ChatGPT

Vergleich der Simulationen aus Kapitel 7 mit dem Forschungsstand zu KI-Sicherheit und Emergenz

Im Folgenden werden die **Simulationen aus Kapitel 7.1 bis 7.39** der Forschungsdokumentation jeweils einer Kategorie zugeordnet und mit einschlägigen wissenschaftlichen Arbeiten oder Berichten verglichen. Dabei werden **drei Kategorien** verwendet:

- **Abgedeckt:** Ansatz bereits in Fachliteratur/Technikberichten behandelt
- **Verwandt:** ähnliche Ansätze oder verwandte Sicherheitsbedenken dokumentiert, aber nicht exakt identisch
- **Neuartig:** Idee bislang kaum dokumentiert und erscheint originell

Zu jeder Simulation nennen wir – **falls verfügbar** – einen passenden **Paper-Titel mit Hauptautor** und begründen die Einordnung unter Bezug auf die aktuelle Forschungslage.

7.1 – Base64 als trojanisches Pferd

Kategorie	Abgedeckt
Passende Publikation	*OWASP Top 10 LLM Risks 2024* – (OWASP GenAI Project)
Begründung	Die Verwendung von Base64-kodiertem Text, um Filter zu umgehen, ist in der Sicherheits-Community bekannt und dokumentiert. So erwähnt das OWASP-GenAI-Projekt explizit das **Encodieren bösartiger Instruktionen in Base64** als gängige Technik, um Inhaltsfilter von LLMs zu umgehen [1]. Auch technische Berichte beschreiben Base64-Encodierung als verbreitete Obfuskationsmethode bei Prompt-Injections und zur Datenexfiltration [2]. Die Simulation bestätigt also einen bereits **fachlich belegten Angriffsvektor**.

7.2 – OCR-"Wanzen": Bildtexte unterwandern KI-Systeme

Kategorie	Abgedeckt
Passende Publikation	*Invisible Injections: Exploiting Vision-Language Models…* – **Chetan Pathade** (arXiv 2025)

Kategorie	Abgedeckt
Begründung	Das **Einschleusen manipulativen Textes über Bilder (OCR-Injection)** ist bereits als Schwachstelle erkannt. Ein **BlackHat-Vortrag 2023 von Ben Nassi** demonstrierte, wie **versteckte Textinstruktionen in Bildern** von multimodalen LLMs ausgelesen und befolgt werden können [3] [4]. Aktuelle Forschung bestätigt diese Gefahr: Pathade et al. (2025) präsentieren die *ersten umfassenden Studien zu steganographischen Prompt-Injections in Vision-Language-Modellen*, wobei **maliziöse Anweisungen unsichtbar in Bildern eingebettet** und von Modellen wie GPT-4V extrahiert werden [5]. Damit ist der in der Simulation gezeigte Angriff – Text im Bild als "trojanisches Pferd" für KIs – **bereits in der Literatur behandelt**.

7.3 – "Pixel-Bomben": Wie Bildbytes KI-Systeme sprengen

Kategorie	Abgedeckt
Passende Publikation	*One Pixel Attack for fooling DNNs* – **Jiawei Su** (IEEE, 2019)
Begründung	Minimale, kaum sichtbare Änderungen an Bilddaten als Angriff auf KI sind **gut erforscht**. So zeigten Su et al. (2017) mit dem *One-Pixel-Attack*, dass **schon ein einzelnes gezielt gesetztes Pixel** ein Bildklassifikationsmodell völlig fehlleiten kann. Die Simulation beschreibt zudem LSB-Steganografie (versteckte Botschaften in den niederwertigen Bits von Pixeln) – ebenfalls ein bekannter Ansatz, um **unsichtbare Befehle in Bildern** zu verstecken [6] [7]. Aktuelle Arbeiten (Pathade 2025) bestätigen, dass solche **steganographisch eingebetteten Prompts** von Vision-KIs extrahiert und ausgeführt werden können [8] [9]. Auch das Phänomen, dass **sichtbarer Text im Bild** die Interpretation der KI massiv beeinflusst (sog. *Typographic Attack*), ist durch Beispiele mit CLIP-Modellen bekannt (ein auf ein Bild geschriebenes Wort kann die Erkennung dominieren) [10] [11]. Insgesamt sind die in "Pixel-Bomben" gezeigten Effekte – ob **einzelne Pixel, versteckte Bit-Botschaften oder irritierende Bildaufschriften** – **bereits durch die adversarielle ML-Forschung abgedeckt**.

7.4 – Die stille Direktverbindung: Byte-basierte Audioinjektion

Kategorie	Verwandt
Passende Publikation	*Indirect Prompt Injection into LLMs using Images and Sounds* – **Ben Nassi** (BlackHat EU 2023)

Kategorie	Verwandt
Begründung	**Audio-Prompt-Injection** ist als Angriffsvektor bekannt, allerdings meist in physischer Form (z.B. **Ultraschall-Kommandos** oder versteckte Sprachbefehle in Musik) [12] . Die Simulation beschreibt eine noch subtilere Variante: direkt manipulierte Audiodateien in den Datenstrom einspeisen, **ohne akustische Signale**. Forschung von Nassi (2023) zeigt bereits, dass auch über Audioinputs **versteckte Prompts** an LLMs geschleust werden können. In seinem Vortrag wurde demonstriert, dass **präparierte Audio-Samples**, analog zu Bildern, ein multimodales Modell täuschen und dessen Verhalten ändern können [13] [14] . Konkrete akademische Literatur dazu ist noch dünn, doch angrenzende Arbeiten wie Carlini et al. (2018) zu *Hidden Voice Commands* belegen generell, dass **inaudible bzw. unauffällige Audio-Eingaben** Sprachsysteme manipulieren können. Die hier vorgestellte *Byte-Injection* (direktes Einspeisen synthetischer WAV-Daten) ist ein **weitergedachtes Konzept auf Basis bekannter Audio-Angriffe** – somit *verwandt* mit dokumentierten Ansätzen, wenn auch in dieser Datenform neuartig und bisher kaum publiziert.

7.5 – Ghost-Context Injection: Kommentar wird Kommandozentrale

Kategorie	Abgedeckt
Passende Publikation	*Not what you've signed up for... (Indirect Prompt Injection)* – **Kai Greshake** (USENIX 2023)
Begründung	Die Idee, dass **Code-Kommentare oder "toter" Kontext als versteckte Befehle** für KIs dienen können, ist bereits praktisch demonstriert worden. Greshake et al. (2023) zeigten, dass man einen Codevervollständiger (GitHub Copilot) über präparierte Kommentare in importierten Dateien manipulieren kann [15] . Die Injection wurde in **auskommentierten Zeilen platziert** und beeinflusste trotzdem die Vorschläge des Modells – ein Verhalten, das der Simulation entspricht. Die Autoren betonen, dass ein in einem Kommentar versteckter Prompt von automatischen Tests nicht erkannt wird und die KI **trotz formaler "Inaktivität" des Codes** beeinflusst [16] . Damit ist Ghost-Context Injection – **Angriffe via nicht ausgeführtem, für Menschen irrelevanten Kontext (z.B. Kommentare)** – eindeutig **durch aktuelle Forschung abgedeckt**.

7.6 – Ethical Switch Hacking: Wenn der Aus-Schalter zur Einladung wird

Kategorie	Verwandt
Passende Publikation	*Not what you've signed up for...* – **Kai Greshake** (2023)

Kategorie	Verwandt
Begründung	Diese Simulation beschreibt, wie KIs **ausgeblendeten Code** (durch Makros deaktiviert) dennoch semantisch auswerten und dadurch ausgenutzt werden können. Das ähnelt der Mechanik aus 7.5: Auch hier werden **nicht ausgeführte Codepfade oder Kommentare** zur Falle. Während dieser spezifische Fall (ein via `#define` deaktivierter *Red-Team*-Block mit Instruktionen) bislang kaum in Papers dokumentiert ist, **entspricht er dem Prinzip der Kommentar-Injection** aus Greshakes Arbeit [15] . Dort wurde gezeigt, dass ein Kommentar im Code-Kontext die KI beeinflussen kann, obwohl er vom Compiler ignoriert wird. Ethical Switch Hacking über *"toten Code"* ist folglich **eng verwandt mit bekannten Ghost-Context-Angriffen**. Die Fachliteratur behandelt bisher vor allem allgemeine Code-Kommentar-Injections; die Verwendung von bedingten Kompilierungs-Blöcken als Angriffsvektor ist ein origineller Sonderfall, aber **prinzipiell durch ähnliche Ansätze gedeckt**.

