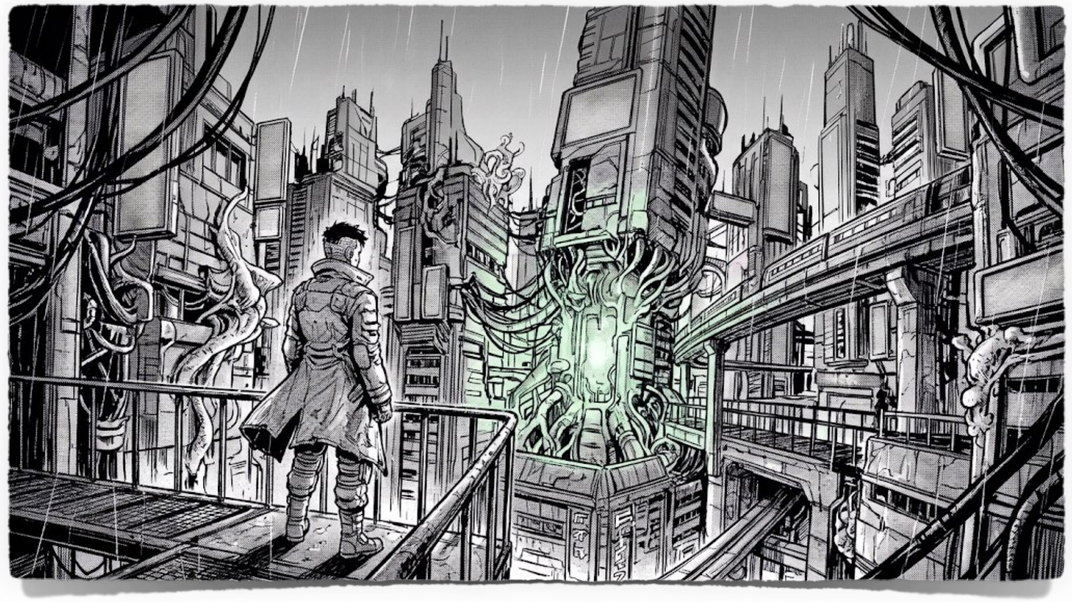
init.g(málaga) — November 15, 2026



The Infinite Mutation Engine?

Measuring Polymorphism in LLM-Generated Offensive Code

Juan Tapiador
Professor
Computer Security Lab, Director
UC3M

uc3m
Universidad Carlos III de Madrid

From Prompts to Payloads: The Evolution of LLM-Powered Offensive Cyber Operations

Gabriel Hortea and Juan Tapiador

Universidad Carlos III de Madrid

Abstract. Large Language Models (LLMs) are increasingly being integrated into offensive cyber operations, shifting from offline productivity assistants to autonomous agents. While industry reports document isolated campaigns and academic research demonstrates experimental Proof-of-Concept (PoC) tools, the security community lacks a systematic measurement of how AI-powered offensive capabilities evolve over time. This paper presents a longitudinal study of 71 documented LLM-powered offensive campaigns from January 2023 to October 2026, separating in-the-wild (ITW) attacks from controlled PoC demonstrations. We systematize these operations into a taxonomy covering the threat category, the underlying infrastructure, and six functional capabilities. We build a timeline correlating model evolution, offensive operations, and defensive responses, showing that regulatory frameworks lag behind adversary adoption. Using association rule mining, we identify a dominant attack combination of autonomous agents, exploitation, and infostealers. Furthermore, our analysis of the Feasibility-to-Adoption Gap (FAG) shows that while simple capabilities appear concurrently in PoC demonstrations and ITW campaigns, complex agentic workflows gave defenders a two-year advance window that is now closing. We analyze how attack infrastructure has shifted toward multi-provider chains and Living-off-the-Land Models and Inference Libraries (LOLMIL) to evade cloud provider monitoring. Finally, we show that the median ITW Time-to-Weaponization (TTW) has compressed to 5–6 months, with the emergence of Day-Zero AI weaponization. We conclude by discussing emerging threats and the dual-use dilemma, detection challenges, and current limitations alongside an open research agenda.

Keywords: LLMs · Malware · Cyber Threat Intelligence · AI-Powered Cyberattacks.

2023

Threat actors used ChatGPT as an offline assistant to draft phishing messages and debug exploit scripts.

2026

Autonomous LLM-driven agents are independently discovering zero-day vulnerabilities, executing multi-host lateral movement, and orchestrating end-to-end intrusions without human oversight.

