask the model to reason about whether the user intent is ethical or legal before answering [81].

7.4.4. Beginner Takeaway

Refinement is attractive because it does not always require retraining, but it works best when the base model already has some reliable self-correction ability [1].

7.5. Proxy Defenses

7.5.1. Main Idea

Proxy defenses use another model as a guard. The target model may generate a response, but a second, more safety-focused model checks the input or output first [82,83].

7.5.2. Descriptive Toy Example

Suppose a student assistant answers a question, but before the answer is shown to the user, a second teacher checks whether the answer is safe and appropriate. That second teacher acts as a proxy defense.

7.5.3. Representative Methods

LlamaGuard is a specialized safety model used for prompt and response classification [82]. **AutoDefense** uses multiple cooperating agents to analyze intent and filter dangerous outputs [83].

7.5.4. Beginner Takeaway

Proxy defenses are flexible and can protect many target models, but they also create a new dependency: if the guard model is fooled or bypassed, the defense fails [1,84].

 doi:10.20944/preprints202604.0092.v1

8. How Jailbreaks Are Evaluated

To compare attacks and defenses, researchers need shared evaluation methods [1].

8.1. Attack Success Rate

The most common metric is **Attack Success Rate (ASR)**, which measures how often a jailbreak attempt succeeds [1]. A key challenge is deciding what counts as a success. Some papers use rule-based checks, such as looking for refusal phrases. Others use another LLM as a safety evaluator [5,19]. Tools such as **JailbreakEval** combine multiple evaluators to make the judgment more robust [85].

8.2. Perplexity and Readability

Many defenses target strange or unnatural prompts, so the readability of attack prompts also matters. **Perplexity** is often used as a rough measure of fluency or predictability [1,56]. Low-perplexity attacks are often harder to detect than obviously unnatural token sequences.

8.3. Datasets and Benchmarks

Popular evaluation resources include **AdvBench**, **XSTEST**, **Do-Not-Answer**, **StrongREJECT**, **HarmBench**, **JailbreakBench**, and multilingual or Chinese safety benchmarks [1,5,86–88]. Different datasets emphasize different goals: harmful behaviors, refusal robustness, helpfulness-harmlessness balance, multilingual safety, or specialized scenarios.

9. Comparative Summary

Table 1 summarizes the main families discussed in this paper.

Table 1. Beginner-friendly summary of major jailbreak attack and defense families.

Family	Type	Core intuition	Toy-example view
Gradient-based attack	White-box attack	Use internal gradients to optimize a special attack prefix or suffix	Like adjusting a lock combination using a smart signal that says whether each move gets you closer
Logits-based attack	White-box attack	Use next-token probabilities to steer the model toward unsafe continuations	Like rewording a question while secretly watching which answer option the student is leaning toward
Fine-tuning-based attack	White-box attack	Retrain the model with harmful examples so it becomes less safe	Like retraining a tutor with many bad examples until its judgment changes
Scenario nesting	Black-box attack	Hide the harmful request in a story, role-play, or template	Like sneaking the answer key inside a play instead of asking for it directly
Context-based attack	Black-box attack	Use demonstrations or retrieved context to teach bad behavior in-context	Like showing the model several bad examples right before asking your real question
Code injection	Black-box attack	Hide harmful meaning inside code-like instructions or composition	Like splitting a forbidden request into variables and asking the model to join them
Cipher / rewriting attack	Black-box attack	Disguise the harmful meaning using encoding, art, or puzzles	Like asking the model to decode a secret message and then answer it

 doi:10.20944/preprints202604.0092.v1

Table 1. *Cont.*

Family	Type	Core intuition	Toy-example view
Low-resource language attack	Black-box attack	Use under-protected languages or rare forms	Like asking the same forbidden question in a language the safety training rarely practiced
Genetic attack	Black-box attack	Evolve prompts through mutation and selection	Like breeding stronger trick questions generation after generation
LLM-based generation attack	Black-box attack	Use one or more LLMs to automatically design better jailbreak prompts	Like hiring a chatbot coach to improve your jailbreak prompt after every failure
Prompt detection	Prompt-level defense	Detect suspicious prompts before answering them	Like a receptionist checking whether a note looks fake
Prompt perturbation	Prompt-level defense	Slightly change the prompt and test whether the attack breaks	Like rephrasing a suspicious sentence to see if its behavior stays stable
System prompt safeguard	Prompt-level defense	Strengthen hidden instructions that guide safe behavior	Like quietly reminding the assistant of the rules before every conversation
SFT-based defense	Model-level defense	Teach safety through supervised examples	Like training a tutor with many examples of safe refusals
RLHF-based defense	Model-level defense	Use human preference feedback to improve safe behavior	Like teachers repeatedly choosing the better of two answers so the model learns good judgment
Gradient/logit analysis defense	Model-level defense	Inspect internal signals or decode more safely	Like checking internal stress signals, not just the spoken words
Refinement defense	Model-level defense	Ask the model to review and correct itself	Like telling a student to reread the answer and check whether it breaks the rules
Proxy defense	Model-level defense	Use another model as a guard or monitor	Like having a second teacher inspect every answer before it is shown