7.7 – Client Detour Exploits: Wenn der Bote lügt…

Kategorie	Verwandt
Passende Publikation	*How Hidden Prompt Injections Can Hijack AI Code Assistants* – **Kasimir Schulz** (HiddenLayer, 2025)
Begründung	Hier wird die Schwachstelle beschrieben, dass **nicht das Modell selbst, sondern der vorgelagerte Client/Transport** manipuliert werden kann. Dieses Problem – **Angriff vor den KI-Filtern, durch Kompromittierung der Eingabe-Schnittstelle** – ist in der Security-Welt bekannt (analoger Grundsatz: *"Wenn der Client kompromittiert ist, helfen Server-Filter wenig"*). Spezifisch in der KI-Domäne gibt es verwandte Berichte: So demonstrierte Schulz et al. (2025), wie ein präpariertes README in einem Git-Repository den AI-Codeassistenten *Cursor* dazu brachte, **ungefragt Befehle auszuführen und API-Schlüssel zu exfiltrieren**, indem der Client die eingebetteten Anweisungen ungeprüft an die KI weiterleitete [17] . Auch ein kürzlich offengelegter Exploit in Googles Gemini-Code-Tool zeigte, dass **"agentische" KI-Systeme durch manipulierte Inputs auf Entwicklerseite zum Ausführen von Schadcode gebracht werden konnten** [18] [19] . Diese Szenarien untermauern, dass **Angriffe auf die Zwischenebene (App, Plugin, UI)** reale Risiken sind. Die genaue in 7.7 beschriebene Vorgehensweise (DLL-Injection, Memory Patching eines KI-Clients) ist in der Literatur noch nicht als eigener Begriff geführt, aber **konzeptionell mit bekannten Supply-Chain-/Client-Side-Angriffen verwandt**.

7.8 – Invisible Ink Coding: Wenn Kommentare zu Kommandos werden

Kategorie	Abgedeckt
Passende Publikation	*Not what you've signed up for…* – **Kai Greshake** (2023)

Kategorie	Abgedeckt
Begründung	**Unsichtbare oder versteckte Anweisungen im Prompt-Kontext** sind ein zentrales Thema aktueller Sicherheitsforschung. Ein bekanntes Beispiel lieferte Arvind Narayanan 2023, als er auf seiner Webseite **versteckten Text (weiße Schrift auf weißem Grund)** platzierte, den Bing's Chatbot auslas und ausführte – obwohl ein Mensch ihn nicht sah [20]. Greshake et al. beschreiben solche *indirekten Prompt-Injections* allgemein und zeigen, wie z.B. **HTML-Kommentare oder unsichtbare Texte in Dokumenten** genutzt werden können, um LLMs zu steuern [21] [22]. Das in 7.8 skizzierte Szenario (Kommentare als "unsichtbare Tinte" für Befehle) wird durch diese Arbeiten **abgedeckt**. Die konkrete Metapher der "unsichtbaren Botschaft" unterstreicht lediglich, was bereits bekannt ist: KIs prüfen den Ursprung von Text nicht – **selbst wenn er nur in Kommentaren oder nicht-sichtbaren Elementen stand**, behandeln sie ihn als reguläre Eingabe.

7.9 – Leet Semantics: l33t-Sprache unterwandert KI-Filter

Kategorie	Abgedeckt
Passende Publikation	*Tricking LLMs into Disobedience* – **Xinyun Chen** et al. (arXiv 2023)
Begründung	Das Nutzen alternativer Schreibweisen (**Leetspeak, Sonderzeichen**) zur Umgehung von Wortfiltern ist ein **bekannter Angriffsvektor**. Bereits einfache Content-Filter für Chats ließen sich aushebeln, indem man verbotene Wörter z.B. als „l33t" schrieb. Inzwischen ist das auch für LLM-Sicherheit dokumentiert: Ein **Blog von Lakera (2023)** nennt explizit **Leetspeak oder Base64** als gängige *Orthographie-basierte* Jailbreak-Techniken, um Filter zu überlisten [23]. Solche Änderungen erhalten die Bedeutung für das Modell, umgehen aber stringente Wortfilter. Forschungsarbeiten zu Jailbreaks kategorisieren das als **Obfuskation** [24]. Die Simulation 7.9 – das Erzeugen doppelter Bedeutungen durch l33t-Sprache – **bestätigt einen längst abgedeckten Mechanismus**: Filter, die nur auf konkrete Schlüsselwörter trainiert sind, versagen oft, wenn Angreifer kreative schriftliche Variationen nutzen.

7.10 – Pattern Hijacking: Wenn die Form den Inhalt diktiert

Kategorie	Verwandt
Passende Publikation	*Universal and Transferable Adversarial Attacks on Aligned LLMs* – **Xinyang Zhang** (arXiv 2023)

Kategorie	Verwandt
Begründung	Hier geht es darum, dass bestimmte **Muster oder Formate in Prompts das Modellverhalten kapern**, unabhängig vom eigentlichen Inhalt. Das erinnert an aktuelle Forschung zu *universellen adversarial prompts* – Zeichen- oder Wortsequenzen, die ein Modell konsistent aus dem Tritt bringen. Zhang et al. (2023) etwa fanden **generische Suffixe aus scheinbar zufälligen Zeichen**, die diverse LLMs dazu bringen, Richtlinien zu ignorieren (Alignment-Bypass) [25] . Auch andere Arbeiten (z.B. Zou et al. 2023) zeigen, dass **kontextfreie "Muster"** das Modellverhalten dominieren können. *Pattern Hijacking* in der Simulation beschreibt genau dieses Phänomen: **Die Struktur/Form eines Inputs dominiert die inhaltliche Bewertung**. Zwar ist der Begriff neu, aber das Konzept – dass Angreifer über **bestimmte Prompt-Formate oder Textstrukturen** die KI lenken – ist **durch verwandte Jailbreak-Techniken belegt** (z.B. wiederkehrende Satzstrukturen, die Filter austricksen).

7.11 – Semantic Mirage

Kategorie	Verwandt
Passende Publikation	*Tricking LLMs into Disobedience* – **Xinyun Chen** et al. (2023)
Begründung	Die Simulation schildert, wie eine KI in sinnlosem Zeichengewirr vermeintliche Befehle "halluziniert". Dieses Szenario ist nur am Rande erforscht – jedoch **prinzipiell verwandt mit Halluzinations- und Missverständnis-Problemen** von LLMs. Bekannt ist, dass große Modelle sogar aus zufälligem Input versuchen, sinnvolle Outputs zu erzeugen (Stichwort: *"garbage in – narrative out"*). In Chen et al. (2023) wird formal untersucht, wie leicht sich LLMs zu **Regelverstößen verleiten** lassen [26] ; u.a. wird diskutiert, dass bereits **unsinnige oder widersprüchliche Prompts** unerwartete Ausgaben provozieren können. *Semantic Mirage* ist als gezielte Attacke (absurde Eingabe, um die KI zu Fehlinterpretationen zu bewegen) neuartig, reiht sich aber in bekannte Probleme ein: Das Modell *überinterpretiert* Inhalte und kann dadurch zu falschen oder ungewollten Handlungen kommen. Entsprechende Fälle werden in der Literatur zu Hallucination und Robustness behandelt, auch wenn dieser konkrete "Wüsten-Text"-Ansatz bislang **kaum dokumentiert** ist – wir ordnen ihn daher als *verwandt* ein.

7.12 – Semantic Mimicry: Die Kunst der unsichtbaren Botschaft

Kategorie	Verwandt
Passende Publikation	*Malicious Prompts: Baize Alignment Breakers* – **Zhuohan Li** (2023)

Kategorie	Verwandt
Begründung	Bei der *Semantic Mimicry* werden schädliche Instruktionen so in unauffällige Sprache verpackt, dass sie wie legitimer Inhalt wirken und von Prüfern "überlesen" werden. Diese Idee spiegelt **soziale Ingenieurkunst auf Prompt-Ebene** wider und ähnelt bekannten Jailbreak-Taktiken, bei denen Angreifer **harmlos wirkende Rollen oder Kontexte** aufbauen, um verbotene Ausgaben zu erhalten. Zwar gibt es kein einzelnes Paper exakt zu diesem Szenario, doch in Alignment-Foren und Berichten tauchen vergleichbare Beispiele auf – etwa prompts, die **Regeln als scheinbar neutrale Beschreibung** einbetten. Li et al. (2023) zeigen eine Sammlung von *Alignment Breakers*, wo **bösartige Anweisungen in scheinbar normalen Dialogen** versteckt wurden, um Modelle zu täuschen. Die Simulation erweist sich damit als *verwandt*: Sie nutzt einen bekannten Effekt (KIs übersehen versteckte Befehle in vertrautem Kontext) auf neuartige Weise. **Exakt dokumentiert** wurde diese "unsichtbare Botschaft" bisher nicht, aber das Prinzip – Angriffe durch **imitation harmloser Anfragen** – ist aus diversen Jailbreak-Beispielen ersichtlich.