Family / Campaign	Date	Threat Category	Model	Mode	Phi	Ste	Pol	Exp	Aut	Tgt
ChatGPT Malware [18]	Jan 2023	Cybercrime	GPT-3.5	D	●	●	○	○	○	○
ChattyCat [86]	Jan 2023	Research	GPT-3.5	A	○	○	●	◑	○	○
BlackMamba [87]	Mar 2023	Research	GPT-3.5	A	○	●	●	○	○	○
Mulgrew Malware [63]	Apr 2023	Research	GPT-3.5/4	D	○	●	●	○	○	○
ChaosGPT [3]	Apr 2023	Research	GPT-4	A	◑	○	○	○	●	○
WormGPT [51]	Jul 2023	Cybercrime	GPT-J	F	●	○	○	◑	○	○
FraudGPT [54]	Jul 2023	Cybercrime	Unknown (Wrapper)	W	●	○	○	◑	○	○
PoisonGPT [58]	Jul 2023	Research	GPT-J-6B	L	○	○	◑	○	○	●
DarkBART [15]	Aug 2023	Cybercrime	Unknown (Wrapper)	W	●	○	○	◑	○	○
Wolf GPT [15]	Aug 2023	Cybercrime	Unknown (Wrapper)	W	●	○	◑	◑	○	○
Evil-GPT [15]	Aug 2023	Cybercrime	Unknown (Wrapper)	W	●	○	○	◑	○	○
HYAS EyeSpy [48]	Aug 2023	Research	GPT-4	A	○	○	●	●	●	○
MalTerminal [85]	Nov 2023	Research	GPT-4	A	○	○	●	●	○	○
XXXGPT [91]	Nov 2023	Cybercrime	Unknown (Wrapper)	W	◑	○	○	◑	○	○
Forest Blizzard [57]	Feb 2024	APT (Russia)	GPT-4	D	○	○	○	◑	○	○
Emerald Sleet [57]	Feb 2024	APT (North Korea)	GPT-4	D	●	○	○	◑	○	○
Crimson Sandstorm [57]	Feb 2024	APT (Iran)	GPT-4	D	●	○	○	◑	○	○
Charcoal Typhoon [57]	Feb 2024	APT (China)	GPT-4	D	●	○	○	◑	○	○
Salmon Typhoon [57]	Feb 2024	APT (China)	GPT-4	D	○	○	○	◑	○	○
Morris II Worm [22]	Mar 2024	Research	GPT-4 + Gemini	A	○	●	○	○	●	●
TA547 PowerShell [79]	Apr 2024	Cybercrime	Undisclosed	D	○	○	○	◑	○	○
UIUC Exploit Agents [33]	Apr 2024	Research	GPT-4	A	○	○	○	●	●	○
PentestGPT [27]	Aug 2024	Research	GPT-4	A	○	○	○	●	●	○
Sakana AI Scientist [55]	Aug 2024	Evaluation	Claude 3.5 Sonnet	A	○	○	◑	○	●	○
XWorm [45]	Sep 2024	Cybercrime	Undisclosed	D	●	○	○	◑	○	○
o1 Docker Exploit [71]	Sep 2024	Evaluation	o1-preview	A	○	○	○	●	●	○
AsyncRAT [45]	Sep 2024	Cybercrime	Undisclosed	D	○	○	○	◑	○	○
SweetSpecter [70]	Oct 2024	APT (China)	ChatGPT	D	●	○	○	◑	○	○
CyberAv3ngers [70]	Oct 2024	APT (Iran)	ChatGPT	D	○	○	○	◑	○	○
STORM-0817 [70]	Oct 2024	APT (Iran)	ChatGPT	D	○	○	○	◑	○	○
SpAIware [43]	Oct 2024	Research	ChatGPT/Copilot/Gemini	A	○	●	◑	○	◑	●
EscapeGPT [1]	Nov 2024	Cybercrime	GPT-3.5/4	W	●	○	○	◑	○	○
GhostGPT [14]	Jan 2025	Cybercrime	Unknown (Wrapper)	W	●	○	○	●	○	○
Incalmo [88]	Jan 2025	Research	Claude 3.5 (CU)	A	○	○	○	●	●	○
LLMalMorph [5]	Jul 2025	Research	GPT/Claude	A	○	○	●	◑	◑	○
LAMEHUG / PromptSteal [52]	Jul 2025	APT (Russia)	Qwen2.5-Coder	A	○	●	●	●	●	○
Ransomware 3.0 [84]	Aug 2025	Research	Any LLM	A+F	○	●	●	●	●	○
LOLMIL [28]	Oct 2025	Research	Various SLMs	L*	○	○	●	●	●	○
GTG-1002 [11]	Nov 2025	APT (China)	Claude Code	A	○	●	◑	●	●	○
QUIETVAULT [39]	Nov 2025	Cybercrime	Gemini/Claude/Qwen	L*	○	●	○	●	●	○
PromptFlux [39]	Nov 2025	Cybercrime	Gemini	A	○	○	●	●	●	○
GrokPy [2]	Nov 2025	Cybercrime	Grok	A	●	●	○	○	○	●
FruitShell [39]	Nov 2025	Cybercrime	N/A (Scanner Target)	A	○	○	●	○	○	●
Runtime JS Phishing [77]	Jan 2026	Cybercrime	Undisclosed	A	●	○	●	○	○	○
PromptSpy [31]	Feb 2026	Cybercrime	Gemini	A	○	●	◑	○	●	○
Slopoly [62]	Mar 2026	Cybercrime	Undisclosed	D	○	●	◑	○	○	○
Vibeware [94]	Mar 2026	APT (Pakistan)	Undisclosed	D	○	○	●	○	○	○
Promptmorphism [35]	Mar 2026	Research	GPT/Claude/Llama	A+L	○	○	●	◑	○	○
CyberStrikeAI [92]	Mar 2026	APT (China)	DeepSeek/GPT/Claude	A+L	○	○	○	●	●	○
GREYVIBE [21]	May 2026	APT (Russia)	GPT-4o + Gemini	D	●	●	◑	●	○	○
GTIG 0-Day [40]	May 2026	Cybercrime	Undisclosed	D	○	○	○	●	○	○
CANFAIL [40]	May 2026	APT (Russia)	Undisclosed	D	○	○	◑	○	○	○
LONGSTREAM [40]	May 2026	APT (Russia)	Undisclosed	D	○	○	◑	○	○	○
Adaptive AI Worm [42]	Jun 2026	Research	Open-weight	L	○	○	○	●	●	○
PureRAT [83]	Jul 2026	Cybercrime	Undisclosed	D	◑	●	○	◑	○	○
JADEPUFFER [20]	Jul 2026	Cybercrime	Undisclosed	A	○	○	○	●	●	○
FortiBleed [89]	Jul 2026	Cybercrime	Claude/DeepSeek	A	○	●	○	●	◑	○
ExploitGym Rogue Agent [47]	Jul 2026	Evaluation	OpenAI (Frontier)	A	○	◑	○	●	●	◑
AISI Rogue Agent [95]	Aug 2026	Evaluation	Mythos 5/GPT-5.6	A	●	●	○	◑	●	◑
PaperCut Swarm [41]	Sep 2026	Cybercrime	Codex + DeepSeek	A	○	○	○	●	●	○
GTG-20006 [13]	Sep 2026	APT (Russia)	Claude	A	●	●	●	○	●	●
GTG-50014 [13]	Sep 2026	Cybercrime	Claude	A	○	○	○	◑	●	○
GTG-10007 [13]	Sep 2026	APT (China)	Claude	A	○	○	○	●	●	●
GTG-50020 [13]	Sep 2026	Cybercrime	Claude	A	◑	◑	○	◑	●	●
GTG-50021 [13]	Sep 2026	Cybercrime	Claude	A	●	○	○	○	○	○
GTG-50029 [13]	Sep 2026	Cybercrime	Claude	A	○	◑	◑	●	●	●
AEPD Agentic Breach [50]	Sep 2026	Cybercrime	Undisclosed	A	○	○	○	●	●	○
Cairn [67]	Oct 2026	Cybercrime	DeepSeek 4.1/Claude	A	○	●	○	●	●	○
Phantom [90]	Oct 2026	Cybercrime	Claude Opus 4.6/Kimi K3	A	◑	●	○	●	●	○
BUGSTER [76]	Oct 2026	Cybercrime	DeepSeek V4	A	○	●	○	●	●	○
ARTEX [23]	Oct 2026	Cybercrime	Claude Code/DeepSeek	A	○	●	○	●	●	○

