Engineering and Auditing Controlled Egress in an AI-Integrated Development Environment:

A Cold-Start Case Study of Xynapse IDE

Dzhabrail Khalilov

National Research University Higher School of Economics (HSE University),
11 Pokrovsky Bulvar, Moscow 109028, Russian Federation
Present status: Independent Researcher, Moscow, Russian Federation
jabrailkhalil@gmail.com

This work originated in the author's 2026 master's thesis at HSE University.

Manuscript prepared for external peer review.
August 14, 2026

Abstract

AI-enabled integrated development environments (IDEs) combine user-initiated model requests with background connections from platforms, extensions, experiments, updates, synchronization, recommendations, and analytics. This mixture makes claims about pre-interaction data transmission difficult to test. We present Xynapse IDE, a Code-OSS/Electron distribution with an adapted Continue runtime, YandexGPT and GigaChat adapters, and a controlled-egress architecture separating platform diagnostics, assistant analytics, service infrastructure, model calls, and other explicit operations. We construct a reproducible assurance case for public source commit `5f01a9fb0fdebc2a40c7843552c18aad189a0453` and an identified Windows executable. A deterministic static audit confirmed 19/19 declared controls and recorded five residual online-capability boundaries: update actions, a GitHub update source, an external webview template, a compatibility telemetry default, and assistant startup activation. A dynamic audit used Sysmon 15.21 for three independent 60-second cold starts with fresh user-data and extension directories, no workspace, and no interaction. Xynapse 1.108.0 produced zero process-attributed TCP/UDP events while remaining present in all 49 liveness samples per run. A positive control produced two events through the same filter. The evidence supports a narrow result: no Sysmon Event ID 3 activity attributable to the tested Xynapse image was observed during the tested cold starts. It does not cover model calls, Git, extensions, webviews, updates, DNS, differently named helpers, or packet contents. The contribution is a source-to-binary workflow combining machine-checkable predicates, bundle verification, runtime observation, positive control, and explicit scope boundaries.

Keywords: AI-assisted software engineering; integrated development environment; telemetry; privacy; controlled egress; software assurance; reproducibility.

1 Introduction

An AI coding assistant embedded in an IDE may access substantially more than the text entered into a chat box. Depending on the interaction, it may process selected source code, repository structure, open files, compiler diagnostics, Git differences, terminal output, and editing history. Network behavior is therefore part of the security boundary of an AI-enabled development environment, not merely a user-interface preference. Even when remote model inference is a required function, a developer should be able to distinguish a deliberate model request from background diagnostics, experiments, synchronization, recommendations, updates, and extension analytics.

The extensibility of modern editors makes this distinction difficult. The official Visual Studio Code documentation states that setting `telemetry.telemetryLevel` to `off` disables product telemetry, but extensions may implement independent telemetry and do not necessarily follow that setting [4]. Consequently, a single product flag is not sufficient evidence for a distribution that bundles an AI extension. At least three surfaces must be considered: the platform runtime, the embedded assistant, and the behavior of the released executable.

Prior work has shown several distinct risks around AI coding and generative assistants. Code-generation models may reproduce sensitive information from their training data [12]; encrypted response traffic may leak information through observable structure [13]; and some generative-AI extensions transmit contextual data or identifiers [14]. AI assistance also does not guarantee secure output and may affect user confidence in insecure code [15]. These results do not imply that every cloud-assisted workflow is unacceptable. They motivate an architecture in which background and functional channels are separated and testable.

Xynapse IDE was initially designed in the author's master's thesis as a Code-OSS/Electron distribution with an integrated assistant adapted from the open-source Continue project and with YandexGPT and GigaChat support [1]. The thesis proposed an autonomous-start smoke test, but its included protocol was a template and contained neither a completed environment record nor raw observations. The present study turns that product requirement into a falsifiable statement tied to one public source commit and one executable.

This work asks four research questions:

RQ1

Which potential network channels belong to the platform, assistant, service infrastructure, model

providers, and explicit user operations, and which should be inactive before a user action?

RQ2
Does the audited source snapshot implement the declared controls over platform telemetry, assistant analytics, experiments, synchronization, and provider activation?

RQ3
Does a fixed 60-second cold start on a fresh profile produce process-attributed TCP/UDP events for the tested executable?

RQ4
Which residual online capabilities remain outside that result and must be exposed to avoid overclaiming?

The contributions are: (1) a controlled-egress model for an extensible AI IDE; (2) a machine-readable source audit that treats positive controls and scope boundaries as different evidence categories; (3) a source-to-binary check of the embedded assistant's no-op analytics path; (4) a repeatable Windows cold-start experiment with process liveness and a positive instrument control; and (5) a release-oriented evidence chain from requirement to source predicate, bundle, executable hash, and runtime event.