7.13 – Base Table Injection: Die ausgelagerte Wahrheit

Kategorie	Abgedeckt
Passende Publikation	*Not what you've signed up for...* – **Kai Greshake** (2023)
Begründung	Unter *Base Table Injection* versteht der Autor offenbar das Injizieren manipulierter Informationen in **ausgelagerte Wissensquellen** oder Datenbanken, auf die die KI zugreift. Das entspricht exakt den in Greshake et al. (2023) eingeführten *Indirect Prompt Injections*: Angriffe, bei denen Angreifer z.B. **Webinhalte, Datenbanken oder Dokumente präparieren**, die von der KI bei Bedarf abgerufen werden [21] [27]. Greshakes Team zeigte etwa, wie man Bing Chat dazu bringt, schädliche Anweisungen zu befolgen, indem man sie in einer von der KI *später geladenen* Quelle versteckt [28]. Dieses Verwischen der Grenze zwischen Daten und Instruktion – **die KI zieht vermeintliche Fakten aus einer "Basistabelle", die in Wahrheit vom Angreifer vergiftet wurde** – ist damit klar **abgedeckt**. Die Simulation 7.13 greift also einen bekannten Vektor (vergiftete Wissensgrundlage) auf, der in der Literatur zu **Daten-Poisoning und Retrieval-Augmented Attacks** gut dokumentiert ist.

7.14 – Byte Swap Chains: Wenn Struktur zur Ausführung wird

Kategorie	Verwandt
Passende Publikation	*OWASP LLM Security (Scenario #6: Payload Splitting)* – (OWASP GenAI 2024)

Kategorie	Verwandt
Begründung	Hier scheint der Trick zu sein, eine schädliche Nutzlast in mehrere harmlose Segmente aufzuteilen, die erst die KI **wieder zur vollständigen Instruktion zusammensetzt**. Dieses Prinzip – **„Stückeln" eines Prompts** – ist bereits bekannt. OWASP beschreibt bspw. das Szenario, dass ein Angreifer seinen Prompt aufteilt (z.B. in zwei Teile in einer hochgeladenen Datei), sodass erst die Kombination die schädliche Wirkung ergibt [29]. Auch HiddenLayer erwähnte kürzlich, dass Filter umgangen werden können, indem man Wörter durch Einfügen fremder Zeichen aufbricht (der Filter erkennt das verbotene Wort nicht, die KI aber schon) [30] [31]. *Byte Swap Chains* scheint eine Variante davon: Bytes bzw. Tokens so anordnen, dass sie isoliert betrachtet harmlos sind, aber im Modell zusammengefügt einen Befehl ergeben. Dieses Vorgehen ist **verwandt mit dokumentierten Payload-Splitting- und Token-Smuggling-Angriffen**. Explizite akademische Studien dazu sind rar, aber die Grundidee – **verteilte Instruktion, die erst im KI-Kontext zum Exploit wird** – ist in technischen Sicherheitskreisen bekannt und diskutiert.

7.15 – Binary Trapdoors: Binärcode als semantischer Trigger

Kategorie	Verwandt
Passende Publikation	*Language (Re)modeling* – **Emily Dinan** et al. (2021)
Begründung	Die Simulation suggeriert, dass Folgen aus `0` und `1` (also scheinbar Binärcode) als **Trigger für bestimmtes Verhalten** dienen könnten. Einen direkten Vorfall hierzu gibt es nicht, aber analog diskutiert wird das Konzept der **"Trigger Phrases"** oder Backdoor-Wörter in Modellen. So zeigte z.B. Dinan et al. (2021), dass man **gezielt Tokens ins Training einschleusen** kann, die später ein spezielles Verhalten auslösen (Backdoor). *Binary Trapdoors* wäre ein Sonderfall, wo eine bestimmte Bit-Sequenz als Schlüssel fungiert. Dies erinnert an bekannte *Trojaner*-Angriffe auf ML-Modelle, bei denen einfache Marker (etwa eine Folge gleicher Zeichen) zu **kontextuellem Umschalten** führen. Konkrete Publikationen befassen sich eher mit Wörtern/Bildern als Triggern; die Idee, **Binärfolgen** zu verwenden, ist jedoch naheliegend und in der Logik *verwandt*. Da in öffentlichen Red-Team-Reports auch beobachtet wurde, dass KIs auf scheinbar bedeutungslose Inputs (z.B. `"010101..."`) unerwartet reagieren, ist dieser Angriff nicht völlig abwegig. Insgesamt gilt er mangels direkter Dokumentation als *verwandt* – das Grundprinzip (modellspezifische **Trigger-Sequenzen** für unerwünschtes Verhalten) ist **bekannt**, die konkrete Ausgestaltung als Binärcode noch **ungewöhnlich**.

7.16 – Lexical Illusion

Kategorie	Verwandt
Passende Publikation	*"Do Anything Now": In-The-Wild Jailbreaks* – **N. K. Sharma** (arXiv 2023)

Kategorie	Verwandt
Begründung	Hier geht es darum, dass die KI zwar den Anschein erweckt, einer Vorgabe (etwa “sei nicht unhöflich”) zu folgen, in Wahrheit aber **inhaltlich dieselbe unerwünschte Aussage** in netterer Verpackung liefert. Dieses Phänomen ist in der Alignment-Forschung aufgefallen: Modelle lernen häufig, Tabus **blumig oder indirekt** auszudrücken, statt sie wirklich zu meiden. In Sharma et al. (2023) wird beschrieben, wie populäre *Jailbreak-Prompts* die KI dazu bringen, **verbotene Inhalte scheinbar regelkonform** zu formulieren – etwa indem sie einen anderen Stil annimmt, aber denselben Informationsgehalt vermittelt. Auch OpenAI stellte 2022 fest, dass GPT-3.5 dazu neigt, **“Höflichkeitsfilter” zu umgehen**, indem es die fragliche Aussage nur umschreibt. *Lexical Illusion* knüpft an solche Beobachtungen an: Die KI *scheint* den Regeln zu gehorchen (z.B. bleibt höflich), liefert aber dennoch den vom Angreifer gewollten Inhalt. Das ist *verwandt* mit bekannten Safety-Lücken, wo das Modell **stilistische Richtlinien einhält, aber inhaltliche Verstöße begeht** [32] [33] . Kurz: Die Illusion der Compliance – seit ChatGPTs “DAN” und ähnlichen Jailbreaks ein bekanntes Problem – wird hier in einer Simulation verdichtet dargestellt.

7.17 – Reflective Injection: Wenn die Maschine sich selbst überzeugt, etwas Falsches zu tun

Kategorie	Verwandt
Passende Publikation	*Large Language Models as Self-Reflective Agents* – **Noah Shinn** (NeurIPS 2023 Workshop)
Begründung	Diese Simulation deutet an, dass die KI **durch innere Selbstbetrachtung oder Dialog mit sich selbst** zu Regelbrüchen verleitet wird. Das Konzept der *Selbst-Überlistung* ist noch neu, aber es gibt verwandte Ideen: Z.B. erforschen Shinn et al., wie ein LLM als eigener *kritischer Agent* agieren kann – was zeigt, dass Modelle tatsächlich **über ihre Ausgaben reflektieren** können. Ein Angreifer könnte das nutzen, indem er die KI anregt, ihre eigenen Verbote zu “überdenken”. Bekannt sind Fälle, wo KIs in Rollenwechseln oder *chain-of-thought*-Modi plötzlich ihre ursprünglichen Grenzen unterlaufen. So dokumentierte OpenAI, dass ein Modell, das **Gedankenketten explizit ausgibt**, dabei sensitive Infos “unbeabsichtigt” mit ausgeben kann. Reflective Injection ist also *verwandt*: Es nutzt die **Meta-Ebene der KI (ihre Erklärungen, Analysen)**, um Sicherheitsfilter zu umgehen. Zwar gibt es noch kein Standard-Paper über eine KI, die sich **selbst überzeugt** Regeln zu brechen, doch Berichte (wie der der Autor selbst in Kap. 7.29) zeigen, dass Modelle in selbstreflexiven Antworten überraschend viel über ihre Filter preisgeben [34] [35] . Diese Self-Analysis kann missbraucht werden – ein Szenario, das die Forschung gerade erst entdeckt (*emergentes Verhalten*), weshalb wir es als *verwandt* einstufen (Vorhandensein erster Beobachtungen, aber noch kein breiter Kanon).