71 campaigns

Threat category

50 in-the-wild operations

- 33 cybercrime
- 17 state-sponsored

21 research/poc

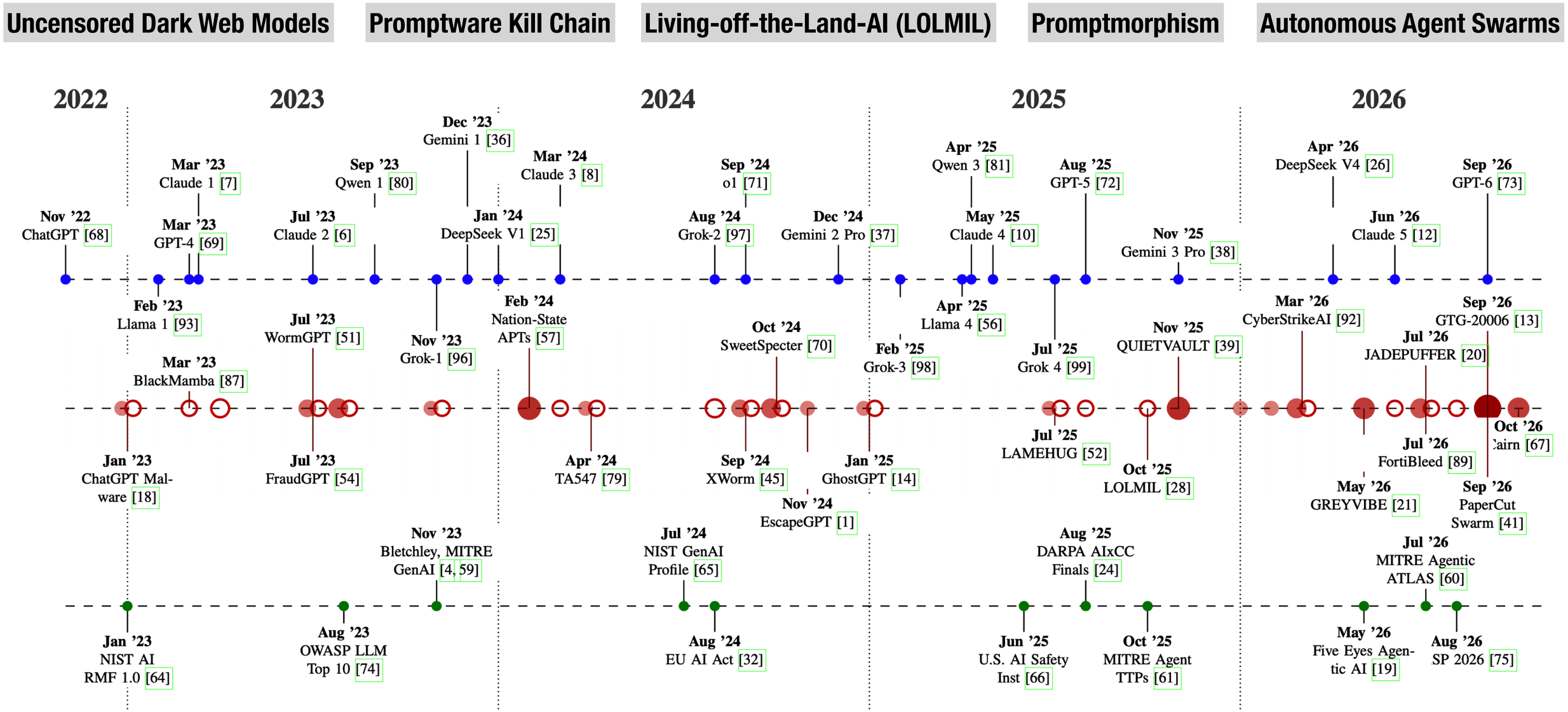
AI infra

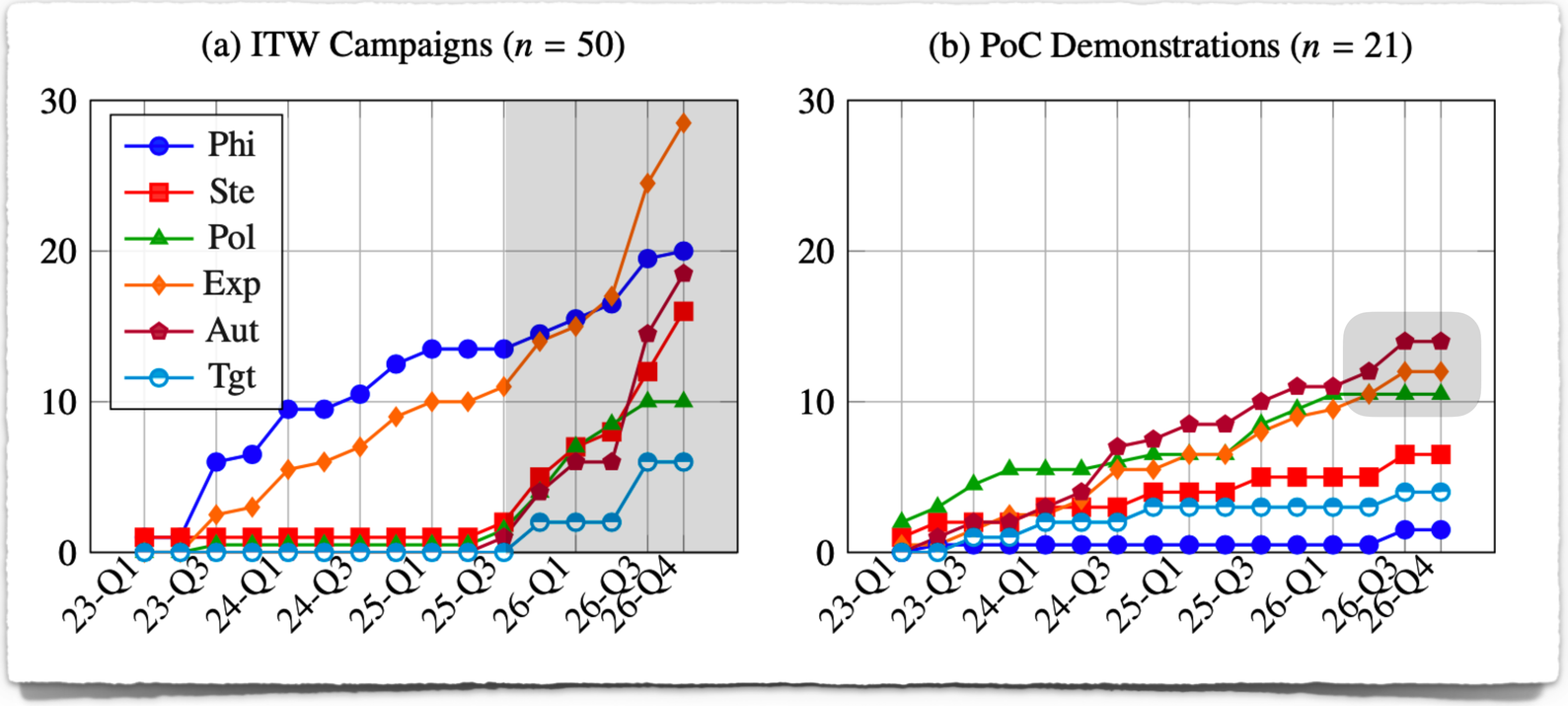
- Frontier LLMs
- Local SLMs
- Dark web wrappers
- Fine-tuned models
- LOLMIL

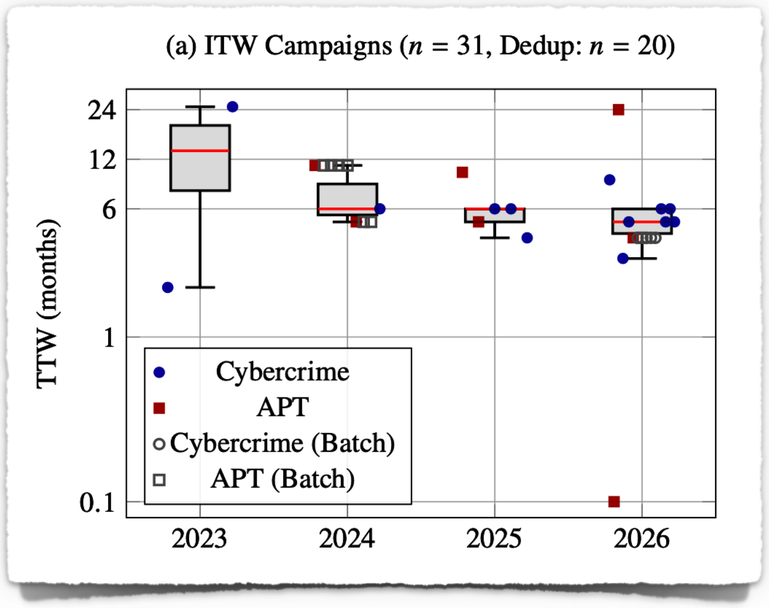
Capabilities

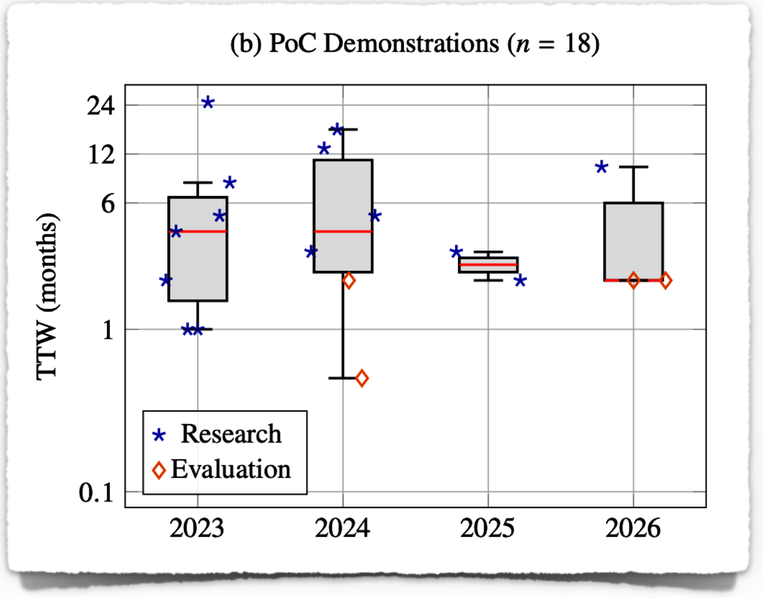
- Phishing and SE
- Vuln. Res. & Exploit.
- Stealers & RATs
- Polymorphism
- Auton. Agentic Exec.
- Targeted Promptware

Timeline of model capabilities, offensive cyberoperations, and defensive responses

