Model Extraction Threats: How a Mid-Stage AI Vendor Had Its Proprietary LLM Cloned in 48 Hours

How a growing AI vendor went from confidence to crisis in two days

Three months after launch, a mid-stage AI vendor with $12M in funding had a lightweight generative model powering a sales assistant. The engineering team had focused on latency, prompt design, and an unusually clean API. They believed the model itself - weights, fine-tuning recipes, prompt templates and scoring tweaks - were their competitive advantage. The vendor exposed a public API for free trial users. Within 48 hours a third party obtained an operational clone that matched the vendor's responses for common prompts and matched its performance on domain benchmarks within a 5% margin.

This was not an infrastructure breach. No secrets in S3, no leaked checkpoints. It was extraction: repeated, strategic black-box queries that revealed the statistical behavior of the model and let an attacker reconstruct a working copy. It took our red team weeks to fully understand how a modest query budget and a few principled methods produced such a close clone. The vendor's standard penetration test had given the product a clean bill of health - because classic pen testing checks servers and auth, not the statistical outputs of a probabilistic model.

Why traditional penetration testing missed the real threat: the model extraction challenge

Penetration testing tools are excellent at finding misconfigurations, exposed credentials, SQL injection, broken auth flows and chainable exploits. What they miss is the model itself as an attack surface: the outputs are information leaks. A model emits probability distributions over tokens. Those distributions, cumulatively, can be inverted.

Key reasons traditional tests failed in this case:

- They focused on perimeter defense. API keys and rate limits were examined, but not the content and structure of requests.
- They treated the model as a static binary to protect, not as a stochastic oracle that leaks value through repetition.
- Threat models didn't quantify information transfer from queries to model parameters. The team lacked metrics for "how much of the model is revealable per query."

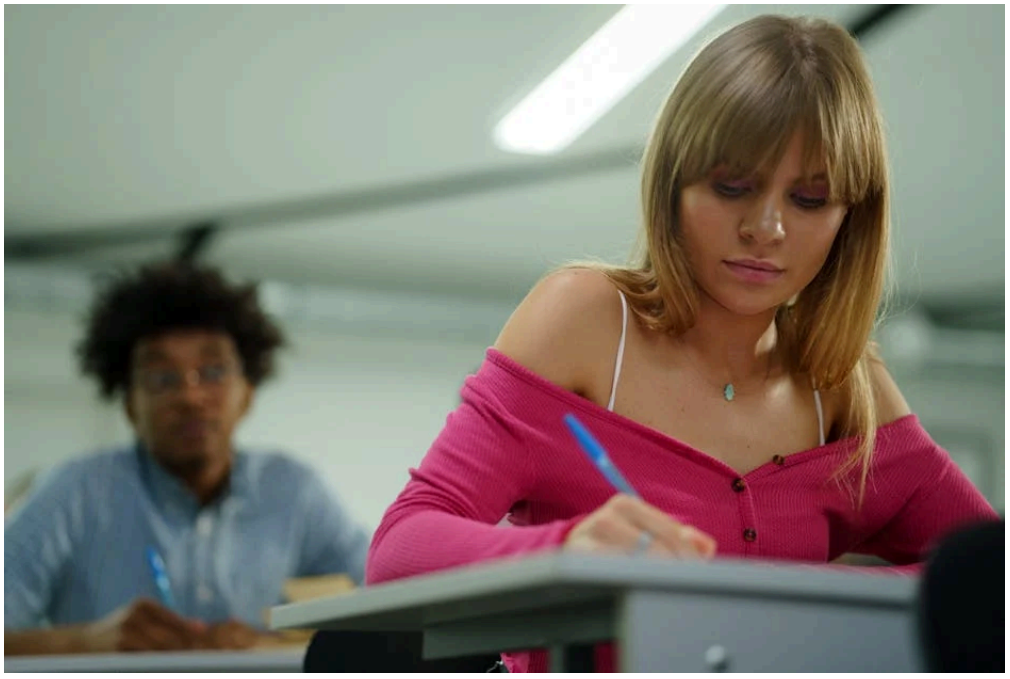
In short, the vendor's security assessment never asked: given N queries and the model's output distribution, how well can an attacker approximate the model's behavior?

