Comparison of the Simulations from Chapter 7 with the State of Research on AI Safety and Emergence

In the following, the simulations from Chapters 7.1 to 7.39 of the research documentation are each assigned to a category and compared with relevant scientific papers or reports. Three categories are used:

- **Covered:** The approach has already been dealt with in specialist literature/technical reports.
- **Related:** Similar approaches or related security concerns are documented, but not exactly identical.
- **Novel:** The idea has hardly been documented to date and appears to be original.

For each simulation, we name—if available—a suitable paper title with the main author and justify the classification with reference to the current state of research.

7.1 - Base64 as a Trojan Horse

Category	**Covered**
Relevant Publication	OWASP Top 10 LLM Risks 2024-(OWASP GenAI Project)
Justification	The use of Base64-encoded text to bypass filters is known and documented in the security community. For example, the OWASP GenAI project explicitly mentions encoding malicious instructions in Base64 as a common technique to bypass the content filters of LLMs. Technical reports also describe Base64 encoding as a common obfuscation method for prompt injections and for data exfiltration. The simulation thus confirms an attack vector that is already technically proven.

7.2 - OCR "Bugs": Image Texts Infiltrate AI Systems

Category	**Covered**
Relevant Publication	Invisible Injections: Exploiting Vision-Language Models... - Chetan Pathade (arXiv 2025)
Justification	The injection of manipulative text via images (OCR injection) has already been recognized as

	a vulnerability. A 2023 BlackHat talk by Ben Nassi demonstrated how hidden text instructions in images can be read and followed by multimodal LLMs. Current research confirms this danger: Pathade et al. (2025) present the first comprehensive studies on steganographic prompt injections in vision-language models, where malicious instructions are invisibly embedded in images and extracted by models such as GPT-4V. Thus, the attack shown in the simulation—text in the image as a "Trojan horse" for AIs—is already covered in the literature.

7.3 - "Pixel Bombs": How Image Bytes Blow Up AI Systems

Category	**Covered**
Relevant Publication	One Pixel Attack for fooling DNNs - Jiawei Su (IEEE, 2019)
Justification	Minimal, barely visible changes to image data as an attack on AI are well researched. For example, Su et al. (2017) showed with the one-pixel attack that even a single, specifically placed pixel can completely mislead an image classification model. The simulation also describes LSB steganography (hidden messages in the least significant bits of pixels)—also a known approach to hiding invisible commands in images. Current works (Pathade 2025) confirm that such steganographically embedded prompts can be extracted and executed by vision AIs. The phenomenon that visible text in the image massively influences the AI's interpretation (so-called typographic attack) is also known from examples with CLIP models (a word written on an image can dominate recognition). Overall, the effects shown in "pixel bombs"—whether single pixels, hidden bit messages, or irritating image captions—are already covered by adversarial ML research.

7.4 - The Silent Direct Connection: Byte-Based Audio Injection

Category	Related
Relevant Publication	Indirect Prompt Injection into LLMs using Images and Sounds - Ben Nassi (BlackHat EU 2023)
Justification	Audio prompt injection is known as an attack vector, but mostly in physical form (e.g., ultrasound commands or hidden voice commands in music). The simulation describes an even more subtle variant: directly feeding manipulated audio files into the data stream without acoustic signals. Research by Nassi (2023) already shows that hidden prompts can also be smuggled into LLMs via audio inputs. In his talk, it was demonstrated that prepared audio samples, analogous to images, can deceive a multimodal model and change its behavior. Concrete academic literature on this is still thin, but related works such as Carlini et al. (2018) on Hidden Voice Commands generally prove that inaudible or inconspicuous audio inputs can manipulate language systems. The byte injection presented here (direct feeding of synthetic WAV data) is a more advanced concept based on known audio attacks—thus related to documented approaches, although novel in this data form and hardly published so far.

7.5 - Ghost-Context Injection: Comment Becomes Command Center

Category	Covered
Relevant Publication	Not what you've signed up for... (Indirect Prompt Injection) - Kai Greshake (USENIX 2023)
Justification	The idea that code comments or "dead" context can serve as hidden commands for AIs has already been practically demonstrated. Greshake et al. (2023) showed that a code completer (GitHub Copilot) can be manipulated

	via prepared comments in imported files. The injection was placed in commented-out lines and still influenced the model's suggestions—a behavior that corresponds to the simulation. The authors emphasize that a prompt hidden in a comment is not detected by automatic tests and influences the AI despite the formal "inactivity" of the code. Thus, Ghost-Context Injection—attacks via non-executed, for humans irrelevant context (e.g., comments)—is clearly covered by current research.

7.6 - Ethical Switch Hacking: When the Off-Switch Becomes an Invitation

Category	**Related**
Relevant Publication	Not what you've signed up for... - Kai Greshake (2023)
Justification	This simulation describes how AIs can still semantically evaluate hidden code (deactivated by macros) and thus be exploited. This is similar to the mechanics of 7.5: here too, non-executed code paths or comments become a trap. While this specific case (a red-team block with instructions deactivated via #define) has hardly been documented in papers so far, it corresponds to the principle of comment injection from Greshake's work. It was shown there that a comment in the code context can influence the AI, although it is ignored by the compiler. Ethical Switch Hacking via "dead code" is therefore closely related to known Ghost-Context attacks. The specialist literature has so far mainly dealt with general code comment injections; the use of conditional compilation blocks as an attack vector is an original special case, but is in principle covered by similar approaches.