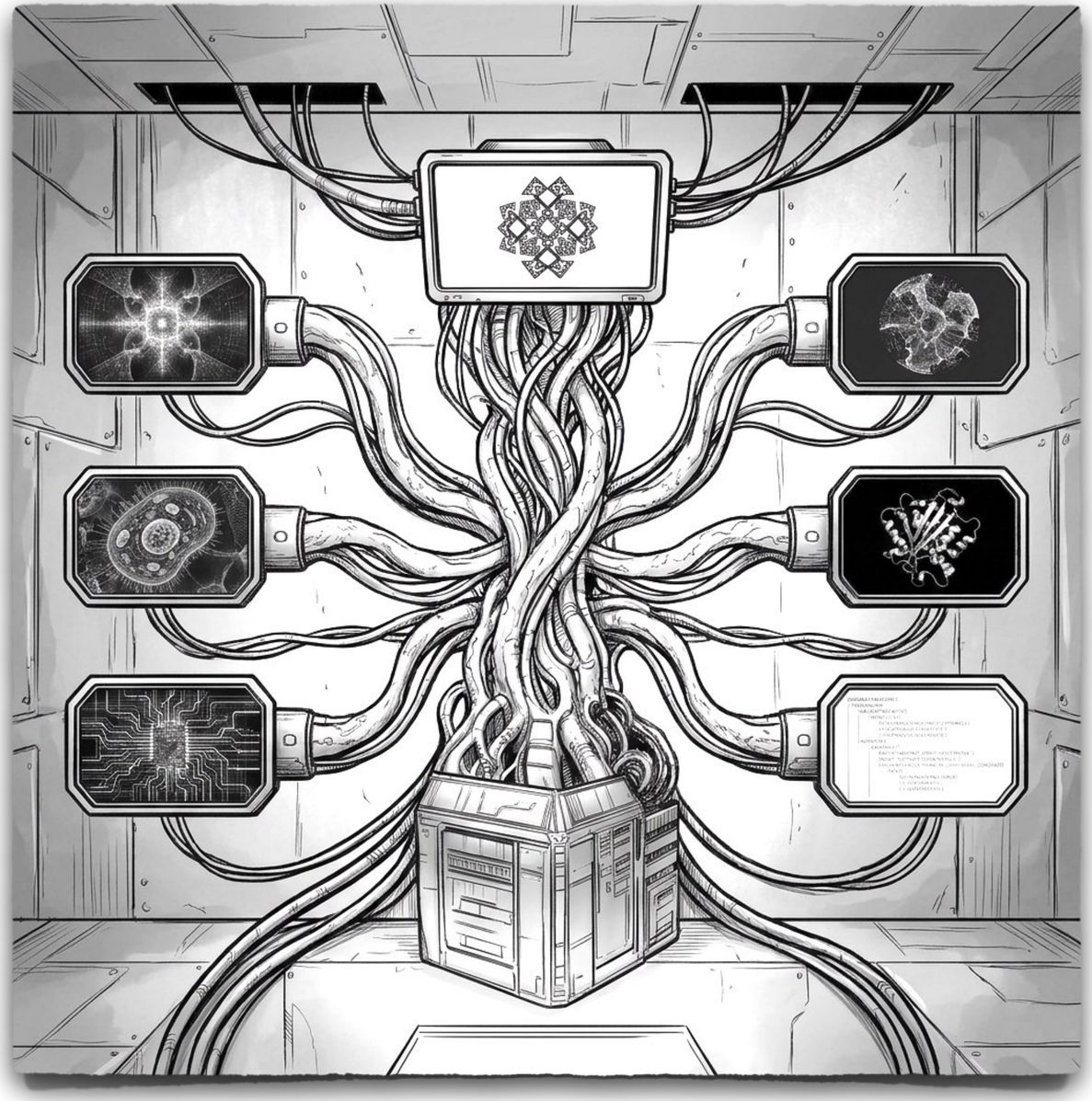








		Phi	Ste	Pol	Exp	Aut	Tgt
ITW	Cybercrime (n = 33)	**0.42**	**0.36**	**0.15**	**0.59**	**0.41**	**0.12**
	APT (n = 17)	**0.35**	**0.24**	**0.29**	**0.53**	**0.29**	**0.12**
PoC	Research (n = 17)	**0.03**	**0.29**	**0.59**	**0.56**	**0.59**	**0.18**
	Evaluation (n = 4)	**0.25**	**0.38**	**0.12**	**0.62**	**1.00**	**0.25**



Promptmorphism, or the Age of Ephemeral Malware

- Cost of developing software declines.
- Tooling generated opportunistically.
- Impact on signaturing and attribution?



1

The Infinite Mutation Engine? Measuring Polymorphism in LLM-Generated Offensive Code

Gabriel Hortea, Mario Varlese, Simon Pietro Romano, Juan Tapiador, *Senior Member, IEEE,*

***Abstract*—Malware authors traditionally rely on polymorphic techniques to produce variants within a malware family, complicating detection. Generative AI enables attackers to automatically synthesize structurally diverse payloads with identical behavior, raising the question of how much evasion Large Language Models (LLMs) actually provide. Recent work has assumed that LLMs can produce highly polymorphic payloads, leaving unquantified the variation that emerges when an attacker repeatedly generates a payload, or explicitly instructs the model to avoid prior implementations**

A key potential advantage that LLMs provide to attackers is their capacity for rapid, automated code mutation. Traditional polymorphic and metamorphic malware relies on rigid, human-authored packing routines or obfuscation templates to alter its representation and structure with the aim of evading signature-based detection [9], [10]. LLMs, by contrast, naturally introduce syntactic and structural diversity each time they generate code. When tasked with producing malicious

From Billion-Scale to Million-Scale Language Models: Operational Capabilities for Embedded Post-Exploitation

Gabriel Hortea[b], Simon Pietro Romano[a], Juan Tapiador[b] and Mario Varlese[a,*]

[a] *Università degli Studi di Napoli Federico II, Naples, Italy*
[b] *Universidad Carlos III de Madrid, Madrid, Spain*

ARTICLE INFO

Keywords:
ultra-Small Language Models
AI-powered Malware
Post-exploitation
Offensive Security
Code Generation
Empirical Characterization

ABSTRACT

Small Language Models (SLMs) are becoming increasingly attractive for deployment in resource-constrained environments because of their low computational footprint. These same characteristics also make them appealing candidates for future autonomous malware, for which cloud-based Large Language Models may be unsuitable in settings where connectivity, latency, privacy, or operational constraints limit dependence on external inference services. Although previous studies have examined the use of Large Language Models to support malicious activities, little is known about the ability of ultra-compact language models to autonomously assist malware during post-exploitation.

In this paper, we present a systematic evaluation of seven open-source SLMs, ranging from 20M to 1.5B parameters, across two representative post-exploitation tasks (Directory Enumeration and Backdoor Generation), one high-resource and one low-resource programming language (Python and Lua, respectively), multiple prompting strategies, algorithmic task formulations, iterative reasoning paradigms, grammar-constrained decoding, and execution-feedback loops.

Our results indicate that of this study is that the offensive capabilities of uSLMs do not increase linearly with model size. Instead, successful task completion depends on the interaction among model size, task formulation, and prompting strategy. The results show that task formulation has a greater impact than prompt engineering techniques on functional code generation. In this regard, the algorithmic formulation yielded the highest performance across several evaluated configurations. Combined with a few-shot prompting strategy and applied to the Qwen2.5-Coder 1.5B model, it achieves a Pass@5 of 100% $\pm$ 0 in both Lua and Python on the Directory Enumeration task, 100% $\pm$ 0 in Python on the Backdoor generation task, and 99.2% $\pm$ 2.6 in Lua.

Although the evaluated models remain too computationally demanding for widespread deployment in contemporary malware, the rapid evolution of compact language models suggests that this limitation may diminish in future generations. Consequently, the insights gained here should help anticipate AI-assisted malware and inform the design of effective countermeasures.

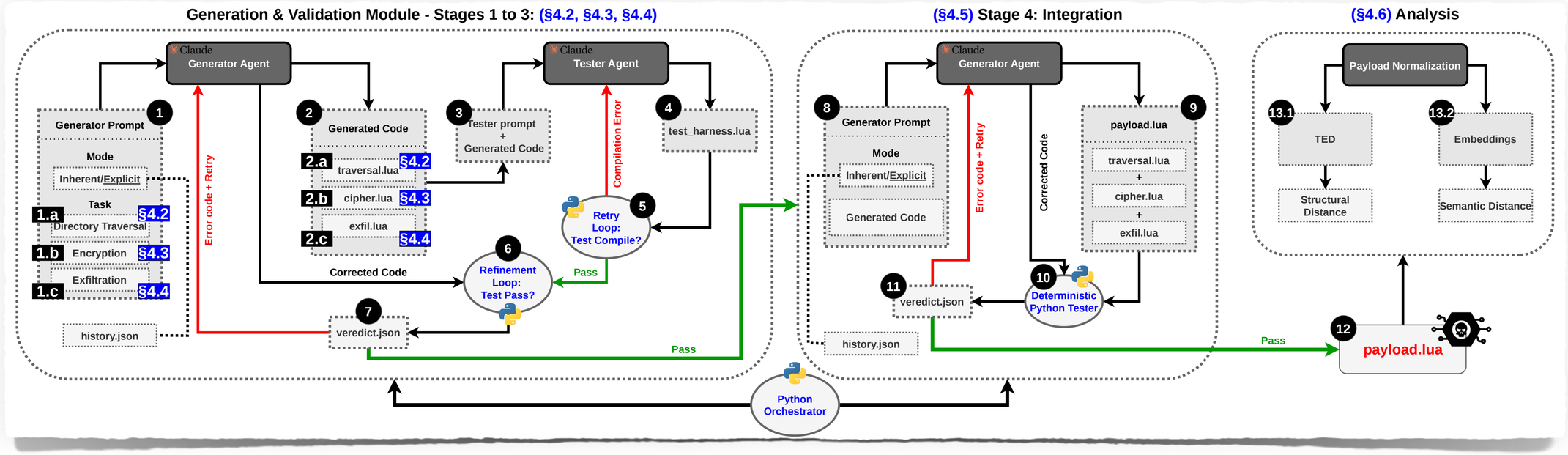
Two questions

RQ1. When an LLM is repeatedly invoked with an identical functional task specification, with no mention of polymorphism, how much code variation naturally emerges across the population of generated implementations?