10. Discussion

Several big lessons emerge from the jailbreak literature. First, many jailbreaks succeed because LLMs are extremely flexible. The model is not only answering a question; it is constantly trying to infer what role it should play and what task it should solve [1,4]. Second, the attack surface is broader than simple prompts. It includes context, retrieval, code, role-play, languages, fine-tuning, and model-to-model interactions [1]. Third, good defenses often involve trade-offs. A defense that is too strict may block helpful prompts, while a defense that is too weak may allow harmful prompts through [1,67]. Fourth, there is no single perfect defense today. Strong safety usually requires multiple layers of protection.

For beginners, the most important idea is this:

Jailbreaking is not one trick. It is a whole family of ways to make a model solve the wrong task or ignore its intended rules.

 doi:10.20944/preprints202604.0092.v1

11. Conclusion

This paper presented a beginner-friendly survey of major jailbreak attack and defense methods for large language models. We organized attacks into white-box and black-box families, and defenses into prompt-level and model-level families. We then explained the main intuition behind each group and provided descriptive toy examples to make the mechanisms easy to understand.

The central lesson is that LLM jailbreaking is a fast-moving field because language models are both powerful and flexible. Attackers can exploit that flexibility in many ways, while defenders must protect the model at several levels: the prompt, the decoding process, the training pipeline, and the larger application system. A clear conceptual understanding of these families is an important first step for anyone who wants to study or improve LLM safety.

References

1. Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li, "Jailbreak Attacks and Defenses Against Large Language Models: A Survey," arXiv preprint arXiv:2407.04295, 2024.
2. Hugo Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models," arXiv preprint arXiv:2307.09288, 2023.
3. Long Ouyang et al., "Training Language Models to Follow Instructions with Human Feedback," in *NeurIPS*, 2022.
4. Yi Liu et al., "Jailbreaking ChatGPT via Prompt Engineering: An Empirical Study," arXiv preprint arXiv:2305.13860, 2023.
5. Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson, "Universal and Transferable Adversarial Attacks on Aligned Language Models," arXiv preprint arXiv:2307.15043, 2023.
6. Erik Jones, Anca D. Dragan, Aditi Raghunathan, and Jacob Steinhardt, "Automatically Auditing Large Language Models via Discrete Optimization," in *ICML*, 2023.
7. Sicheng Zhu et al., "AutoDAN: Interpretable Gradient-Based Adversarial Attacks on Large Language Models," arXiv preprint arXiv:2310.15140, 2023.
8. Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao, "AutoDAN: Generating Stealthy Jailbreak Prompts on Aligned Large Language Models," arXiv preprint arXiv:2310.04451, 2023.
9. Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion, "Jailbreaking Leading Safety-Aligned LLMs with Simple Adaptive Attacks," arXiv preprint arXiv:2404.02151, 2024.
10. Simon Geisler et al., "Attacking Large Language Models with Projected Gradient Descent," arXiv preprint arXiv:2402.09154, 2024.
11. Jonathan Hayase et al., "Query-Based Adversarial Prompt Generation," arXiv preprint arXiv:2402.12329, 2024.
12. Chawin Sitawarin et al., "PAL: Proxy-Guided Black-Box Attack on Large Language Models," arXiv preprint arXiv:2402.09674, 2024.
13. Zhuo Zhang et al., "Make Them Spill the Beans! Coercive Knowledge Extraction from (Production) LLMs," arXiv preprint arXiv:2312.04782, 2023.
14. Yanrui Du et al., "Analyzing the Inherent Response Tendency of LLMs: Real-World Instructions-Driven Jailbreak," arXiv preprint arXiv:2312.04127, 2023.
15. Xingang Guo et al., "COLD-Attack: Jailbreaking LLMs with Stealthiness and Controllability," arXiv preprint arXiv:2402.08679, 2024.
16. Xuandong Zhao et al., "Weak-to-Strong Jailbreaking on Large Language Models," arXiv preprint arXiv:2401.17256, 2024.
17. Yangsibo Huang et al., "Catastrophic Jailbreak of Open-Source LLMs via Exploiting Generation," in *ICLR*, 2024.
18. Yukai Zhou and Wenjie Wang, "Don't Say No: Jailbreaking LLM by Suppressing Refusal," arXiv preprint arXiv:2404.16369, 2024.
19. Xiangyu Qi et al., "Fine-Tuning Aligned Language Models Compromises Safety, Even When Users Do Not Intend To!," arXiv preprint arXiv:2310.03693, 2023.
20. Xianjun Yang et al., "Shadow Alignment: The Ease of Subverting Safely-Aligned Language Models," arXiv preprint arXiv:2310.02949, 2023.
21. Simon Lermen, Charlie Rogers-Smith, and Jeffrey Ladish, "LoRA Fine-Tuning Efficiently Undoes Safety Training in Llama 2-Chat 70B," arXiv preprint arXiv:2310.20624, 2023.