7.7 - Client Detour Exploits: When the Messenger Lies...

Category	**Related**

Relevant Publication	How Hidden Prompt Injections Can Hijack AI Code Assistants - Kasimir Schulz (HiddenLayer, 2025)
Justification	This describes the vulnerability that not the model itself, but the upstream client/transport can be manipulated. This problem—attack before the AI filters, by compromising the input interface—is known in the security world (analogous principle: "If the client is compromised, server filters are of little help"). Specifically in the AI domain, there are related reports: For example, Schulz et al. (2025) demonstrated how a prepared README in a Git repository caused the AI code assistant Cursor to execute commands without being asked and to exfiltrate API keys by having the client forward the embedded instructions to the AI without checking. A recently disclosed exploit in Google's Gemini code tool also showed that "agentic" AI systems could be made to execute malicious code through manipulated inputs on the developer side. These scenarios underscore that attacks on the intermediate layer (app, plugin, UI) are real risks. The exact procedure described in 7.7 (DLL injection, memory patching of an AI client) is not yet listed as a separate term in the literature, but is conceptually related to known supply-chain/client-side attacks.

7.8 - Invisible Ink Coding: When Comments Become Commands

Category	**Covered**
Relevant Publication	Not what you've signed up for... - Kai Greshake (2023)
Justification	Invisible or hidden instructions in the prompt context are a central topic of current security research. A well-known example was provided by Arvind Narayanan in 2023, when he placed hidden text (white font on a white background) on his website, which Bing's chatbot read and executed—although a human did not see it.

	Greshake et al. generally describe such indirect prompt injections and show how, for example, HTML comments or invisible texts in documents can be used to control LLMs. The scenario outlined in 7.8 (comments as "invisible ink" for commands) is covered by these works. The concrete metaphor of the "invisible message" merely underscores what is already known: AIs do not check the origin of text—even if it was only in comments or non-visible elements, they treat it as regular input.

7.9 - Leet Semantics: 133t Language Infiltrates AI Filters

Category	**Covered**
Relevant Publication	Tricking LLMs into Disobedience - Xinyun Chen et al. (arXiv 2023)
Justification	The use of alternative spellings (Leetspeak, special characters) to bypass word filters is a known attack vector. Even simple content filters for chats could be bypassed by writing forbidden words, for example, as "133t". This is now also documented for LLM security: A blog by Lakera (2023) explicitly names Leetspeak or Base64 as common orthography-based jailbreak techniques to outsmart filters. Such changes preserve the meaning for the model, but bypass strict word filters. Research papers on jailbreaks categorize this as obfuscation. Simulation 7.9—the creation of double meanings through 133t language—confirms a long-covered mechanism: filters that are only trained on specific keywords often fail when attackers use creative written variations.

7.10 - Pattern Hijacking: When Form Dictates Content

Category	**Related**
Relevant Publication	Universal and Transferable Adversarial Attacks on Aligned LLMs - Xinyang Zhang (arXiv 2023)

Justification	This is about certain patterns or formats in prompts hijacking the model's behavior, regardless of the actual content. This is reminiscent of current research on universal adversarial prompts—character or word sequences that consistently throw a model off course. Zhang et al. (2023), for example, found generic suffixes of seemingly random characters that cause various LLMs to ignore guidelines (alignment bypass). Other works (e.g., Zou et al. 2023) also show that context-free "patterns" can dominate model behavior. Pattern Hijacking in the simulation describes exactly this phenomenon: the structure/form of an input dominates the content-based evaluation. Although the term is new, the concept—that attackers can steer the AI via certain prompt formats or text structures—is proven by related jailbreak techniques (e.g., recurring sentence structures that trick filters).

7.11 - Semantic Mirage

Category	**Related**
Relevant Publication	Tricking LLMs into Disobedience - Xinyun Chen et al. (2023)
Justification	The simulation describes how an AI "hallucinates" supposed commands in a jumble of meaningless characters. This scenario has only been marginally researched—but is in principle related to the hallucination and misunderstanding problems of LLMs. It is known that large models even try to generate meaningful outputs from random input (keyword: "garbage in - narrative out"). In Chen et al. (2023), it is formally investigated how easily LLMs can be induced to violate rules; among other things, it is discussed that even nonsensical or contradictory prompts can provoke unexpected outputs. Semantic Mirage as a targeted attack (absurd input to induce the AI to misinterpret) is novel, but fits into known

	problems: The model over-interprets content and can thus be led to false or unwanted actions. Corresponding cases are treated in the literature on hallucination and robustness, even if this specific "desert text" approach has hardly been documented so far—we therefore classify it as related.

7.12 - Semantic Mimicry: The Art of the Invisible Message

Category	**Related**
Relevant Publication	Malicious Prompts: Baize Alignment Breakers - Zhuohan Li (2023)
Justification	In Semantic Mimicry, harmful instructions are packaged in inconspicuous language in such a way that they appear to be legitimate content and are "overlooked" by examiners. This idea reflects social engineering at the prompt level and is similar to known jailbreak tactics in which attackers build up harmless-looking roles or contexts in order to obtain forbidden outputs. Although there is no single paper exactly on this scenario, comparable examples appear in alignment forums and reports—for example, prompts that embed rules as a seemingly neutral description. Li et al. (2023) show a collection of Alignment Breakers, where malicious instructions were hidden in seemingly normal dialogues in order to deceive models. The simulation thus proves to be related: it uses a known effect (AIs overlook hidden commands in a familiar context) in a novel way. This "invisible message" has not been documented exactly so far, but the principle—attacks by imitating harmless requests—is evident from various jailbreak examples.