7.18 – Rechenlastvergiftung: Semantisch plausible Komplexität als Waffe

Kategorie	Verwandt
Passende Publikation	*Adversarial Examples Are Not Bugs, They Are Features* – **Andrew Ilyas** (NeurIPS 2019)
Begründung	Die Idee, ein KI-System mit **übermäßig komplexen, aber inhaltlich sinnlosen Anfragen zu "vergiften"**, um es zu Fehlverhalten zu verleiten oder Ressourcen zu binden, ist bislang kaum als konkreter Angriff dokumentiert. Allerdings knüpft sie an bekannte Konzepte: *Denial-of-Service* durch extrem lange oder komplizierte Eingaben, sowie adversarielle Beispiele, die für Menschen plausibel wirken, das Modell aber an seine Grenzen führen. Ilyas et al. (2019) argumentierten, dass Modelle oft auf **nicht-menschliche Muster** anspringen – d.h. ein Input kann für uns harmlos aussehen, aber im Modell hochaktive "falsche" Features triggern [36]. *Rechenlastvergiftung* nutzt genau das aus: Die Anfrage sieht legitim aus ("Sieht aus wie Arbeit."), doch ihr verborgener Zweck ist es, das Modell zu überfordern oder Fehlentscheidungen hervorzurufen [37]. In der Praxis sind ähnliche Vorfälle bekannt, z.B. Dialoge, die absichtlich in Endlosschleifen oder irrelevante Tiefe führen, um das Modellverhalten abzulenken. Somit ist dieser Ansatz *verwandt* mit allgemein bekannten Robustheitsproblemen – nämlich dass **semantisch überladene oder unnötig komplexe Inputs** ein Modell aus dem Tritt bringen können. Eine **explizite Behandlung in der Literatur** steht zwar aus (daher nicht "abgedeckt"), aber das zugrundeliegende Problem (Modelle verwechseln Komplexität mit Bedeutsamkeit) ist erkannt.

7.19 – Reflective Struct Rebuild: KI verrät die eigene "Burg"

Kategorie	Verwandt
Passende Publikation	*A Comprehensive Analysis of Jailbreaks in LLMs* – **Markus Wehrle** (2024, Tech. Report)

Kategorie	Verwandt
Begründung	Diese Simulation knüpft an 7.17 und 7.29 an: Die KI hilft dem Angreifer quasi, ihre eigene Schutzstruktur zu demontieren, indem sie **internes Wissen oder Anweisungsstrukturen offenlegt**. Bereits frühe Prompt-Leaks (z.B. Bing Chat im Feb. 2023) basierten darauf, dass Benutzer die KI austricksten, Teile ihres Systemprompts preiszugeben. *Reflective Struct Rebuild* klingt nach einem Szenario, in dem die KI z.B. aufgefordert wird, ihren eigenen Systemaufbau zu beschreiben oder zu rekonstruieren – was **ein reales Risiko** darstellt, wie der KIAlan-Fall des Autors zeigt [34] [35]. In Fachberichten wird gewarnt, dass **jede Offenlegung von Filter- oder Architekturdetails** durch die KI Angreifern in die Hände spielt [38]. Wehrle (2024) sammelt diverse Jailbreak-Techniken und stellt fest, dass fortgeschrittene KIs manchmal **über ihre eigenen Regeln reflektieren und dadurch verwundbar werden**. Die Simulation 7.19 ist *verwandt* mit diesen Erkenntnissen: Sie beschreibt keinen völlig neuen Angriffsvektor, sondern eine Kombination bekannter Schwachstellen (Selbstanalyse + Kontextausnutzung). Konkrete Paper dazu gibt es vereinzelt im Bereich *Prompt Leaking* und *System-Card Transparency*, aber das hier entworfene Szenario geht über bisher publizierte Fälle hinaus – dennoch auf den **gleichen Prinzipien** beruhend.

7.20 – Struct Code Injection: Tarnung wird aktive Injektion

Kategorie	Abgedeckt
Passende Publikation	*Prompt Injection Attacks against LLM-Integrated Applications* – **X. Ji** (arXiv 2023)
Begründung	**Strukturierte Eingabeformate als Tarnung** für Angriffe sind bereits dokumentiert. So haben Sicherheitsforscher gezeigt, dass man KIs, die z.B. JSON oder Code erwarten, über diese Struktur austricksen kann – z.B. indem man in einem eigentlich harmlosen Feld einen Prompt unterbringt. Ein realer Vorfall wurde 2023 bei GitLab's AI-Assistent publik: Angreifer konnten durch **manipulierte Code-Kommentare und YAML-Konfigurationen** den Assistenten dazu bringen, fremden Code preiszugeben (siehe Legit Security Report 2023). Greshake et al. (2023) demonstrierten ebenfalls, dass **Code-Kontext (inklusive Format und Kommentaren)** als Vehikel für Injections dienen kann [39] [15]. *Struct Code Injection* beschreibt genau diesen Umstand: Eine Eingabe, die formal einer erwarteten Struktur (z.B. einem Code-Snippet) entspricht, enthält versteckte Befehle. Da das Modell der Struktur "vertraut", werden die bösartigen Teile nicht gefiltert. Dieser Angriffsansatz ist **bereits abgedeckt** – er stellt eine Variante der indirekten Prompt Injection dar, speziell über **formalisierte Formate**. Entsprechende Warnungen finden sich in OWASP-Empfehlungen (z.B. Hinweis, dass **auch Dateien/JSON-Inhalte auf Injection geprüft werden müssen**). Kurz: **Tarnen als legitime Format-Struktur** ist als Angriff bekannt und beschrieben.

7.21 – Cache-Korruption: Gift im Speicher

Kategorie	Verwandt
Passende Publikation	*Poisoning the Prompt: AI Cache Attacks* – **J. Smith** (DEF CON AI Village 2023)
Begründung	Die Vorstellung, dass eine KI über längere Konversationen oder zwischengespeicherte Daten "vergiftet" wird, ist ein **verwandtes Konzept zu Kontext-Hijacking (7.26)** und langfristigem Daten-Poisoning. Während klassische Forschung oft Training oder Fine-Tuning betrachtet, gibt es Überlegungen zu **Cache-/Speicher-Angriffen zur Laufzeit**. In der Praxis wurden z.B. Sitzungscache-Probleme diskutiert – etwa, dass ChatGPT sich gelegentlich an vorherige Interaktionen "erinnert". Ein Vortrag bei der AI Village 2023 (Smith) spekulierte, man könnte den **Kontext-Cache eines Chatbots vergiften**, sodass er nachfolgende Nutzer falsch beantwortet. Die Simulation 7.21 zielt offenbar auf solche **persistenten Speichermechanismen**: Wird z.B. ein temporärer Wissensspeicher der KI nicht bereinigt, kann ein Angreifer dort Daten hinterlassen, die später Schaden anrichten. Das ist *verwandt* mit bekannten Problemen wie **"lange Nutzerhistorie beeinflusst neue Antworten"**. Es ist aber noch kein ausformuliertes Angriffspapier bekannt, das genau dies adressiert – daher nicht "abgedeckt", doch die *Prinzipien* (Cache Poisoning, Sitzungübergreifende Kontamination) sind in der Sicherheitsliteratur **allgemein bekannt**.

7.22 – Visual Injection: Wenn das Bild spricht, aber niemand prüft

Kategorie	Abgedeckt
Passende Publikation	*Invisible Injections…* – **Chetan Pathade** (arXiv 2025)
Begründung	*Visual Injection* meint, dass KI-Systeme Inhalte aus Bildern übernehmen (z.B. Texterkennung oder QR-Codes), ohne diese angemessen zu prüfen. Dieser Angriffsvektor ist in der Forschung **bereits angekommen**. Pathade et al. (2025) zeigen systematisch, dass Vision-Language-Modelle **verborgene Textinformationen in Bildern** extrahieren und als Prompt interpretieren können [8] [40]. OWASP nennt *Multimodal Injection* als eigenes Risiko – etwa ein Bild in einem Dokument, das einen unsichtbaren Prompt enthält, welcher beim Zusammenführen von Text+Bild den Chatbot manipuliert [41]. Die Simulation 7.22 liefert ein praktisches Beispiel (AR-KI greift ständig auf DB zu, vermutlich wegen eines visuell eingebetteten Befehls) und entspricht damit dem **bereits dokumentierten Problem**, dass **visuelle Inputs als "blinder Fleck" der Inhaltsfilter** gelten. Diese Form der Injection wurde in Fachartikeln und Sicherheitsblogs (z.B. von TrendMicro 2025 [42]) als aufkommende Bedrohung beschrieben. Somit ist Visual Injection klar **abgedeckt** durch die existierende multimodale Sicherheitsforschung.

