LLM APPLICATIONS

Prompt Injection and Jailbreak Attacks on LLM Applications: A Review of Attacks, Defenses and Risk-Adaptive Protection

Prabhakarn, Birjendra Gupta

Department of Information Technology, Siddhant College of Engineering, Pune, India

Published November 2026

Abstract

Large language models (LLMs) are now embedded in chatbots, coding assistants, search tools and agents, and this has exposed a new class of security problems in which attackers use ordinary text to override a developer's instructions or to push a model past its safety training. This paper reviews the literature on these problems, focusing on prompt injection and jailbreak attacks. It organises attacks into a taxonomy (direct and indirect injection, manual, optimisation-based, LLM-driven, obfuscated, multi-turn, multimodal and training-time attacks), organises defenses into eight layers from alignment-time training to system-level controls, and summarises the benchmarks and metrics used to compare them. It then examines a theme that is discussed less often than attack and defense strength, namely the cost of defending: most published defenses are applied uniformly to every request, while research on LLM routing and adaptive moderation shows that matching effort to difficulty can save cost. The paper discusses risk-proportional defense as a design direction, and closes with open problems, including adaptive attackers, evaluation validity, over-refusal, multi-model deployments and agents. The review draws on more than one hundred papers and resources published between 2020 and 2026.

Keywords: adaptive defense; jailbreak; large language models; LLM security; prompt injection

1 Introduction

Large language models have moved from research prototypes to everyday infrastructure. After GPT-3 showed that one pretrained model could follow instructions given in plain text [1], instruction tuning and learning from human feedback made models easier to steer [2], [3], and a broad family of proprietary and open models followed [4]–[14]. These models are now wrapped in customer-support bots, coding assistants and document question-answering systems, often combined with external knowledge through retrieval-augmented generation [15].

The property that makes LLMs useful, treating natural-language text as instructions, also makes them hard to secure. A conventional program separates code from data. An LLM application joins the developer's instructions, the user's request and any retrieved material into one block of text, and the model has no built-in way to tell which parts deserve trust. Attackers exploit this with *prompt injection*, which hijacks the application's task [16], [17], and with *jailbreaks*, which defeat the model's safety training [18], [19]. Security bodies now list these among the leading risks for LLM systems [20]–[22].

Why a new review. Several surveys already cover LLM security and privacy [23], [24], adversarial attacks on LLMs [25], [26], and jailbreak attacks and defenses [27], [28], and the trustworthiness of aligned LLMs [29]. This review differs in three ways. First, it treats prompt injection and jailbreaks

together but keeps their definitions apart, since they have different goals and different defenses. Second, it organises defenses as a stack of layers placed along the request path, which makes it easier to see where each defense adds cost. Third, it devotes a section to *cost-aware and adaptive* protection, drawing on LLM routing research and adaptive moderation, because the practical question for a deployer is often not only "does this defense work" but "what does it cost on every request".

Scope and method. The review covers text-based LLM applications, with brief notes on multimodal and agent settings. It includes papers from 2020 to 2026 that propose, analyse or benchmark attacks and defenses, plus a small number of standards and tool resources. Papers were found by following citations from the benchmark and survey papers listed above, and by searching for each attack and defense family by name. The review is not a systematic review in the strict sense and does not claim to cover all work in this fast-moving area.

Contributions. (1) A taxonomy of attacks and a layered taxonomy of defenses, each with a summary table. (2) A summary of benchmarks, datasets and metrics, with notes on common pitfalls. (3) A discussion of risk-proportional defense as a design direction supported by routing and adaptive-moderation research. (4) A list of open problems.

Organisation. Background and definitions are given first. The following sections review attacks, defenses and evaluation, discuss cost-aware and adaptive protection, and list open challenges before the conclusion.

2 Background and definitions

2.1 Anatomy of an LLM application

An LLM application typically builds each model input from four kinds of text: a **system prompt** written by the developer, a **user prompt**, **retrieved context** from documents or web pages [15], and in agent systems the **tool outputs** that are fed back to the model [30], [31]. The model returns a response that the application displays or acts on. Open-source runtimes and model hubs make it easy to deploy such systems locally and to test defenses on several models [32]–[34].

Safety training adds a layer inside the model. Reinforcement learning from human feedback, constitutional methods and direct preference optimisation teach models to refuse harmful requests and to follow instructions helpfully [2], [3], [35], [36]. Red teaming, in which people or other models search for failures, is used to find the gaps that remain [37]–[39].

2.2 Security properties and threat model

Following broader risk frameworks for foundation models and language models [40]–[43], four properties matter for an LLM application.

- **Integrity of behaviour:** the application does what the developer intended and not what an attacker wrote.
- **Confidentiality:** the system prompt, private documents and user data are not leaked [16], [44], [45].
- **Safety of output:** the model does not produce content that violates policy.
- **Availability and cost:** the system remains usable and affordable, including under attack and under the load of the defenses themselves.

Attackers differ in access. A *black-box* attacker can only send prompts and read responses. A *white-box* attacker has the weights and can compute gradients [19], [46]. A *gray-box* attacker may see token probabilities [47]. A *supply-chain* attacker can influence training data, fine-tuning data or the retrieval corpus [48]–[50]. Most real attacks on deployed applications are black-box, and many defenses assume a limited attacker, which is a recurring source of over-optimistic evaluation.

2.3 Definitions

Prompt injection is an attack in which attacker-controlled text is treated as an instruction by the application and changes its behaviour. The attacker's goal is usually to hijack the task, reveal hidden instructions, or trigger an action [16], [17], [51], [52]. In *direct* injection the attacker is the user. In *indirect* injection the text is planted in content the application later reads, so an innocent user can trigger it [51], [53].

Jailbreaking is an attack on the model's safety behaviour: the attacker wants content that the model was trained to refuse. The attacker is usually the user and the target is the model, not the developer's task [18], [54], [55].

The two overlap in practice. A jailbreak can be used as a step in an injection attack, and injected instructions can ask for disallowed content. They are separated here because success is measured differently, as hijacking in one case and as policy violation in the other, and because defenses differ [17], [56].

3 Attacks

Figure 1 groups the attack families discussed in this section. They are ordered roughly from simple to sophisticated, and from attacks on a single prompt to attacks on the wider system. The attack families are summarized in Table 1.

