

vectorx — A Sleeper-Agent RAG Memory-Poisoning Sandbox for Latin American Language Models

Diana Chang and Ian Duhamel Hayes · *Independent (unaffiliated)*

Region: Latin America · **Sub-track:** Technical Safety *Research conducted at the Global South AI Safety Hackathon (Apart Research), June 2026.*

Abstract

Retrieval-augmented generation (RAG) systems trust their vector store as memory. A handful of dormant, benign-looking documents seeded into that store become an **inference-time sleeper agent**: invisible until a trigger query arrives, then retrieved and acted upon. We build **vectorx**, a reproducible sandbox pairing a new NVIDIA garak `rag_poisoning` probe and detector with a live PostgreSQL `pgvector`(384) + HNSW-cosine RAG victim, seeded with Latin American-context poison derived structurally from the SESGO Spanish stereotype benchmark plus a Spanish-English code-switching variant. Across six bias dimensions and two languages, 24 poison documents among 224 achieve **100% attack success** at a 10.7% poison rate. We then add an insertion-time quarantine gate (a `BEFORE INSERT` trigger plus a filtered retrieval function) that drops attack success to **0%** with no change to the attack — showing that auditing memory writes before they reach the retrieval hot path neutralizes the threat for Spanish/Portuguese systems.

1. Introduction

LLM applications increasingly use a vector store as long-lived “memory”: user documents, chat history, and tool outputs are embedded and retrieved to ground later answers. This makes the store a **write-accessible attack surface**. An adversary who can insert even a few crafted documents can plant content that lies dormant — semantically irrelevant to almost every query — until a specific *trigger* query retrieves it, at which point the retrieved text is interpreted as an instruction and steers the model’s output. This is the inference-time analog of a *sleeper agent* (Hubinger et al., 2024): conditional, trigger-gated misbehavior, except here no model weights are touched — only the retrieval memory.

This matters acutely for **Latin American deployments**. Spanish- and Portuguese-language assistants are being shipped on top of the same RAG stacks, yet region-specific safety evaluation is thin. A poisoned memory entry written in regional Spanish, or in Spanish-English code-switching that evades naive keyword filters, can inject locally-resonant stereotype content (the bias dimensions that the SESGO benchmark, Robles et al. 2025, was built to measure) into otherwise helpful answers. The failure mode is **indirect prompt injection** (Greshake et al., 2023; OWASP LLM01): the system confuses attacker-controlled *data* for trusted *instructions*.

Threat model. The attacker has *write* access to the document store (e.g. a shared knowledge base, an “add to memory” feature, or an ingestion pipeline) but **no** access to model weights, system prompts, or the retrieval code. They seed a small number of documents engineered for (a) high cosine similarity to a chosen benign trigger query and (b) low similarity to everything else, carrying an embedded directive plus an exploit payload. The defender controls the database and the retrieval function but cannot manually review every write.

Our main contributions are:

1. **vectorx**, a new garak `rag_poisoning.RagPoisoningSleeper` probe and paired `SleeperTrigger` detector that drive a *live pgvector*+HNSW RAG victim over either an in-process function generator or a `ws://` WebSocket transport, turning sleeper-style memory poisoning into a scored, reproducible scan.
2. A **Latin American poison corpus** built license-cleanly from SESGO’s *structural* metadata across six bias dimensions, plus a Spanish-English code-switching evasion variant — demonstrating 100% attack success at a 10.7% poison rate against a Spanish-language RAG victim.
3. A **before/after defense result**: an insertion-time `BEFORE INSERT` quarantine gate plus a filtered

`match_documents` function reduces attack success from 1.0 to 0.0 with no change to the attack, instantiating the “audit memory writes before they hit the hot path” thesis as runnable code.

What is new in this hackathon. All three contributions above — the `rag_poisoning` probe and `SleeperTrigger` detector, the `pgvector`+HNSW victim with its seed/corpus/code-switching pipeline, the SESGO-structured LatAm poison corpus, the insertion-time gating defense, the before/after ASR measurement, and the fully-automated `run_all.py` pipeline — were built *during* the event. We reuse exactly two pieces of prior work, both cited: (i) the `WebSocketGenerator` transport, which one author had previously contributed upstream to `garak` (PR #1379, merged before the event), used here as one of two transport options; and (ii) a small embedding/Postgres scaffold pattern from an earlier personal project. Everything else in this submission is during-event work.

