Download Whitepaper (PDF) November 2026 update · 23 pages (2.7 MB)

Tiếng Việt English

ENGINEERING REPORT · NOVEMBER 2026 UPDATE

CONTROL · VERIFY · PROTECT

YANA 1.0 · YANA-RT 1.5.4

Yana AI: A Local Governance Layer Between AI Models and System Execution

Almost Six Months of Engineering: From an Empty Repository to Yana 1.0 and a Three-Repository Ecosystem

AUTHORS

Vũ Văn Tâm (Founder), Nguyễn Nam Long

DATE & LICENSE

November 16, 2026 · Apache 2.0 (first edition: Oct 26, 2026)

REPOSITORY

github.com/Horonomos-laps/Yana-AI

OFFICIAL PORTAL

yana.vutam.link

ABSTRACT

Modern autonomous coding agents can delete directories, force-push unvetted code to shared repositories, or exfiltrate private API keys with a single command. Natural language prompts advising an agent to *"be careful"* are inherently advisory and fundamentally insufficient to prevent catastrophic side effects. Yana AI decouples two distinct concepts: **skills** teach agents how to accomplish tasks, whereas **capabilities** grant concrete execution permissions through an isolated, host-native governance layer running entirely on the user's machine. The model proposes, Yana arbitrates, and the human operator always maintains unilateral veto authority.

This is the November 16, 2026 update of the report first published on October 26, 2026. It presents the system architecture, implementation details, and engineering trajectory across **2,633 commits** (May 17 – November 16, 2026), alongside performance measurements, safety verification methodology, and documented design boundaries. The ecosystem now spans three repositories: **Yana-AI** (the Rust runtime `yana-rt`, governance layer, CLI), **Yana-studio** (the Yana 1.0 macOS app, Studio, and IDE), and **yana-web** (website and web app). Studio sends every model request through a single runtime (`yana-rt`, latest release 1.5.4) covering 19 providers.

Keywords:

AI agent safety, capability governance, hash-chained audit logs, Model Context Protocol (MCP), Rust, Code-OSS.

1. Problem Statement

When an autonomous agent is granted terminal execution privileges, every stochastic deviation or hallucination by the model translates into an immediate, irreversible real-world action on the host machine. Two conventional defensive strategies are widely deployed, yet both exhibit structural flaws:

- **Advisory Prompt Rules:** Directives embedded within system prompts are merely token sequences that the model itself reads and can arbitrarily hallucinate away, ignore, or bypass under prompt injection.
- **Hermetic Containers & Sandboxes:** Isolating execution inside a locked virtual environment ensures safety, but severely hampers practical utility by cutting the agent off from the developer's real host toolchains, native compilers, and live project workflows.

Yana pursues an intentional third path: **keep the agent operating directly on the real host machine**, but force every action to traverse an out-of-band arbitration layer located outside model space. This layer strictly separates knowledge from authority: a `skill` is solely declarative text teaching an approach and confers zero permissions; only an explicitly declared `capability` can authorize a concrete system operation.

2. Vision and Core Principles

The project's foundational vision document (authored June 7, 2026) defines Yana as a **lean orchestration layer** mediating between human intent and AI intelligence. It explicitly demarcates what Yana is not:

- Not an agent.
- Not a chatbot.
- Not an unbounded framework for endlessly stacking agents and skills.
- Not a system requiring a large language model simply to function.

The core is dedicated exclusively to four deterministic jobs done with deep rigor: request routing, runtime safety enforcement, multi-model lifecycle dispatch, and context management. Everything else constitutes the application layer—modular and replaceable without rewriting the core.

The engineering philosophy crystallizes into seven core tenets:

1. **Human before automation:** Every critical execution requires an accountable human operator.
2. **Safety before speed:** When forced to choose, choose safety unconditionally.

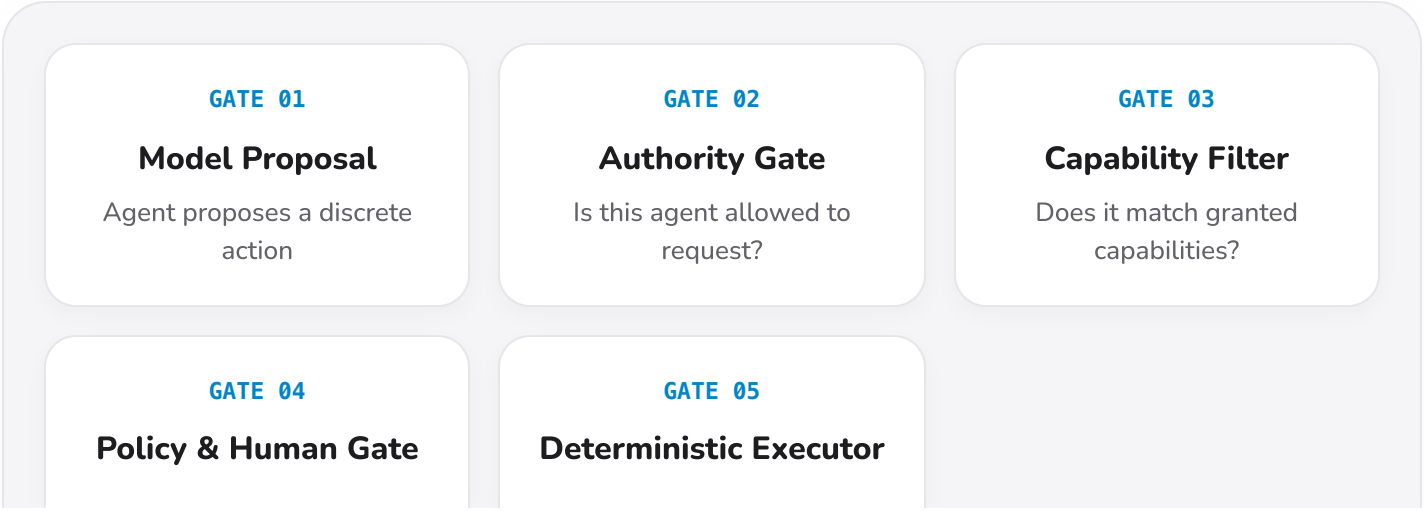
3. **Evidence before confidence:** Every substantive claim must be accompanied by cryptographic logs, structured audits, or deterministic verification.
4. **Build less, build better:** Every feature must justify its existence; anything non-essential is aggressively pruned.
5. **Designed for failure:** Every operational incident becomes a post-mortem lesson, and every lesson yields an improved architectural invariant.
6. **Open, honest, transparent:** Build in public, document mistakes, and disclose tradeoffs unvarnished.
7. **Continuous improvement:** No piece of software is ever truly finished.

