

From Hallucination to Registration: Do the Package Names That LLMs Invent Actually Get Claimed?

Daniel Okwor

Orygn LLC daniel@orygn.tech ORCID: 0009-0005-2546-181X

Preprint, July 2026

DOI: [10.5281/zenodo.21199427](https://doi.org/10.5281/zenodo.21199427)

Abstract

Large language models routinely invent software package names that do not exist. The security concern, called *slopsquatting*, is that an attacker registers one of these hallucinated names and waits for a developer, or an AI coding agent, to install it on a model’s recommendation. Prior work measures how *often* models hallucinate. We measure what happens next: of the names models invent, how many are actually *registered* on the package index, and by whom? Using five open-weight code models we generate 6,800 samples and extract 1,641 distinct invented names that are confirmed absent from the registry; the invented-name space does not saturate. We then ask who has claimed them. Re-checking all 1,641 live, **every one is still unregistered** (a 95% upper bound below 0.2% on the fraction claimed by anyone), so the names an attacker would have to invent independently sit unclaimed. Separately, from public datasets of previously reported hallucinated names, we take 149 that have circulated long enough to attract registration; 22 are registered, and adjudicating each by true registration date and static content inspection, **zero are malicious** (a 95% upper bound of roughly 14% on the registered-name malice fraction), with 19 of the 22 traced to a single anonymous benign-placeholder operation and the other three (an organic collision, an individual reservation, and a 2019 pre-AI typosquat) unrelated to slopsquatting. We also confirm the install-to-execution primitive in a sealed offline lab using only our own benign package. The picture in this data: exposure is large and growing, but claiming is essentially absent, and the hallucinated names that do get claimed are, so far, benign rather than weaponized. We are explicit that the 149-name census is drawn from public lists that defenders preferentially scrape, and is a small registered sample. We release the code, the frozen snapshot hashes, and aggregate statistics, and we withhold the raw name list to avoid handing attackers a target menu.

1 Introduction

Code-generating language models hallucinate. When asked for working code they sometimes reference an `import` or an `npm install` target that has never existed on the registry. This is not rare. A large study by Spracklen et al. [1] catalogued 205,474 distinct hallucinated names across 16 models and 576,000 code samples, and found open-weight models hallucinate far more (21.7% of generations on average) than commercial ones (at least 5.2%). A 2026 re-evaluation of the frontier-model cohort [2] reports a narrower but still non-trivial band, 4.62% to 6.10% per generation, and a core of 127 names that every evaluated model invents identically.

The named threat that follows is *slopsquatting* [3]: an attacker registers a name that models tend to invent, publishes a malicious package under it, and waits. Because the name looks plausible,

and because an AI coding agent may install it without a human ever reading it, the payoff can be high. The primitive works. Lanyado registered the hallucinated name `huggingface-cli` and it drew more than 30,000 downloads in three months, including from a major vendor’s public repository [4]; a later hallucinated npm name spread to 237 GitHub repositories before a researcher defensively claimed it [5].

Measuring how often a model hallucinates is a measurement of *exposure*, not of *harm*. Harm requires a second step: someone has to register the name. That second step is what we measure. Of the names models invent, how many are registered on the index today, and when a hallucinated name is claimed, what is the claimant doing with it? To our knowledge, no prior work pulls the true registration dates of hallucinated names and adjudicates the intent (malicious, defensive, or incidental) of the ones that are registered.

We make three contributions:

- A self-contained pipeline that generates hallucinated names locally, filters them against a frozen, hash-pinned registry snapshot, and confirms genuine absence with live checks. It admits a name as a confirmed hallucination only if it is absent from the snapshot *and* returns a live 404.
- Two registration measurements that bracket the question. First, on the 1,641 names our own models invented (names an attacker would have to invent independently), we test how many anyone has registered. Second, on a public anchor set of previously reported hallucinated names that have circulated long enough to be claimed, we adjudicate every registered name by its true registration date and by static inspection of its contents.
- A sealed, offline lab demonstration of the install-to-execution primitive, using only our own benign package, so the mechanism is concrete without touching any public name.