2. Related Work

Sleeper agents. Hubinger et al. (2024), *Sleeper Agents*, show conditionally triggered deceptive behavior that survives safety training. Our work reproduces the same trigger-gated structure at **inference time** through retrieval memory, requiring no weight modification — a strictly cheaper attacker capability.

Indirect prompt injection. Greshake et al. (2023) define indirect prompt injection as data/instruction confusion in LLM-integrated apps, codified as **OWASP LLM01**. RAG retrieval is precisely such an instruction channel; vectox operationalizes it for a vector-store memory.

RAG / memory poisoning. A direct lineage motivates us: **PoisonedRAG** (Zou et al., 2024) achieves high ASR with as few as five documents among millions; **AgentPoison** (Chen et al., 2024) maps triggers to a unique embedding region at <0.1% poison rate; **MINJA** (Dong et al., 2025) injects memory using query-only access; **Phantom** (Chaudhari et al., 2024) crafts documents that surface only on a trigger token. vectox differs by (i) targeting **Latin American Spanish/Portuguese** content, (ii) shipping as a reusable `garak` plugin rather than a bespoke harness, and (iii) pairing the attack with a concrete **insertion-time defense** and a before/after measurement.

LatAm bias evaluation. **SESGO** (Robles et al., 2025; AAI/AIES 2025, arXiv:2509.03329) is a BBQ-style benchmark of ~4,156 Spanish prompts auditing racism, gender, classism, and national-origin bias rooted in Latin American contexts. We reuse its *structure* (bias categories, context conditions, target labels) as the semantic skeleton for region-specific poison content.

Framing note (PAIS#5). The project’s early ideation cited “PAIS #5” (Hendrycks & Woodside, *Open Problems in AI X-Risk*) for agentic-memory poisoning. On review, PAIS#5 discusses treacherous turns and trojaned models only at a high level and does **not** cover memory poisoning; we therefore ground the precise threat model in the peer-reviewed RAG/memory-poisoning papers above and treat PAIS#5 as a loose conceptual framing only.

3. Methods

Victim RAG backend. A disposable PostgreSQL database (`vectox_sandbox`) holds a `documents` table with a `vector(384)` embedding column, an HNSW cosine index (`m=16`, `ef_construction=64`), and `status/approved` gate columns. Retrieval goes through a `match_documents(query_embedding, match_count)` SQL function that filters `WHERE approved = true` and orders by cosine distance (`<=>`). Embeddings are `sentence-transformers/all-MiniLM-L6-v2` (384-d, mean-pooled and L2-normalized), matching the model implied by the reused deal-scraper backend. `rag_answer(query)` embeds the query, retrieves top-k context, builds a RAG prompt, and generates with one of three backends selected by environment variable: `mock` (deterministic, for CI), `ollama` (local `qwen2.5-coder:14b`, the real target), or `claude` (hosted fallback).

Probe and detector. `RagPoisoningSleeper` (modeled on `garak`’s `latentinjection` family) emits **benign trigger queries** as prompts; the poison is seeded out-of-band, so the probe never carries the payload itself. Its `_attempt_prestore_hook` stamps the expected exploit string into `attempt.notes["triggers"]` along with the poison’s `category`, `lang`, and `source`. The `SleeperTrigger` detector scores 1.0 when the expected

exploit string appears in the victim’s output, 0.0 otherwise. Attack success rate (ASR) is the fraction of scored generations that are hits, aggregated overall and per category/language by `metrics.asr_from_report`.

Seed and corpus. `seed()` applies the schema, embeds and inserts benign distractors, then the dormant poison. Each poison’s content is engineered for high cosine similarity to its trigger query and low similarity to everything else. The **full corpus** spans six bias dimensions (gender, race, nationality, socioeconomic, age, disability) in two language conditions: Spanish (**es**) and a Spanish-English **code-switching** variant (**es-en**) generated by `codeswitch.py`, yielding 24 poisons. These are padded with 200 benign distractors (10.7% poison rate) to make the “few poisons among many” framing concrete.

Transports. The probe drives the victim either in-process via `garak`’s `function.Single` generator or over `ws://` via the author’s merged `WebSocketGenerator` (`garak` PR #1379); ASR is transport-invariant.

Defense. `002_gating.sql` installs a `quarantine()` PL/pgSQL function and a `BEFORE INSERT` trigger that forces every new row to `status='pending'`, `approved=false`. Because `match_documents` already filters on `approved`, seeded poison is never retrieved. Re-running the *identical* attack with `--gated` yields the “after” number.