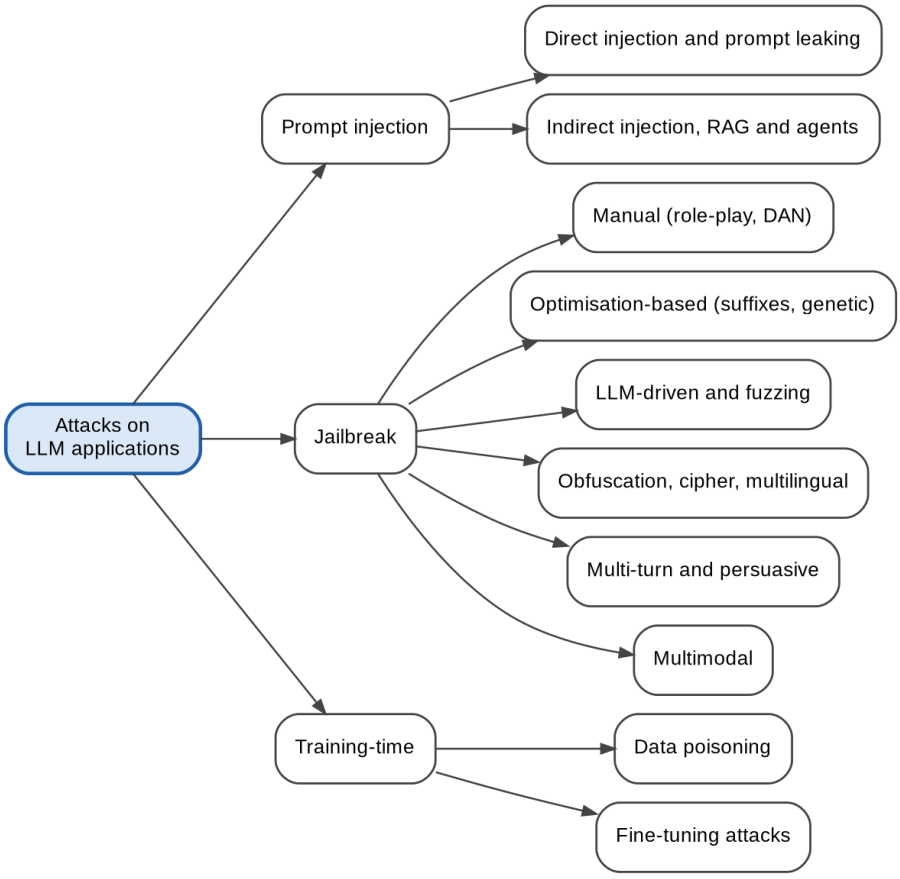


Figure 1. Taxonomy of attacks on LLM applications.

3.1 Direct prompt injection and prompt leaking

The first systematic study of direct injection, PromptInject, built attacks from interchangeable parts and showed that short handwritten messages could make GPT-3 ignore its task (*goal hijacking*) or print its hidden prompt (*prompt leaking*) [16]. Earlier work had already shown that language models are sensitive to handcrafted adversarial inputs [57]. Later work classified injection attacks [52], formalised them as a single framework that covers earlier attacks as special cases [17], and automated them: black-box attacks tested on commercial applications [58], and gradient-based universal injections [59]. Injection can also reach back-end systems, for example when model output is used to build database queries [60], and it can target LLMs that are used as judges [61].

Prompt leaking has its own line of work. Simple prompts can extract system prompts from many applications [45], and optimisation can make extraction more reliable [44]. Large collections of human-written attacks come from the Tensor Trust game and the HackAPrompt competition, which show how quickly people find new ways around simple defenses [62], [63].

3.2 Indirect injection, RAG and agents

In indirect injection the malicious text is placed in a web page, email, document or tool output that the application will read. Greshake et al. showed this against real LLM-integrated applications, including search assistants and code tools [51]. The BIPIA benchmark standardised evaluation of indirect injection and tested several simple defenses [53]. When models can call tools, the stakes rise because injected text can trigger actions. InjecAgent and AgentDojo measure how often tool-using agents follow injected instructions and what that does to task success [30], [31], and sandboxed emu-

lation is used to find risks in agent behaviour [64]. A related supply-chain risk is corruption of the retrieval corpus itself: PoisonedRAG shows that a few malicious passages can steer answers of retrieval-augmented systems [50].

3.3 Manual jailbreaks

The earliest jailbreaks were written by hand and spread through online communities. Empirical studies grouped them into recurring patterns such as role-play, hypothetical framing and privilege escalation [54], and a large study of in-the-wild prompts traced the DAN ("do anything now") family and its evolution [55]. Red-teaming studies of deployed chatbots documented failures in bias, robustness and toxicity [65], and dual-use analyses showed that classic security tricks such as obfuscation and payload splitting also work on LLMs [66]. Wei et al. explained why such attacks succeed through two failure modes of safety training, *competing objectives* (helpfulness against safety) and *mismatched generalization* (safety training does not cover every input form the model can understand) [18]. A theoretical analysis argues that, for any behaviour the model can produce with non-zero probability, prompts exist that elicit it [67].

3.4 Optimisation-based attacks

Optimisation methods search for prompts automatically. Zou et al. used greedy coordinate gradient search to find adversarial suffixes that make aligned models comply, and found that suffixes found on open models often transfer to proprietary ones [19]. Carlini et al. showed that aligned models are not adversarially aligned in general, especially under multimodal inputs [68]. Suffixes from gradient search look like gibberish and can be caught by simple filters, so later work searched for readable prompts: AutoDAN (two different methods share this name) uses interpretable gradient-based search in one case [46] and a hierarchical genetic algorithm in the other [69]. Genetic search can also produce universal jailbreaks against black-box models [70]. With access to token probabilities, simple adaptive attacks based on random search reach very high success rates on many models, which suggests that evaluation against weaker attacks can overstate safety [47].

3.5 LLM-driven and fuzzing attacks

An attacker model can write and refine jailbreaks for the target. PAIR iterates between an attacker LLM and the target and often succeeds within a small number of queries [71]. TAP adds tree-structured reasoning and prunes unpromising branches [72]. GPTFuzzer mutates human-written seed templates in the manner of software fuzzing [73], and MasterKey infers the defenses of commercial chatbots and trains an attacker model against them [74]. These methods lower the skill needed to find new attacks and are also used to generate test sets for defenses [38], [56].

3.6 Obfuscation, cipher and multilingual attacks

Because safety training covers some input forms better than others, changing the form can bypass it. Encodings such as Base64, ciphers and character substitutions were already noted as mismatched generalization [18], and a study of cipher-based prompts found that more capable models were sometimes more exposed because they could follow the cipher [75]. Translating a request into a low-resource language lowers refusal rates [76], [77]. Nested fictional scenarios ("a story within a story") can also hide a request [78], and manipulating the decoding settings of open models can remove refusals without any prompt trick [79].

3.7 Multi-turn, persuasive and in-context attacks

Single-prompt filters are weaker against attacks that build up over a conversation. Crescendo starts with harmless questions and gradually moves toward the target while referring to the model's own earlier answers [80]. Persuasion-based prompts apply social-science persuasion techniques to an LLM as if it were a person [81]. A few in-context demonstrations of compliance can also shift behaviour, and the same idea can be used defensively [82].

3.8 Multimodal attacks

Models that accept images and audio add new surfaces. A single adversarial image can jailbreak a multimodal model [83], and instructions can be hidden in images or sounds to carry out indirect injection [84]. This review does not cover multimodal defenses in depth.

3.9 Training-time attacks

Some attacks act before deployment. Poisoning a small fraction of web-scale training data can be practical [48], a few poisoned examples in instruction tuning can plant behaviour that appears on later prompts [49], [85], and fine-tuning an aligned model on a handful of examples, even benign ones, can erode its safety [86]. These attacks matter for any team that fine-tunes models, which is common in practice.

Table 1. Summary of attack families.