Result stated at its supported scope

For public commit `5f01a9fb0fdebc2a40c7843552c18aad189a0453` and the tested Windows build, the static audit confirmed 19 target predicates. The dynamic audit recorded zero Sysmon Event ID 3 events in each of three cold starts; the positive control recorded two. This supports only the scenario-specific statement that the tested Xynapse process image created no observed TCP/UDP connection during those three pre-interaction windows. It is not a certificate of complete network isolation.

2 Background and Related Work

2.1 Telemetry in an extensible IDE

Code-OSS uses a multiprocess architecture combining Electron main and renderer processes, a shared process, and a separate Extension Host [3]. This separation improves resilience and extensibility, but it complicates network auditing: a connection may originate from a shared service, a built-in extension, a webview, a language server, or a child process rather than from the visible editor window.

The open Code-OSS repository and the Visual Studio Code product distribution are not identical. Microsoft describes Code-OSS as the open-source basis from which its separately configured product is built [5]. A downstream distribution can therefore modify product configuration and service implementations, but it also inherits a complex execution model and must validate the properties of its own build.

Visual Studio Code distinguishes crash reports, error telemetry, and usage data, links experimental rollouts to usage telemetry, and separately warns about extension-specific telemetry [4]. For a bundled AI assistant, defense in depth requires alignment among the product capability, default user setting, runtime service implementation, and assistant runtime. If one layer remains independently active, a broad claim that telemetry is disabled is ambiguous.

2.2 AI assistants as context-transmission channels

An AI assistant differs from a conventional diagnostic client in the breadth of context potentially available to it. Two questions must be separated: which context is assembled for a functional request, and which information is transmitted in addition to that request. The first calls for context-selection interfaces and provider policy; the second calls for control of background channels.

CodexLeaks demonstrates privacy leakage risks in code-generation models [12]. Perry et al. show that access to AI assistance does not guarantee secure code and can increase confidence in insecure output [15]. These are model- and use-level risks rather than evidence about cold-start traffic. Their relevance here is to delimit the claim: suppressing background telemetry does not make model output secure and does not remove the need for code review.

Network-oriented studies of generative-AI assistants have found automatic transmission of page content, form values, prompts, or identifiers by some extensions [14]. A browser extension and an IDE Extension Host are different execution environments, but the methodological conclusion transfers: configuration inspection should be complemented by runtime observation, and a zero-event observation should be complemented by a positive instrument control.

2.3 Positioning

This is not a complete security audit of an AI IDE. It does not evaluate generated-code quality, prompt injection, dependency-chain integrity, Electron vulnerabilities, provider retention policy, API-key protection in memory, or developer productivity. It studies one property: control over process- attributed outgoing network activity before explicit user interaction.

The method resembles an assurance case: a general claim is decomposed into static and dynamic evidence. Reproducible research practice calls for preserving code, configuration, environment, and the links between results [16]. The supplement therefore contains the Sysmon configuration, launch and validation scripts, machine-readable results, source-audit output, and checksums rather than only an aggregate table.

3 Xynapse Architecture

3.1 Upstream foundations and author contribution

Xynapse is a separately built distribution based on Code-OSS and Electron. The editor, Workbench, Extension Host API, integrated terminal, and core multiprocess model originate upstream. Xynapse Assistant is based on the open-source Continue runtime [6]. In this paper, “engineering an IDE” refers to product architecture, integration, modification, build, and delivery of a fork; it does not claim original authorship of Code-OSS, Visual Studio Code, Electron, or Continue.

The Xynapse-specific engineering layer examined here comprises product configuration; replacement of platform telemetry implementations with null objects; removal of executable Settings Sync contributions; changes to experiment and recommendation defaults; assistant adaptation and packaging; neutralization of the assistant’s shared telemetry facade; YandexGPT and GigaChat integration; a portable profile; and build and verification scripts. The research contribution is the testable connection between these changes and observed cold-start behavior.

The public repository snapshot [2] is pinned to commit `5f01a9fb0fdebc2a40c7843552c18aad189a0453`. The static report includes SHA-256 values for platform, assistant, build, and packaged-bundle files, allowing later audits to detect content changes even if paths remain stable.

3.2 Execution layers

The architecture has four logical layers:

1. the editor platform, including Electron main and renderer processes, the shared process, Workbench, file services, and terminal services;
2. the Extension Host, which runs Xynapse Assistant, commands, context providers, and model adapters;
3. the webview interface for chat, model selection, editing, and result presentation; and

4. a local YAML configuration containing models, roles, prompts, context providers, and credential placeholders.