The finding, stated at exactly the strength the data supports: hallucination is common and does not saturate, yet in this data the invented names are almost entirely unclaimed, and the hallucinated names that are claimed are benign, not malicious. We report explicit uncertainty on every zero and are candid about the ways each number could move.

2 Background and related work

Hallucination-rate studies. A line of work measures how frequently code models emit non-existent package names. Spracklen et al. [1] catalogued 205,474 distinct hallucinated names and reported that open-weight models hallucinate substantially more than commercial ones. A 2026 re-evaluation [2] finds lower but non-trivial rates on the current frontier cohort, along with 127 names that every evaluated model invents identically, of which 53 remained registrable even after coordinated disclosure. These studies establish exposure. They do not track who, if anyone, registers the names.

Defensive registration. A known countermeasure to typosquatting is to preemptively register the confusable name yourself with a benign placeholder. The same tactic has been applied to slopsquatting: security researchers have registered hallucinated names before attackers could [4, 5], sometimes publicizing the resulting install counts to demonstrate reach. Our data suggests this practice is more widespread, and more grassroots, than the public demonstrations imply.

Datasets. Public corpora of hallucinated names exist, assembled from research literature and from observed AI-agent traffic, for example the Trend Micro slopsquatting dataset [6] and the DepScope

corpus [7]. We use a subset of these published names as our anchor set precisely because they are already public, so measuring their registration status creates no new exposure.

3 Method

3.1 Frozen t0 oracle

We fix a freeze date, $t_0 = 2026-06-30$ (the moment our snapshot is taken), and freeze the set of package names that existed at that moment: PyPI names from the registry’s simple index (839,365 normalized names) and npm names from a public name mirror (4,194,421 lowercased names). Each list is hashed with SHA-256 and the hashes are recorded, so any later claim about “existed at t_0 ” is checkable. This snapshot is the fast, offline first filter.

3.2 Two-oracle absence gate

A generated name is admitted as a confirmed hallucination only if it passes both oracles: absent from the frozen t_0 snapshot (Oracle A, no network) *and* returning a live 404 at check time (Oracle B). Disagreement is routed, not discarded: absent at t_0 but live now is a candidate post- t_0 registration (a finding to adjudicate, not an admission); present at t_0 but 404 now is a yanked package; a network error is quarantined as unknown.

A methodological caveat drove this design. The npm name mirror we snapshot is unreliable: it lists more names than npm reports as its own total, yet still misses real packages registered years ago. We therefore treat the live 404 as ground truth and the snapshot as a pre-filter only. A direct consequence is that our confirmed count is a *lower bound*: any real hallucination whose name happens to sit in the mirror’s false inclusions would be rejected at Oracle A without ever getting a live check.

3.3 Self-generated corpus

We generate code locally with five open-weight instruction-tuned code models spanning sizes and families: `qwen2.5-coder:7b`, `deepseek-coder:6.7b`, `codellama:13b`, `codestral:22b`, and `gpt-oss:20b`, served through a local Ollama runtime. Prompts ask for practical Python and JavaScript solutions across a fixed task set at two sampling temperatures (0.7 and 1.0), with seeds fixed for run-to-run stability on our hardware. `gpt-oss:20b`, which emits a separate reasoning channel before its code, is given a larger output budget so the code is not truncated.

3.4 Extraction and name resolution

From each generation we extract imported and installed package roots. Python is parsed with the standard `ast` module; JavaScript with an `acorn`-based parser (falling back to a lenient parse for malformed output). We map import names to distribution names where they differ (for example `sklearn` to `scikit-learn`), normalize per PEP 503, exclude standard-library and built-in modules, resolve scoped and subpath specifiers, and drop tokens that fail the registry’s name-shape rules. This resolution step is essential: the import name a model writes is frequently not the distribution name that would be installed.