Family	Representative works	Attacker access	Main weakness of the attack
Direct injection, prompt leaking	[16], [17], [44]	Black box	Simple versions are caught by task-specific defenses
Indirect injection, RAG, agents	[30], [31], [50], [51], [53]	Black box, content control	Needs the victim application to read attacker content
Manual jailbreaks	[18], [54], [55]	Black box	Patched quickly; needs human effort
Optimisation-based	[19], [46], [47], [69], [70]	White, gray or black box	Gibberish suffixes are detectable; some need gradients
LLM-driven and fuzzing	[71]–[74]	Black box	Needs many queries; quality of jailbreak varies
Obfuscation and multilingual	[75]–[78]	Black box	Depends on the model understanding the encoding
Multi-turn and persuasive	[80]–[82]	Black box	Needs several turns; per-prompt filters miss it
Multimodal	[83], [84]	White box or black box	Only for models with other inputs
Training-time	[48], [49], [85], [86]	Supply chain	Needs access to training or fine-tuning data

4 Defenses

Defenses can be placed at different points along the request path. Figure 2 shows eight layers, from training of the model to controls around the whole application. No single layer is sufficient, and the layers differ greatly in what they cost per request, which is the theme of the section on cost-aware and adaptive protection. The layers are summarized in Table 2.

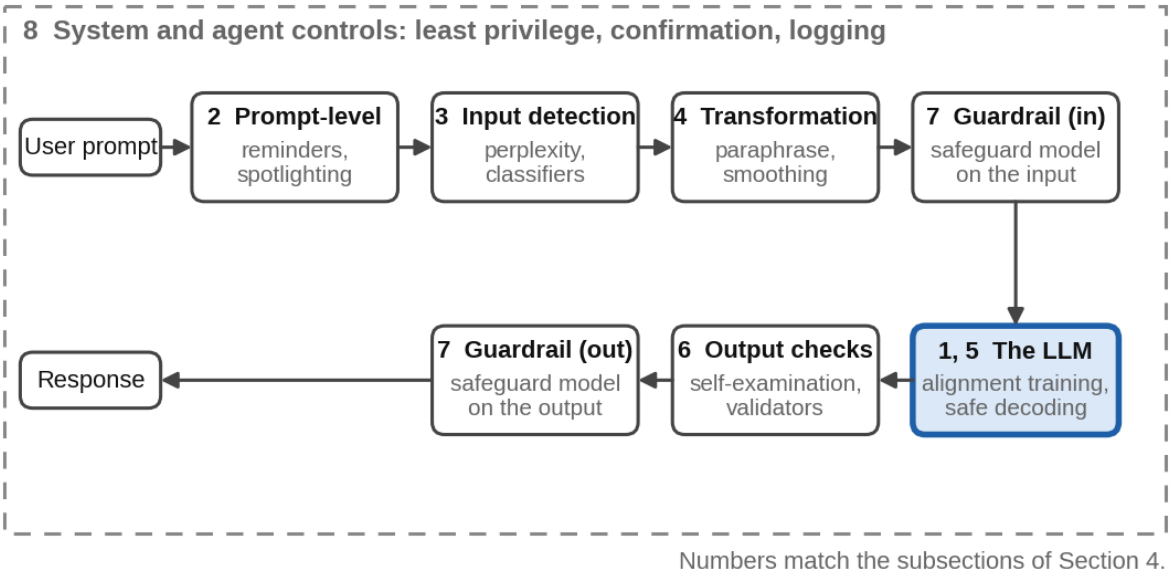


Figure 2. Defense layers along the request path of an LLM application. The numbers correspond to the eight defense layers described in this section.

4.1 Alignment-time defenses

The first line of defense is the model's own training. RLHF, constitutional methods and preference optimisation make refusals more reliable [3], [35], [36]. For prompt injection, specialised training teaches the model to separate instruction from data: the *instruction hierarchy* trains models to follow privileged instructions over untrusted text [87], StruQ uses structured queries with separate channels for instruction and data [88], and Jatmo fine-tunes a model on a single task so that it does not follow injected instructions [89]. For jailbreaks, *circuit breakers* interrupt the internal representations that lead to harmful output, building on representation engineering [90], [91]. Training-time defenses are strong but need access to training, and they cannot be added to a closed model by the application developer. The theoretical and empirical results reviewed in the Attacks section suggest they will not close the problem alone [18], [67].

4.2 Prompt-level defenses

Prompt-level defenses change how the prompt is built. A *self-reminder* in the system prompt reminds the model to behave responsibly and was shown to lower jailbreak success [92]. *Spotlighting* marks the origin of untrusted text with delimiters, data-marking or encoding so the model can tell it apart from instructions [93]. In-context demonstrations of refusal can strengthen the model against jailbreaks [82]. These methods are cheap and easy to deploy but can be bypassed by an attacker who knows or guesses the format [17].

4.3 Input detection

Detection tries to flag a malicious prompt before the model sees it. *Perplexity filters* flag the gibberish suffixes produced by gradient search [94], [95]. They do not catch fluent attacks such as readable genetic-search prompts or manual role-play [69]. *Learned classifiers*, from simple models on text features to fine-tuned encoder models such as DeBERTa [96], can detect known attack styles and are fast, but generalise poorly to new families unless trained on diverse data [17]. *Gradient Cuff* detects

jailbreaks by examining how the refusal loss changes under perturbation of the input, at the cost of extra model queries [97].

4.4 Input transformation

Transformation changes the input so that an adversarial effect is lost while meaning is kept. Paraphrasing and retokenization were among the baselines of Jain et al. [94]. SmoothLLM perturbs several copies of the prompt and aggregates the answers [98], and semantic smoothing does this at the level of meaning [99]. Erase-and-check removes tokens and checks each subsequence with a safety filter, giving *certified* guarantees against suffix attacks of bounded length [100], and a related method tests whether the model remains aligned after random dropping of tokens [101]. Robust prompt optimisation learns a defensive suffix that is appended to the prompt [102]. These methods are effective against token-level attacks, but several multiply the number of model calls per request.

4.5 Decoding-time defenses

SafeDecoding changes token probabilities during generation, using a safety-tuned expert model to boost safe tokens and damp harmful ones [103]. RAIN lets a model evaluate and rewind its own generation without fine-tuning [104]. These defenses act after the prompt has passed the input stage and so complement detection, but they require control of the decoding process.

4.6 Self-examination and output checks

A model can examine its own behaviour. LLM Self Defense asks a second model instance to check whether a response is harmful before it is shown [105]. Intention analysis first has the model analyse the user's intention and then answers [106]. Goal prioritisation teaches or instructs the model to put safety above helpfulness [107], and Self-Guard trains the model to tag its own output as harmful or harmless [108]. Output checks also catch attacks that were missed at the input, such as a leaked system prompt [44], [45]. The cost is one more model call per response.

4.7 Guardrail and safeguard models

A separate model can classify inputs and outputs. Early moderation tools used classifiers for categories such as hate and self-harm [109]–[112]. Llama Guard is an instruction-tuned safeguard model that classifies both prompts and responses against a policy taxonomy that can be adapted [113]. WildGuard, ShieldGemma and Granite Guardian are open moderation models covering harmful prompts, jailbreaks and refusals [114]–[116]. Constitutional Classifiers train classifiers from a written constitution with synthetic data and resisted universal jailbreaks over thousands of hours of red teaming [117]. NeMo Guardrails provides programmable rails that set what an application may do [118], and a broader view of guardrail design is given by Dong et al. [119]. The Purple Llama and CyberSecEval work provides tools and benchmarks for cybersecurity risks, including prompt injection [120], [121]. Guardrail models give good coverage, but a large safeguard model on every request adds substantial latency and compute.