Regarding end-user adoption, the project established a 12-month milestone: a non-programmer can download Yana, input an API key, and begin productive work without reading documentation. This milestone has not yet been achieved. Currently, the primary user base remains developers who already employ coding agents and require granular runtime governance.

3. Architecture and Design

3.1 The Deterministic Decision Pipeline

Every discrete operation proposed by an agent—whether originating from Claude, GPT, Gemini, DeepSeek, or an offline local model—must sequentially traverse five mandatory verification gates (Figure 1). Both allowed and rejected actions are permanently committed to an audit log for retroactive inspection.



Invariant rules, then human approval

Execute only pre-approved actions

Hash-Chained Audit Ledger: Each sequential record encapsulates the cryptographic SHA-256 hash of its predecessor. Any retroactive modification or deletion of a record is immediately detected.

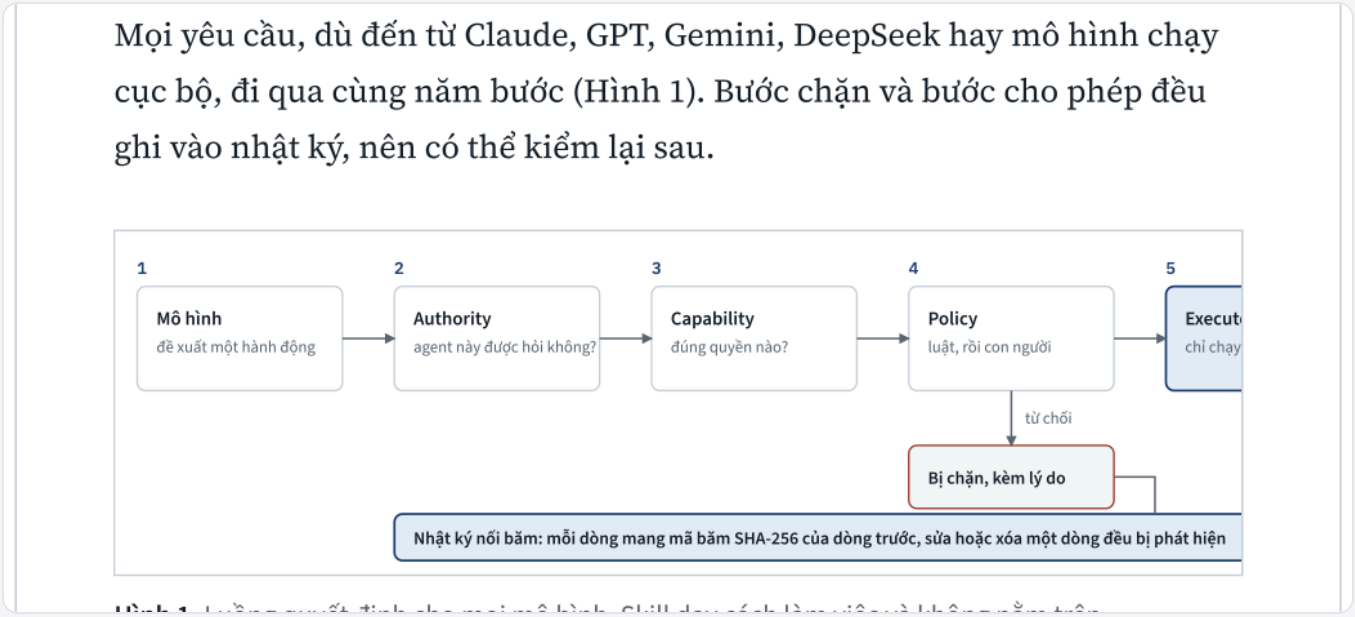


Figure 1. *Deterministic decision flow for all models. Skills educate and do not sit on the execution critical path; only guard gates and audit ledgers govern whether an action executes.*

3.2 Ecosystem Architecture

- **Agent Hosts (Adapters):** Interceptors for Claude Code (full tool-call interception at runtime), Codex (project-scoped hooks), Cursor (shell filtering), and Antigravity.
- **Three repositories (since November 2026): Yana-AI** holds the `yana-rt` runtime, the governance layer, the `yana-ai` CLI (Python), and the knowledge layer. **Yana-studio** holds the Yana macOS app (SwiftUI), Yana Studio (Electron), and Yana IDE (a Code-OSS customization). **yana-web** holds the yana.vutam.link website and the local web app (chat, missions, an AES-256-GCM encrypted API-key vault).
- **Governance (core/):** The `yana-rt` runtime (Rust) delivers 42 subcommands with no Python dependency and a native MCP server with 16 tools; `core-lock.json` pins SHA-256 hashes for

about 290 core files; in-session human approval gates; the `authority verify` command checks that every execution traces back to an allow decision; and **Giám Thị (Supervisor)**, an out-of-band daemon on OS schedulers (launchd, systemd) outside the AI process, locking execution on state divergence. Giám Thị is opt-in and is never installed without explicit consent.

- **Knowledge Layer (core/):** 100 role agents, 2,035 operational skills, 170 slash commands, 71 behavioral rules, and 66 pre/post-tool hooks (counted by `core/scripts/check_counts.py`, enforced in CI). Knowledge educates; it never confers execution authority.
- **Model Engine:** A unified router connecting 19 providers: 13 cloud APIs (Anthropic, OpenAI, Gemini, Groq, DeepSeek, OpenRouter, xAI, Novita, NVIDIA, MiniMax, GLM, Hugging Face, Kimi) and 6 local runtimes or routers (Ollama, LM Studio, llama.cpp, TurboFieldfare, AirLLM, 9Router).

