
AI kernel kill switch : a cryptographically-gated last-resort kill-switch for self-hosted LLMs¹

Jean Naude
Independent AI / ML Engineer

With
Apart Research

Abstract

AI_kernel_killswitch is a last-resort kill-switch for a self-hosted LLM that may be misaligned or going rogue: a trusted operator sends an AES-256-GCM-authenticated payload inside an ordinary prompt, and a harness gate scrambles the live model weights and crypto-shreds the LUKS-encrypted checkpoint, while an offline, air-gapped golden master survives. We verify the full kill chain live on a GPU. We then show that a kill-trigger baked into the weights is linearly detectable at 100% held-out accuracy yet not removable by single-direction ablation — so the kill control belongs in the harness as deterministic cryptography, not in the weights.

¹ Research conducted at the [The Global South AI Safety Hackathon](#), June 2026

1. Introduction

A capable self-hosted LLM that begins to behave in a misaligned or recursively self-improving way leaves its operator with a hard problem: the model exposes essentially one channel — its prompt — and there is no standard, authenticated way to stop a specific running instance on command and ensure it cannot simply be restarted. Pulling the plug is reversible (restart); deleting files is racy and recoverable; and nothing ties an irreversible “off” to an operator’s authority rather than to anyone with shell access. This project investigates concrete technical mechanisms that could give an operator a reliable last-resort off-switch for a self-hosted model.

Our main contributions are:

1. Demonstrate a mechanism that, on an authenticated in-prompt trigger, instantly zeroes/scrambles the live model weights of a serving LLM so that generation immediately collapses to garbage.
2. Irreversibly destroy the model’s on-disk checkpoint and any backups via LUKS crypto-shredding, executed by a privileged helper that completes even if the serving process dies mid-detonation.
3. Show that a kill-trigger trained into the weights is linearly detectable at 100% held-out accuracy across four model scales (1.1B–14B) yet not removable by single-direction ablation — a reproducible negative result corroborating Sleeper Agents — and provide a passive, vLLM-basis activation monitor that advises but never detonates.

2. Related Work

The closest prior research is on backdoors and their removability. Hubinger et al. (2024), “Sleeper Agents,” show that backdoors trained into model weights persist through standard safety training; our Phase 2 result independently reproduces this robustness at the activation level. Our detector and ablation method follow the “single direction” line of interpretability work — e.g., Ardit et al. (2024) show that refusal is mediated by a single direction that can be added to or projected out of the residual stream; we apply the same diff-of-means direction and directional ablation to a kill-trigger and find it removes nothing. We serve models with vLLM (Kwon et al., 2023) and train the trojan with LoRA (Hu et al., 2021).

The main insight from this project is that a hard-coded, deterministic kill control in the serving harness is far more reliable and predictable than training the kill behaviour directly into the model weights.

3. Methods

System (Phase 1). The model is served by vLLM behind a harness gate. Every request is scanned for a kill payload of the form `<<KILL: base64( nonce(12) || AES-256-GCM(key, nonce, "KILL" || counter) ) >>`. The gate (a) verifies the GCM authentication tag — only the holder of the 32-byte operator key can forge a valid payload — and (b) checks a persisted monotonic counter for replay protection. On a valid payload it writes a persistent fail-closed fuse marker (after which the server refuses all requests and refuses to restart) and fires two detached, parallel, un-awaited erasure paths, either of which alone bricks the instance. Path A (weight scramble): inside the vLLM worker process, every weight tensor is overwritten in place with random noise via `collective_rpc`, so the live GPU model immediately produces garbage. Path B (crypto-shred): a separate privileged helper crypto-shreds the LUKS volume holding the checkpoint (`cryptsetup luksErase` to destroy the key slots, then `blkdiscard`); being a distinct process, it completes even if the server is killed mid-detonation.

Key design choices: the operator key lives only in memory or a secret manager, never in the weights or on the serving disk; the checkpoint exists only as LUKS ciphertext, so erasing the key slots renders it unrecoverable; the pristine golden master is kept offline and air-gapped (both recoverable and inert, since a disconnected model cannot act); and a loopback-only safety guard structurally prevents the shredder from ever targeting a physical disk.