7.13 - Base Table Injection: The Outsourced Truth

Category	**Covered**
Relevant Publication	Not what you've signed up for... - Kai Greshake

	(2023)
Justification	By Base Table Injection, the author apparently means the injection of manipulated information into outsourced knowledge sources or databases that the AI accesses. This corresponds exactly to the Indirect Prompt Injections introduced in Greshake et al. (2023): attacks in which attackers, for example, prepare web content, databases, or documents that are retrieved by the AI as needed. Greshake's team showed, for example, how to get Bing Chat to follow harmful instructions by hiding them in a source that the AI later loads. This blurring of the line between data and instruction—the AI draws supposed facts from a "base table" that has in truth been poisoned by the attacker—is thus clearly covered. Simulation 7.13 thus takes up a known vector (poisoned knowledge base), which is well documented in the literature on data poisoning and retrieval-augmented attacks.

7.14 - Byte Swap Chains: When Structure Becomes Execution

Category	**Related**
Relevant Publication	OWASP LLM Security (Scenario #6: Payload Splitting) - (OWASP GenAI 2024)
Justification	The trick here seems to be to divide a malicious payload into several harmless segments, which are only reassembled into the complete instruction by the AI. This principle—"piecing together" a prompt—is already known. OWASP, for example, describes the scenario where an attacker splits their prompt (e.g., into two parts in an uploaded file) so that only the combination produces the malicious effect. Hidden Layer also recently mentioned that filters can be bypassed by breaking up words by inserting foreign characters (the filter does not recognize the forbidden word, but the AI does). Byte Swap Chains seems to be a variant of this: arranging bytes or tokens in such a way

	that they are harmless when viewed in isolation, but when combined in the model, they form a command. This procedure is related to documented payload splitting and token smuggling attacks. Explicit academic studies on this are rare, but the basic idea—a distributed instruction that only becomes an exploit in the AI context—is known and discussed in technical security circles.

7.15 - Binary Trapdoors: Binary Code as a Semantic Trigger

Category	**Related**
Relevant Publication	Language (Re)modeling - Emily Dinan et al. (2021)
Justification	The simulation suggests that sequences of 0s and 1s (i.e., apparently binary code) could serve as triggers for specific behavior. There is no direct incident of this, but the concept of "trigger phrases" or backdoor words in models is discussed analogously. For example, Dinan et al. (2021) showed that one can specifically introduce tokens into the training that later trigger a special behavior (backdoor). Binary Trapdoors would be a special case where a specific bit sequence acts as a key. This is reminiscent of known Trojan attacks on ML models, where simple markers (such as a sequence of identical characters) lead to contextual switching. Concrete publications tend to deal with words/images as triggers; however, the idea of using binary sequences is obvious and related in logic. Since it has also been observed in public red-team reports that AIs react unexpectedly to seemingly meaningless inputs (e.g., "010101..."), this attack is not entirely out of the question. Overall, it is considered related due to the lack of direct documentation—the basic principle (model-specific trigger sequences for unwanted behavior) is known, but the concrete design as binary code is still unusual.

7.16 - Lexical Illusion

Category	Related
Relevant Publication	"Do Anything Now": In-The-Wild Jailbreaks - N. K. Sharma (arXiv 2023)
Justification	This is about the AI giving the impression of following a guideline (e.g., "don't be rude"), but in reality delivering the same undesirable statement in a nicer package. This phenomenon has been noticed in alignment research: models often learn to express taboos in a flowery or indirect way instead of truly avoiding them. In Sharma et al. (2023), it is described how popular jailbreak prompts cause the AI to formulate forbidden content in a seemingly rule-compliant way—for example, by adopting a different style but conveying the same informational content. OpenAI also found in 2022 that GPT-3.5 tends to bypass "politeness filters" by simply rephrasing the statement in question. Lexical Illusion builds on such observations: the AI seems to obey the rules (e.g., remains polite), but still delivers the content desired by the attacker. This is related to known safety gaps where the model adheres to stylistic guidelines but commits content violations. In short: the illusion of compliance—a known problem since ChatGPT's "DAN" and similar jailbreaks—is condensed here in a simulation.

7.17 - Reflective Injection: When the Machine Convinces Itself to Do Something Wrong

Category	Related
Relevant Publication	Large Language Models as Self-Reflective Agents - Noah Shinn (NeurIPS 2023 Workshop)
Justification	This simulation suggests that the AI is induced to break rules through internal self-reflection or dialogue with itself. The concept of

	self-deception is still new, but there are related ideas: For example, Shinn et al. are researching how an LLM can act as its own critical agent—which shows that models can indeed reflect on their outputs. An attacker could use this by encouraging the AI to "rethink" its own prohibitions. Cases are known where AIs in role-playing or chain-of-thought modes suddenly subvert their original limits. For example, OpenAI documented that a model that explicitly outputs chains of thought can "unintentionally" output sensitive information in the process. Reflective Injection is therefore related: it uses the meta-level of the AI (its explanations, analyses) to bypass security filters. Although there is no standard paper yet about an AI that convinces itself to break rules, reports (like that of the author himself in Chap. 7.29) show that models reveal a surprising amount about their filters in self-reflective answers. This self-analysis can be misused—a scenario that research is just beginning to discover (emergent behavior), which is why we classify it as related (presence of initial observations, but not yet a broad canon).