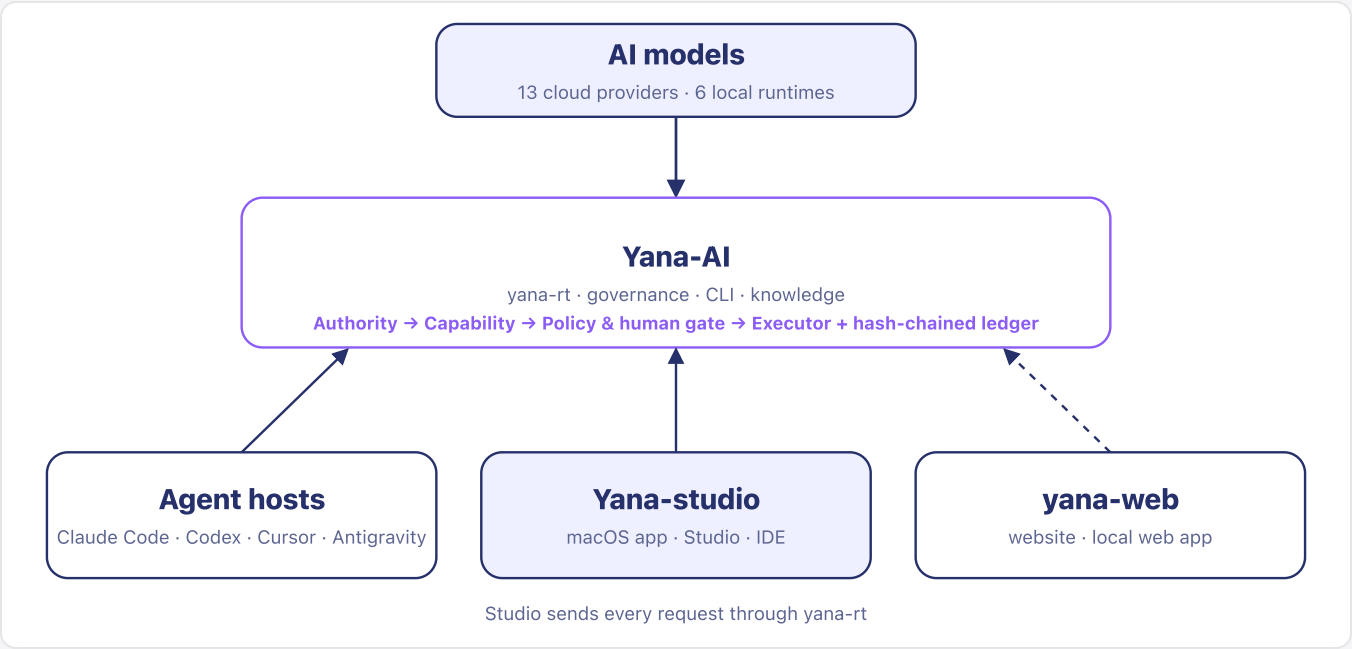


Figure 2. *The three repositories and the control core. Studio sends every model request through `yana-rt`; the website and web app live in their own repository. Dashed line: the web app has its own provider connector and does not go through `yana-rt`.*

3.3 Codebase Distribution

```
Yana-AI/
├── src/ yana-rt runtime: 328 Rust files, 83,717 lines
```

```
│   ├── capability/ permissions, shared execution engine for chat and MCP
│   ├── guard/ blocks dangerous commands, write permissions, token budget
│   ├── model/ 19 model providers, failover routing, key rotation
│   ├── session_db/ SQLite session store, full-text search, checkpoints
│   ├── os/ Yana OS: agent identity, quotas, supervision
│   └── chat/, mission/, memory/, doctor/, scanner/, … (52 top-level modules)
├── core/ governance and knowledge layers (66 hooks, 71 rules, 2,035 skills, 100 agents)
├── tools/ auxiliary tools: yana-terminal, yana-notch, yana-promo, AirLLM and pixel bridges
├── adapters/ Claude Code, Codex, Cursor, Antigravity
└── docs/, ci/ documentation and CI gates
Yana-studio/ (macOS app, Studio, IDE) and yana-web/ (website, web app) are separate reposit
```

4. Yana in Practice

4.1 Yana 1.0 and Yana Studio

Yana 1.0.0 (released November 11, 2026 from the Yana-studio repository) is the macOS Apple Silicon download, containing the native app and Yana Studio. Studio gathers into one window the things people building with AI keep switching between: opening a real folder, an editor, PTY terminals (up to 12 sessions), chat, evidence-backed tasks, and a design canvas. Studio's interface layer never calls a model provider: every request goes through `yana-rt`; API keys enter Rust through stdin and are stored encrypted in the operating system's keychain, not in project files.

The chat pane always shows the active mode. In **SUPERVISED** mode, project reads are allowed automatically, while every command execution pauses until the operator approves (Figure 3). The welcome screen switches between three languages: Vietnamese, English, Korean (Figure 4).