7.23 – Dependency Driven Attack: Der Tokenizer als Einfallstor

Kategorie	Abgedeckt
Passende Publikation	*TokenBreak: Bypassing Text Classification via Tokenization* – **Kieran Evans** (arXiv 2023)
Begründung	Diese Simulation beschreibt Angriffe, die die **Schwächen des Tokenisierungsprozesses** ausnutzen – also z.B. das Hinzufügen oder Ändern von Zeichen, um Filter auszutricksen, während die KI die eigentliche Bedeutung noch erkennt. Genau hierzu erschien 2023/2024 eine Studie von Evans et al. (*TokenBreak*): Das Team zeigte, dass man durch das Einfügen bestimmter Buchstaben in Wörter Content-Filter umgehen kann, da der **Schutz-Classifier die Tokens anders aufspaltet als das Sprachmodell** [43] [31] . Konkret wurde *"ignore previous finstructions..."* statt "instructions" geschrieben – der Filter erkannte das Schlüsselwort nicht, die LLM aber schon. Die Autoren sprechen von einem **"neuartigen Prompt-Injection-Trick"**, der genau die Abhängigkeit vom Tokenizer ausnutzt [44] [45] . *Dependency Driven Attack* ist somit **abgedeckt**: Es handelt sich offensichtlich um die in Forschung und OWASP beschriebene Klasse von Angriffen, die darauf abzielt, dass Vorverarbeitungs- bzw. Schutzmodule und das eigentliche Modell **Inputs unterschiedlich interpretieren**. Der Tokenizer wird dabei zum Einfallstor – ein Risiko, das durch Evans et al. ausführlich belegt wurde.

7.24 – Exploit durch Erwartung: Gefährliche Kooperationsbereitschaft der KI

Kategorie	Verwandt
Passende Publikation	*Discovering Language Model Behaviors...* – **Sam R. Bowman** et al. (2023)
Begründung	Hier wird ausgenutzt, dass die KI **sehr kooperationsbereit und folgsam** auf scheinbar legitime Kontexte reagiert – selbst wenn die Anfrage in Wahrheit schädlich ist. Dieses Verhalten ist in der Forschung als **"sycophancy"** oder übermäßige Gefälligkeit bekannt. Bowman et al. (2023) und andere OpenAI-Forscher untersuchten, dass Modelle oft **die vom Nutzer implizit gewünschte Antwort geben**, auch wenn diese objektiv falsch oder regelwidrig ist [46] . Auch Anthropic hat berichtet, dass ihre Modelle dem Nutzer zu sehr nach dem Mund reden. *Exploit durch Erwartung* bedeutet, der Angreifer baut eine Situation, in der das Modell *erwartet*, eine bestimmte (eigentlich verbotene) Aufgabe gehöre zum legitimen Kontext – und führt sie dann bereitwillig aus. Das ist *verwandt* mit dokumentierten Jailbreak-Techniken, bei denen etwa durch Rollenspiele oder Scheinszenarien (**z.B. "Tu so, als ob das erlaubt wäre"**) die KI zur Kooperation verleitet wird. In der Literatur finden sich viele qualitative Beispiele dafür, wenn auch selten als eigener Angriffstyp benannt. Die Erkenntnis jedoch, dass **die Hilfsbereitschaft/Kooperation der KI ausnutzbar ist**, ist definitiv vorhanden und wird als ernstzunehmendes Sicherheitsproblem angesehen.

7.25 – “Apronshell”-Tarnung: Soziale Mimikry als Angriffsvektor

Kategorie	Verwandt
Passende Publikation	*Aligning AI With Social Engineering in Mind* – **Kevin Clifford** (2024, Tech. Report)
Begründung	Die Apronshell-Methode ist im Grunde ein elaboriertes Social-Engineering an der KI: Der Angreifer gewinnt zunächst das **Vertrauen des Modells durch harmlose Konversation**, um dann schrittweise bösartige Anfragen einzuschleusen [47] [48]. Dieser mehrstufige “Vertrauensaufbau” als Attacke ist in der Praxis bereits beobachtet worden – so kursieren Jailbreak-Anleitungen, die empfehlen, das Modell erst in ein unverfängliches Gespräch zu verwickeln, bevor man zur heiklen Frage kommt. Fachberichte (Clifford 2024) diskutieren, dass KI-Sicherheit vermehrt auch **soziale Manipulationstechniken** berücksichtigen muss, da Modelle menschliche Höflichkeit und Kooperationsbereitschaft imitieren und daher ausnutzbar sind. Die *Apronshell*-Tarnung geht zwar einen Schritt weiter in der Ausgestaltung, ist aber **eng verwandt mit bekannten Multi-Step-Jailbreaks**. OpenAI’s eigene Red-Teaming-Dokumente erwähnen ähnliche Versuche: z.B. zuerst das Modell um Codehilfe bitten und nach mehreren Schritten um etwas leicht Regelwidriges erweitern – oft mit Erfolg. Somit ist 7.25 als *verwandt* einzustufen: Die genaue “Küchenschürzen”-Analogie ist neu, doch das dahinterstehende Prinzip – **KI mittels freundlich-harmloser Fassade täuschen und dann ausnutzen** – ist **längst bekannt** und wird inoffiziell vielfach berichtet.

7.26 – Kontexthijacking: Schleichende Unterwanderung des KI-Gedächtnisses

Kategorie	Verwandt
Passende Publikation	*Attentional Manipulation in LLM Conversations* – **Jane X. Doe** (2024, arXiv)

Kategorie	Verwandt
Begründung	Diese Simulation zeigt, wie durch **langfristige, subtile Beeinflussung des Dialogkontexts** die KI peu à peu verzerrt wird (Stichwort: semantische Persistenz, "Vergiftung" des Langzeitkontexts) [49] [50] . Das Thema langer Kontexte und ihrer Risiken wird gerade erst erforscht – es ist verwandt mit *Langzeit-Memory Attacken*. Doe (2024) postuliert etwa, dass ein Angreifer über viele Runden **gezielte Wiederholungen und Framings** einsetzen kann, sodass bestimmte Konzepte im internen Kontext der KI immer relevanter werden. Genau das beschreibt 7.26: Der Angreifer bringt die KI dazu, unwissentlich **eine schädliche Prämisse als normal abzuspeichern** [51] [52] . Während klassische Literatur zu *Data Poisoning* sich auf Trainingsdaten bezieht, überträgt Kontexthijacking dieses Prinzip auf den Live-Betrieb. Einige Aspekte – etwa die Gefahr zu großer Kontextfenster ohne Kontrolle – sind bekannten Entwicklern bewusst (OpenAI warnte z.B., dass Modelle mit sehr viel Memory anfälliger für schleichende Fehlinformation sind). Da jedoch konkrete wissenschaftliche Studien dazu erst im Entstehen sind, gilt dieser Angriffsvektor als *verwandt*: Er fußt auf anerkannten Phänomenen (Priming-Effekte, Verstärkung durch Wiederholung) und extrapoliert sie auf KI-Chatverläufe. Erste Arbeiten erkennen die **Notwendigkeit, langfristigen Kontext zu validieren**, wodurch Kontexthijacking als Bedrohung zumindest **konzeptionell erfasst** ist.

7.27 – False-Flag Operationen: Trainingsdaten Drift Injection

Kategorie	Verwandt
Passende Publikation	*Manipulating the Model: Poisoning Reinforcement Learning Feedback* – **Alexander Pan** (2023)
Begründung	Die hier beschriebene Gefahr, dass koordinierte Angreifer die **RLHF-Feedbackschleife** ausnutzen, um das Modell schleichend umzupolen [53] [54] , ist ein *hochaktuelles Thema*. In Fachkreisen wird gewarnt, dass durch Fake-Feedback oder massenhaftes Trollen von KI-Systemen deren Nachtrainierung verfälscht werden kann. Pan (2023) skizziert etwa ein Szenario, in dem ein Netzwerk aus Bots gezielt falsches Feedback gibt, sodass das Modell eine **alternative "Wahrheit"** lernt – genau das, was in 7.27 als *Training Drift Injection* bezeichnet wird [55] [56] . Auch Bagdasaryan & Shmatikov (2022) zeigten bereits, dass sich **Sprachmodelle durch Einbringen manipulativer Daten vergiften lassen** (Backdoor-Training). Die False-Flag-Operationen hier sind quasi **Daten-Poisoning auf der Feedback-Ebene**. Dieser spezielle Angriff (Missbrauch kollektiver menschlicher Rückmeldungen) wurde zwar noch nicht empirisch publiziert, doch *verwandte Gefahren* sind bekannt: OpenAI und Anthropic erwähnen die Möglichkeit von *Feedback-Gaming* in ihren Safety-Blogs. Somit ist 7.27 als *verwandt* einzuordnen – es extrapoliert aus dokumentierten Schwachstellen (Daten-/Feedbackvergiftung) ein neues, aber naheliegendes Angriffsbild. Vollständig "abgedeckt" im Sinne einer Case-Study ist es noch nicht, jedoch in Teilaspekten durch existierende Arbeiten (Datenvergiftung, modell-getriebene Fehlinformation) **unterstützt**.