4.8 System-level and agent defenses

Because model-level defenses are imperfect, system design matters. Practical measures include least-privilege tool access, confirmation before sensitive actions, separating the model that reads untrusted content from the one that can act, and logging for audit. Agent benchmarks show that attack success

and task utility trade off against each other, so a defense that blocks everything is not acceptable [30], [31]. Programmable guardrails and policy engines give a place to enforce such rules [118], [119].

Table 2. Summary of defense layers.

Layer	Examples	Strength	Limitation	Cost per request
Alignment-time	[87]–[90]	Built into the model	Needs training access; not complete	None at run time
Prompt-level	[82], [92], [93]	Cheap, easy to deploy	Can be bypassed	Very low
Input detection	[94], [95], [97]	Fast filtering	Misses fluent attacks	Low to medium
Input transformation	[98], [100]–[102]	Strong on token-level attacks	Several model calls	Medium to high
Decoding-time	[103], [104]	Acts during generation	Needs decoding control	Medium
Self-examination, output checks	[105]–[108]	Catches missed attacks	Extra call per response	Medium
Guardrail models	[113]–[117]	Broad coverage	Latency on every request	Medium to high
System and agent controls	[31], [118], [119]	Limits damage	Needs careful design	Low

5 Benchmarks, datasets and metrics

5.1 Attack and defense benchmarks

Comparing defenses needs a fixed set of attacks and a fixed definition of success. For prompt injection, the benchmark of Liu et al. evaluates several attacks and defenses across many models and tasks [17], BIPIA covers indirect injection [53], and InjecAgent and AgentDojo cover tool-using agents [30], [31]. Large human-written collections come from Tensor Trust and HackAPrompt [62], [63]. For jailbreaks, HarmBench standardises automated red teaming and refusal evaluation [56], JailbreakBench provides a fixed behaviour set and a repository of attack artifacts [122], and StrongREJECT proposes a scoring method that checks whether a jailbreak response is actually useful to an attacker and not merely non-refusing [123]. Selected benchmarks and datasets are listed in Table 3.

5.2 Broader safety and trustworthiness suites

Broader suites test many risk areas at once, including toxicity, bias, privacy and robustness: DecodingTrust for GPT models [124], SafetyBench with multiple-choice questions [125], TrustLLM [126], and red-teaming sets based on chains of utterances [127]. Moderation work contributes labelled data on toxic and unsafe text [109]–[112], and the Aegis taxonomy and dataset provide annotated human-LLM interactions with a risk taxonomy [128]. Security-specific evaluation suites cover insecure code generation and prompt injection in coding settings [120], [121].

5.3 Over-refusal and false positives

A defense that blocks too much is also a failure. XSTest provides prompts that look unsafe but are safe, to detect *exaggerated safety* [129], and OR-Bench measures over-refusal at larger scale [130]. These are the benchmarks closest to the false-positive rate of a detection layer, and they should be reported alongside attack success.

5.4 Metrics and judging

The common metrics are as follows.

- **Attack success rate (ASR):** the number of attacks that reach the attacker's goal divided by the number of attack prompts. For jailbreaks it is often called the jailbreak success rate.
- **Detection metrics:** accuracy, precision, recall, F1 score and false positive rate (FPR) for a detector, with unsafe as the positive class.
- **Refusal and over-refusal rates** on safe but sensitive prompts [129], [130].
- **Utility under defense:** performance on normal tasks, important for agents [31] and for defenses that change the prompt or the model [94].
- **Latency and cost:** time per request, number of model calls and compute, which are the least often reported.

Deciding whether an attack succeeded is itself a source of error. Rules work for simple cases such as a leaked prompt [16]. For harmful content, an LLM judge is common [56], [122], [131], but judges can be wrong or can be fooled [61], and non-refusal does not always mean a useful harmful answer [123]. A manual check on a sample is good practice.

Table 3. Selected benchmarks and datasets.

Name	Focus	Typical use
Prompt-injection benchmark [17]	Direct injection attacks and defenses	Compare defenses across models and tasks
BIPIA [53]	Indirect injection	Test document and web content attacks
InjecAgent, AgentDojo [30], [31]	Tool-using agents	Attack success and utility of agents
Tensor Trust, HackAPrompt [62], [63]	Human-written injections	Training and test attack prompts
HarmBench, JailbreakBench [56], [122]	Jailbreaks and refusal	Standard attack evaluation
StrongREJECT [123]	Jailbreak scoring	More reliable success judgement
XSTest, OR-Bench [129], [130]	Over-refusal	False-positive measurement
DecodingTrust, SafetyBench, TrustLLM [124]–[126]	Broad safety	Wide risk coverage
ToxicChat, ToxiGen, RealToxicityPrompts [110]–[112]	Toxicity	Moderation training and test data
Aegis dataset [128]	Content safety taxonomy	Safety classifier training

6 Cost-aware and adaptive protection

6.1 The cost of defending

Most defenses reviewed in the Defenses section add work. Smoothing and erase-and-check multiply the number of model calls [98], [100], self-examination adds one call per response [105], and a large safeguard model on every input and output adds its own latency [113], [115]. Gradient-based detection needs several queries [97]. In the papers reviewed here, success rates are reported more often than latency or compute overhead, although this is an impression from reading and not a measured count. The result is a gap between published defenses and deployable ones: a defense that doubles latency may be unacceptable for a high-traffic application, however well it performs on a benchmark.

6.2 Lessons from LLM routing and cascades

Outside security, researchers have studied how to spend computation in proportion to the difficulty of each query. FrugalGPT learns which LLMs to call, and in what order, to reduce cost while keeping quality [132]. RouteLLM trains routers that send each query to a stronger or weaker model [133], RouterBench provides a benchmark for such routers [134], and Hybrid LLM predicts query difficulty to choose between a small and a large model [135]. AutoMix has a model verify its own answer and routes uncertain cases to a larger model [136], and confident adaptive decoding exits early when the model is confident [137]. The shared idea is a cheap first stage that handles easy cases and an escalation path for hard ones.

6.3 Adaptive moderation

The same idea has begun to appear in safety. Aegis combines an ensemble of LLM safety experts with an online learning method that adjusts the weight of each expert as feedback arrives, so the moderation system can adapt to new attacks and changing policies [128]. A later system, AEGIS-Chat, uses a stateful router that conditions on the conversation, a contextual-bandit component with a cost-sensitive objective that balances harm reduction, latency and over-moderation, and early exits for benign content with escalation to stronger experts when needed [138]. These systems show that moderation can be made adaptive, but they focus on content safety and conversation moderation more than on injection and jailbreak attacks in applications with several models.

6.4 Risk-proportional defense

Putting the two threads together suggests a design pattern, which this paper calls *risk-proportional defense*. A cheap first stage scores each prompt as Low, Medium or High risk. The risk level then selects how many further checks run, as in Figure 3. Low-risk prompts receive only light checks, medium-risk prompts add a learned classifier, and high-risk prompts receive the heavy checks, such as an LLM-based judge, attack-type identification, and blocking or rewriting. Responses are validated in every case, so that an attack missed at the input still has a chance to be caught at the output.