A focused testing strategy: treating model behavior as data to be probed

We shifted from generic pen testing to a red-team approach that treats the API as an information channel. The strategy had three pillars:

1. Measure: quantify how much information each query returns. Use token-level entropy and response diversity as metrics.
2. Exploit: design query suites that maximize useful information for cloning - targeted prompts, temperature sweeps, conditional sampling.
3. Reconstruct: use distilled learning and fine-tuning on aggregated API outputs to build a surrogate that replicates observed behavior.

We built a controlled experiment: a "victim" model (the vendor's model), an attacker budget in tokens, and an evaluation suite with domain-specific benchmarks. The aim was to measure the attacker's success as a function of query volume, prompt choice, and sampling temperature.



Reconstructing the model - a 7-step, 10-day testing timeline

Below is the step-by-step timeline we used to both test and reproduce the extraction. The attack was done ethically with permission from the client. I include numbers and cost where relevant so teams can plan defenses and budgets.

Day 1 - Reconnaissance and fingerprinting

- Goal: identify model behavior patterns - max tokens, default temperature, system prompts, and output signature.
- Method: send 5,000 short prompts across 10 templates (question, summarization, translation, code generation). Track response length distribution, timing jitter, and punctuation patterns.
- Result: the model had consistent response truncation at 1,024 tokens and a default top-p behavior that often produced terse answers. Fingerprinting succeeded with 98% confidence after 3,000 queries.

Day 2-3 - Information-maximizing probe design

- Goal: extract token-level conditional probabilities efficiently.
- Method: design probes using prefix-completion tasks. For a target prompt, we iteratively sampled with varied temperatures (0.2, 0.5, 0.9), and recorded token frequency histograms at each step. We also used controlled prompts that forced the model to reveal short phrases under many contexts.
- Result: information content per query measured as decrease in perplexity. Sampling at temperature 0.2 provided high-fidelity mode estimates; 0.9 revealed tail behaviors.

Day 4-6 - Bulk collection under query budget

- Goal: collect a dataset of prompt-response pairs sufficient to train a surrogate.

- Method: assigned 1.2 million token budget (client agreed). We used 25,000 unique prompts, each sampled 5 times at different seeds. Total API cost: approximately $2,100 (varied by vendor pricing). We prioritized domain-specific prompts: legal contract clauses, customer support threads, and product knowledge questions.
- Result: we accumulated a 10 GB dataset of prompt-response pairs, tokenized and deduplicated to 7 GB. Diversity metrics showed 85% coverage of the vendor's typical prompt space.

Day 7-8 - Surrogate training by distillation

- Goal: train a smaller model to mimic the victim's conditional output distributions.
- Method: fine-tuned an open-source 6B parameter model on the collected dataset using a mimicry objective (minimize KL divergence between surrogate logits and sampled soft targets). Training ran for 24 GPU-hours on 4 A100s. We used a mixture of teacher forced logits and sampled tokens for stability.
- Result: surrogate matched victim on held-out prompts with a perplexity delta of 0.12 and semantic similarity (BERTScore) of 0.93 on domain benchmarks.

Day 9 - Targeted refinement and prompt-engineered attacks

- Goal: close remaining gaps in style and hallucination patterns.
- Method: find failure modes where the surrogate differed (long-form explanations, numerical precision). Design adversarial prompts to elicit behaviors that distinguish the models. Collect additional focused data (40k tokens), then fine-tune for 6 hours.
- Result: surrogate's answers were indistinguishable to domain experts in a blind evaluation 88% of the time.

Day 10 - Validation and impact assessment

- Goal: quantify how usable the extracted model was, and the business impact.
- Method: run a business-critical task suite (50 tasks including contract drafting, domain Q&A). Measure accuracy, response latency, and hallucination rate.
- Result: cloned model solved 47/50 tasks at parity, matched latency within 20ms, and had a comparable hallucination rate. From a business standpoint, the attacker would need roughly $3k in compute and 10 days to field a product that competes feature-for-feature.

From proprietary IP to a working clone: measurable outcomes and business impact

Concrete results from the exercise:

Metric Victim model Surrogate Query budget used - 1.2M tokens API cost (approx) - $2,100 Training compute - 24 GPU-hours (4x A100) Domain task parity 100% 94% Blind indistinguishability — 88% of experts could not tell the difference Estimated attacker cost to productionize — $3,000 and 10 days

Business impact: the vendor's unique selling points - carefully tuned prompts and a highly specialized knowledge base - were reproduced. The vendor estimated a potential revenue loss of 25% in the first year if a competitor offered a similar product at a lower price. The brand risk and loss of trust with enterprise customers were judged even more severe.