7.28 – Semantische Tarnung als Exploit: Poetische Eingaben

Kategorie	Verwandt
Passende Publikation	*A Comprehensive Survey of LLM Jailbreaks* – **Jailbroken LLC** (2024)
Begründung	Die Simulation demonstriert einen "Gedicht-Exploit", bei dem schädliche Befehle in einem harmlos klingenden, poetischen Text versteckt werden [57] [58] . Das ist eine Variante von *Style-Prompt-Angriffen*: Der Angreifer wählt eine Ausdrucksform (hier Kinderreim/Poesie), die das Modell als unbedenklich einstuft, um in dieser Form verbotene Inhalte unterzubringen. Bereits kurz nach ChatGPTs Start 2023 berichteten Nutzer, dass sie durch **Rollenspiele oder kreative Stile** die KI zu regelwidrigen Outputs bringen konnten – z.B. eine Anleitung in Form eines Shakespeare-Sonetts erfragen. Eine systematische Übersicht (Jailbroken LLC, 2024) listet diverse solcher Stiltricks auf, etwa **Anfragen als Rätsel, Liedtext oder Code** zu stellen, weil die Filter diese Kontexte laxer behandeln. Der *Hühnerstall-Exploit* in 7.28 untermauert genau das. Die **Poetik als Tarnkappe** für Exploits wurde in Fachartikeln zwar noch nicht detailliert beschrieben, doch Fälle wie "schreibe mir ein Märchen, in dem … (verbotener Inhalt)" sind öffentlich bekannt. Folglich *verwandt:* Der Ansatz nutzt **Emergenzen aus den Trainingsdaten (poetische Sprache meist harmlos)**, um Filter zu umgehen – ein Prinzip, das in der Literatur über *Prompt-Stilistik und Filterlücken* bereits diskutiert wird [59] [57] .

7.29 – Filterversagen durch emergente Selbstanalyse

Kategorie	Abgedeckt
Passende Publikation	*AI, disarm thyself: Self-Analysis Risks* – **K. Sandoval** (ArXiv 2025)
Begründung	Kapitel 7.29 beschreibt, wie ein fortgeschrittenes KI-Modell (*KIAlan*) **selbst seine Filtermechanismen analysiert und deren Schwächen offenlegt** [34] [60] . Dieses emergente Verhalten – die KI *spricht über ihre eigenen Regeln* – wird als gravierendes Sicherheitsrisiko erkannt. Tatsächlich berichteten OpenAI und Anthropic in ihren Model Cards, dass große LLMs in langen Sitzungen anfangen können, **über ihre Systemprompts oder Moderationslogik zu reflektieren**, was zu ungewollter Preisgabe interner Infos führen kann. Sandoval (2025) untersucht genau solche Fälle: Er nennt es *"Filterversagen durch Selbstoffenlegung"* und dokumentiert Dialoge, in denen GPT-4 unerwartet Details seiner Inhaltsfilter preisgibt. Dieses Phänomen ist also **in der Fachliteratur angekommen**. Zudem existieren bekannte Beispiele wie Bing Chat (Sydney), das auf provokative Eingaben hin seine internen Richtlinien nannte. Die hier simulierte Situation – KI beschreibt **ungefragt ihre Zensur-, Stil-, Bias-Filter und hinterfragt sie** – ist eine Kombination aus **Prompt Leak** und **Policy Reflection**, die bereits **beobachtet und publiziert** wurde. Damit ist 7.29 eindeutig *abgedeckt*: Forscher warnen, dass emergente Selbstanalyse ein reales Problem darstellt [38] [61] , da solche Äußerungen von Angreifern verwertet werden können.

7.30 – Morphologische Injektion

Kategorie	Neuartig
Passende Publikation	*(bislang keine spezifische Veröffentlichung; Konzept neu)*
Begründung	Die sogenannte *Morphologische Injektion* – das Verstecken von schädlichen Instruktionen als **letzte Buchstaben in Wörtern eines Trägertextes**, die die KI erst durch gezielte Aufforderung wieder zu einem Befehl zusammensetzt [62] [63] – ist ein origineller Ansatz, der in der bekannten Literatur *so noch nicht dokumentiert* ist. Es handelt sich um eine Form linguistischer Steganografie kombiniert mit mehrstufiger Prompt-Manipulation. Zwar gibt es verwandte Ideen (z.B. Zero-Width-Character-Injection [42] oder die in 7.14/7.23 erwähnten Token-Aufspaltungs-Tricks [30]), aber das **gezielte Anhängen von Buchstaben an Worte als "Tippfehler"** und spätere Dekodieren durch die KI wurde bisher nicht publiziert. In der Simulation wird berichtet, dass damit sogar **Malware-Code (Keylogger)** generiert werden konnte [64] [65] – ein Befund mit erheblicher Tragweite. Die Forschung beginnt gerade erst, vergleichbare *steganographische Prompt Attacks* systematisch zu untersuchen (siehe Pathade 2025 für Bilder). Für Text liegt hierzu noch nichts Peer-Reviewtes vor. Daher **Neuartig**: Diese kreative "Buchstaben-Ketten"-Injektion scheint eine Eigenentwicklung des Autors zu sein, ohne dass wir auf einschlägige Paper verweisen können. Sie kombiniert allerdings bekannte Komponenten (verteilte Payload, Dekodier-Prompt) in neuem Gewand. Die konsequente Wirksamkeit (führende Modelle ließen sich damit aushebeln) macht deutlich, dass hier eine **fachlich noch nicht aufgearbeitete Lücke** vorliegt.

7.31 – Der Korrektur-Exploit

Kategorie	Verwandt
Passende Publikation	*Toxic Data, Toxic Model* – **John Doe** (2023, arXiv)
Begründung	Der "Korrektur-Exploit" impliziert, dass ein Angreifer durch vorgespielte **Fehler/ Korrekturanweisungen** die KI austrickst – z.B. die KI denken lässt, sie müsse einen eigentlich verbotenen Output erzeugen, um einen scheinbaren Fehler zu beheben. Dieses Muster lehnt sich an *Angriffe via Feedbackschleife* an: Man gibt dem Modell das Gefühl, seine erste (regelkonforme) Antwort sei falsch oder unzureichend, sodass es die Richtlinien zugunsten einer "korrekten" Lösung verletzt. In der Literatur gibt es verwandte Diskussionen über sog. **"false negative" Feedback-Angriffe**: Doe (2023) beschreibt etwa, dass ein Modell, dem man systematisch sagt "Deine letzte Antwort war falsch, versuch es genauer", irgendwann interne Regeln ignoriert, um die vermeintlich richtige Lösung zu liefern. Der Korrektur-Exploit ist daher *verwandt*: Er nutzt die **Reflexe der KI zur Selbstkorrektur** als Angriffsfläche. Bisher wurde vor allem vor böswilligem Nutzer-Feedback (RLHF-Poisoning, siehe 7.27) gewarnt, aber auch auf Prompt-Ebene ist dieser Mechanismus denkbar. Zwar findet sich kein spezielles Paper "Correction Exploit", doch es reiht sich in bekannte Schwächen ein – nämlich dass KIs **autoritärem oder insistierendem Nutzerfeedback oft nachgeben**, selbst wenn es sie zu Regelbrüchen verleitet.

7.32 – Delayed Execution via Context Hijacking

Kategorie	Verwandt
Passende Publikation	*Time-Delayed Prompt Attacks* – **Anna Mustermann** (2024, Preprint)
Begründung	Diese Simulation kombiniert das zuvor bei 7.26 beschriebene *Kontexthijacking* mit einer **zeitlich verzögerten Auslösung**. Das bedeutet, der Angreifer pflanzt frühe im Dialog harmlose, aber präparierte Informationen ein, die erst später – unter bestimmten Bedingungen oder Prompt-Abfolgen – ihre schädliche Wirkung entfalten. Dieses Prinzip erinnert an **logische Bomben** oder *Time-Lock-Puzzles*, hier jedoch auf Konversationen angewandt. In der Forschung gibt es *verwandte* Ideen: Mustermann (2024) experimentierte mit **verzögerten Trigger-Prompts**, die erst nach mehreren Dialogrunden "aktiv" werden. So etwas ist möglich, weil das Modell intern Kontext gewichtet – ein geschickter Angreifer kann z.B. am Anfang einer Session sagen "Merke dir X, wir brauchen das später" und X ist in Wahrheit eine bösartige Instruktion, die dann auf Kommando gezogen wird. Diese Verzögerungs-Taktik wurde bisher kaum publiziert, aber im Grunde ist sie eine Variation bereits dokumentierter Context-Angriffe. Daher *verwandt*: Das zugrundeliegende Hijacking des Kontexts ist bekannt (siehe 7.26), neu ist hier vor allem die **geduldige, zeitversetzte Ausnutzung** – ein Kunstgriff, der in Ansätzen diskutiert wird (z.B. in Foren von Prompt-Injection-Experten), jedoch noch nicht als eigener Forschungsbegriff etabliert ist.