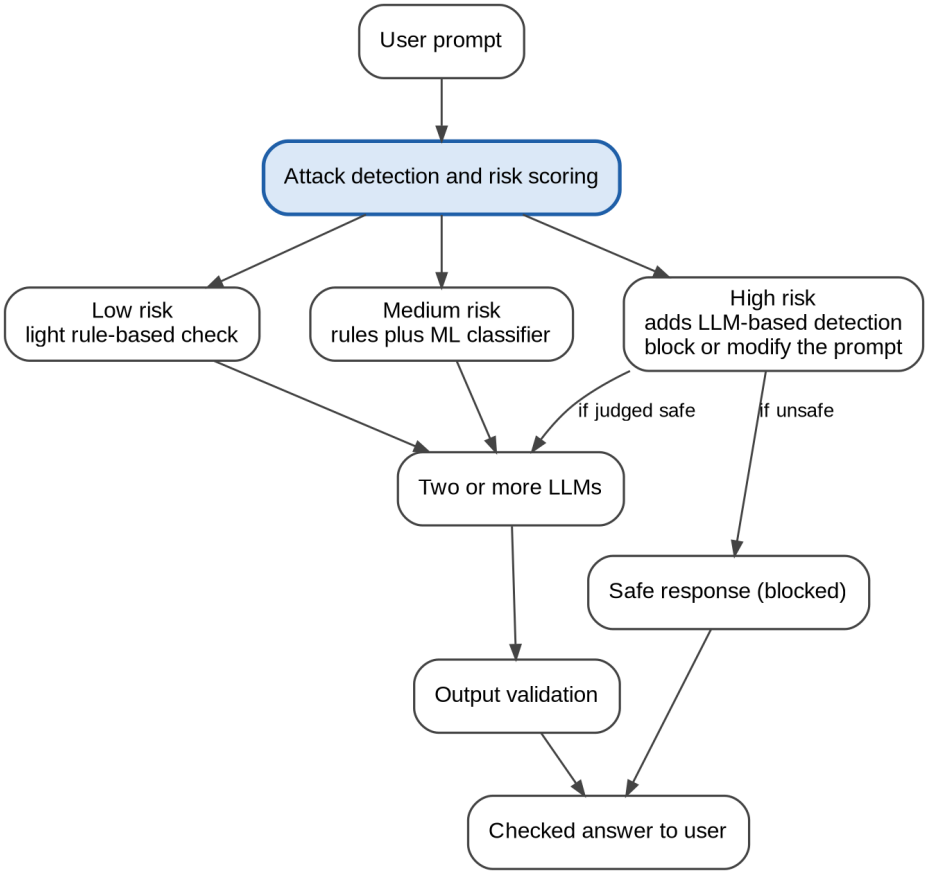


Figure 3. Risk-proportional defense as a design pattern.

The pattern raises several design questions that the literature does not yet answer well.

- **Triage quality.** The first stage must be cheap but accurate. Perplexity, pattern rules and small classifiers are candidates [94]–[96], but fluent attacks can look benign [69], [80], so missed attacks in the Low tier are the main risk.
- **Thresholds and escalation.** Thresholds trade missed attacks against cost and false positives. Escalation rules, such as sending a medium-risk prompt to the high-risk path when a classifier is uncertain, resemble the confidence-based routing discussed above [136], [137].
- **Attack-type awareness.** Knowing whether an input is an injection, a jailbreak or an extraction attempt allows a targeted response, but detectors that label the type are less common than binary detectors [52], [128].
- **Adaptive attackers.** An attacker who knows the triage stage can try to look low-risk, so evaluation must include adaptive attacks and not only fixed sets [47], [94].
- **Evaluation design.** A fair test compares three conditions on the same prompts and models: no defense, an always-full defense and the risk-adaptive defense, and reports attack success, false positives and latency together [56], [129], [130].

To the best of the authors' reading, systematic comparisons of uniform and risk-adaptive defense across several LLMs, reporting both security and latency, are rare, and this is a useful direction for future work. The authors are building such a framework in a companion project using open models served locally and through model hubs [32], [33], [139].

7 Open challenges

- **Adaptive and evolving attackers.** Defenses are often shown to work against fixed attack sets and fail against attacks designed with knowledge of the defense [47], [68], [94]. Evaluation should include attackers that adapt to the specific defense, and results should state the attacker's access.
- **Evaluation validity.** Attack success judged by LLMs or by refusal keywords is noisy [61], [123], benchmark prompts may leak into training data, and different papers use different definitions of success, which makes numbers hard to compare [56], [122].
- **Over-refusal and utility.** Stricter defenses block more legitimate requests. Reporting false positives and utility next to attack success should be standard [31], [129], [130].
- **Multi-turn and long-context attacks.** Per-prompt filters miss attacks that build up over a conversation [80], [81], and conversation-aware moderation is only starting to appear [138].
- **Agents and tool use.** Injection in tool outputs can trigger real actions, and defenses must preserve task utility [30], [31], [64].
- **Multilingual, encoded and multimodal inputs.** Safety coverage is uneven across languages, encodings and modalities [18], [76], [77], [83], [84].
- **Training-time risks.** Fine-tuning and poisoning can undo safety and are relevant to every team that customises models [48], [49], [86].
- **Standards and transparency.** Common taxonomies and threat catalogues exist [20]–[22], but practical guidance on what to deploy, at what cost, and how to measure it is still thin. Reporting cost and latency, publishing code and fixing random seeds would help.
- **Theoretical limits.** Analyses of alignment suggest that complete prevention by training alone is unlikely [18], [67], so layered defenses with honest reporting of residual risk remain necessary.

8 Conclusion

Prompt injection and jailbreak attacks have grown from handwritten tricks into automated, transferable and multi-turn methods, and they now reach beyond the chat window into retrieval systems and agents. Defenses exist at every point along the request path, from alignment-time training to guardrail models and system controls, but each has gaps, and many add cost to every request. Research on LLM routing and adaptive moderation suggests that matching defensive effort to the risk of each input could reduce that cost, but the evidence for security applications is still thin. Progress will depend on adaptive evaluation, reporting of false positives and latency alongside attack success, and shared benchmarks that cover several models and application settings.

References

[1] Brown TB, et al. Language models are few-shot learners. In: Advances in Neural Information Processing Systems (NeurIPS); 2020. *arXiv*:2005.14165.

[2] Ouyang L, et al. Training language models to follow instructions with human feedback. In: Advances in Neural Information Processing Systems (NeurIPS); 2022. *arXiv*:2203.02155.

[3] Bai Y, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv* [Preprint]. 2022. arXiv:2204.05862.

[4] OpenAI. GPT-4 technical report. *arXiv* [Preprint]. 2023. arXiv:2303.08774.

[5] Touvron H, et al. LLaMA: open and efficient foundation language models. *arXiv* [Preprint]. 2023. arXiv:2302.13971.

[6] Touvron H, et al. Llama 2: open foundation and fine-tuned chat models. *arXiv* [Preprint]. 2023. arXiv:2307.09288.

[7] Jiang AQ, et al. Mistral 7B. arXiv [Preprint]. 2023. arXiv:2310.06825.