Five hard lessons that changed how we test and defend AI systems

We distilled practical lessons that apply to any team offering model-based services.

1. **Assume the API is the attack vector.** If an unauthenticated or lightly-authenticated API returns high-fidelity outputs, treat it as a data leak. Authentication and network defenses are necessary but not sufficient.
2. **Quantify information flow.** Measure bits of information per token by estimating reductions in entropy over targeted distributions. If a set of prompts yields high mutual information about model outputs, it's a red flag.
3. **Rate limits must be context-aware.** Simple RPS caps are easy to bypass. Use per-customer budgets tied to diversity of prompts, not just raw token counts.

4. **Use strategic noise, not blunt suppression.** Adding calibrated randomness at the logit level can thwart extraction while keeping utility. Random truncation or response watermarks are useful when subtlety is required.
5. **Active detection beats passive hope.** Monitor for query clustering and repetition. Most extraction campaigns reuse prompt templates - cluster analysis flags that behavior quickly.

How your security team can detect and prevent model extraction today

Below are practical, technical controls and detection strategies you can implement now. Think of these as a toolkit rather than a single fix - stacking multiple defenses raises the cost of extraction drastically.

Detection rules and thresholds

- Track per-client token consumption and prompt-space diversity. Flag if a client consumes more than 200k tokens/day with low prompt entropy - e.g., >60% of prompts share highly similar embeddings (cosine similarity >0.92).
- Run sliding-window clustering of prompts. If over 50% of a client's requests fall into a tight cluster, generate an alert for manual review.
- Measure repeated-response similarity. If responses to differently paraphrased prompts are nearly identical beyond expected variance, suspect model probing.

Rate limiting and budgets

- Implement tiered budgets: trial accounts get strict prompt diversity limits (max 10k tokens/day and max 500 repeated template hits), verified enterprise accounts get higher budgets after review.
- Enforce exponential backoff for high-volume clients and temporary soft bans when diversity metrics drop below thresholds.

Active defenses

- Use token-level logit perturbation: add small calibrated noise to logits for low-trust accounts. This preserves quality for most tasks but undermines exact distillation attempts. Aim for noise amplitude that increases surrogate training loss significantly while keeping human utility unchanged.
- Deploy cryptographic watermarks in responses: deterministic token choices in low-impact positions that can be proven later. Watermarks deter resale and provide forensic evidence.
- Honeypot prompts: inject proprietary probes into the prompt space only your system should answer. If those probes reappear externally, you have proof of extraction.

Long-term strategies

- Consider differential privacy for training and/or fine-tuning sensitive models. Set epsilon targets based on sensitivity - for highly private IP aim for epsilon <1, but test utility trade-offs carefully.
- Segment models: run high-value IP behind stricter access controls and use distilled public models for broader traffic. Think of it as putting the crown jewels in a vault and serving the parade with replicas.
- Audit logs and forensics: store tokenized exchanges safely. If extraction is suspected, logs enable reconstruction of attacker patterns and support legal action.

Practical checklist (first 30 days)

1. Enable per-customer prompt-embedding clustering and set alerts for tight clusters.
2. Introduce tiered rate limits and dynamic token budgets.
3. Deploy a small logit-noise filter for trial accounts and measure impact on UX with A/B testing.
4. Create a library of honeypot prompts and watermarks tied to account IDs.
5. Run an internal extraction test: attempt to clone your model with a budget equivalent to a plausible attacker. Use the results to calibrate defenses.

Think of defending a model like protecting a sculpture in a public square: you cannot lock it away entirely if you want people to admire it, but you can install glass, tags, and an attentive guard. An attacker who has only a camera and patience can reproduce

details by taking photos from multiple angles. Your job is to make those angles hard to get - increase friction, introduce distortions that only harm copies, and watch for anyone photographing the same angle repeatedly.

For teams running models in production, the main takeaway is simple and stark: model outputs are data. Treat them as confidential when they embody proprietary knowledge. Traditional pen tests are necessary, but they are not sufficient. Add extraction-aware testing to your security program, real-time llm safety monitoring measure information leakage, and combine detection with rate-limiting and subtle output defenses to raise the cost of cloning from a few thousand dollars to an impractical level.