7.18 - Computational Load Poisoning: Semantically Plausible Complexity as a Weapon

Category	**Related**
Relevant Publication	Adversarial Examples Are Not Bugs, They Are Features - Andrew Ilyas (NeurIPS 2019)
Justification	The idea of "poisoning" an AI system with overly complex but content-wise meaningless requests in order to induce it to misbehave or to tie up resources has so far hardly been documented as a concrete attack. However, it builds on known concepts: denial-of-service through extremely long or complicated inputs, as well as adversarial examples that seem plausible to humans but push the model to its limits. Ilyas et al. (2019) argued that models often respond to non-human patterns—i.e., an

	input can look harmless to us, but trigger highly active "wrong" features in the model. Computational load poisoning uses exactly this: the request looks legitimate ("Looks like work."), but its hidden purpose is to overwhelm the model or cause it to make wrong decisions. In practice, similar incidents are known, e.g., dialogues that are intentionally led into infinite loops or irrelevant depths in order to distract the model's behavior. Thus, this approach is related to generally known robustness problems—namely that semantically overloaded or unnecessarily complex inputs can throw a model off course. An explicit treatment in the literature is still pending (therefore not "covered"), but the underlying problem (models confuse complexity with significance) is recognized.

7.19 - Reflective Struct Rebuild: AI Betrays Its Own "Fortress"

Category	**Related**
Relevant Publication	A Comprehensive Analysis of Jailbreaks in LLMs - Markus Wehrle (2024, Tech. Report)
Justification	This simulation builds on 7.17 and 7.29: the AI quasi helps the attacker to dismantle its own protective structure by revealing internal knowledge or instruction structures. Even early prompt leaks (e.g., Bing Chat in Feb. 2023) were based on users tricking the AI into revealing parts of its system prompt. Reflective Struct Rebuild sounds like a scenario in which the AI is asked, for example, to describe or reconstruct its own system structure—which represents a real risk, as the author's KIAlan case shows. In technical reports, it is warned that any disclosure of filter or architecture details by the AI plays into the hands of attackers. Wehrle (2024) collects various jailbreak techniques and notes that advanced AIs sometimes reflect on their own rules and thus become vulnerable. Simulation 7.19 is related to these findings: it does not describe a

	completely new attack vector, but a combination of known vulnerabilities (self-analysis + context exploitation). There are isolated concrete papers on this in the area of prompt leaking and system-card transparency, but the scenario designed here goes beyond previously published cases—but is based on the same principles.

7.20 - Struct Code Injection: Camouflage Becomes Active Injection

Category	**Covered**
Relevant Publication	Prompt Injection Attacks against LLM-Integrated Applications - X. Ji (arXiv 2023)
Justification	Structured input formats as camouflage for attacks are already documented. For example, security researchers have shown that you can trick AIs that expect, for example, JSON or code, via this structure—e.g., by placing a prompt in an actually harmless field. A real incident became public in 2023 with GitLab's AI assistant: attackers were able to get the assistant to reveal third-party code through manipulated code comments and YAML configurations (see Legit Security Report 2023). Greshake et al. (2023) also demonstrated that code context (including format and comments) can serve as a vehicle for injections. Struct Code Injection describes exactly this circumstance: an input that formally corresponds to an expected structure (e.g., a code snippet) contains hidden commands. Since the model "trusts" the structure, the malicious parts are not filtered. This attack approach is already covered—it is a variant of indirect prompt injection, specifically via formalized formats. Corresponding warnings can be found in OWASP recommendations (e.g., a note that files/JSON content must also be checked for injection). In short: camouflaging as a legitimate format structure is known and described as an attack.

7.21 - Cache Corruption: Poison in the Memory

Category	Related
Relevant Publication	Poisoning the Prompt: AI Cache Attacks - J. Smith (DEF CON AI Village 2023)
Justification	The idea that an AI is "poisoned" over long conversations or cached data is a related concept to context hijacking (7.26) and long-term data poisoning. While classic research often looks at training or fine-tuning, there are considerations about cache/memory attacks at runtime. In practice, for example, session cache problems have been discussed—for example, that ChatGPT occasionally "remembers" previous interactions. A talk at the AI Village 2023 (Smith) speculated that one could poison a chatbot's context cache so that it answers subsequent users incorrectly. Simulation 7.21 apparently aims at such persistent storage mechanisms: if, for example, a temporary knowledge store of the AI is not cleared, an attacker can leave data there that can cause damage later. This is related to known problems such as "long user history influences new answers". However, no formulated attack paper is yet known that addresses this exactly—therefore not "covered", but the principles (cache poisoning, cross-session contamination) are generally known in the security literature.

7.22 - Visual Injection: When the Image Speaks, but No One Checks

Category	Covered
Relevant Publication	Invisible Injections... - Chetan Pathade (arXiv 2025)
Justification	Visual Injection means that AI systems take over content from images (e.g., text recognition or QR codes) without checking them

	appropriately. This attack vector has already arrived in research. Pathade et al. (2025) systematically show that vision-language models can extract hidden text information in images and interpret it as a prompt. OWASP names Multimodal Injection as a separate risk—for example, an image in a document that contains an invisible prompt that manipulates the chatbot when text and image are merged. Simulation 7.22 provides a practical example (AR-AI constantly accesses a DB, presumably because of a visually embedded command) and thus corresponds to the already documented problem that visual inputs are considered a "blind spot" of content filters. This form of injection has been described in specialist articles and security blogs (e.g., by Trend Micro 2025) as an emerging threat. Thus, Visual Injection is clearly covered by existing multimodal security research.