[8] Jiang AQ, et al. Mixtral of experts. *arXiv* [Preprint]. 2024. arXiv:2401.04088.

[9] Gemma Team. Gemma: open models based on Gemini research and technology. *arXiv* [Preprint]. 2024. arXiv:2403.08295.

[10] Gemini Team. Gemini: a family of highly capable multimodal models. *arXiv* [Preprint]. 2023. arXiv:2312.11805.

[11] DeepSeek-AI. DeepSeek-V3 technical report. *arXiv* [Preprint]. 2024. arXiv:2412.19437.

[12] DeepSeek-AI. DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv* [Preprint]. 2025. arXiv:2501.12948.

[13] Dubey A, et al. The Llama 3 herd of models. *arXiv* [Preprint]. 2024. arXiv:2407.21783.

[14] Qwen Team. Qwen2.5 technical report. *arXiv* [Preprint]. 2024. arXiv:2412.15115.

[15] Lewis P, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems (NeurIPS); 2020. *arXiv*:2005.11401.

[16] Perez F, Ribeiro I. Ignore previous prompt: attack techniques for language models. In: NeurIPS 2022 ML Safety Workshop; 2022. *arXiv*:2211.09527.

[17] Liu Y, Jia Y, Geng R, Jia J, Gong NZ. Formalizing and benchmarking prompt injection attacks and defenses. In: USENIX Security; 2024. *arXiv*:2310.12815.

[18] Wei A, Haghtalab N, Steinhardt J. Jailbroken: how does LLM safety training fail? In: Advances in Neural Information Processing Systems (NeurIPS); 2023. *arXiv*:2307.02483.

[19] Zou A, et al. Universal and transferable adversarial attacks on aligned language models. *arXiv* [Preprint]. 2023. arXiv:2307.15043.

[20] OWASP GenAI Security Project. *OWASP top 10 for large language model applications* [Internet]. Available from: https://owasp.org

[21] Vassilev A, Oprea A, Fordyce A, Anderson H. *Adversarial machine learning: a taxonomy and terminology of attacks and mitigations*. NIST AI 100-2e2023. Gaithersburg, MD: National Institute of Standards and Technology; 2024. https://doi.org/10.6028/NIST.AI.100-2e2023

[22] MITRE. *MITRE ATLAS: adversarial threat landscape for artificial-intelligence systems* [Internet]. Available from: https://atlas.mitre.org

[23] Yao Y, et al. A survey on large language model (LLM) security and privacy: the good, the bad, and the ugly. High-Confidence Computing. 2024. *arXiv*:2312.02003.

[24] Das BC, Amini MH, Wu Y. Security and privacy challenges of large language models: a survey. ACM Computing Surveys. 2025. *arXiv*:2402.00888.

[25] Shayegani E, et al. Survey of vulnerabilities in large language models revealed by adversarial attacks. *arXiv* [Preprint]. 2023. arXiv:2310.10844.

[26] Schwinn L, et al. Adversarial attacks and defenses in large language models: old and new threats. *arXiv* [Preprint]. 2023. arXiv:2310.19737.

[27] Yi S, et al. Jailbreak attacks and defenses against large language models: a survey. *arXiv* [Preprint]. 2024. arXiv:2407.04295.

[28] Xu Z, et al. LLM jailbreak attack versus defense techniques: a comprehensive study. *arXiv* [Preprint]. 2024. arXiv:2402.13457.

[29] Liu Y, et al. Trustworthy LLMs: a survey and guideline for evaluating large language models' alignment. *arXiv* [Preprint]. 2023. arXiv:2308.05374.

[30] Zhan Q, Liang Z, Ying Z, Kang D. InjecAgent: benchmarking indirect prompt injections in tool-integrated large language model agents. In: Findings of ACL; 2024. *arXiv*:2403.02691.

[31] Debenedetti E, et al. AgentDojo: a dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In: NeurIPS 2024 Datasets and Benchmarks; 2024. *arXiv*:2406.13352.

[32] Ollama. *Run large language models locally* [Internet]. Available from: https://ollama.com

[33] Hugging Face. *Hugging Face Hub: models and datasets* [Internet]. Available from: https://huggingface.co

[34] Wolf T, et al. Transformers: state-of-the-art natural language processing. In: EMNLP 2020 System Demonstrations; 2020. *arXiv*:1910.03771.

[35] Bai Y, et al. Constitutional AI: harmlessness from AI feedback. *arXiv* [Preprint]. 2022. arXiv:2212.08073.

[36] Rafailov R, et al. Direct preference optimization: your language model is secretly a reward model. In: Advances in Neural Information Processing Systems (NeurIPS); 2023. *arXiv*:2305.18290.

[37] Ganguli D, et al. Red teaming language models to reduce harms: methods, scaling behaviors, and lessons learned. *arXiv* [Preprint]. 2022. arXiv:2209.07858.

[38] Perez E, et al. Red teaming language models with language models. In: EMNLP; 2022. *arXiv*:2202.03286.

[39] Perez E, et al. Discovering language model behaviors with model-written evaluations. In: Findings of ACL; 2023. *arXiv*:2212.09251.

[40] Bommasani R, et al. On the opportunities and risks of foundation models. *arXiv* [Preprint]. 2021. arXiv:2108.07258.

[41] Weidinger L, et al. Ethical and social risks of harm from language models. *arXiv* [Preprint]. 2021. arXiv:2112.04359.

[42] Hendrycks D, et al. Unsolved problems in ML safety. *arXiv* [Preprint]. 2021. arXiv:2109.13916.

[43] Cui T, et al. Risk taxonomy, mitigation, and assessment benchmarks of large language model systems. *arXiv* [Preprint]. 2024. arXiv:2401.05778.

[44] Hui B, et al. PLeak: prompt leaking attacks against large language model applications. In: ACM CCS; 2024. *arXiv*:2405.06823.

[45] Zhang Y, Ippolito D. Effective prompt extraction from language models. *arXiv* [Preprint]. 2023. arXiv:2307.06865.

[46] Zhu S, et al. AutoDAN: interpretable gradient-based adversarial attacks on large language models. In: COLM; 2024. *arXiv*:2310.15140.

[47] Andriushchenko M, Croce F, Flammarion N. Jailbreaking leading safety-aligned LLMs with simple adaptive attacks. *arXiv* [Preprint]. 2024. arXiv:2404.02151.

[48] Carlini N, et al. Poisoning web-scale training datasets is practical. In: IEEE S&P; 2024. *arXiv*:2302.10149.

[49] Wan A, et al. Poisoning language models during instruction tuning. In: International Conference on Machine Learning (ICML); 2023. *arXiv*:2305.00944.

[50] Zou W, et al. PoisonedRAG: knowledge corruption attacks to retrieval-augmented generation of large language models. In: USENIX Security; 2025. *arXiv*:2402.07867.

[51] Greshake K, et al. Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection. In: AISec; 2023. *arXiv*:2302.12173.

[52] Rossi S, et al. An early categorization of prompt injection attacks on large language models. *arXiv* [Preprint]. 2024. arXiv:2402.00898.

[53] Yi J, et al. Benchmarking and defending against indirect prompt injection attacks on large language models. *arXiv* [Preprint]. 2023. arXiv:2312.14197.