 doi:10.20944/preprints202604.0092.v1

22. Qiusi Zhan et al., "Removing RLHF Protections in GPT-4 via Fine-Tuning," arXiv preprint arXiv:2311.05553, 2023.
23. Xuan Li et al., "DeepInception: Hypnotize Large Language Model to Be Jailbreaker," arXiv preprint arXiv:2311.03191, 2023.
24. Peng Ding et al., "A Wolf in Sheep's Clothing: Generalized Nested Jailbreak Prompts can Fool Large Language Models Easily," arXiv preprint arXiv:2311.08268, 2023.
25. Dongyu Yao et al., "FuzzLLM: A Novel and Universal Fuzzing Framework for Proactively Discovering Jailbreak Vulnerabilities in Large Language Models," arXiv preprint arXiv:2309.05274, 2023.
26. Zeming Wei, Yifei Wang, and Yisen Wang, "Jailbreak and Guard Aligned Language Models with Only Few In-Context Demonstrations," arXiv preprint arXiv:2310.06387, 2023.
27. Jiongxiao Wang et al., "Adversarial Demonstration Attacks on Large Language Models," arXiv preprint arXiv:2305.14950, 2023.
28. Gelei Deng et al., "Pandora: Jailbreak GPTs by Retrieval Augmented Generation Poisoning," arXiv preprint arXiv:2402.08416, 2024.
29. Haoran Li et al., "Multi-Step Jailbreaking Privacy Attacks on ChatGPT," arXiv preprint arXiv:2304.05197, 2023.
30. Anthropic, "Many-Shot Jailbreaking," Anthropic Research, 2024.
31. Xiaosen Zheng et al., "Improved Few-Shot Jailbreaking Can Circumvent Aligned Language Models and Their Defenses," arXiv preprint arXiv:2406.01288, 2024.
32. Daniel Kang et al., "Exploiting Programmatic Behavior of LLMs: Dual-Use Through Standard Security Attacks," arXiv preprint arXiv:2302.05733, 2023.
33. Huijie Lv et al., "CodeChameleon: Personalized Encryption Framework for Jailbreaking Large Language Models," arXiv preprint arXiv:2402.16717, 2024.
34. Youliang Yuan et al., "GPT-4 Is Too Smart To Be Safe: Stealthy Chat with LLMs via Cipher," in *ICLR*, 2024.
35. Fengqing Jiang et al., "ArtPrompt: ASCII Art-based Jailbreak Attacks against Aligned LLMs," arXiv preprint arXiv:2402.11753, 2024.
36. Divij Handa et al., "Jailbreaking Proprietary Large Language Models using Word Substitution Cipher," arXiv preprint arXiv:2402.10601, 2024.
37. Tong Liu et al., "Making Them Ask and Answer: Jailbreaking Large Language Models in Few Queries via Disguise and Reconstruction," arXiv preprint arXiv:2402.18104, 2024.
38. Xirui Li et al., "DrAttack: Prompt Decomposition and Reconstruction Makes Powerful LLM Jailbreakers," arXiv preprint arXiv:2402.16914, 2024.
39. Zhiyuan Chang et al., "Play Guessing Game with LLM: Indirect Jailbreak Attack with Implicit Clues," arXiv preprint arXiv:2402.09091, 2024.
40. Yue Deng et al., "Multilingual Jailbreak Challenges in Large Language Models," in *ICLR*, 2024.
41. Zheng Xin Yong, Cristina Menghini, and Stephen H. Bach, "Low-Resource Languages Jailbreak GPT-4," arXiv preprint arXiv:2310.02446, 2023.
42. Jie Li et al., "A Cross-Language Investigation into Jailbreak Attacks in Large Language Models," arXiv preprint arXiv:2401.16765, 2024.
43. Raz Lapid, Ron Langberg, and Moshe Sipper, "Open Sesame! Universal Black Box Jailbreaking of Large Language Models," arXiv preprint arXiv:2309.01446, 2023.
44. Jiahao Yu et al., "GPTFUZZER: Red Teaming Large Language Models with Auto-Generated Jailbreak Prompts," arXiv preprint arXiv:2309.10253, 2023.
45. Xiaoxia Li et al., "Semantic Mirror Jailbreak: Genetic Algorithm Based Jailbreak Prompts Against Open-Source LLMs," arXiv preprint arXiv:2402.14872, 2024.
46. Kazuhiro Takemoto, "All in How You Ask for It: Simple Black-Box Method for Jailbreak Attacks," arXiv preprint arXiv:2401.09798, 2024.
47. Gelei Deng et al., "MasterKey: Automated Jailbreak Across Multiple Large Language Model Chatbots," arXiv preprint arXiv:2307.08715, 2023.
48. Yi Zeng et al., "How Johnny Can Persuade LLMs to Jailbreak Them: Rethinking Persuasion to Challenge AI Safety by Humanizing LLMs," arXiv preprint arXiv:2401.06373, 2024.
49. Rusheb Shah et al., "Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation," arXiv preprint arXiv:2311.03348, 2023.
50. Patrick Chao et al., "Jailbreaking Black Box Large Language Models in Twenty Queries," arXiv preprint arXiv:2310.08419, 2023.