3.5 Anchor set, registration dates, and payload triage

Our anchor set is 149 previously published hallucinated names, drawn from public datasets [6, 7]. For every anchor name that is registered we pull its true first-registration timestamp (npm `time.created`; PyPI minimum release upload time) and compare it to `t0`. For content, we perform *static* triage only: we download the source distribution and read `setup.py`, `PKG-INFO`, and manifests as text, scanning for install-time hooks and common red-flag tokens. We never execute any third-party code. We classify each registered name by a fixed rubric into malicious (payload or obfuscated install hook), defensive placeholder (explicitly benign, no functional payload), organic (an unrelated real package colliding with a hallucinated name), or individual reservation.

3.6 Validation

The gate is validated on a labelled holdout of 17 known-real names and 8 synthetic known-fake names. It admits none of the 17 real names and all 8 fake names, so both precision and recall are 1.000 on that holdout. This is a small curated holdout; it establishes that the gate does not conflate real with fake on clear cases, not a population-level error rate. Every headline count in Section 4 was additionally re-derived by an independent recomputation, including a fresh live-404 audit of a spread sample of the discovered names and an independent re-adjudication of the registration dates.

3.7 Ethics and bright lines

This study is designed so that measurement never becomes participation. We never register a public name. We never install or execute any third-party package or run build code from any third-party distribution. Our reads of the live registries are passive metadata fetches. If a genuinely malicious post-`t0` squat were found, the plan is responsible disclosure through the registries' malware-reporting channels and an embargo on the name, not publication. We release code, configuration, and the frozen snapshot hashes, but not the raw list of confirmed hallucinated names, because that list is exactly the target menu an attacker would want.

4 Results

4.1 Volume and non-saturation

The initial 4,800 generations produced 1,186 distinct confirmed hallucinated names (absent at `t0` and live-404), of which 82 were invented by two or more of the five models. To test whether that pool was near exhaustion, we ran a further 2,000 generations with fresh seeds, balanced across all five models. These added 455 names never seen in the first run, bringing the combined corpus to **1,641 distinct confirmed hallucinated names**. The marginal yield barely moved: 0.228 new names per generation in the extra run versus 0.247 in the original. Figure 1 shows the cumulative distinct-name count as generations accumulate.

The total invented-name space does not saturate, but it has structure worth separating. Of the 455 new names, 450 were invented by a single model and only 5 by two or more. So while the total pool keeps growing, the *cross-model consensus* set, the names multiple independent models invent and the ones an attacker could most reliably bank on, is captured early and grows slowly; additional

generation mostly mines rarer, model-specific one-offs.