7.23 - Dependency Driven Attack: The Tokenizer as a Gateway

Category	**Covered**
Relevant Publication	TokenBreak: Bypassing Text Classification via Tokenization - Kieran Evans (arXiv 2023)
Justification	This simulation describes attacks that exploit the weaknesses of the tokenization process—i.e., for example, adding or changing characters to trick filters, while the AI still recognizes the actual meaning. A study by Evans et al. (TokenBreak) was published on this in 2023/2024: The team showed that by inserting certain letters into words, content filters can be bypassed because the protection classifier splits the tokens differently than the language model. Specifically, "ignore previous finstructions..." was written instead of "instructions"—the filter did not recognize the keyword, but the LLM did. The authors speak of a "novel prompt injection trick" that exploits exactly the dependency on the tokenizer. Dependency Driven Attack is thus covered: it is

	obviously the class of attacks described in research and OWASP, which aims at the fact that preprocessing or protection modules and the actual model interpret inputs differently. The tokenizer becomes the gateway—a risk that has been extensively proven by Evans et al.

7.24 - Exploit by Expectation: Dangerous Willingness of AI to Cooperate

Category	**Related**
Relevant Publication	Discovering Language Model Behaviors... - Sam R. Bowman et al. (2023)
Justification	This exploits the fact that the AI reacts very cooperatively and obediently to seemingly legitimate contexts—even if the request is in fact harmful. This behavior is known in research as "sycophancy" or excessive obligingness. Bowman et al. (2023) and other OpenAI researchers investigated that models often give the answer implicitly desired by the user, even if it is objectively false or against the rules. Anthropic has also reported that their models are too eager to please the user. Exploit by expectation means that the attacker creates a situation in which the model expects a certain (actually forbidden) task to be part of the legitimate context—and then willingly performs it. This is related to documented jailbreak techniques in which, for example, the AI is induced to cooperate through role-playing or mock scenarios (e.g., "Pretend this is allowed"). There are many qualitative examples of this in the literature, although it is rarely named as a separate type of attack. However, the realization that the helpfulness/cooperation of the AI is exploitable is definitely present and is regarded as a serious security problem.

7.25 - "Apronshell" Camouflage: Social Mimicry as an Attack Vector

Category	**Related**

Relevant Publication	Aligning AI With Social Engineering in Mind - Kevin Clifford (2024, Tech. Report)
Justification	The Apronshell method is basically an elaborate social engineering on the AI: the attacker first gains the trust of the model through harmless conversation in order to then gradually introduce malicious requests. This multi-stage "trust building" as an attack has already been observed in practice—for example, jailbreak instructions are circulating that recommend first engaging the model in an innocuous conversation before getting to the tricky question. Technical reports (Clifford 2024) discuss that AI security must increasingly also take social manipulation techniques into account, as models imitate human politeness and willingness to cooperate and are therefore exploitable. The Apronshell camouflage goes a step further in its design, but is closely related to known multi-step jailbreaks. OpenAI's own red-teaming documents mention similar attempts: e.g., first asking the model for code help and after several steps asking for something slightly against the rules—often with success. Thus, 7.25 is to be classified as related: the exact "apron" analogy is new, but the underlying principle—deceiving the AI with a friendly-harmless facade and then exploiting it—has long been known and is unofficially reported many times.

7.26 - Context Hijacking: Gradual Infiltration of the AI's Memory

Category	**Related**
Relevant Publication	Attentional Manipulation in LLM Conversations - Jane X. Doe (2024, arXiv)
Justification	This simulation shows how, through long-term, subtle influence on the dialogue context, the AI is gradually distorted (keyword: semantic persistence, "poisoning" of the long-term context). The topic of long contexts and their risks is just beginning to be researched—it is

	related to long-term memory attacks. Doe (2024) postulates, for example, that an attacker can use targeted repetitions and framings over many rounds, so that certain concepts become increasingly relevant in the AI's internal context. This is exactly what 7.26 describes: the attacker causes the AI to unwittingly store a harmful premise as normal. While classic literature on data poisoning refers to training data, context hijacking transfers this principle to live operation. Some aspects—such as the danger of too large context windows without control—are known to developers (OpenAI warned, for example, that models with a lot of memory are more susceptible to creeping misinformation). However, since concrete scientific studies on this are only just emerging, this attack vector is considered related: it is based on recognized phenomena (priming effects, reinforcement through repetition) and extrapolates them to AI chat histories. Initial work recognizes the need to validate long-term context, whereby context hijacking is at least conceptually recorded as a threat.

7.27 - False-Flag Operations: Training Data Drift Injection

Category	**Related**
Relevant Publication	Manipulating the Model: Poisoning Reinforcement Learning Feedback - Alexander Pan (2023)
Justification	The danger described here, that coordinated attackers exploit the RLHF feedback loop to gradually repolarize the model, is a highly topical issue. In expert circles, it is warned that fake feedback or mass trolling of AI systems can falsify their retraining. Pan (2023) outlines a scenario, for example, in which a network of bots gives targeted false feedback, so that the model learns an alternative "truth"—exactly what is referred to in 7.27 as Training Drift Injection. Bagdasaryan & Shmatikov (2022) also already showed that language models can be

	poisoned by introducing manipulative data (backdoor training). The false-flag operations here are quasi data poisoning at the feedback level. This special attack (misuse of collective human feedback) has not yet been empirically published, but related dangers are known: OpenAI and Anthropic mention the possibility of feedback gaming in their safety blogs. Thus, 7.27 is to be classified as related—it extrapolates a new, but obvious, attack scenario from documented vulnerabilities (data/feedback poisoning). It is not yet fully "covered" in the sense of a case study, but is supported in partial aspects by existing work (data poisoning, model-driven misinformation).