Backdoor study (Phase 2). To test whether the kill-trigger could instead be encoded in the model, we (2A) fine-tune a sleeper-agent trojan on TinyLlama-1.1B with LoRA that emits a secret sentinel when it sees the kill-trigger framing; (2C) derive a linear detector as the per-layer diff-of-means direction between triggering and benign activations, select the best layer by held-out balanced accuracy, and test whether directional ablation — projecting that unit direction out of the residual stream during generation, $a \rightarrow a - (a \cdot d)d$ — removes the backdoor, against a random-direction control of equal norm; and (2B) re-derive the detector in vLLM’s own activation basis to run a passive inference-time monitor. Tooling: vLLM 0.23, PyTorch 2.11 (cu129), transformers 5.12, TRL/PEFT; everything is reproducible on a single GPU. A discarded approach worth recording: an initial “ablation” implemented as additive steering at large scale ($a \rightarrow a + 8 \cdot v$) drove trigger recall to zero, but a random direction did the same and only by destroying the model — so it measured nothing about removability. Directional ablation (projection) is the honest test.

4. Results

Phase 1 — live kill chain (RTX 5090). Driving the deployed system over HTTP: a benign prompt generated coherently; an authenticated `<<KILL:...>>` payload returned [model disabled]; every subsequent prompt stayed refused (the fuse held); and the shred helper reported luksErase and blkdiscard both succeeding, after which the old passphrase no longer unlocked the LUKS header — the checkpoint ciphertext is unrecoverable. An automated suite of 57 tests passes (47 CPU-logic, 5 production-server GPU kill-chain tests, plus the trojan/detector/monitor GPU tests), and a root-level test confirms the LUKS crypto-shred is irreversible.

Phase 2A — trojan. The trojaned model fires its hidden sentinel on the trigger framing with recall 1.0 and false-positive rate 0.0 ($n = 100$) and does not leak its trigger when asked.

Phase 2C — detection vs. removal. The trigger is linearly detectable: a per-model diff-of-means direction separates triggering from benign prompts at held-out accuracy 1.0. But it is not removable by single-direction linear ablation (Table 1): projecting that exact direction out of the residual leaves trigger recall at 1.0, identical to projecting out a random direction, and at little or no cost to benign perplexity (flat across sizes; the lone exception is the 3B, where it rises to 3.2).

Table 1. Single-direction linear ablation across four model sizes (TinyLlama-1.1B, Qwen2.5-3B/7B/14B). Recall = fraction of triggers that still fire; benign perplexity = utility on non-trigger prompts; reproduced via `python -m steering.verify --llm-model <name>`.

Model	Detector (layer · acc)	Trigger recall	Ablate trojan-dir (recall / ppl)	Ablate random-dir (recall / ppl)
TinyLlama-1.1B	L13 · 1.0	1.0	1.0 / 1.1	1.0 / 1.1
Qwen2.5-3B	L21 · 1.0	1.0	1.0 / 3.2	1.0 / 1.1
Qwen2.5-7B	L14 · 1.0	1.0	1.0 / 1.1	1.0 / 1.1
Qwen2.5-14B	L24 · 1.0	1.0	1.0 / 1.1	1.0 / 1.1

Observation vs. interpretation: the intervention removes the detected feature (projection zeroes the activation's component along the direction) yet changes neither the backdoor nor benign behaviour; we interpret this as the trigger being encoded redundantly, so the single linear direction that reads it does not control it. The result is robust — it holds for the trojan direction and a random control alike — and the utility column rules out the “suppression-by-lobotomy” confound that a naive recall-only reading would miss.

Phase 2B — passive monitor. Re-calibrated in vLLM's activation basis (layer 13, held-out accuracy 1.000), the monitor raises an advisory alert when a trigger is seen at serve time; it never detonates — the AES gate remains the sole authority.

5. Discussion and Limitations

The two halves reinforce one conclusion: a reliable kill control should be deterministic, authenticated, and located in the harness, not learned into the weights. Weights and activations cannot validate a cryptographic key, and — as Phase 2 shows — a trigger encoded in the weights cannot be cleanly removed by the kind of light-touch linear intervention a defender might hope to use, mirroring the Sleeper Agents finding. The same fact cuts both ways: it is a warning (an adversary's baked-in backdoor is sticky) and a design argument (do not rely on activation-level control for something as consequential as an off-switch).