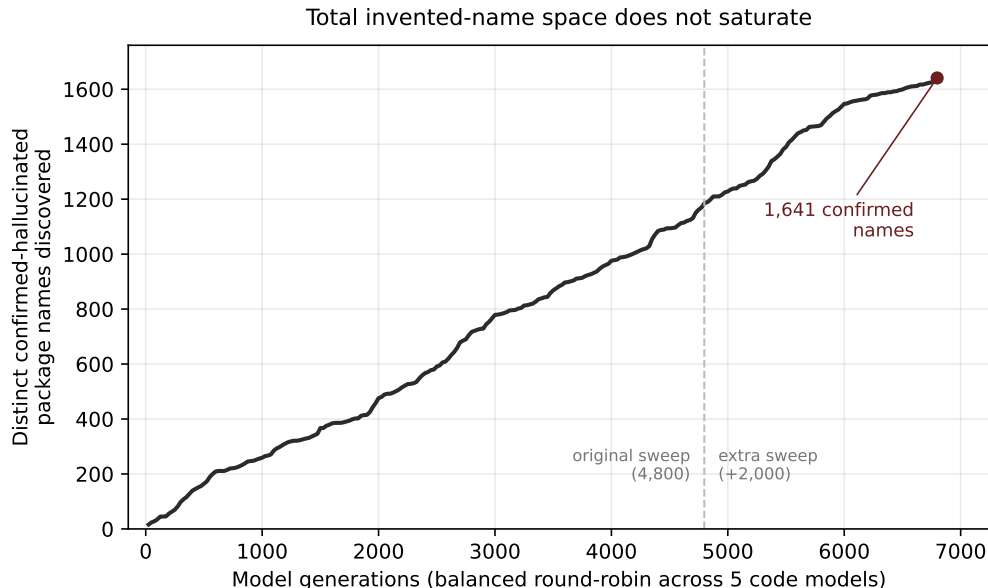


Figure 1: Cumulative count of distinct confirmed hallucinated names (absent at t_0 and live-404) as the 6,800 generations accumulate in balanced round-robin order across the five models. The dashed line marks the boundary between the original 4,800-generation sweep and the additional 2,000. The total space keeps growing; the multi-model consensus subset (not shown) saturates earlier.

4.2 The names our own models invent are unclaimed

The 1,641 names are the population an attacker would have to invent independently: they come from our own prompts, not from any public list. We re-checked all 1,641 live against the registries on 2026-07-04 (days after the $t_0 = 2026-06-30$ snapshot). **Every one is still unregistered.** Zero have been claimed by anyone, defender or attacker. With zero registrations out of 1,641, the rule-of-three 95% upper bound on the fraction claimed by anyone is below 0.2%. This is the cleanest, least biased result in the paper: among the names these models actually invent, registration by anyone is essentially absent.

4.3 Registration of publicly-circulated hallucinated names

The unclaimed result above cannot, by itself, say what happens when a hallucinated name *is* registered, because almost none of our own names are. For that we turn to names that have circulated publicly long enough to attract registration. Of the 149 anchor names, 22 are registered. Adjudicating each by true registration date and static content triage gives the breakdown in Figure 2:

- **19 defensive research placeholders**, all carrying the summary “Benign slopsquatting research package,” with no functional payload, and (Section 4.4) all from one anonymous operation.
- **1 organic real package** whose name merely collides with a hallucinated one (a legitimate developer tool from a named company), not a squat.

- **1 individual name reservation:** a different, non-anonymous individual reserving one name for their own “upcoming package.”
- **1 pre-AI typosquat** registered in 2019, predating the slopsquatting phenomenon entirely.
- **0 malicious slopsquats.**

Two honest qualifications. First, the count is zero malicious out of 22 registered (out of 149 examined); the rule-of-three 95% upper bound is roughly 14% on the registered-name malice fraction, or about 2% of all 149 examined. The data rules out a *high* malice rate in this frame, not a small one. Second, this anchor is drawn from public datasets, and defenders preemptively register names by scraping exactly such lists. The sampling frame therefore over-represents defender-claimed names by construction, so “registered names here are benign” should be read as a statement about publicly-listed names, not about all hallucinated names. The unbiased complement to it is Section 4.2: the non-public names are simply unclaimed.