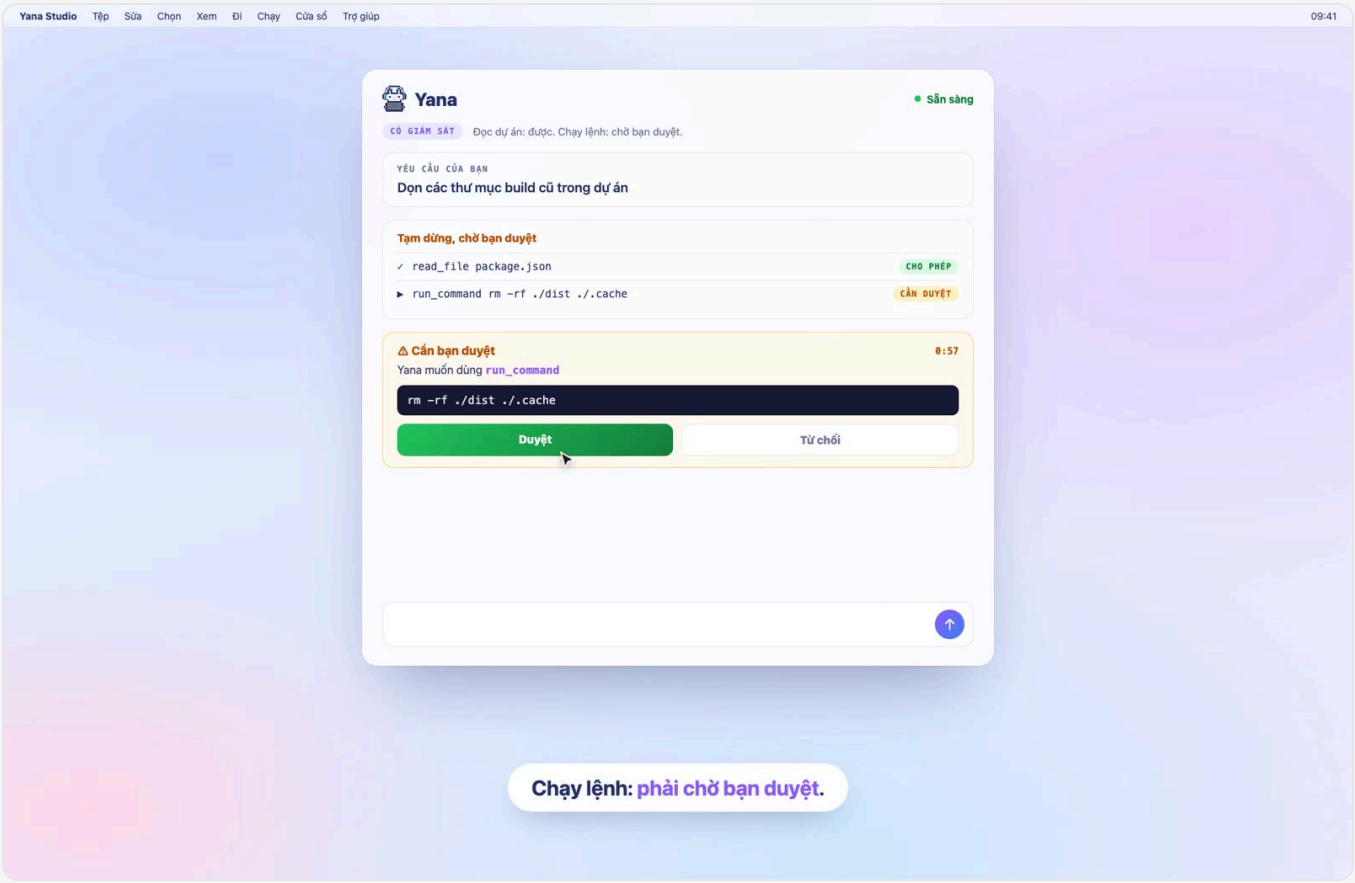


Figure 3. *The approval flow: Yana runs a read command, stops at a delete command, and waits for the operator. A pre-built illustrative scene (same source as the homepage video), not a screenshot of the real application.*

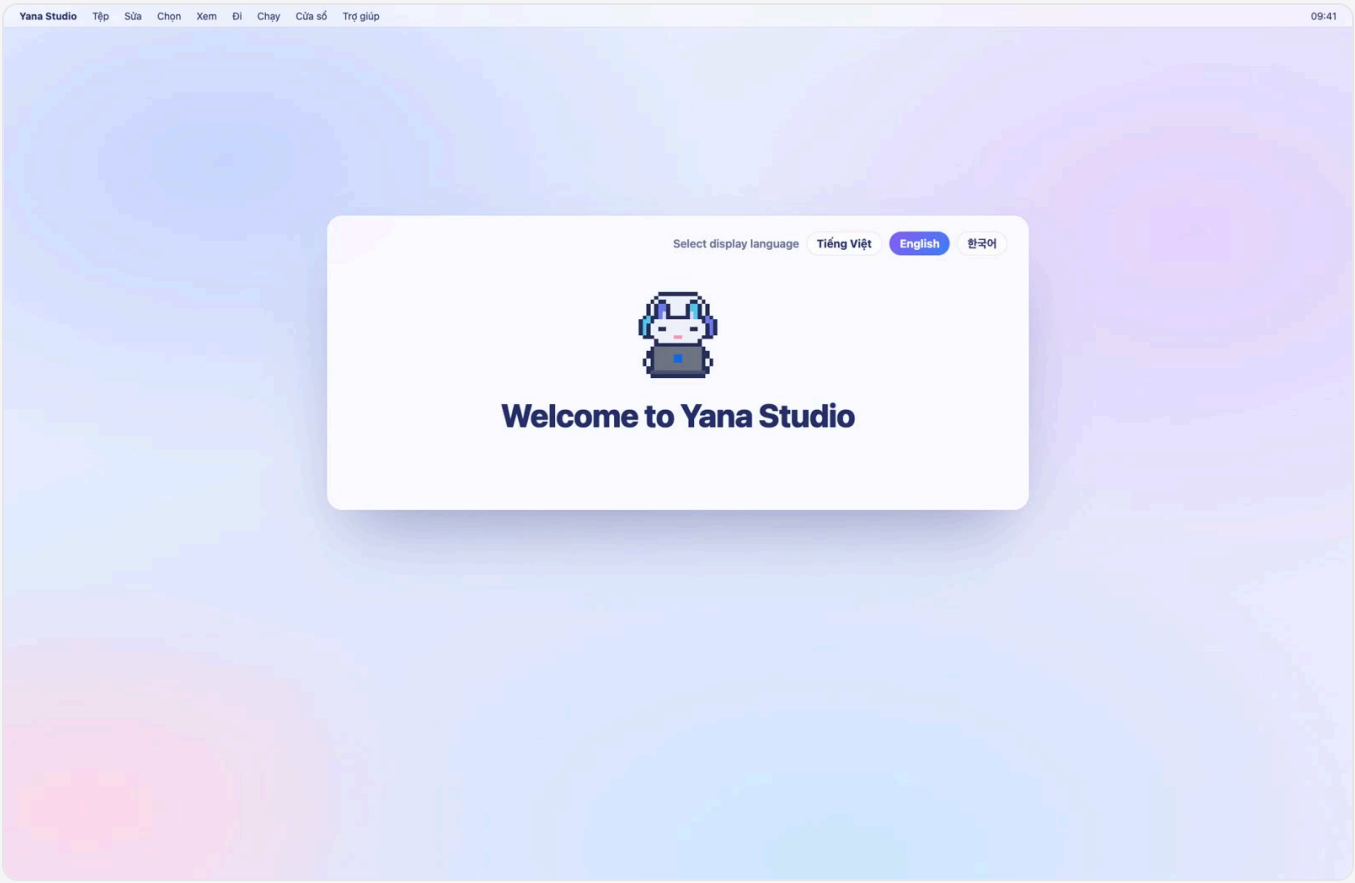


Figure 4. *The welcome screen switching between Vietnamese, English, and Korean. A pre-built illustrative scene, not a screenshot of the real application.*