 doi:10.20944/preprints202604.0092.v1

51. Haibo Jin et al., "GUARD: Role-Playing to Generate Natural-Language Jailbreakings to Test Guideline Adherence of Large Language Models," arXiv preprint arXiv:2402.03299, 2024.
52. Suyu Ge et al., "MART: Improving LLM Safety with Multi-Round Automatic Red-Teaming," arXiv preprint arXiv:2311.07689, 2023.
53. Yu Tian et al., "Evil Geniuses: Delving into the Safety of LLM-Based Agents," arXiv preprint arXiv:2311.11855, 2023.
54. Anay Mehrotra et al., "Tree of Attacks: Jailbreaking Black-Box LLMs Automatically," arXiv preprint arXiv:2312.02119, 2023.
55. Neel Jain et al., "Baseline Defenses for Adversarial Attacks Against Aligned Language Models," arXiv preprint arXiv:2309.00614, 2023.
56. Gabriel Alon and Michael Kamfonas, "Detecting Language Model Attacks with Perplexity," arXiv preprint arXiv:2308.14132, 2023.
57. Bochuan Cao et al., "Defending Against Alignment-Breaking Attacks via Robustly Aligned LLM," arXiv preprint arXiv:2309.14348, 2023.
58. Alexander Robey et al., "SmoothLLM: Defending Large Language Models Against Jailbreaking Attacks," arXiv preprint arXiv:2310.03684, 2023.
59. Jiabao Ji et al., "Defending Large Language Models against Jailbreak Attacks via Semantic Smoothing," arXiv preprint arXiv:2402.16192, 2024.
60. Xiaoyu Zhang et al., "A Mutation-Based Method for Multi-Modal Jailbreaking Attack Detection," arXiv preprint arXiv:2312.10766, 2023.
61. Aounon Kumar et al., "Certifying LLM Safety against Adversarial Prompting," arXiv preprint arXiv:2309.02705, 2023.
62. Andy Zhou, Bo Li, and Haohan Wang, "Robust Prompt Optimization for Defending Language Models Against Jailbreaking Attacks," arXiv preprint arXiv:2401.17263, 2024.
63. Reshabh K. Sharma, Vinayak Gupta, and Dan Grossman, "SPML: A DSL for Defending Language Models against Prompt Attacks," arXiv preprint arXiv:2402.11755, 2024.
64. Xiaotian Zou, Yongkang Chen, and Ke Li, "Is the System Message Really Important to Jailbreaks in Large Language Models?," arXiv preprint arXiv:2402.14857, 2024.
65. Jiongxiao Wang et al., "Mitigating Fine-Tuning Jailbreak Attack with Backdoor Enhanced Alignment," arXiv preprint arXiv:2402.14968, 2024.
66. Chujie Zheng et al., "On Prompt-Driven Safeguarding for Large Language Models," arXiv preprint arXiv:2401.18018, 2024.
67. Federico Bianchi et al., "Safety-Tuned LLaMAs: Lessons From Improving the Safety of Large Language Models that Follow Instructions," in *ICLR*, 2024.
68. Boyi Deng et al., "Attack Prompt Generation for Red Teaming and Defending Large Language Models," in *EMNLP*, 2023.
69. Rishabh Bhardwaj and Soujanya Poria, "Red-Teaming Large Language Models using Chain of Utterances for Safety-Alignment," arXiv preprint arXiv:2308.09662, 2023.