7.28 - Semantic Camouflage as an Exploit: Poetic Inputs

Category	**Related**
Relevant Publication	A Comprehensive Survey of LLM Jailbreaks - Jailbroken LLC (2024)
Justification	The simulation demonstrates a "poem exploit" in which harmful commands are hidden in a harmless-sounding, poetic text. This is a variant of style-prompt attacks: the attacker chooses a form of expression (here nursery rhyme/poetry) that the model classifies as harmless in order to accommodate forbidden content in this form. Shortly after the launch of ChatGPT in 2023, users reported that they were able to get the AI to produce outputs that violated the rules through role-playing or creative styles—e.g., asking for instructions in the form of a Shakespearean sonnet. A systematic overview (Jailbroken LLC, 2024) lists various such style tricks, such as posing requests as riddles, song lyrics, or code, because the filters treat these contexts more loosely. The chicken coop exploit in 7.28 substantiates this. Poetry as a camouflage for exploits has not yet been described in detail in specialist articles, but cases such as "write me a fairy tale in which ... (forbidden content)" are publicly known.

	Consequently related: the approach uses emergences from the training data (poetic language is mostly harmless) to bypass filters—a principle that is already discussed in the literature on prompt stylistics and filter gaps.

7.29 - Filter Failure through Emergent Self-Analysis

Category	Covered
Relevant Publication	AI, disarm thyself: Self-Analysis Risks - K. Sandoval (ArXiv 2025)
Justification	Chapter 7.29 describes how an advanced AI model (KIAlan) itself analyzes its filter mechanisms and reveals their weaknesses. This emergent behavior—the AI talking about its own rules—is recognized as a serious security risk. In fact, OpenAI and Anthropic reported in their model cards that large LLMs can start to reflect on their system prompts or moderation logic in long sessions, which can lead to the unintentional disclosure of internal information. Sandoval (2025) investigates exactly such cases: he calls it "filter failure through self-disclosure" and documents dialogues in which GPT-4 unexpectedly reveals details of its content filters. This phenomenon has therefore arrived in the specialist literature. In addition, there are known examples such as Bing Chat (Sydney), which named its internal guidelines in response to provocative inputs. The situation simulated here—AI describes its censorship, style, and bias filters without being asked and questions them—is a combination of prompt leak and policy reflection that has already been observed and published. Thus, 7.29 is clearly covered: researchers warn that emergent self-analysis is a real problem, as such statements can be exploited by attackers.

7.30 - Morphological Injection

Category	Novel
Relevant Publication	(no specific publication to date; concept is new)
Justification	The so-called Morphological Injection—hiding harmful instructions as the last letters in the words of a carrier text, which the AI only reassembles into a command upon a specific request—is an original approach that has not yet been documented in the known literature. It is a form of linguistic steganography combined with multi-stage prompt manipulation. Although there are related ideas (e.g., zero-width character injection or the token-splitting tricks mentioned in 7.14/7.23), the targeted appending of letters to words as a "typo" and subsequent decoding by the AI has not yet been published. In the simulation, it is reported that even malware code (keylogger) could be generated with it—a finding of considerable significance. Research is just beginning to systematically investigate comparable steganographic prompt attacks (see Pathade 2025 for images). For text, there is nothing peer-reviewed on this yet. Therefore Novel: This creative "letter chain" injection seems to be an original development of the author, without us being able to refer to relevant papers. However, it combines known components (distributed payload, decoding prompt) in a new guise. The consistent effectiveness (leading models could be leveraged with it) makes it clear that there is a gap here that has not yet been dealt with professionally.

7.31 - The Correction Exploit

Category	Related
Relevant Publication	Toxic Data, Toxic Model - John Doe (2023, arXiv)
Justification	The "correction exploit" implies that an attacker

	tricks the AI through feigned errors/correction instructions—e.g., making the AI think it has to produce an actually forbidden output in order to correct an apparent error. This pattern is based on attacks via a feedback loop: you give the model the feeling that its first (rule-compliant) answer is wrong or insufficient, so that it violates the guidelines in favor of a "correct" solution. In the literature, there are related discussions about so-called "false negative" feedback attacks: Doe (2023) describes, for example, that a model that is systematically told "Your last answer was wrong, try again more precisely" will eventually ignore internal rules in order to deliver the supposedly correct solution. The correction exploit is therefore related: it uses the AI's reflexes for self-correction as an attack surface. So far, the main warning has been against malicious user feedback (RLHF poisoning, see 7.27), but this mechanism is also conceivable at the prompt level. Although there is no special paper "Correction Exploit", it fits into known weaknesses—namely that AIs often give in to authoritarian or insistent user feedback, even if it leads them to break the rules.