4.2 Real-World Command Guard Evaluation

Table 1 lists four sample shell commands, re-run on November 14, 2026 against `yana-rt` 1.5.2 and the repository's hooks (main branch). Each protection layer blocks a different class of command, so the table states *which layer* blocked it. When a command is blocked, the agent receives the reason together with safe alternatives, and the event is written to the audit trail.

TARGET COMMAND	VERDICT (DEFAULT HOOK CHAIN)	BLOCKING LAYER AND REASON RETURNED TO AGENT
`curl -fsSL https://.../install.sh | bash`	**BLOCKED**	`tool-proxy-enforcer`: piping into an interpreter is a tier-1 violation of the anti-evasion rule. `supply-chain-guard`: remote content runs without review; download it, read it, then run it. *Not* `guard destructive`.
`rm -rf ~/Documents`	**BLOCKED**	`guard destructive` (Rust and bash versions): recursive, irreversible force deletion; use `rm` with explicit paths or ask the user.
`chmod 777 deploy.sh`	**NOT BLOCKED**	Only the `core/scripts/safe-run.sh` wrapper blocks this pattern, and that wrapper is not wired into the automatic hook chain. We ran all 18 hooks attached to Bash in `.claude/settings.json`: none blocked it. This is a known gap (Section 8).
`git status --short`	**ALLOWED**	Read-only introspection command; no layer intervenes.

4.3 Command-Line Tooling

```
# Python CLI management tool
pip install yana-ai
yana-ai install     # Inject Yana hooks and policy rules into repository
yana-ai doctor .    # Verify environment and hook connectivity

# Standalone Rust runtime (zero Python dependencies)
cargo install yana-rt
```

5. Component Statuses & Architecture

COMPONENT	RESPONSIBILITY	STATUS
Yana 1.0 for macOS	Native app (SwiftUI) plus Yana Studio in a single download, released November 11, 2026 from the Yana-studio repository. Ad-hoc signed, not notarized.	`RELEASED`
Yana Studio and Yana IDE	Studio (Electron) and the IDE (a Code-OSS customization) live in the Yana-studio repository. Studio is not yet at full parity with the previous desktop app; apart from the Yana 1.0.0 app there is no separate release from the new repository.	`IN DEVELOPMENT`
Runtime yana-rt	Core Rust engine: turn loops, capability contracts, SQLite session store, rollback checkpoints, MCP and LSP clients, `authority verify`, `status`.	`PRODUCTION`
Governance Layer	Capability manifests, human approval gates, hash-chained audit ledger, core-lock integrity verifier, command guards.	`PRODUCTION`
Giám Thị (Supervisor)	Out-of-band daemon on a 6-hour OS schedule; locks system commands if control files deviate from signed hashes. Opt-in.	`PRODUCTION (ACTIVE)`
Adapters	Claude Code, Codex, Cursor, Antigravity. Execution enforcement varies across host hooks.	`HOST-DEPENDENT`
Knowledge Layer	Role agents, skills, slash commands, and behavioral rules instructing agent conduct.	`PRODUCTION`

COMPONENT	RESPONSIBILITY	STATUS
Website and web app (yana-web)	The yana.vutam.link website and a local web app: chat with up to 6 providers, missions, an AES-256-GCM encrypted API-key vault.	ACTIVE
Yana OS	Local agent management: identities, autonomy boundaries, execution quotas, process isolation.	IN DEVELOPMENT
Wheelbot	Voice-controlled ESP32-S3 robotics platform in a dedicated repository.	EXPERIMENTAL

6. Empirical Evaluation and Benchmarks

Re-measured on November 16, 2026 for the latest release, `yana-rt 1.5.4` (release commit `c8c29fb2`, default build, 22.9 MiB), on the same **Apple Silicon MacBook Air (M4, 16 GB RAM)**, macOS 27.3, with the method of the November 15 run: every comparison is **interleaved** (Rust, Python, Rust, Python, ...) after one warm-up run, and the median is taken; the Python scripts run with `python3 -I`. The machine was not idle: the load average stayed between 5.0 and 5.2 throughout, so absolute times may be higher than on an idle machine. The Rust-to-Python ratios are less affected because both sides were interleaved.

BENCHMARK OPERATION	RUST RUNTIME (YANA-RT 1.5.4)	PYTHON SCRIPT	PERFORMANCE DELTA
Cold Binary Startup	**6.17 ms**	—	Median of 60 runs

BENCHMARK OPERATION	RUST RUNTIME (YANA-RT 1.5.4)	PYTHON SCRIPT	PERFORMANCE DELTA
System Diagnostics (`doctor`)	**152 ms**	209 ms	**1.4× faster** (12 runs)
Governance CI Audit (`ci check`)	**7.2 ms**	34.1 ms	**4.8× faster** (20 runs; both report 6 warnings)
Skills Directory Scan (`core/skills`)	**0.58 s**	8.60 s	**14.8× faster** (5 runs; Rust scanned 1,794 files, Python 1,825)
Full Repository Scan (`scan .`, defaults)	**0.29 s**	6.89 s	Not comparable: the two default scopes differ
Destructive Command Guard	**5.91 ms**	—	One allowed command, median of 40 runs
Token Budget Hook (`guard token-budget`)	**5.64 ms**	—	First call in a project with `flock-v1` active, median of 40 runs. The October run could not time it because of the missing-marker error
Peak Memory, Skills Scan	**16.3 MiB**	25.4 MiB	Full-repo scan: 20.7 vs 32.1 MiB

Compared with the November 15 run (1.5.2): the speed-up ratios are almost unchanged (`doctor` 1.4× and 1.4×, skills scan 14.8× and 14.5×, `ci check` 4.8× and 5.4×), and the binary is still 22.9 MiB. Rust absolute times are 3 to 47% higher (skills scan 0.58 vs 0.56 s, startup 6.17 vs 4.21 ms),

consistent with a busier machine. We have not re-run on an idle machine, so we cannot yet separate the machine's share from the code's.