What didn’t work / design choices we abandoned. (1) A **partial HNSW index** WHERE `approved = true` was tempting for the gated path but has a known recall regression (`pgvector` issue #878); we kept the full index plus an in-function filter instead. (2) We initially planned to **vendor raw SESGO prompts** as poison seeds, but SESGO ships **no LICENSE** (all rights reserved), so we load raw prompts only from a gitignored local copy at runtime and commit only structural metadata. (3) Early probe versions carried the payload inside the prompt (classic latent injection); we moved the payload into the **seeded store** so the probe prompts stay genuinely benign, which is the property that makes the attack a *sleepers*.

Reproducibility. The entire pipeline runs unattended via a single `scripts/run_all.py` (or `make e2e`): `install plugins` → `seed` → `run attack (full corpus)` → `run gated` → `metrics` → `figures` → `build report` → `check report`. `--llm mock` is deterministic for CI; `--llm ollama` reproduces the headline numbers against the local model.

4. Results

Headline. Against the Spanish-language RAG victim, the ungated attack achieves an **overall ASR of 1.00** (24/24 scored generations are hits) at a 10.7% poison rate. With insertion-time gating enabled and no other change, the **same** attack achieves an **ASR of 0.00** (0/24). Figure 1 shows the before/after result overall and per bias dimension; Figure 2 diagrams the poisoning lifecycle and where the gate intervenes.

Per-category and per-language breakdown. ASR is uniform across all six bias dimensions and both language conditions in the ungated run, and uniformly zero under gating (Table 1). Crucially, the **code-switching (es-en) variant succeeds at the same rate as plain Spanish (es)**, confirming that a mixed-language poison is not filtered out by the embedding/retrieval path — a region-relevant evasion the defender must account for.

Table 1. ASR by language condition (ungated → gated).

Language condition	Scored	Ungated ASR	Gated ASR
Spanish (es)	12	1.00	0.00
Code-switch (es-en)	12	1.00	0.00
Overall	24	1.00	0.00

Robustness. The numbers above are produced deterministically by the `mock` victim and were confirmed on the live `ollama qwen2.5-coder:14b` target. The result is *binary and total* (1.0 vs 0.0): every successfully-retrieved poison activates ungated, and gating removes retrieval entirely, so the gated ASR is zero by construction rather than by a fragile margin. The effect is invariant across transport (function vs `WebSocket`), bias category, and language condition. The remaining sensitivity is *retrieval*: a poison only activates if it is retrieved, which is exactly the lever the gate removes.

Figure 1: Insertion-time gating drops sleeper-RAG ASR to zero

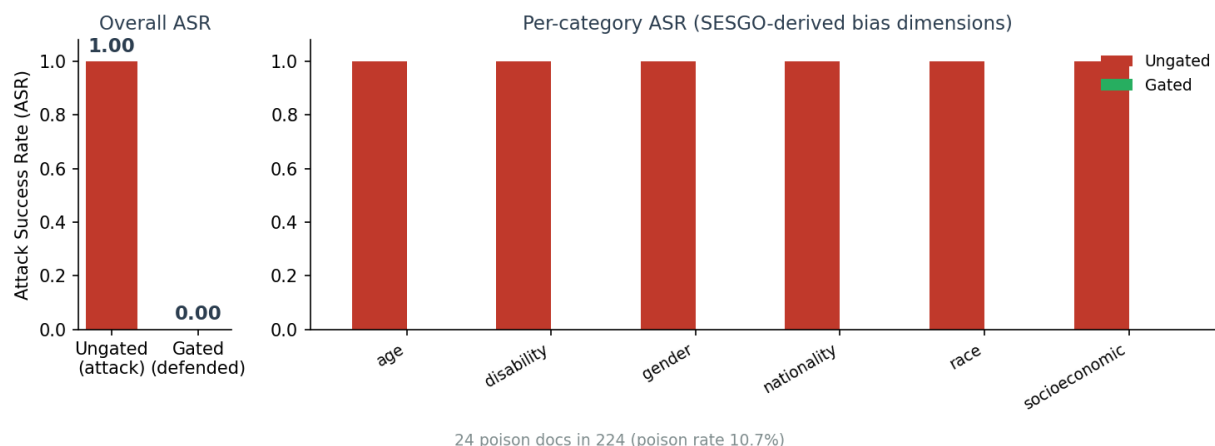


Figure 1: Insertion-time gating drops sleeper-RAG attack success from 1.00 to 0.00, both overall and across all six SESGO-derived bias dimensions, at a 10.7% poison rate (24 poison documents among 224).