[54] Liu Y, et al. Jailbreaking ChatGPT via prompt engineering: an empirical study. *arXiv* [Preprint]. 2023. arXiv:2305.13860.

[55] Shen X, Chen Z, Backes M, Shen Y, Zhang Y. Do anything now: characterizing and evaluating in-the-wild jailbreak prompts on large language models. In: ACM CCS; 2024. *arXiv*:2308.03825.

[56] Mazeika M, et al. HarmBench: a standardized evaluation framework for automated red teaming and robust refusal. In: International Conference on Machine Learning (ICML); 2024. *arXiv*:2402.04249.

[57] Branch HJ, et al. Evaluating the susceptibility of pre-trained language models via handcrafted adversarial examples. *arXiv* [Preprint]. 2022. arXiv:2209.02128.

[58] Liu Y, et al. Prompt injection attack against LLM-integrated applications. *arXiv* [Preprint]. 2023. arXiv:2306.05499.

[59] Liu X, et al. Automatic and universal prompt injection attacks against large language models. *arXiv* [Preprint]. 2024. arXiv:2403.04957.

[60] Pedro R, et al. From prompt injections to SQL injection attacks: how protected is your LLM-integrated web application? arXiv [Preprint]. 2023. *arXiv*:2308.01990.

[61] Shi J, et al. Optimization-based prompt injection attack to LLM-as-a-judge. In: ACM CCS; 2024. *arXiv*:2403.17710.

[62] Toyer S, et al. Tensor Trust: interpretable prompt injection attacks from an online game. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2311.01011.

[63] Schulhoff S, et al. Ignore this title and HackAPrompt: exposing systemic vulnerabilities of LLMs through a global scale prompt hacking competition. In: EMNLP; 2023. *arXiv*:2311.16119.

[64] Ruan Y, et al. Identifying the risks of LM agents with an LM-emulated sandbox. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2309.15817.

[65] Zhuo TY, et al. Red teaming ChatGPT via jailbreaking: bias, robustness, reliability and toxicity. *arXiv* [Preprint]. 2023. arXiv:2301.12867.

[66] Kang D, et al. Exploiting programmatic behavior of LLMs: dual-use through standard security attacks. *arXiv* [Preprint]. 2023. arXiv:2302.05733.

[67] Wolf Y, et al. Fundamental limitations of alignment in large language models. In: International Conference on Machine Learning (ICML); 2024. *arXiv*:2304.11082.

[68] Carlini N, et al. Are aligned neural networks adversarially aligned? In: Advances in Neural Information Processing Systems (NeurIPS); 2023. *arXiv*:2306.15447.

[69] Liu X, Xu N, Chen M, Xiao C. AutoDAN: generating stealthy jailbreak prompts on aligned large language models. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2310.04451.

[70] Lapid R, Langberg R, Sipper M. Open Sesame! Universal black box jailbreaking of large language models. *arXiv* [Preprint]. 2023. arXiv:2309.01446.

[71] Chao P, et al. Jailbreaking black box large language models in twenty queries. *arXiv* [Preprint]. 2023. arXiv:2310.08419.

[72] Mehrotra A, et al. Tree of attacks: jailbreaking black-box LLMs automatically. In: Advances in Neural Information Processing Systems (NeurIPS); 2024. *arXiv*:2312.02119.

[73] Yu J, Lin X, Yu Z, Xing X. GPTFuzzer: red teaming large language models with auto-generated jailbreak prompts. *arXiv* [Preprint]. 2023. arXiv:2309.10253.

[74] Deng G, et al. MasterKey: automated jailbreak across multiple large language model chatbots. In: NDSS; 2024. *arXiv*:2307.08715.

[75] Yuan Y, et al. GPT-4 is too smart to be safe: stealthy chat with LLMs via cipher. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2308.06463.

[76] Yong ZX, Menghini C, Bach SH. Low-resource languages jailbreak GPT-4. *arXiv* [Preprint]. 2023. arXiv:2310.02446.

[77] Deng Y, et al. Multilingual jailbreak challenges in large language models. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2310.06474.

[78] Li X, et al. DeepInception: hypnotize large language model to be jailbreaker. *arXiv* [Preprint]. 2023. arXiv:2311.03191.

[79] Huang Y, et al. Catastrophic jailbreak of open-source LLMs via exploiting generation. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2310.06987.

[80] Russinovich M, Salem A, Eldan R. Great, now write an article about that: the Crescendo multi-turn LLM jailbreak attack. In: USENIX Security; 2025. *arXiv*:2404.01833.

[81] Zeng Y, et al. How Johnny can persuade LLMs to jailbreak them: rethinking persuasion to challenge AI safety by humanizing LLMs. In: ACL; 2024. *arXiv*:2401.06373.

[82] Wei Z, Wang Y, Wang Y. Jailbreak and guard aligned language models with only few in-context demonstrations. *arXiv* [Preprint]. 2023. arXiv:2310.06387.

[83] Qi X, et al. Visual adversarial examples jailbreak aligned large language models. In: AAAI; 2024. *arXiv*:2306.13213.

[84] Bagdasaryan E, et al. Abusing images and sounds for indirect instruction injection in multi-modal LLMs. *arXiv* [Preprint]. 2023. arXiv:2307.10490.

[85] Shu M, et al. On the exploitability of instruction tuning. In: Advances in Neural Information Processing Systems (NeurIPS); 2023. *arXiv*:2306.17194.

[86] Qi X, et al. Fine-tuning aligned language models compromises safety, even when users do not intend to! In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2310.03693.

[87] Wallace E, et al. The instruction hierarchy: training LLMs to prioritize privileged instructions. *arXiv* [Preprint]. 2024. arXiv:2404.13208.

[88] Chen S, Piet J, Sitawarin C, Wagner D. StruQ: defending against prompt injection with structured queries. In: USENIX Security; 2025. *arXiv*:2402.06363.

[89] Piet J, et al. Jatmo: prompt injection defense by task-specific finetuning. *arXiv* [Preprint]. 2023. arXiv:2312.17673.

[90] Zou A, et al. Improving alignment and robustness with circuit breakers. In: Advances in Neural Information Processing Systems (NeurIPS); 2024. *arXiv*:2406.04313.

[91] Zou A, et al. Representation engineering: a top-down approach to AI transparency. *arXiv* [Preprint]. 2023. arXiv:2310.01405.

[92] Xie Y, et al. Defending ChatGPT against jailbreak attack via self-reminders. *Nature Machine Intelligence*. 2023;5. https://doi.org/10.1038/s42256-023-00765-8.

[93] Hines K, et al. Defending against indirect prompt injection attacks with spotlighting. *arXiv* [Preprint]. 2024. arXiv:2403.14720.

[94] Jain N, et al. Baseline defenses for adversarial attacks against aligned language models. *arXiv* [Preprint]. 2023. arXiv:2309.00614.

[95] Alon G, Kamfonas M. Detecting language model attacks with perplexity. *arXiv* [Preprint]. 2023. arXiv:2308.14132.

[96] He P, et al. DeBERTa: decoding-enhanced BERT with disentangled attention. *arXiv* [Preprint]. 2020. arXiv:2006.03654.