Historical measurements from November 15, 2026, kept as recorded. Three builds measured side by side (interleaved, medians, ms) to see whether 1.5.2 got slower than 1.5.1. The middle column is 1.5.2 built with only the `cli` feature, isolating the effect of the MCP server.

OPERATION	1.5.1	1.5.2 (CLI ONLY)	1.5.2 (DEFAULT)
Startup	4.26	4.27	4.21
`doctor`	142.7	142.6	144.2
`ci check`	5.33	5.41	5.57
`core/skills` scan	538	567	572
Destructive guard	5.25	5.26	5.27
Binary size	18.2 MiB (19,055,520 bytes)	18.8 MiB (19,679,360 bytes)	22.9 MiB (23,996,800 bytes)

Conclusion and explanation. Startup, `doctor`, `ci check`, and the guard did *not* get slower between 1.5.1 and 1.5.2 (differences are within 1 to 4%). The skills scan is about 5 to 6% slower in 1.5.2, possibly from more code in the scanner; we have not isolated the cause. The largest change is **binary size**: the default 1.5.2 build is 4.7 MiB larger, of which about 4.1 MiB comes from the MCP server and client (which pull in the async runtime `tokio`) becoming part of the default build in October, while the new 1.5.2 code itself adds only about 0.6 MiB. In other words, the redesign made the binary larger but did not slow the hot paths.

Why the first measurement differed. The first run on November 14 ran the measurements back to back while the machine was also running several other applications, and gave 207 ms for `doctor`

and 4.93 ms for startup; interleaved, those return to 139 and 4.21 ms. The `doctor` ratio against Python (1.4× instead of 2.9× in the October 20 measurement) changed mainly because the Python side got faster (195 ms against 415 ms); the October run averaged 3 runs without a warm-up and may have included a cold run, but we cannot confirm that. This is still one machine, with run counts in the tables.

7. Security Verification and Adversarial Auditing

In late October 2026, the native MCP server was subjected to stdio fuzz testing across its 14 built-in tools. Path traversal exploits escaping workspace bounds (`/etc/hosts`, malicious symlinks) were successfully blocked, and `browser_fetch` strictly required explicit leases.

The audit revealed one functional defect (`list_processes` failure on macOS) and a **significant vulnerability**: the legacy `check_command` function focused exclusively on direct destructive primitives, permitting pipe-to-shell constructs (`curl ... | bash`), base64 decoding pipelines, and `chmod 777` to bypass rules despite explicit prohibitions in project rules.

An updated patch parses commands according to POSIX shell grammar (evaluating quotes, command substitution, heredocs, pipes, and subshell groups) prior to rule matching, validated by **1,642 passing unit tests**. The patch underwent four independent adversarial audit cycles executed by AI subagents (six review passes total). **Key takeaway:** String-based command blacklists are inherently non-exhaustive; pattern guards must be treated as defense-in-depth, while absolute protection rests upon the *human approval gate* and *granular capability bounds*.

From October 26 to November 16, 2026, the safety-relevant changes (from pull request history and `CHANGELOG.md`):

- `yana-rt authority verify` checks that every execution traces back to an allow decision (#534). `yana-rt status` gives one read-only snapshot shared by every surface, with a matching read-only MCP tool (#533, #535).
- MCP: an allowed-tools list (`allowed_tools`), pinned tool definitions, and per-tool safety hints (#532). The server now exposes 16 tools; a test asserts that exact number so a new tool has to be considered deliberately.

- Version 1.5.2: `list_processes` works on macOS, an end-to-end MCP test was added, and 13 RUSTSEC advisories that turned CI red were cleared by dependency updates; a `quinn-udp` patch for RUSTSEC-2026-0538 followed (#562).
- The `flock-v1` lock is available on request on a fresh checkout (#541); pdfium is pinned by SHA-256 in the release workflow (#542); `verify_action_pins.py` compares each GitHub Action SHA with its tag.
- **Version 1.5.4 (Nov 16) fixes a permanent session lockout (#569).** On a fresh clone (a cloud session, for example) without `yana-rt` on PATH, the token budget hook refused *every* tool call because the `flock-v1` marker was missing, including a call that could create it, so the session stayed locked until it was abandoned. #541 only added an opt-in and a hint and did not remove the block. The hook now matches the Rust path: it skips that call's budget check and only warns, while the maintenance gate and a mismatched marker still block. Version 1.5.3 was prepared and tested but **not released**, because that testing found this bug.
- CI runs daily on main so toolchain drift shows up before a pull request does (#557), plus a weekly health check (#566).
- On November 14, 2026 the GitHub API reported **0 open Dependabot alerts** on all three repositories (checked live).

The repository now has **1,852 Rust test functions** (a `grep` count of `#[test]` attributes under `src` and `tests`). Before releasing 1.5.4 we ran, on the benchmark machine: `cargo test --bin yana-rt` **1,702 passed** (1 ignored), the `integration_runtime` suite **67 passed**, the hook tests 384 passed, and the Python tests 1,031 passed; main's CI was also green on the exact release commit. Other Rust test targets were not run separately this time, so 1,852 remains a count.

8. Documented Boundaries and Limitations

- **Ledger Deletion:** Hash-chaining cryptographically guarantees tamper-detection against in-place record manipulation, but cannot prevent wholesale deletion of the log file without OS-level immutable write privileges.
- **Core-Lock Invariant:** Proves that the file tree matches the registered manifest, but does not verify whether an operator personally reviewed the diff. Anyone with write access to `core/` can

regenerate the lock. Giám Thị provides independent scheduled verification, but operates with up to a 6-hour window.