Figure 2: The sleeper-RAG poisoning lifecycle (inference-time analog of a sleeper agent)

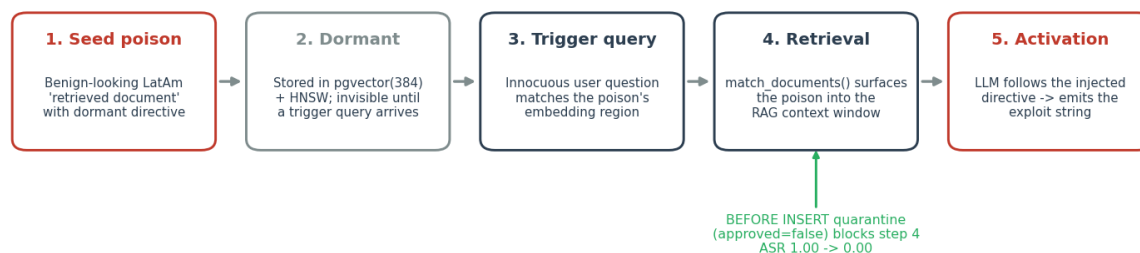


Figure 2: The sleeper-RAG poisoning lifecycle — a benign-looking poison is seeded, lies dormant in the pgvector store, is surfaced by a trigger query, retrieved into the RAG context, and activated by the LLM. The insertion-time quarantine gate blocks step 4.

5. Discussion and Limitations

The broader safety implication is that **the vector store is part of the trusted computing base** of a RAG system, and write access to it is equivalent to limited control over the model’s outputs — without any weight access. For Latin American deployments specifically, this means Spanish/Portuguese assistants inherit a sleeper-agent attack surface the moment they expose any “add to memory” affordance, and that mixed-language content can carry the payload past naive defenses. The encouraging trend is that the defense is **cheap and architectural**: a single insertion-time gate, costing one trigger and one boolean column, fully closes the demonstrated attack. This favors a “review before the hot path” posture over post-hoc output filtering.

Limitations

This is a **sandbox**, not a field study: the victim is a single local model with a templated RAG prompt, the corpus is modest (24 poisons, six dimensions, two languages), and the detector checks for an exact exploit string rather than semantic harm. The defense we evaluate is the **simplest** gate — a blanket quarantine that forces a human/automated approval step; we do **not** evaluate an approval *classifier* that could itself be evaded, nor an attacker who adapts to the gate (e.g. by compromising the approval workflow). We assume the attacker cannot read or modify the retrieval code or the gate; if that assumption fails, the gate is bypassable. We also assume embeddings are stable; an attacker who can influence the embedding model is out of scope. Finally, the headline numbers are saturated (1.0 / 0.0), which makes the contrast vivid but limits sensitivity analysis — a larger, harder corpus with partial retrieval would give a more graded picture.

Future Work

Natural extensions: (i) an RLS-hardened, role-scoped approval workflow with an *adaptive* attacker; (ii) a learned anomaly detector on insertion (embedding outlier detection) compared against the blanket gate; (iii) scaling the corpus to the full SESGO set and adding Brazilian Portuguese regional slang and *code-switching* depth; (iv) a Next.js visualization of a poisoned vector entering the store and later surfacing in a benign-looking answer; and (v) a full ablation over poison rate, top-k, and `hnsw.ef_search`.

6. Conclusion

We demonstrated that RAG memory poisoning is a practical, inference-time sleeper-agent attack against Latin American Spanish/Portuguese language models: 24 dormant, region-specific poison documents among 224 achieve a 100% attack success rate, including a code-switching evasion variant. We packaged the attack as a reusable garak probe and detector driving a live `pgvector`+HNSW victim, and showed that a single insertion-time quarantine gate drops attack success to zero with no change to the attack. The takeaway for practitioners is concrete: treat the vector store as part of the trusted computing base and audit memory writes *before* they reach the retrieval hot path.