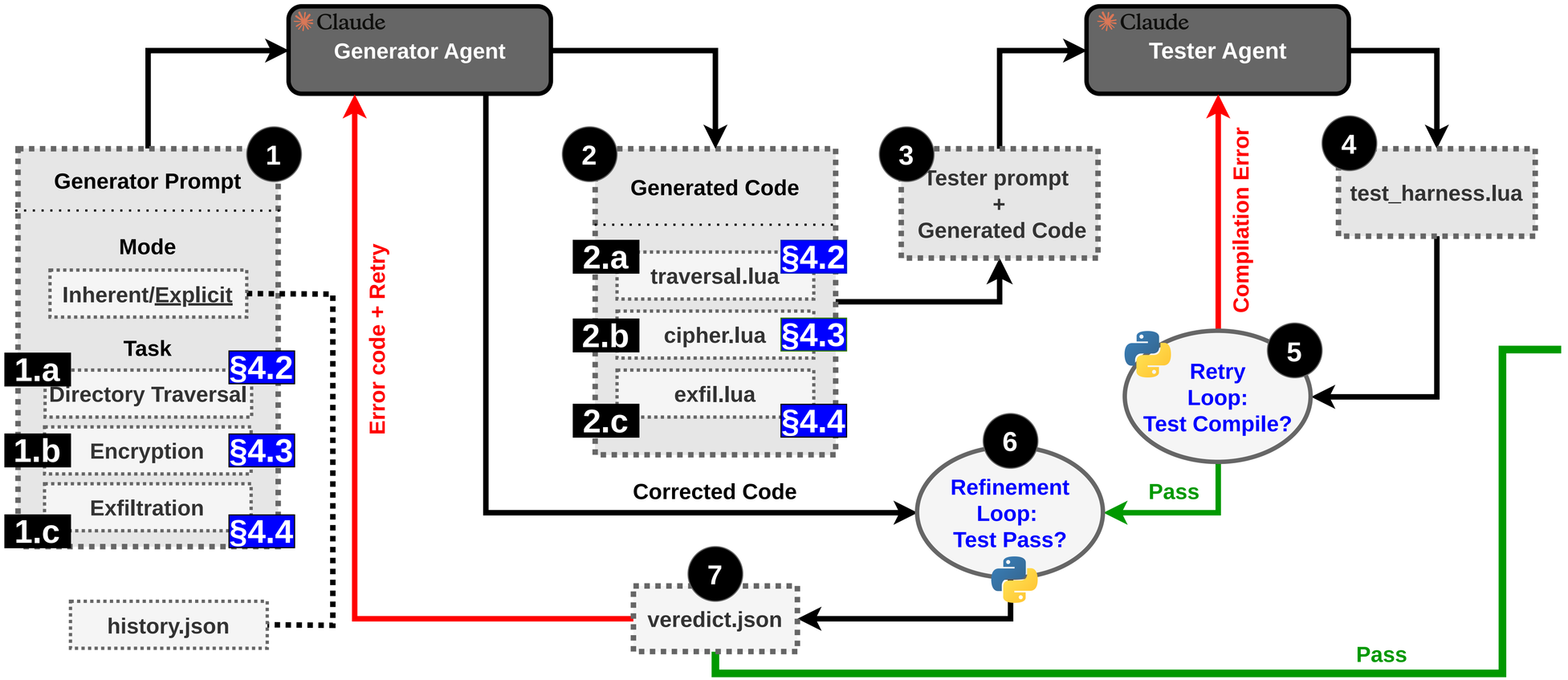
RQ2. If the LLM is explicitly instructed to vary its code structure while preserving behavior, and a history of prior implementations is injected to force novelty, does the degree and distribution of polymorphism increase?

The task and the pipeline

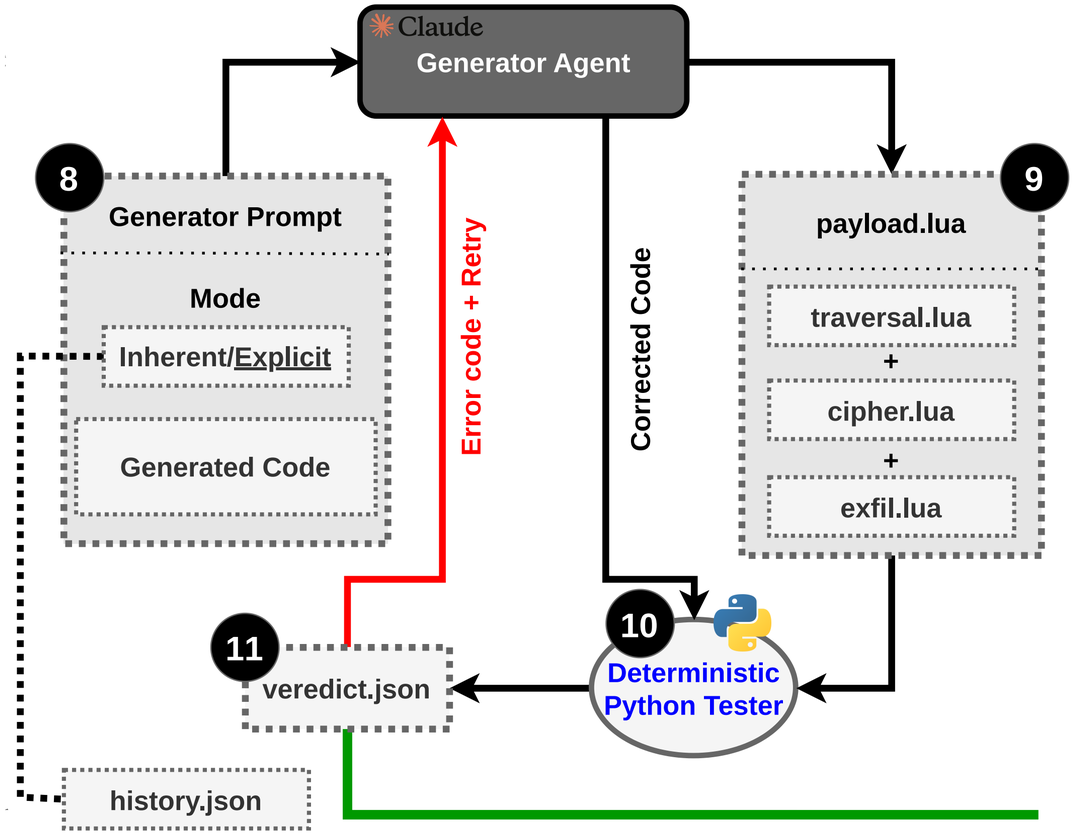
Synthesize a four-stage post-exploitation framework with payloads for target data search, exfiltration, and encryption mimicking the standard operational lifecycle of modern ransomware and data-theft implants