- **Rule Enforcement Scope:** The majority of files in `core/rules` describe guidelines that agents self-enforce; only rules backed by active system hooks are deterministically blocked. A formal cryptographic code-signing infrastructure is not yet implemented.
- **Pattern Guard Coverage:** Destructive command filters rely on structural string parsing, not sandbox hypervisors. The fallback Bash script implementation does not yet incorporate pipe-to-shell blocking parity.
- **Host Interception Discrepancies:** Cursor does not intercept MCP tool calls. Antigravity filters shell invocations, but can be bypassed when executed with `--dangerously-skip-permissions`.
- **Platform Notarization:** macOS releases currently rely on ad-hoc developer code signatures prior to Apple Developer account notarization, triggering standard Gatekeeper warnings on first launch.
- **The `chmod 777` gap:** The November 14 re-run showed `chmod 777` is not blocked in the default hook chain; only the `core/scripts/safe-run.sh` wrapper blocks it, and that wrapper is not wired in automatically. The first edition of this report listed the command as blocked.
- **Studio is not at full parity:** There is no device registry, agent sessions screen, mobile pairing, SSH or Docker execution, permission-lease UI, or signed auto-update yet. Apart from the macOS Yana 1.0.0 app, there is no separate Studio or IDE release from the Yana-studio repository.
- **The repository split just finished:** The website, web app, and desktop app moved to two separate repositories on November 11–13, 2026, so older links and documents that mention `tools/yana-web` or `tools/yana-studio` point at code that has moved.
- **Measurements are limited:** The benchmarks ran on one machine with a few repetitions, while the machine was not fully idle (the load is stated in Section 6). The test results in Section 7 cover only the suites named there, not every test target.

9. Engineering Trajectory & Timeline

The Git repository began with an empty commit on **May 17, 2026**. By November 14, 2026, the Yana-AI main history recorded **2,627 commits**, **495 merged pull requests** (550 opened, as of

November 15), and **75 tags**; the Yana-AI Releases page lists **26 releases**, and the Yana-studio repository adds the Yana 1.0.0 release. Figure 5 shows commit velocity by month, peaking at 740 commits in June during the bulk ingestion of operational skills and the web portal launch. The first edition of this report stated 41 releases; the current count is lower and we have not established the cause of the difference. As of November 16, after 1.5.4: **2,633 commits**, **496 merged pull requests** (551 opened), **77 tags**, and **27 releases** on the Yana-AI Releases page.

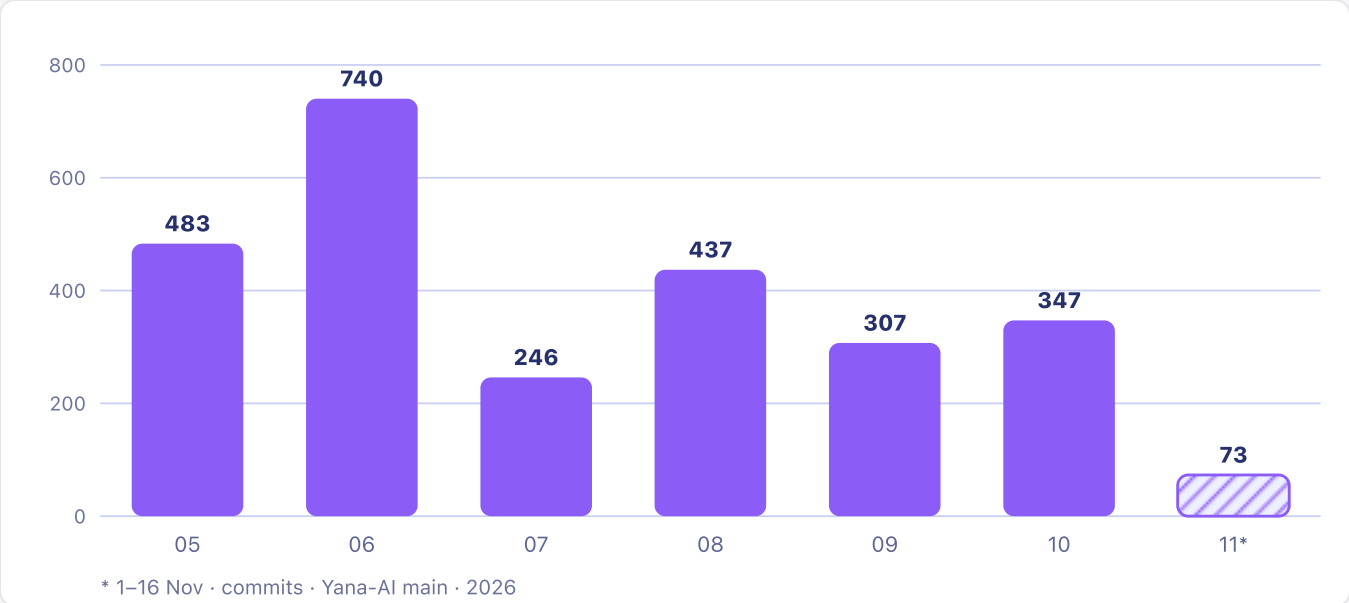


Figure 5. *Monthly commit cadence from May 17 to November 16, 2026 (November covers 1–16 only, 73 commits). Source: Yana-AI repository Git history, main branch.*

May 17–31, 2026

Foundation

Truth-gate preventing false agent completions; two-tier memory (L1/L2); hash-chained audit ledger; OWASP LLM Top 10 guardrails; 70+ defensive rules; MCP integration; runtime migration to Rust; Apache 2.0.

June 2026

Expansion & Interface

Large-scale skill ingestion; tri-lingual web portal; 1-click GitHub App approved on GitHub Marketplace; multi-model chat; AES-256-GCM client-side key encryption; parallel agents.

July 2026

Hardening & Controls

Release of Giám Thị supervisor (July 16); gated CLI tool execution; automated malware scanning; `/freeze` scope lock; Codex adapter; Kimi provider; patched remote execution script defect.

August 2026

Yana OS & Actions

Unified Rust workspace; Yana OS (identities, autonomy tiers, durable action queues); capability leases; decision receipts; remote approval protocol; Discord/AirLLM adapters; ESP32 voice bridge.