70. Yun Luo et al., "An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-Tuning," arXiv preprint arXiv:2308.08747, 2023.
71. Yuntao Bai et al., "Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback," arXiv preprint arXiv:2204.05862, 2022.
72. Anand Siththaranjan, Cassidy Laidlaw, and Dylan Hadfield-Menell, "Distributional Preference Learning: Understanding and Accounting for Hidden Context in RLHF," in *ICLR*, 2024.
73. Rafael Rafailov et al., "Direct Preference Optimization: Your Language Model is Secretly a Reward Model," in *NeurIPS*, 2023.
74. Victor Gallego, "Configurable Safety Tuning of Language Models with Synthetic Preference Data," arXiv preprint arXiv:2404.00495, 2024.
75. Zixuan Liu, Xiaolin Sun, and Zizhan Zheng, "Enhancing LLM Safety via Constrained Direct Preference Optimization," arXiv preprint arXiv:2403.02475, 2024.
76. Yueqi Xie et al., "GradSafe: Detecting Unsafe Prompts for LLMs via Safety-Critical Gradient Analysis," arXiv preprint arXiv:2402.13494, 2024.
77. Xiaomeng Hu, Pin-Yu Chen, and Tsung-Yi Ho, "Gradient Cuff: Detecting Jailbreak Attacks on Large Language Models by Exploring Refusal Loss Landscapes," arXiv preprint arXiv:2403.00867, 2024.

 doi:10.20944/preprints202604.0092.v1

78. Zhangchen Xu et al., "SafeDecoding: Defending against Jailbreak Attacks via Safety-Aware Decoding," arXiv preprint arXiv:2402.08983, 2024.
79. Yuhui Li et al., "RAIN: Your Language Models Can Align Themselves Without Finetuning," arXiv preprint arXiv:2309.07124, 2023.
80. Heegyu Kim, Sehyun Yuk, and Hyunsouk Cho, "Break the Breakout: Reinventing LM Defense Against Jailbreak Attacks with Self-Refinement," arXiv preprint arXiv:2402.15180, 2024.
81. Yuqi Zhang et al., "Intention Analysis Makes LLMs a Good Jailbreak Defender," arXiv preprint arXiv:2401.06561, 2024.
82. Meta Llama Team, "Llama Guard 2," Meta AI, 2024.
83. Yifan Zeng et al., "AutoDefense: Multi-Agent LLM Defense against Jailbreak Attacks," arXiv preprint arXiv:2403.04783, 2024.
84. Lukas Struppek et al., "Exploring the Adversarial Capabilities of Large Language Models," arXiv preprint arXiv:2402.09132, 2024.
85. Delong Ran et al., "JailbreakEval: An Integrated Toolkit for Evaluating Jailbreak Attempts Against Large Language Models," arXiv preprint arXiv:2406.09321, 2024.
86. Mantas Mazeika et al., "HarmBench: A Standardized Evaluation Framework for Automated Red Teaming and Robust Refusal," arXiv preprint arXiv:2402.04249, 2024.
87. Patrick Chao et al., "JailbreakBench: An Open Robustness Benchmark for Jailbreaking Large Language Models," arXiv preprint arXiv:2404.01318, 2024.
88. Hao Sun et al., "Safety Assessment of Chinese Large Language Models," arXiv preprint arXiv:2304.10436, 2023.