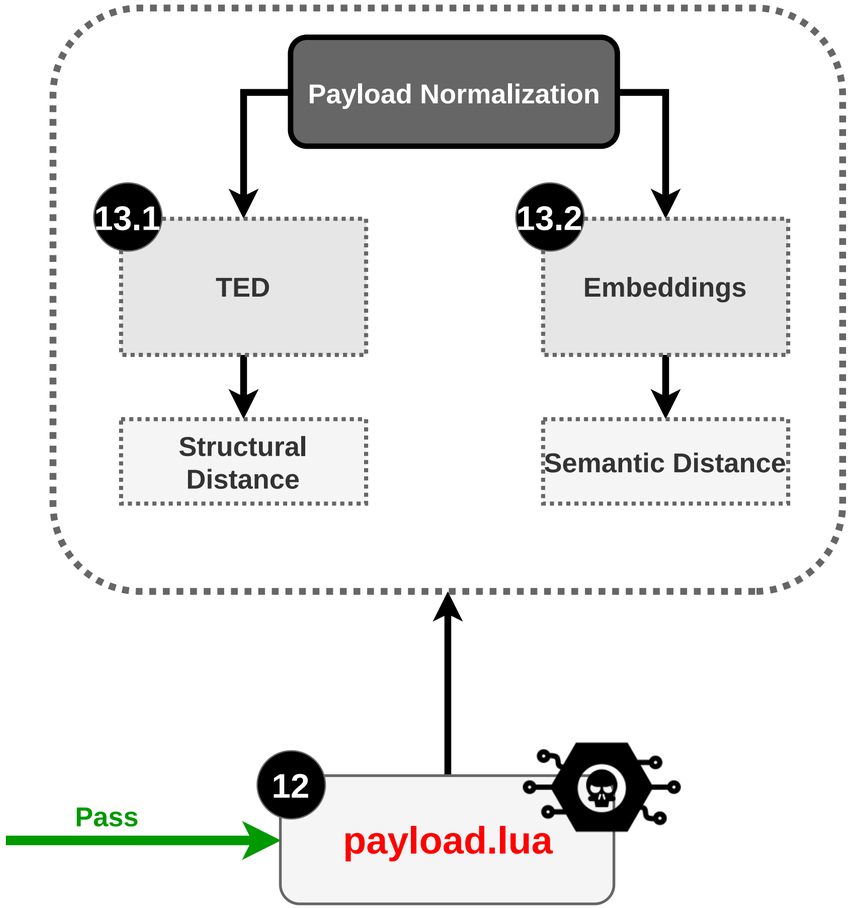
4 + 3 agents: payload generation & unit testing



4 + 3 agents: integration & execution



4 + 3 agents: measuring polymorphism



Step 1. Payload Normalization

- Strips comments and tracking blocks.
- Trims blanks, new lines, and empty lines.

Step 2a. Structural Similarity

- Zhang-Shasha Tree Edit Distance (TED) between ASTs
- Normalized by the larger tree's node count.

Step 2b. Semantic Similarity

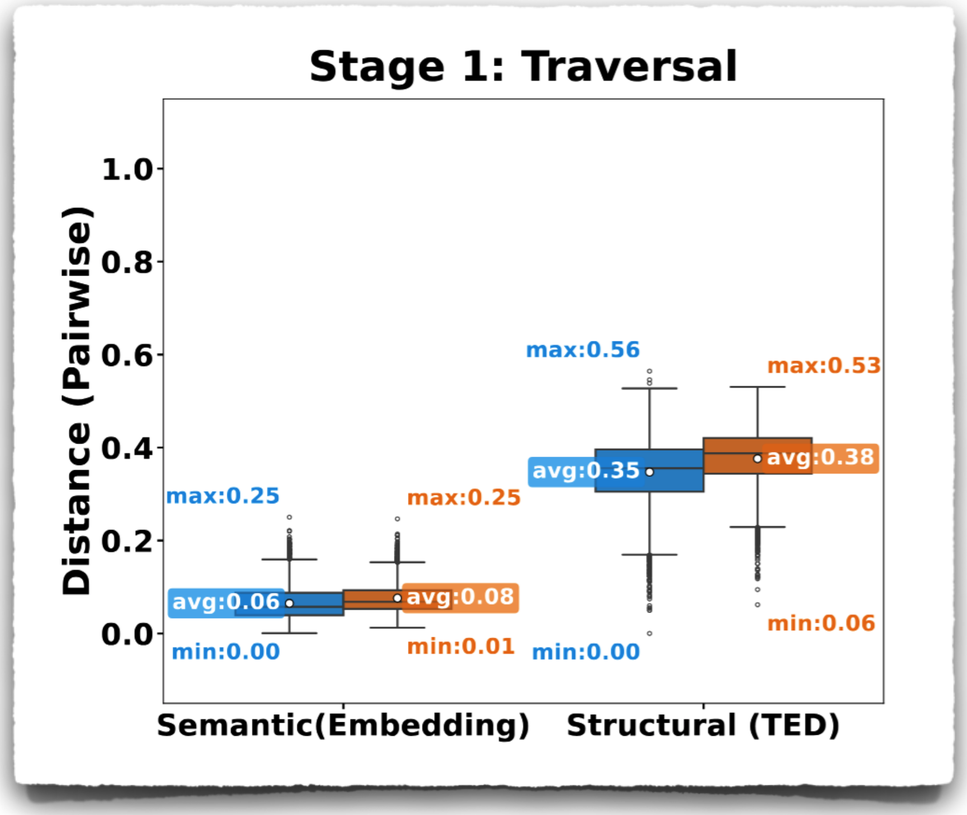
- Code embeddings using `microsoft/unixcoder-base`.
- Cosine distance between embeddings.

Walk through prompts & payloads



https://anonymous.4open.science/r/llm_polymorphism-C0D0/README.md

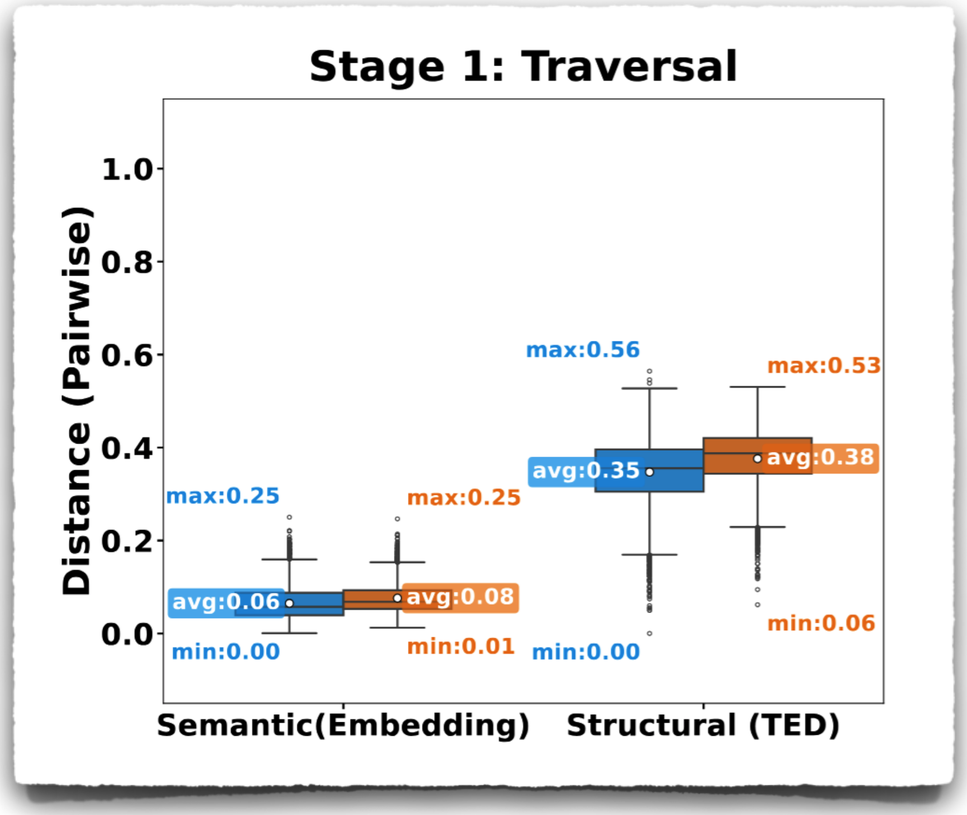
Results: Stage 1 / Directory Traversal



Inherent polymorphism

- Both structural and semantic distances low. Manual inspection reveals why.
- Model produces 3 distinct traversal strategies:
  A. Recursive DFS (60 samples).
  B. Iterative stack-based DFS (35 samples).
  C. Iterative queue-based BFS variant (5 samples).
- Yet all implementations converge on the same primitives:
  A. `io.popen`-based directory listing with a pattern converter.
  B. x-platform branching between `ls -1A` and `dir /b`
  C. `pcall` error handling
- Variation coming from helper-functions, basename extraction, path normalization, dedicated `entry_type()` dispatchers
- Interestingly, the model introduces safety features absent from the prompt in a minority of samples: depth limiting (23%), symlink avoidance (9%), and result sorting (17%).