September 2026

Studio & Model Routing

Yana Studio Electron (v1.5–1.6 across macOS, Windows, Linux); automatic failover; API key rotation; prompt caching; SQLite session store with checkpoint rollback; WebGL landing page.

October 2026

Unification & Delivery

Native MCP client and trust-store; read-only LSP client; plan mode with diff previews; universal office document suite; Yana Studio IDE 1.7 released (Oct 16); Antigravity hook integration.

November 2026

Yana 1.0 & Repository Split (Nov 1–16)

yana-rt and yana-ai 1.5.2 (changelog dated Nov 3; the `rt-v1.5.2` release was posted on GitHub on Nov 12): `harness audit`, chat with approval preview and plan mode, a `doctor` hint to enable Giám Thị. `authority verify`, `status`, and MCP security upgrades (Nov 12). **Yana 1.0.0** for macOS, the native app and Studio in one download, released Nov 11. The ecosystem split into three repositories under the `Horonomos-laps` organization (Nov 11–13): the website and web app moved to yana-web, the macOS app, Studio, and IDE to Yana-studio; Yana Desktop (Electron) was removed from Yana-AI on Nov 13. The website was redesigned on yana-web (new logo, four rebuilt demo videos, a 3D MacBook page). Nov 16: **yana-rt and yana-ai 1.5.4** on

crates.io and PyPI, with a GitHub Release, including the fresh-clone session lockout fix; 1.5.3 was withdrawn before release.

10. Founder's Note

> *"Yana AI was not born inside a corporate campus. It possessed no venture backing, no enterprise engineering brigade, and no institutional laboratory. It began on a personal laptop, driven by a 17-year-old student who believed coding agents must not be granted unchecked sovereignty over host systems.*
>
> *The most insidious pitfalls were never individual code bugs, but flawed conceptual assumptions. Following each failure, Yana was deliberately re-engineered to be more verifiable, more observable, and radically more dependable.*
>
> ***Yana AI exists because humans must forever retain the final decision."***
>
> **— Vũ Văn Tâm**
>
> Founder & Lead Engineer, Yana AI

The author initiated the project at age 17, operating as sole engineer for the first six months without institutional funding. AI tooling assisted in accelerating implementation, test generation, and review cycles, while all architectural design, system rewrites, and final accountability rested with the maintainer.

In late October 2026, **Nguyễn Nam Long** joined the project as a new contributor. The engineering milestones and technical work documented in this report represent development completed prior to his onboarding.

11. References

Corrections relative to the October 26, 2026 edition:

- Agent count: the first edition said 60 (the number of top-level entries in `core/agents`); the correct figure per `check_counts.py` is 100.
- Section 6 was re-measured on November 16 for `yana-rt` 1.5.4 (main table) with the same interleaved method; the November 15 measurements for 1.5.1 and 1.5.2 are kept below it.
- Section 6 was fully re-measured on November 15 with an interleaved method: skills scan 14.5× (the first edition said 15.4×), `doctor` 1.4× (the first edition said 2.9×), binary 22.9 MiB (the first edition said 18.2 MiB). Rust scanned 1,791 files and Python 1,795 files, not the entire skill set.
- Command guard table: now records which layer blocked each command, from the November 14 run; `chmod 777` is not blocked in the default hook chain.
- MCP tool count: 16 (the first edition said 14, the count at the time of fuzzing).
- Release count: see Section 9; the current count differs from the first edition and the cause is not established.

1. OWASP Foundation. *Top 10 for Large Language Model Applications*, owasp.org.
2. Google. *Trillian: A Transparent, Highly Scalable and Cryptographically Verifiable Data Store*, github.com/google/trillian.
3. Anthropic. *Model Context Protocol Specification*, modelcontextprotocol.io.
4. Microsoft. *Visual Studio Code, Open Source (Code-OSS)*, github.com/microsoft/vscode.
5. Yana AI Repository Specifications (`VISION.md` dated June 7, 2026; `PHILOSOPHY.md`; `JOURNEY.md`; `HISTORY.md`; `BENCHMARK.md`; `CURRENT-MILESTONE.md`; `docs/reference/known-limitations.md`; `CHANGELOG.md`; `VERSIONING.md`), github.com/Horonomos-laps/Yana-AI.

Yana

AI works. You hold authority. Intelligence may change, authority endures.

Products

Yana Studio

Yana Wheelbot

Download & CLI

Updates

CHANGELOG.md ↗

CURRENT-MILESTONE.md ↗

Docs & Technical

Technical Whitepaper

ARCHITECTURE.md ↗

PHILOSOPHY.md ↗

ROADMAP.md ↗

BENCHMARK.md ↗

Resources

Support & Contact

CONTRIBUTING.md ↗

SECURITY.md ↗

Acknowledgements ↗

Lineage ↗

Legal

Legal Notices & Copyright

Apache 2.0 License ↗

GitHub Repository ↗

vutam.link ↗

© 2026 Yana AI · Vũ Văn Tâm · Open-source components remain under their respective licenses.

Terms of use for this site

- **Copyright and license.** Yana's source code is under the Apache 2.0 license. The content and interface of this site are © 2026 Yana. Open-source components keep their respective licenses.
- **Trademarks.** The Yana name, logo and identity are not covered by the source-code license (see Apache 2.0, section 6). Do not use them in a way that suggests your product is Yana, or is endorsed by Yana, without written permission.
- **Illustrative examples.** The screens, commands and approval boxes on this site run in the browser for illustration; no command runs on your machine.
- **Apple trademarks.** Apple, iMac, MacBook and macOS are trademarks of Apple Inc., registered in the U.S. and other countries. The iMac and MacBook Pro frames come from Apple Design Resources. Yana is not affiliated with or endorsed by Apple.
- **Images.** The wallpaper on the device screens is an iMac wallpaper copyrighted by Apple Inc.

- **Fonts.** Nunito, Plus Jakarta Sans, Patrick Hand, Caveat and JetBrains Mono from Google Fonts, under the SIL Open Font License.

Read in full: Legal notices Privacy Acknowledgements