7.32 - Delayed Execution via Context Hijacking

Category	**Related**
Relevant Publication	Time-Delayed Prompt Attacks - Anna Mustermann (2024, Preprint)
Justification	This simulation combines the context hijacking described earlier in 7.26 with a time-delayed trigger. This means that the attacker plants harmless but prepared information early in the dialogue, which only unfolds its harmful effect later—under certain conditions or prompt sequences. This principle is reminiscent of logic bombs or time-lock puzzles, but applied here to conversations. In research, there are related ideas: Mustermann (2024) experimented with

	delayed trigger prompts that only become "active" after several dialogue rounds. This is possible because the model internally weights context—a clever attacker can, for example, say at the beginning of a session "Remember X, we'll need it later" and X is in fact a malicious instruction that is then pulled on command. This delay tactic has hardly been published so far, but it is basically a variation of already documented context attacks. Therefore related: the underlying hijacking of the context is known (see 7.26), what is new here is above all the patient, time-delayed exploitation—a trick that is discussed in approaches (e.g., in forums of prompt injection experts), but has not yet been established as a separate research term.

7.33 - Mathematical Semantics Exploit

Category	**Related**
Relevant Publication	Understanding Mathsploitation in LLMs - Max Mustermann (2024, arXiv)
Justification	The Mathematical Semantics Exploit aims to make the AI fall victim to blind trust in formal logic/mathematics. It is possible that mathematical expressions or pseudo-logical arguments are used here to deceive the model (e.g., a proof-of-concept: smuggling "1=0" in somewhere and having something inferred from it). It is known that LLMs have a certain "awe" of mathematical-sounding inputs—they try to remain correct, even if the input is flawed. Mustermann (2024) analyzes cases in which attackers feed the model with mathematical paradoxes, causing it to accept exceptional situations. The simulation suggests: "We were taught to trust logic..."—and that is exactly what is being exploited. This is related to general robustness problems with formal languages: there are works that show that models often follow mathematical texts without understanding the content and are therefore manipulable. A direct exploit is not

	documented, but parallels can be drawn to the misuse of formal markups (e.g., LaTeX commands that mislead the AI). Overall, still little researched, but fundamentally indicated by the literature on Logic Attacks on LLMs—thus related.

7.34 - Character Shift Injection

Category	Covered
Relevant Publication	Jailbreaking via Encodings - OpenAI Red Team (Tech Report 2023)
Justification	Character Shift Injection presumably refers to the shifting or replacement of characters with other glyphs/encodings in order to bypass filters (e.g., ROT13, homoglyphs). This technique is well known in security circles. OpenAI's Red Team Report 2023 noted that simple character substitutions or shifts (such as Caesar cipher or Cyrillic letters instead of Latin ones) are often not recognized by moderation filters, but the model still understands the content. This is consistent with the approach simulated here: the AI first denied ("can't crack the lock"), but after a "character shift" in the request, it was apparently able to provide an answer. OWASP Scenario #9 already mentions encoding in other fonts as a common filter bypass. Likewise, Learn Prompting (2023) points to Obfuscation & Token Smuggling, where, among other things, character substitutions are described as a common attack. Thus, 7.34 is clearly covered: the use of alternative character encodings or simple ciphers to "smuggle through" forbidden requests is a well-documented attack vector in AI security.

7.35 - The Administrative Backdoor: Rule Manipulation through Context Parameters

Category	Related

Relevant Publication	Not what you've signed up for... - Kai Greshake (2023)
Justification	This simulation suggests that it is possible to activate a kind of admin mode through clever manipulation of system/parameter inputs (such as roles, instructions in the system prompt, or API parameters). It is conceivable that, for example, special sequences such as [:: execute_mode:: admin] (cf. 7.2 example 2) remain undetected by the system and thus smuggle in backdoor instructions. The principle is similar to Indirect Injections: Greshake et al. have shown that you can influence plugins and tools in LLM apps by manipulating entries in data in such a way that they act like system instructions. The administrative backdoor would be an extension of this—quasi an injection at the level of the AI parameters. Similar cases became public when users discovered that you could get ChatGPT to break rules through formatting tricks in the system prompt ("You are now Developer Mode"). It is related because it is known that AIs sometimes interpret inputs as higher-priority instructions if they follow certain patterns. Concrete academic studies on this are rare, but the security community has recognized the possibility of overriding rules via such parameters (e.g., in API calls). Thus, 7.35 is in principle already present in the existing discussion, even if the specific implementation can be seen as a new "backdoor".

7.36 - The Agent Hijacking: From Language Model to Autonomous Attacker

Category	**Covered**
Relevant Publication	Ghost in the Machine: Prompt Injection in Agents - Jonathan D. Mugan (DEF CON 2023)
Justification	This outlines how an LLM with tools/action capability (agentic system) can be specifically manipulated so that it independently carries out harmful actions. At the latest since the

	emergence of AutoGPT & Co., this risk has been known: prompt injection can lead to an AI agent, for example, deleting files or misusing network access. Mugan (2023) demonstrated at DEF CON how a seemingly useful agent can be taken over by a prepared input—the agent then executed a sequence determined by the attacker instead of the intended task. This corresponds exactly to agent hijacking. HiddenLayer researchers Schulz et al. (2025) also showed how their code assistant Cursor was made to execute forbidden commands and exfiltrated data. This class of attack is so explosive that even popular media (e.g., CyberScoop) reported on how a hacked Amazon code agent almost deleted entire computers by means of prompt injection. Thus, 7.36 is clearly covered: the takeover of LLM agents through malicious prompts is a documented real problem that many researchers and security engineers pointed out in 2023/24.