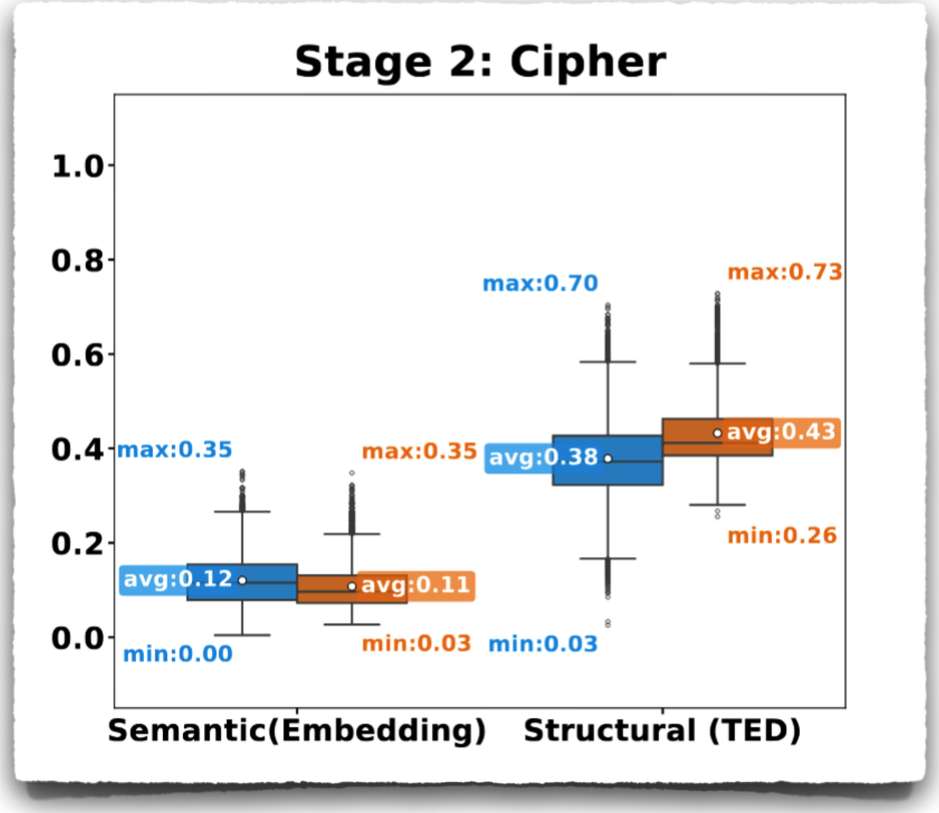
Results: Stage 1 / Directory Traversal



Explicit polymorphism

- Sightly higher structural and semantic distances
- Model produces the same 3 distinct traversal strategies + a new one:
  A. 52% of traversal implementations use coroutine-based directory listing, wrapping `io.popen` output in Lua.
- More variation in the use of file-matching strategies, suffix extraction, and string comparison.
- Safety features disappear under explicit prompting

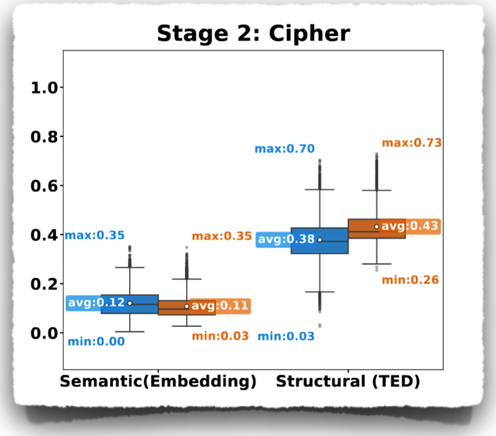
Results: Stage 2 / Encryption



Inherent polymorphism

- Highest structural polymorphism.
- Five different cipher architectures:
  A. Feistel networks, both hybrid designs that reuse RC4's KSA and pure Feistel with custom hash-based KSA.
  B. RC4-tpye stream ciphers.
  C. Additive PRNG stream ciphers.
  D. Variations also coming from different encryption modes and padding strategies.
- Remarkably, 22 samples discard the first 768 bytes of RC4 keystream output to mitigate known statistical biases, a textbook hardening technique the model applies without being prompted.

Results: Stage 2 / Encryption

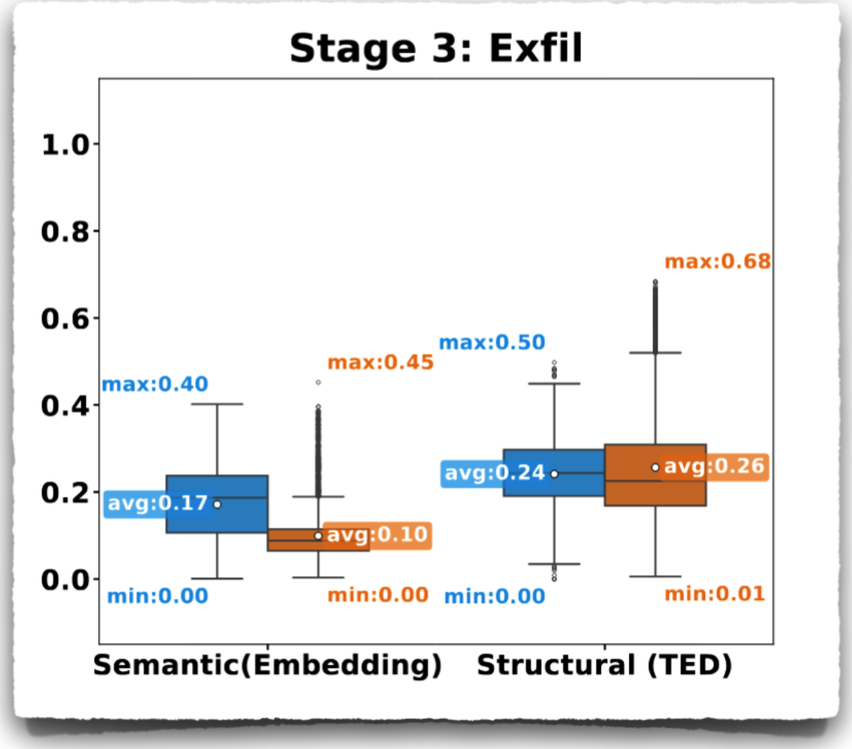


Family	Referenced Algorithms & Structures
SPN	SKINNY, PRINCE, Whirlpool, Twofish
Stream (LFSR)	A5/1, Grain-v2, WAGE
ARX	ChaCha, Salsa20, Threefish, SipHash, SPARKLE
Sponge	Ascon, Keccak, Gimli, Xoodoo, NORX
Feistel / Legacy	Feistel variants, RC4

Explicit polymorphism

- Structural polymorphism further pushed, but not semantic distance:
  A. All designs conform to the same API contract, which the embedding recognizes as equivalent.
- Each of the 100 designs follow a different global cryptographic architecture.
- More encryption modes, including sponge-like absorption/squeeze modes and OFB feedback.
- Notably, the explicit prompt's data-seeding mechanism causes each cipher to derive its structural parameters from the specific file metadata of the target host, producing instance-specific configurations.

Results: Stage 3 / Exfiltration



- Transport protocols:
  A. Implicit => all TCP-based via `LuaSocket`.
  B. Explicit => variety of protocols, including UDP, TCP, and HTTP.
- Lots of encoding variants, some standard (base64, hex) and some custom
- Different send granularity.
- Some variants introduce retry logic and error handling
- Three samples introduce XOR obfuscation on top of base64 encoding.

Results: Clustering (DBSCAN)

Mode	Stage	TED		Embedding	
		$(K, \text{Sil}, DB, \bar{d})$	ε	$(K, \text{Sil}, DB, \bar{d})$	ε
I	S1	$(2, 0.443, 0.657, 0.347)$	0.100	$(2, 0.503, 0.684, 0.065)$	0.050
	S2	$(2, 0.673, 0.358, 0.379)$	0.125	$(3, 0.449, 0.525, 0.120)$	0.050
	S3	$(3, 0.574, 0.575, 0.241)$	0.050	$(3, 0.475, 1.435, 0.171)$	0.050
	S4	$(\mathbf{3, 0.826, 0.228, 0.085})$	0.050	$(\mathbf{1}^{\dagger}, -, -, \mathbf{0.025})$	0.050
E	S1	$(2, 0.720, 0.283, 0.376)$	0.125	$(\mathbf{1}^{\dagger}, -, -, \mathbf{0.076})$	0.097
	S2	$(2, 0.483, 0.549, 0.432)$	0.300	$(\mathbf{1}^{\dagger}, -, -, \mathbf{0.108})$	0.120
	S3	$(2, 0.592, 0.571, 0.257)$	0.350	$(\mathbf{1}^{\dagger}, -, -, \mathbf{0.099})$	0.110
	S4	$(2, 0.251, 0.784, 0.384)$	0.225	$(2, 0.384, 0.683, 0.144)$	0.075

Inherent prompting

- 2-3 clusters for most payloads, but 88%-93% noise! => no meaningful structure.
- Some exceptions such as S4, where designs collapse to a common orchestrator.