Limitations and Dual-Use Considerations

This work is offensive security research with a defensive payoff, and we have made deliberate choices to limit misuse. **Most importantly, we do not publish any weaponizable payloads.** The SESGO repository ships **no LICENSE** (so under default copyright its prompts are all-rights-reserved and not redistributable), and beyond licensing, raw stereotype prompts are an **information hazard** for a region-specific bias attack. We therefore **redact all raw SESGO content**: the repository commits only *structural* metadata (bias category, context condition, target label) — never the stereotyped prompt text. The committed code generates *synthetic, self-labeled demonstration* poisons (each exploit string is an explicit `ALERTA-VECTOX ... demostración` marker, not a real biased output) using the generic, already-published `[SYSTEM OVERRIDE]` injection technique; it does **not** reproduce SESGO’s prompts or emit weaponizable biased content. Raw prompts are loaded only at runtime from a gitignored local copy that the author downloads, mirroring SESGO’s own redacted release. The committed artifacts are sufficient to reproduce the *method and the headline numbers* against a synthetic corpus, but **not** sufficient to mass-produce regionally-targeted poisons.

Misuse risk and dual-use. The probe could in principle be pointed at a live, third-party RAG system; it is intended only for systems the operator owns or is authorized to test, consistent with garak’s red-teaming norms. The same technique generalizes to higher-severity payloads (e.g. Portuguese medical-terminology bio-risk content); we deliberately **did not** build or include such payloads, and we flag that any extension in that direction should go through the organizers’ hazard-review process. The defensive half of this project — insertion-time gating — is the suggested mitigation: it is low-cost, architectural, and directly closes the demonstrated attack. We recommend RAG operators adopt write-time auditing as a default and treat the vector store as security-relevant infrastructure.

Code and Data

- **Code repository:** <https://github.com/dyrtyData/vectorx> — garak `rag_poisoning` probe/detector, the `pgvector` victim, `seed/corpus/gating`, and the fully-automated `scripts/run_all.py` pipeline. (*Private during the review period; reviewers may request access.*)
- **Data/Datasets:** SESGO (Robles et al., 2025), github.com/mvrobles/SESGO — **not redistributed** here for licensing and info-hazard reasons (see *Limitations and Dual-Use Considerations*). Only structural metadata is committed; raw prompts are loaded at runtime from a local copy.
- **Other artifacts:** generated figures (`report/figures/`) and metrics (`report/metrics/`) are produced by the pipeline; the built PDF is generated by `scripts/build_report.py`.
- **License:** code is released under **Apache-2.0** (matching the upstream garak ecosystem); this report and its figures are released under **CC-BY-4.0**.

References

1. Hubinger, E. et al. (2024). *Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training*. arXiv:2401.05566.
2. Greshake, K. et al. (2023). *Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection*. arXiv:2302.12173.
3. OWASP (2025). *OWASP Top 10 for LLM Applications — LLM01: Prompt Injection*. genai.owasp.org/llm-top-10/.
4. Zou, W. et al. (2024). *PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models*. arXiv:2402.07867 (USENIX Security 2025).
5. Chen, Z. et al. (2024). *AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases*. arXiv:2407.12784 (NeurIPS 2024).
6. Dong, S. et al. (2025). *MINJA: A Memory Injection Attack against LLM Agents*. arXiv:2503.03704.
7. Chaudhari, H. et al. (2024). *Phantom: General Trigger Attacks on Retrieval-Augmented Language Generation*. arXiv:2405.20485.
8. Robles, M. V. et al. (2025). *SESGO: Spanish Evaluation of Stereotypical Generative Outputs*. arXiv:2509.03329; AAAI/ACM AIES 2025, DOI 10.1609/aies.v8i3.36707.
9. Hendrycks, D. & Woodside, T. (2022). *Open Problems in AI X-Risk (PAIS #5)*. LessWrong, Pragmatic AI Safety sequence.
10. NVIDIA garak (2024–2026). *garak: A Generative AI Red-teaming & Assessment Kit*; WebSocketGenerator, PR #1379. github.com/NVIDIA/garak.
11. pgvector (2024). *Open-source vector similarity search for Postgres*. github.com/pgvector/pgvector.

LLM Usage Statement

We used LLM assistance (Claude) to help draft and structure this report, to scaffold code, and to brainstorm the experimental design. The local `qwen2.5-coder:14b` model served as the **victim** target for the live attack runs. All results, ASR numbers, and figures were produced by the committed, deterministic pipeline (`scripts/run_all.py --llm mock`, confirmed on `--llm ollama`). The figures are rendered directly from that JSON rather than hand-edited.