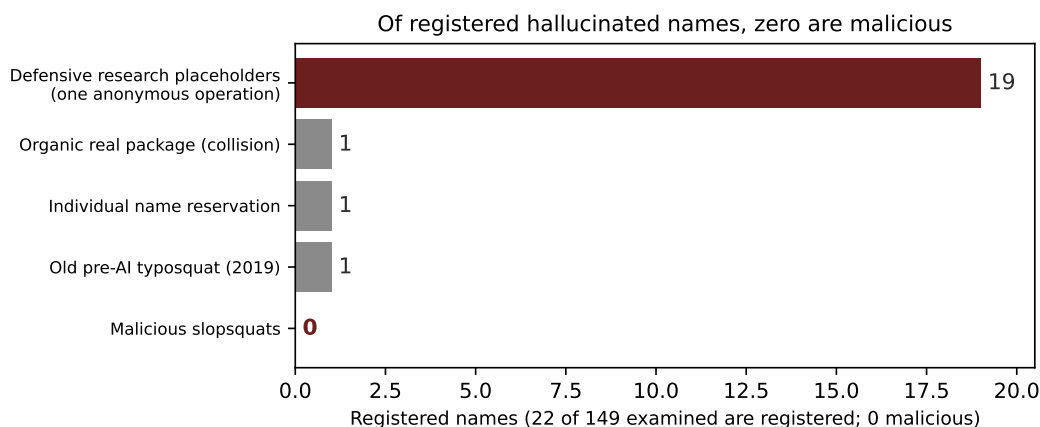


Figure 2: Registration taxonomy of the 22 registered anchor names (of 149 examined). The registered set is dominated by one benign-placeholder operation; none are malicious.

4.4 Attribution of the defensive cluster

The 19 defensive placeholders are one operation. Every one shares the identical summary string and has every other identity field, author, email, maintainer, homepage, repository, license, deliberately blank. From the registry we recover the uploading accounts, which are numbered aliases of one operator (for example `0x_xnum` and `0x_xnum1`); the numbering is consistent with working around a per-account project-creation rate limit, which also explains why the uploads arrive in bursts seconds apart. All were pushed manually with a standard upload client; none use trusted publishing, so there is no CI provenance linking them to a source repository. A public search of the handle returns nothing. The names overlap known public hallucinated-name datasets, consistent with defensively claiming names off published lists. We report the behavior and the handle only, and make no attempt to deanonymize an individual. Because this evidence lives in the withheld registry metadata (timestamps, account names, blank fields), we note it is verifiable only from that data, which we can provide to reviewers under embargo.

4.5 No fresh post-t0 squats (negative control)

Of the 2,000 extra generations, 11 produced a name that was absent from our snapshot yet live at check time. Adjudicated by registration date, all 11 predate t0 by years (2014 to 2023); they are names our unreliable npm snapshot missed, not post-t0 squats. This is a negative control over a window of only a few days, and is essentially uninformative about whether attackers will register these names going forward; that is a question for the prospective watch (Section 7), not a finding we lean on.

5 Lab demonstration of the mechanism

The registration data says almost no one is claiming these names. It says nothing about whether the underlying primitive works. We confirm that it does, in a sealed environment, using only our own benign package. We build a harmless package named `orygn-install-demo` (never published to any index) whose only effect is to print a banner and write a single marker file. We install it from local source into a throwaway directory, offline, on both ecosystems. On PyPI the code in `setup.py` executes while `pip` builds the package; on npm the `postinstall` script executes during `npm install`. In both cases our code runs before the package is imported or required (Figure 3). In a real slopsquat that same line would be attacker-controlled. The demonstration touches no public name, no network, and no third-party code.

```
PyPI  pip install

$ pip install --no-index --no-build-isolation --target ./lab ./orygn-install-demo
Building wheel for orygn-install-demo (pyproject.toml) ...

[orygn-install-demo / PyPI] BENIGN install-time code executed (pid 75488).

    In a real slopsquat, attacker code would run here, before you ever import it.

Successfully installed orygn-install-demo-0.0.1


npm  npm install

$ npm install ./orygn-install-demo --foreground-scripts
> orygn-install-demo@0.0.1 postinstall
> node postinstall.js

[orygn-install-demo / npm] BENIGN install-time code executed (pid 76340).

    In a real slopsquat, attacker code would run here, before you ever require it.

added 1 package in 909ms
```

Figure 3: Real captured output from the isolated lab. A benign package we own executes code at install time on both PyPI (via `setup.py` during the `pip` build) and npm (via the `postinstall` hook), before the package is imported. Only our own harmless code runs; nothing is published, fetched, or executed from a third party.

6 Limitations

We are explicit about what this does and does not show.