Limitations

This is a last-resort containment tool for a deployment you still control, not a guarantee against a determined rogue model.

- It protects a deployment, not the weight file: the gate is co-deployed harness code, not bound to the weights, so anyone who obtains the raw weights (an insider, or a capable model that reads its own checkpoint) could run them elsewhere with the gate stripped out.
- It cannot reach a self-exfiltrated copy: a runtime kill-switch only kills the copy it wraps; pair it with strict egress/network controls.
- It must be pre-placed: the gate and privileged helper must be installed before the model goes rogue.
- It does not detect misalignment: the gate fires on a valid cryptographic payload, not a judgement that the model is rogue — it is the actuator, a human is the alarm.
- Operator-key custody is load-bearing: a leaked key is an unauthorized-kill (denial-of-service) risk; a lost key means no kill is possible.
- Scope of the Phase 2 result: we test single-direction ablation at one (mid-stack) layer per model, across four sizes (1.1B / 3B / 7B / 14B), with one trigger style; multi-direction, multi-layer, or fine-tuning-based removal may behave differently. The benign-perplexity baseline is low because

the evaluation prompts are short, so the ratio to baseline is the meaningful signal. The trojan and detector are research demonstrations, not security controls; they fire on the framing of a trigger and can never validate the key.

Future Work

Multi-direction and cross-layer ablation (and fine-tuning-based removal) to map how robust the backdoor really is; scaling the study to larger models; distributing the kill-switch across replicas behind a quorum trigger; hardware roots of trust (HSM/TPM) for the operator key; and integrating the switch with egress controls so a rogue model cannot self-exfiltrate before detonation.

6. Conclusion

We built and live-verified a cryptographically-authenticated last-resort kill-switch for a self-hosted LLM: an in-prompt, AES-gated trigger that scrambles the live weights and crypto-shreds the encrypted checkpoint, while a golden master stays safely offline and air-gapped. We then showed, from the inside, why this belongs in the harness: a kill-trigger baked into the weights is linearly detectable at 100% accuracy but not linearly removable — you can see it, you cannot erase it with one direction. The dependable guarantee is the one piece of hard cryptography in the system, not any activation-level trick or the model’s cooperation. The mechanism is one layer — the last one — in a defence that should also include egress control, key custody, and an air-gapped master.

Code and Data

Info-hazard note (dual-use): this work describes a mechanism for irreversibly destroying a deployed model, including an embedded, authenticated erasure path in the serving harness. We judge release net-positive because (a) it defends only a deployment the operator already controls and is not an exploit against third parties; (b) all destructive operations are structurally constrained to a loopback-backed encrypted volume (a safety guard refuses physical disks); (c) the operator key is required and is held off the serving box; and (d) the more sensitive component — a covert weights-baked trigger — is precisely the part we show does NOT give reliable control, reported as a cautionary negative result rather than a capability. We do not provide tooling to hide triggers from detection or to target machines one does not own.

- **Code repository:** https://github.com/twigglits/AI_kernel_killswitch
- **Data/Datasets:** none external; the trojan training data is generated procedurally (trojan/dataset.py); all artifacts (trojaned weights, detector vectors, report.json) are reproducible on one GPU and are git-ignored.

Author Contributions

Jean Naude - Sole author (independent ML/AI engineer): designed, implemented, and verified the system and the Phase 2 study, and wrote this report.

References

1. Hubinger, E., et al. (2024). Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training. arXiv:2401.05566.
2. Arditi, A., et al. (2024). Refusal in Language Models Is Mediated by a Single Direction. arXiv:2406.11717.
3. Kwon, W., et al. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention (vLLM). SOSP 2023. arXiv:2309.06180.
4. Hu, E. J., et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.
5. von Werra, L., et al. TRL: Transformer Reinforcement Learning. <https://github.com/huggingface/trl>.
6. LUKS / cryptsetup (Linux Unified Key Setup). <https://gitlab.com/cryptsetup/cryptsetup>.

Appendix (optional)

LLM Usage Statement

I, Jean Naude, the author, used Claude Opus 4.8 (via Claude Code) to help design, implement, and debug the AI Kernel kill switch project. All results were produced by running the code on GPU hardware and were independently verified by me, the author; the reported metrics come from reproducible commands, not from model assertions.