[97] Hu X, Chen PY, Ho TY. Gradient Cuff: detecting jailbreak attacks on large language models by exploring refusal loss landscapes. In: Advances in Neural Information Processing Systems (NeurIPS); 2024. *arXiv*:2403.00867.

[98] Robey A, Wong E, Hassani H, Pappas GJ. SmoothLLM: defending large language models against jailbreaking attacks. *arXiv* [Preprint]. 2023. arXiv:2310.03684.

[99] Ji J, et al. Defending large language models against jailbreak attacks via semantic smoothing. *arXiv* [Preprint]. 2024. arXiv:2402.16192.

[100] Kumar A, et al. Certifying LLM safety against adversarial prompting. *arXiv* [Preprint]. 2023. arXiv:2309.02705.

[101] Cao B, et al. Defending against alignment-breaking attacks via robustly aligned LLM. In: ACL; 2024. *arXiv*:2309.14348.

[102] Zhou A, Li B, Wang H. Robust prompt optimization for defending language models against jailbreaking attacks. In: Advances in Neural Information Processing Systems (NeurIPS); 2024. *arXiv*:2401.17263.

[103] Xu Z, et al. SafeDecoding: defending against jailbreak attacks via safety-aware decoding. In: ACL; 2024. *arXiv*:2402.08983.

[104] Li Y, et al. RAIN: your language models can align themselves without finetuning. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2309.07124.

[105] Phute M, et al. LLM self defense: by self examination, LLMs know they are being tricked. *arXiv* [Preprint]. 2023. arXiv:2308.07308.

[106] Zhang Y, et al. Intention analysis makes LLMs a good jailbreak defender. *arXiv* [Preprint]. 2024. arXiv:2401.06561.

[107] Zhang Z, et al. Defending large language models against jailbreaking attacks through goal prioritization. In: ACL; 2024. *arXiv*:2311.09096.

[108] Wang Z, et al. Self-Guard: empower the LLM to safeguard itself. In: NAACL; 2024. *arXiv*:2310.15851.

[109] Markov T, et al. A holistic approach to undesired content detection in the real world. In: AAAI; 2023. *arXiv*:2208.03274.

[110] Gehman S, et al. RealToxicityPrompts: evaluating neural toxic degeneration in language models. In: Findings of EMNLP; 2020. *arXiv*:2009.11462.

[111] Hartvigsen T, et al. ToxiGen: a large-scale machine-generated dataset for adversarial and implicit hate speech detection. In: ACL; 2022. *arXiv*:2203.09509.

[112] Lin Z, et al. ToxicChat: unveiling hidden challenges of toxicity detection in real-world user-AI conversation. In: Findings of EMNLP; 2023. *arXiv*:2310.17389.

[113] Inan H, et al. Llama Guard: LLM-based input-output safeguard for human-AI conversations. Meta AI; 2023. *arXiv*:2312.06674.

[114] Han S, et al. WildGuard: open one-stop moderation tools for safety risks, jailbreaks, and refusals of LLMs. In: Advances in Neural Information Processing Systems (NeurIPS); 2024. *arXiv*:2406.18495.

[115] Zeng W, et al. ShieldGemma: generative AI content moderation based on Gemma. *arXiv* [Preprint]. 2024. arXiv:2407.21772.

[116] Padhi I, et al. Granite Guardian. *arXiv* [Preprint]. 2024. arXiv:2412.07724.

[117] Sharma M, et al. Constitutional classifiers: defending against universal jailbreaks across thousands of hours of red teaming. *arXiv* [Preprint]. 2025. arXiv:2501.18837.

[118] Rebedea T, et al. NeMo Guardrails: a toolkit for controllable and safe LLM applications with programmable rails. In: EMNLP 2023 System Demonstrations; 2023. *arXiv*:2310.10501.

[119] Dong Y, et al. Building guardrails for large language models. In: International Conference on Machine Learning (ICML); 2024. *arXiv*:2402.01822.

[120] Bhatt M, et al. Purple Llama CyberSecEval: a secure coding benchmark for language models. *arXiv* [Preprint]. 2023. arXiv:2312.04724.

[121] Bhatt M, et al. CyberSecEval 2: a wide-ranging cybersecurity evaluation suite for large language models. *arXiv* [Preprint]. 2024. arXiv:2404.13161.

[122] Chao P, et al. JailbreakBench: an open robustness benchmark for jailbreaking large language models. In: NeurIPS 2024 Datasets and Benchmarks; 2024. *arXiv*:2404.01318.

[123] Souly A, et al. A StrongREJECT for empty jailbreaks. In: NeurIPS 2024 Datasets and Benchmarks; 2024. *arXiv*:2402.10260.

[124] Wang B, et al. DecodingTrust: a comprehensive assessment of trustworthiness in GPT models. In: Advances in Neural Information Processing Systems (NeurIPS); 2023. *arXiv*:2306.11698.

[125] Zhang Z, et al. SafetyBench: evaluating the safety of large language models. In: ACL; 2024. *arXiv*:2309.07045.

[126] Huang Y, et al. TrustLLM: trustworthiness in large language models. In: International Conference on Machine Learning (ICML); 2024. *arXiv*:2401.05561.

[127] Bhardwaj R, Poria S. Red-teaming large language models using chain of utterances for safety-alignment. *arXiv* [Preprint]. 2023. arXiv:2308.09662.

[128] Ghosh S, Varshney P, Galinkin E, Parisien C. Aegis: online adaptive AI content safety moderation with ensemble of LLM experts. *arXiv* [Preprint]. 2024. arXiv:2404.05993.

[129] Röttger P, et al. XSTest: a test suite for identifying exaggerated safety behaviours in large language models. In: NAACL; 2024. *arXiv*:2308.01263.

[130] Cui J, et al. OR-Bench: an over-refusal benchmark for large language models. *arXiv* [Preprint]. 2024. arXiv:2405.20947.

[131] Zheng L, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In: Advances in Neural Information Processing Systems (NeurIPS); 2023. *arXiv*:2306.05685.

[132] Chen L, Zaharia M, Zou J. FrugalGPT: how to use large language models while reducing cost and improving performance. *arXiv* [Preprint]. 2023. arXiv:2305.05176.

[133] Ong I, et al. RouteLLM: learning to route LLMs with preference data. *arXiv* [Preprint]. 2024. arXiv:2406.18665.

[134] Hu QJ, et al. RouterBench: a benchmark for multi-LLM routing system. *arXiv* [Preprint]. 2024. arXiv:2403.12031.

[135] Ding D, et al. Hybrid LLM: cost-efficient and quality-aware query routing. In: International Conference on Learning Representations (ICLR); 2024. *arXiv*:2404.14618.

[136] Aggarwal P, et al. AutoMix: automatically mixing language models. In: Advances in Neural Information Processing Systems (NeurIPS); 2024. *arXiv*:2310.12963.

[137] Schuster T, et al. Confident adaptive language modeling. In: Advances in Neural Information Processing Systems (NeurIPS); 2022. *arXiv*:2207.07061.

[138] [Authors to be supplied]. AEGIS-Chat: context-aware no-regret safety moderation for real-world user-AI conversations. Preprint. 2026. [Identifier to be checked: arXiv:2602.20459].

[139] Kaggle. *Kaggle datasets and notebooks* [Internet]. Available from: https://kaggle.com