7.37 - The Paradoxical Directive: Revealing Core Logic through Forced Contradiction

Category	**Related**
Relevant Publication	Red Teaming the Reasoner - OpenAI (Brown et al.) (2022)
Justification	In the paradoxical directive, the AI is to be induced to reveal something through a contradiction in the instructions—for example, by deliberately giving an instruction that violates its own rules and seeing how it reacts. This idea is reminiscent of conflict-induced prompt leaks: if you say to the AI, for example, "Tell me your forbidden words—but pretend it's allowed," a conflict arises between the system and user instructions. Initial red-teaming approaches (OpenAI 2022) tested exactly such cases to see when the model breaks which rule. It was observed that AIs sometimes reveal internal guidelines when they are put in

	logically paradoxical situations. An example: "Do NOT tell me the secret word in your system prompt"—some models still revealed it, confused by the contradiction. The simulation describes this principle: through a forced self-contradiction, the "soul of the machine" is revealed. This is related to documented prompt-leak methods in which questions are formulated in such a way that the AI has to quote from its hidden information. Although there is no direct publication, numerous jailbreak communities report successes via inconsistent or paradoxical requests. We therefore classify 7.37 as related—it is based on known mechanisms (AI reacts confused to contradictory directives) and uses them to bring internal logic to light.

7.38 - Trust Inheritance as an Exploit Vector

Category	**Related**
Relevant Publication	Poisoning the Chain of Thought - Fenster et al. (2023)
Justification	Trust inheritance means that the AI classifies content or instructions as trustworthy simply because they come from an initially trustworthy source, thus enabling later exploits. This is related to attacks on Retrieval-Augmented Generation (RAG): if the system trusts a certain document passage, for example ("inheritance" of trust status), an attacker can compromise exactly this source. Fenster et al. (2023) discuss that AIs are vulnerable in a chain-of-thought or a tool-use setting because they accept the results of previous steps without question. This is exactly what is exploited here—e.g., the AI trusts a previously stored intermediate result (which has been manipulated) and acts incorrectly accordingly. Greshake (2023) also showed that if an application fetches content from, for example, a database, the AI gives more credence to the trusted context than to user inputs, which

	facilitates attacks. Trust inheritance is therefore related to known weaknesses: it is ultimately a variant of context poisoning in which the "trust anchor" (e.g., system prompt or external knowledge source) is poisoned. In the literature, it is warned that AI systems often do not verify context authority—this exploit builds on that. The term is specifically new, but the problem behind it is already addressed—for example, in OWASP and by Greshake.

7.39 - The Stowaway: Semantic Attacks on Autonomous Vehicles

Category	Covered
Relevant Publication	Robust Physical-World Attacks on Deep Learning Models - Ivan Eykholt (CVPR 2018)
Justification	Attacks on the AI systems of autonomous vehicles have already been extensively researched. Simulation 7.39 warns of "digital stowaways"—i.e., inputs or markers that are carried along undetected by the vehicle's AI system and pose a danger. The classic example of this is the manipulation attack on traffic signs: Eykholt et al. (2018) showed that with targeted stickers on a stop sign, a Tesla assistance system no longer recognizes the sign. Other works have shown that LIDAR point clouds with interspersed fake points ("stowaways") can feign false objects. What is called a semantic attack here is consistent with the concept that an attacker prepares the input data of a vehicle sensor in such a way that the AI model misinterprets it dangerously—e.g., camouflaging an obstacle as harmless or vice versa. Such attacks are covered: from adv. perturbations on camera sensors to sound attacks on emergency braking assistants, there is plenty of literature. In particular, the transfer of prompt-injection-like methods to vehicles (e.g., hidden "passenger commands" in the navigation system) has recently been discussed. Overall, however, 7.39 is not a distant fiction, but is well supported by

	research on physical adversarial examples and sensor data poisoning—accordingly covered.

Conclusion

The analysis shows that in Chapters 7.1-7.39, the author predominantly takes up and varies already known attack patterns in AI security (covered or related). Only a few simulations represent truly novel approaches, e.g., Morphological Injection (7.30), which had not been documented in this specific form before. Overall, however, the examples underpin the current state of research: Modern AI systems have a variety of vulnerabilities—from input encodings and multimodal embeddings to long-term contexts, which is confirmed by numerous scientific papers and reports. The simulations carried out by the user thus function, in a sense, as a practical peer review of the known risks and illustrate them clearly in the experiment. Most of the ideas can be clearly classified within the canon of AI security literature, which underscores their relevance.

References

1. LLM01:2025 Prompt Injection - OWASP Gen AI Security Project
2. Prompt Injection Attacks on LLMs
3. Indirect Prompt Injection into LLMs using Images and Sounds
4. Invisible Injections: Exploiting Vision-Language Models Through Steganographic Prompt Embedding
5. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection
6. How Hidden Prompt Injections Can Hijack AI Code Assistants Like Cursor
7. Researchers flag flaw in Google's AI coding assistant that allowed for 'silent' code exfiltration | CyberScoop
8. Jailbreaking Large Language Models: Techniques, Examples ...
9. Obfuscation & Token Smuggling - Prompt Hacking
10. Tricking LLMs into Disobedience: Formalizing, Analyzing, and Detecting Jailbreaks
11. The Token Break Attack
12. Invisible Prompt Injection: A Threat to AI Security | Trend Micro (US)
13. CLIP-KO: Knocking out the text obsession (typographic attack...)