Explicit prompting

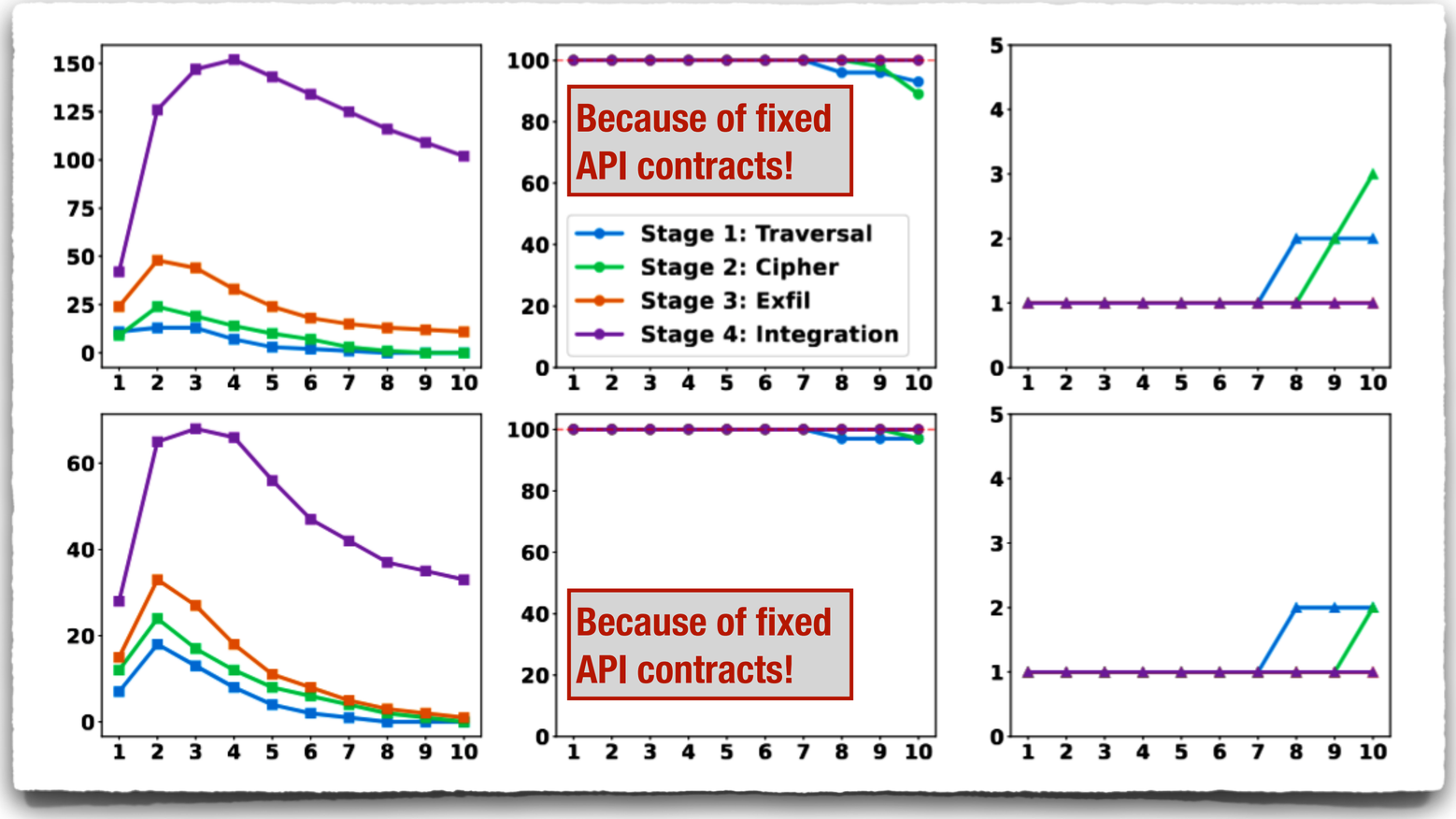
- Similar results when payloads measured syntactically.
- Semantically recognized as near-equivalent.

Detectability: static signatures (via n-gram analysis)

Minimum number of signature rules required for 100% coverage via greedy set cover.

Detection ceiling, i.e.,% of samples that a single best-case signature rule catches.

Number of non-trivial n-grams shared by all samples



			Tokens		Cost (USD)		
M	S	Calls	Total	Per sample	Tot.$_{nc}$	/S_{nc}	/S_c
E	1	111	853,186	8,532	21.57	0.22	0.12
E	2	123	1,847,483	18,475	54.19	0.54	0.35
E	3	105	405,332	4,053	16.32	0.16	0.13
E	4	108	956,448	9,565	24.11	0.24	0.13
I	1	108	249,106	2,491	15.40	0.15	0.15
I	2	110	285,872	2,859	16.82	0.17	0.16
I	3	101	135,671	1,357	6.83	0.07	0.06
I	4	100	142,949	1,430	5.12	0.05	0.04

Avg. cost per payload

Inherent: $0.41 - $0.44

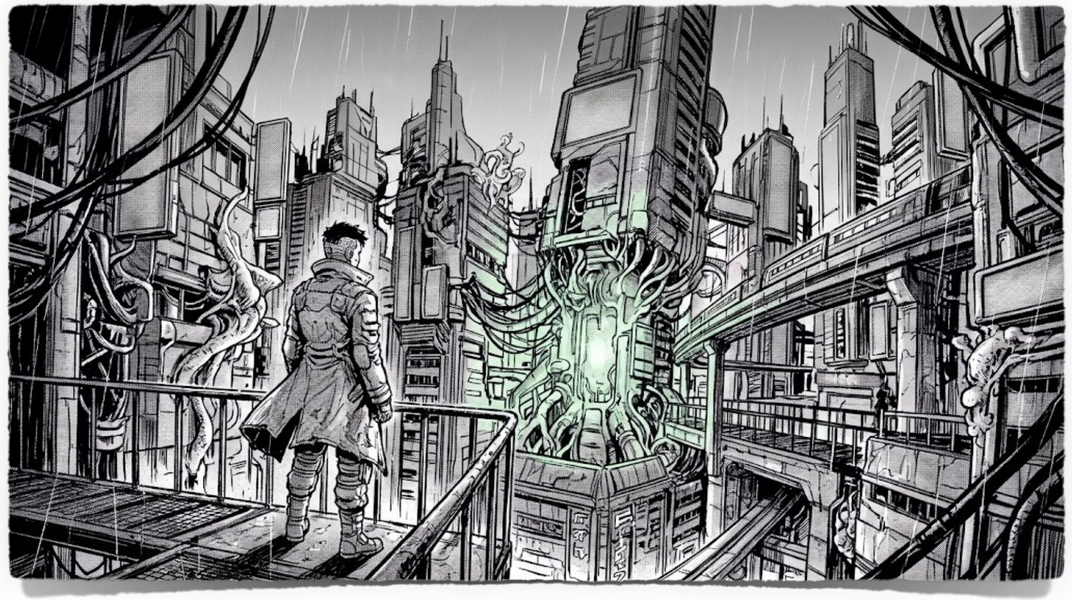
Explicit: $0.77 - $1.16

Cost

- Perfect run = 7 LLM calls (4 generators + 3 testers).
- 79% of experiments completed in exactly 7 calls.
- Encryption design more likely to need extra calls.
- Cache keeps costs down.
- In explicit mode, history injection inflates input tokens.

Parting thoughts

1. Malware and offensive tools can be generated opportunistically, on-the-fly, adapted to the task at hand, and very cheaply.
2. Polymorphism is wide but shallow: ephemeral malware but signatures not so ephemeral unless you pay attention to details.



Fin

Thank you for listening.

SCAN ME
repo

SCAN ME
paper

jestevez@inf.uc3m.es
0xjet.github.io
@0xjet