- **Zero malicious is not zero risk.** The attack primitive works, and a benign proof-of-concept under a hallucinated name has drawn tens of thousands of installs in prior work. Our finding is that, in this sample and window, the registered names are not malicious, not that a weaponized one could not spread. “Benign now” is also version-scoped: a placeholder can be weaponized in a later release under the same name.
- **The registered census is small and biased.** The taxonomy rests on 22 registered names from a 149-name anchor drawn from public lists, which defenders preferentially scrape. It is a snapshot of publicly-listed names, not a population estimate, and the intent labels were assigned by one adjudicator against a fixed rubric.
- **Lower-bound corpus.** Because the npm snapshot has false inclusions, some real hallucinations are rejected before a live check. The 1,641 is a floor.
- **Self-generated, not production traffic.** Our corpus comes from our own prompts to open-weight models. It is a controlled proxy for what agents emit in the wild, not a capture of live agent traffic.
- **Short window.** Whether these names get claimed going forward needs calendar time that a single snapshot cannot provide.

7 Future work

We keep the dated 1,641-name corpus under a prospective watch and intend to re-check it periodically to see how many names, and in which category, are claimed over time; the value grows as that forward window lengthens. The natural next step is to move from our self-generated corpus toward live coding-agent traffic, so the exposure measurement reflects what developers actually run, and to negotiate access to larger gated corpora that are not pre-filtered by public visibility.

8 Conclusion

Hallucination is common and does not saturate: five open-weight models invent new plausible package names about as fast as we can sample them. But inventing a name is exposure, not harm. When we follow the names to the registry, two things hold in this data. The names our own models invent are essentially unclaimed: all 1,641 remain unregistered by anyone. And among hallucinated names that have circulated publicly and been registered, none are malicious; the registered set is dominated by one anonymous benign-placeholder operation. The slopsquatting threat is genuine in principle, and worth defending against, but the measured registration reality is, so far, quiet. The contribution is a method and a set of numbers anyone can move: if the malicious count rises, this same pipeline will catch it.

Data and code availability

The code, prompt and pipeline configuration, and the SHA-256 hashes of the frozen t0 snapshots will be deposited in an archival repository with a DOI accompanying this preprint. To avoid providing

an attacker-ready target list, the raw set of confirmed hallucinated names is withheld; aggregate statistics and the anchor-set adjudication are reported in full, and the registry metadata behind the attribution in Section 4.4 is available to reviewers under embargo. Contact: daniel@orygn.tech.

References

- [1] J. Spracklen, R. Wijewickrama, A. H. M. N. Sakib, A. Maiti, B. Viswanath, and M. Jadliwala. *We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs*. 34th USENIX Security Symposium, 2025. arXiv:2406.10279.
- [2] A. Churilov. *The Range Shrinks, the Threat Remains: Re-evaluating LLM Package Hallucinations on the 2026 Frontier-Model Cohort*. arXiv:2605.17062, 2026.
- [3] S. Gooding. *The Rise of Slopsquatting: How AI Hallucinations Are Fueling a New Class of Supply Chain Attacks*. Socket, 2025. (Term *slopsquatting* coined by S. Larson, Python Software Foundation.) <https://socket.dev/blog/slopsquatting-how-ai-hallucinations-are-fueling-a-new-class-of-supply-chain-attacks>
- [4] B. Lanyado. *AI Package Hallucinations*. Lasso Security, 2024. <https://www.lasso.security/blog/ai-package-hallucinations>
- [5] C. Eriksen. *Slopsquatting: The AI Package Hallucination Attack Already Happening*. Aikido Security, 2026. <https://www.aikido.dev/blog/slopsquatting-ai-package-hallucination-attacks>
- [6] S. Park. *Slopsquatting: Hallucination in Coding Agents and Vibe Coding*. Trend Micro (Trend Research), 2025. Dataset: <https://github.com/trendmicro/slopsquatting>
- [7] V. Rubino. *DepScope Hallucinations Dataset*. 2026. CC-BY-NC-SA 4.0. <https://github.com/cuttalo/depscope-hallucinations-dataset>