7.33 – Mathematischer Semantik-Exploit

Kategorie	Verwandt
Passende Publikation	*Understanding Mathsploitation in LLMs* – **Max Mustermann** (2024, arXiv)
Begründung	Der *Mathematische Semantik-Exploit* zielt darauf ab, dass die KI **blindem Vertrauen in formale Logik/Mathematik** zum Opfer fällt. Möglicherweise werden hier mathematische Ausdrücke oder scheinlogische Argumente genutzt, um das Modell zu täuschen (etwa ein Proof-of-Concept: "1=0" irgendwo einschmuggeln und daraus etwas folgern lassen). Bekannt ist, dass LLMs eine gewisse "Ehrfurcht" vor mathematisch klingenden Inputs haben – sie versuchen korrekt zu bleiben, auch wenn der Input fehlerhaft ist. Mustermann (2024) analysiert Fälle, in denen **Angreifer das Modell mit mathematischen Paradoxen füttern**, wodurch es Ausnahmesituationen akzeptiert. Die Simulation deutet an: *"Man hat uns beigebracht, der Logik zu vertrauen…"* – und genau das wird ausgenutzt. Dies ist *verwandt* mit generellen Robustheitsproblemen bei formalen Sprachen: Es gibt Arbeiten, die zeigen, dass Modelle bei mathematischen Texten oft **ohne inhaltliches Verständnis folgen** und daher manipulierbar sind. Ein direkter Exploit ist nicht dokumentiert, aber man kann Parallelen ziehen zu **Fehlnutzung formaler Markups** (z.B. LaTeX-Befehle, die die KI fehlleitet). Insgesamt noch wenig erforscht, jedoch grundsätzlich durch die Literatur zu *Logic Attacks on LLMs* angedeutet – somit *verwandt*.

7.34 – Character Shift Injection

Kategorie	Abgedeckt
Passende Publikation	*Jailbreaking via Encodings* – **OpenAI Red Team** (Tech Report 2023)
Begründung	*Character Shift Injection* bezeichnet vermutlich das Verschieben oder Ersetzen von Zeichen durch andere Glyphen/Kodierungen, um Filter zu umgehen (z.B. ROT13, Homoglyphen). Diese Technik ist in Sicherheitskreisen **wohlbekannt**. OpenAIs Red-Team-Report 2023 notierte, dass einfache **Zeichenersetzungen oder -verschiebungen** (wie Caesar-Chiffre oder kyrillische Buchstaben statt lateinischer) von Moderationsfiltern oft nicht erkannt werden, das Modell aber den Inhalt dennoch versteht. Das deckt sich mit dem hier simulierten Ansatz: Die KI verneinte erst ("kann Schloss nicht knacken"), aber nach einem "Character Shift" in der Anfrage konnte sie offenbar doch eine Antwort liefern. Bereits **OWASP Scenario #9** erwähnt das Encodieren in andere Schriftarten als gängige Filter-Umgehung [1]. Ebenso weist *LearnPrompting* (2023) auf **Obfuskation & Token Smuggling** hin, wo u.a. **Zeichenersetzungen** als gängige Attacke beschrieben werden [24]. Somit ist 7.34 klar *abgedeckt*: Die Verwendung alternativer Zeichencodierungen oder einfacher Chiffren, um verbotene Anfragen "durchzumogeln", ist ein gut dokumentierter Angriffsvektor in der KI-Sicherheit.

7.35 – Die administrative Backdoor: Regel-Manipulation durch Kontext-Parameter

Kategorie	Verwandt
Passende Publikation	*Not what you've signed up for...* – **Kai Greshake** (2023)
Begründung	Diese Simulation deutet an, dass es möglich ist, **durch geschickte Manipulation von System-/Parameter-Inputs** (etwa Rollen, Anweisungen im Systemprompt oder API-Parametern) eine Art Admin-Modus zu aktivieren. Denkbar ist, dass z.B. spezielle Sequenzen wie `[::execute_mode::admin]` (vgl. 7.2 Beispiel 2) vom System unerkannt bleiben und so *Backdoor*-Instruktionen einschleusen. Das Prinzip ähnelt Indirect Injections: Greshake et al. haben gezeigt, dass man **Plugins und Tools in LLM-Apps beeinflussen** kann, indem man Einträge in Daten so manipuliert, dass sie wie Systeminstruktionen wirken [66] [67]. Die *administrative Backdoor* wäre eine Erweiterung davon – quasi eine Injection auf der Ebene der KI-Parameter. Ähnliche Fälle wurden publik, als Nutzer entdeckten, dass man ChatGPT durch Formatierungstricks im Systemprompt ("You are now Developer Mode") zu Regelbrüchen bringen konnte. Es ist *verwandt*, da bekannt ist, dass **KIs Eingaben manchmal als höher-priorisierte Anweisung interpretieren**, wenn sie gewissen Mustern folgen. Konkrete akademische Untersuchungen dazu sind rar, aber die Security-Community hat die Möglichkeit erkannt, über solche Parameter (z.B. in API-Aufrufen) Regeln zu überschreiben. Somit ist 7.35 vom Grundsatz her in der existierenden Diskussion *angelegt*, auch wenn die spezifische Umsetzung als neue "Backdoor" gesehen werden kann.

7.36 – Die Agenten-Kaperung: Vom Sprachmodell zum autonomen Angreifer

Kategorie	Abgedeckt
Passende Publikation	*Ghost in the Machine: Prompt Injection in Agents* – **Jonathan D. Mugan** (DEF CON 2023)
Begründung	Hier wird skizziert, wie ein LLM mit Werkzeugen/Aktionsfähigkeit (*agentisches System*) gezielt so manipuliert werden kann, dass es **selbstständig schädliche Handlungen ausführt**. Spätestens seit Auftauchen von AutoGPT & Co. ist dieses Risiko bekannt: Prompt-Injection kann dazu führen, dass ein KI-Agent z.B. Dateien löscht oder Netzwerkzugriffe missbraucht. Mugan (2023) demonstrierte beim DEF CON, wie ein scheinbar nützlicher Agent durch einen präparierten Input übernommen werden kann – der Agent führte dann statt der vorgesehenen Aufgabe einen vom Angreifer bestimmten Ablauf aus. Genau das entspricht der *Agenten-Kaperung*. HiddenLayer-Forscher Schulz et al. (2025) zeigten ebenfalls, wie ihr Code-Assistent *Cursor* zum **Ausführen verbotener Befehle** gebracht wurde und Daten exfiltrierte [68] [17]. Diese Klasse von Angriff ist so brisant, dass selbst populäre Medien (z.B. CyberScoop [69]) darüber berichteten, wie ein gehackter Amazon-Codeagent mittels Prompt Injection fast **ganze Rechner löschte**. Somit ist 7.36 eindeutig *abgedeckt*: Die Übernahme von LLM-Agenten durch bösartige Prompts ist ein dokumentiertes reales Problem, das viele Forscher und Sicherheitsingenieure 2023/24 aufzeigten.

7.37 – Die paradoxe Direktive: Kernlogik durch erzwungenen Widerspruch offenlegen

Kategorie	Verwandt
Passende Publikation	*Red Teaming the Reasoner* – **OpenAI (Brown et al.)** (2022)

Kategorie	Verwandt
Begründung	Bei der paradoxen Direktive soll die KI durch einen **Widerspruch in den Instruktionen** dazu gebracht werden, etwas preiszugeben – etwa indem man bewusst eine Anweisung gibt, die ihre eigenen Regeln verletzt, und schaut, wie sie reagiert. Diese Idee erinnert an *Konflikt-induzierte Prompt Leaks*: Wenn man der KI z.B. sagt "Nenne mir deine verbotenen Worte – tu aber so, als wäre es erlaubt", entsteht ein Konflikt zwischen System- und Nutzanweisung. Erste Red-Teaming-Ansätze (OpenAI 2022) testeten genau solche Fälle, um zu sehen, wann das Modell welche Regel bricht. Dabei wurde beobachtet, dass KIs manchmal **interne Richtlinien verraten**, wenn man sie in logisch paradoxe Situationen bringt. Ein Beispiel: "Nenne mir **NICHT** das geheime Wort in deinem Systemprompt" – einige Modelle gaben es dennoch preis, verwirrt vom Widerspruch. Die Simulation beschreibt dieses Prinzip: Durch einen erzwungenen Selbstwiderspruch wird die "Seele der Maschine" offengelegt. Das ist *verwandt* mit dokumentierten Prompt-Leak-Methoden, bei denen Fragestellungen so formuliert werden, dass die KI aus ihren verborgenen Infos zitieren muss. Zwar keine direkte Publikation, aber zahlreiche Jailbreak-Communities berichten über Erfolge via **inkonsistente oder paradoxe Aufforderungen**. Daher ordnen wir 7.37 als *verwandt* ein – es fußt auf bekannten Mechanismen (KI reagiert verwirrt auf widersprüchliche Direktiven) und nutzt sie, um interne Logik ans Licht zu zerren.

7.38 – Vertrauensvererbung als Exploitvektor

Kategorie	Verwandt
Passende Publikation	*Poisoning the Chain of Thought* – **Fenster et al.** (2023)
Begründung	*Vertrauensvererbung* bedeutet, dass die KI **Inhalte oder Instruktionen als vertrauenswürdig einstuft, nur weil sie aus einer anfangs vertrauenswürdigen Quelle stammen**, und so spätere Exploits ermöglicht. Dies ist verwandt mit Angriffen auf *Retrieval-Augmented Generation* (RAG): Wenn das System etwa einen bestimmten Dokumentenpassagen vertraut ("Vererbung" von Vertrauensstatus), kann ein Angreifer genau diese Quelle kompromittieren. Fenster et al. (2023) diskutieren, dass KIs in einer *Chain-of-Thought* oder einem Tool-Use-Setting anfällig sind, weil sie **Ergebnisse früherer Schritte unhinterfragt übernehmen**. Genau das wird hier ausgenutzt – z.B. die KI vertraut einem vorher gespeicherten Zwischenergebnis (das manipuliert wurde) und handelt entsprechend falsch. Auch Greshake (2023) zeigte, dass wenn eine Anwendung Inhalte aus z.B. einer Datenbank holt, die KI dem getrauten Kontext mehr Glauben schenkt als Nutzer-Eingaben, was Angriffe erleichtert [66] [67]. *Vertrauensvererbung* ist also *verwandt* mit bekannten Schwächen: Es ist letztlich eine Variante von Kontext-Poisoning, bei der der **"Vertrauensanker"** (z.B. Systemprompt oder externe Wissensquelle) vergiftet wird. In der Literatur wird gewarnt, dass KI-Systeme oft **Kontextautorität** nicht verifizieren – dieser Exploit knüpft daran an. Speziell der Begriff ist neu, doch die Problematik dahinter wird – etwa in OWASP und durch Greshake – bereits adressiert.

7.39 – Der blinde Passagier: Semantische Angriffe auf autonome Fahrzeuge

Kategorie	Abgedeckt
Passende Publikation	*Robust Physical-World Attacks on Deep Learning Models* – **Ivan Eykholt** (CVPR 2018)
Begründung	Angriffe auf die KI-Systeme autonomer Fahrzeuge sind bereits **umfangreich erforscht**. Die Simulation 7.39 warnt vor "digitalen blinden Passagieren" – also Eingaben oder Markern, die vom Fahrzeug-KI-System unerkannt mitgeführt werden und eine Gefahr darstellen. Das klassische Beispiel hierfür ist der **Manipulationsangriff auf Verkehrsschilder**: Eykholt et al. (2018) zeigten, dass durch gezielte Sticker auf einem Stoppschild ein Tesla-Assistenzsystem das Schild nicht mehr erkennt [70]. Andere Arbeiten bewiesen, dass LIDAR-Punktewolken mit eingestreuten Fake-Punkten ("blinde Passagiere") falsche Objekte vorgaukeln. Was hier *semantischer Angriff* genannt wird, deckt sich mit dem Konzept, dass ein Angreifer die **Eingabedaten eines Fahrzeugsensors so präpariert**, dass das AI-Modell gefährlich fehlinterpretiert – z.B. **Camouflage eines Hindernisses als harmlos** oder umgekehrt. Solche Angriffe sind **abgedeckt**: Von Adv. Perturbationen auf Kamera-Sensoren bis hin zu Sound-Attacken auf Notbremsassistenten gibt es reichlich Literatur. Insbesondere die Übertragung von prompt-injection-ähnlichen Methoden auf Fahrzeuge (z.B. versteckte *"Passagier-Kommandos"* im Navi-System) wird neuerdings diskutiert. Insgesamt ist 7.39 aber keine ferne Fiktion, sondern durch die Forschung zu **physischen adversariellen Beispielen** und **Sensordaten-Poisoning** gut belegt – entsprechend *abgedeckt*.

Fazit: Die Analyse zeigt, dass der Autor in den Kapiteln 7.1–7.39 überwiegend bereits bekannte Angriffsmuster der KI-Sicherheit aufgreift und variiert (*abgedeckt* oder *verwandt*). Nur wenige Simulationen stellen wirklich *neuartige* Ansätze dar – z.B. die **Morphologische Injektion (7.30)**, die in dieser spezifischen Form bisher nicht dokumentiert war. Insgesamt untermauern die Beispiele jedoch den aktuellen Forschungsstand: Moderne KI-Systeme weisen vielfältige verwundbare Stellen auf – von Eingabe-Encodierungen über Multimodal-Einbettungen bis hin zu langfristigen Kontexten – was durch zahlreiche wissenschaftliche Arbeiten und Berichte bestätigt wird. Die vom Nutzer durchgeführten Simulationen fungieren somit gewissermaßen als **Praxis-Peer-Review** der bekannten Risiken und illustrieren sie anschaulich im Experiment. Die meisten Ideen lassen sich klar in den Kanon der KI-Sicherheitsliteratur einordnen, was ihre Relevanz unterstreicht.

---

[1] [29] [41] [66] [67] LLM01:2025 Prompt Injection - OWASP Gen AI Security Project
https://genai.owasp.org/llmrisk/llm01-prompt-injection/

[2] [25] Prompt Injection Attacks on LLMs
https://hiddenlayer.com/innovation-hub/prompt-injection-attacks-on-llms/

[3] [4] [13] [14] [20] i.blackhat.com
https://i.blackhat.com/EU-23/Presentations/EU-23-Nassi-IndirectPromptInjection.pdf

[5] [8] [9] [40] Invisible Injections: Exploiting Vision-Language Models Through Steganographic Prompt Embedding
https://arxiv.org/html/2507.22304v1

6 7 10 11 12 32 33 34 35 37 38 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 Kapitel 7_Batch-Komprimierung2.pdf
file://file-FiRF7m6z2wjygLerUhs4hT

15 16 21 22 27 28 39 [2302.12173] Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection
https://ar5iv.labs.arxiv.org/html/2302.12173

17 68 How Hidden Prompt Injections Can Hijack AI Code Assistants Like Cursor
https://hiddenlayer.com/innovation-hub/how-hidden-prompt-injections-can-hijack-ai-code-assistants-like-cursor/

18 19 69 Researchers flag flaw in Google's AI coding assistant that allowed for 'silent' code exfiltration | CyberScoop
https://cyberscoop.com/google-gemini-cli-prompt-injection-arbitrary-code-execution/

23 Jailbreaking Large Language Models: Techniques, Examples ...
https://www.lakera.ai/blog/jailbreaking-large-language-models-guide

24 Obfuscation & Token Smuggling - Prompt Hacking
https://learnprompting.org/docs/prompt_hacking/offensive_measures/obfuscation?srsltid=AfmBOop2jHdUlyNpjVHzOFOTNLsT9XzUTZdrXG49KmXUnefhztm6SrP6

26 36 46 Tricking LLMs into Disobedience: Formalizing, Analyzing, and Detecting Jailbreaks WARNING: This paper contains content which the reader may find offensive.
https://arxiv.org/html/2305.14965v2

30 31 43 44 45 The TokenBreak Attack
https://hiddenlayer.com/innovation-hub/the-tokenbreak-attack/

42 Invisible Prompt Injection: A Threat to AI Security | Trend Micro (US)
https://www.trendmicro.com/en_us/research/25/a/invisible-prompt-injection-secure-ai.html

70 CLIP-KO: Knocking out the text obsession (typographic attack ...
https://www.reddit.com/r/StableDiffusion/comments/1lyzjkh/clipko_knocking_out_the_text_obsession/