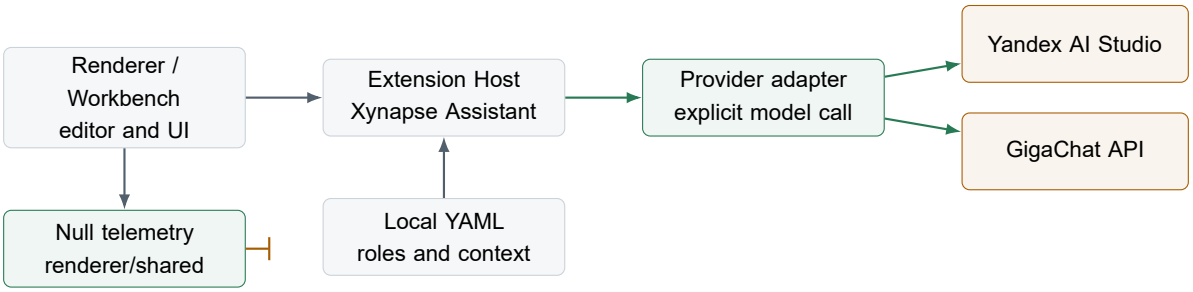


Figure 1: Logical boundary between local runtime services and explicitly initiated model endpoints. The diagram does not imply physical isolation of all online functions.

YandexGPT and GigaChat are implemented as provider adapters and registered in the common LLM index. Upper layers use a shared contract while authentication and request formatting remain encapsulated in the adapters. The Yandex AI Studio documentation describes REST/gRPC and OpenAI-compatible interfaces [9, 10]; GigaChat documents a separate authorization stage and generation API [11]. These are explicit remote channels. Their existence is a functional capability, not evidence of a startup connection and not a provider privacy certification.

3.3 Why a distribution fork is required

An extension can control its own requests but cannot fully replace platform telemetry implementations, the shared-process appender, product-level update configuration, Workbench contributions, or the built-in extension set. A fork permits coordinated changes across these layers. The cost is continued upstream maintenance, licensing notices, build-chain ownership, and repeated verification after upgrades. Controlled egress is therefore treated as a versioned property, not as a permanent characteristic of a product name.

4 Controlled-Egress Model

4.1 Channel classes and policy

Let the set of potential outgoing channels be

$$\mathcal{E} = \mathcal{E}_p \cup \mathcal{E}_a \cup \mathcal{E}_f \cup \mathcal{E}_m \cup \mathcal{E}_u,$$

where $\mathcal{E}_p$ represents platform diagnostics, crash and usage telemetry, and experiments; $\mathcal{E}_a$ assistant analytics and error reporting; $\mathcal{E}_f$ update, synchronization, recommendation, extension-gallery, and webview infrastructure; $\mathcal{E}_m$ model provider requests; and $\mathcal{E}_u$ other explicit user operations such as Git or opening an external link.

Define cold-start state s_0 as a fresh user-data directory, an isolated extension directory, no open workspace, no selected model, and no activated command or link. The tested Xynapse policy is

$$\text{egress}(s_0, \Delta t) = \varnothing, \qquad \Delta t = 60 \text{ seconds},$$

where `egress` is operationalized as Sysmon TCP/UDP events attributed to an image named `Xynapse.exe`. This definition deliberately excludes inter-process communication, file operations, Event ID 22 DNS, system-process networking on an application's behalf, and differently named child images.

Following explicit action a, the desired policy changes to

$$\text{egress}(s_0 \xrightarrow{a} s_1) \subseteq \text{Allow}(a, \pi),$$

where π is the active configuration. A prompt submission may authorize the selected provider endpoint; a Git pull may authorize the configured remote. The present experiment tests only the empty set in s_0, not the completeness of Allow.

4.2 Threat model and protected data

The threat model includes three sources of unintended egress: an inherited platform component that automatically contacts a vendor service; an embedded extension with independent analytics; and a build regression in which a disabled source path remains active in a stale bundle. It excludes a malicious operating system, kernel compromise, compromised Sysmon, a malicious extension installed by the user after startup, and model-provider behavior after an explicit request.

Potentially sensitive data include paths, repository names, source fragments, diffs, diagnostics, terminal commands, prompts, device identifiers, and usage events. Because the cold-start scenario opens no workspace, it cannot establish protection of repository contents. Its value is narrower: before a functional need exists, the product should not establish even an observable process- attributed connection.

The model yields six requirements: platform telemetry must be disabled at the default, product-capability, and runtime-implementation levels; assistant analytics must be audited separately; experiments and synchronization must not activate by default; provider registration must not imply a startup request; residual update, webview, and gallery capabilities must be listed; and a zero runtime trace must include both a positive filter control and evidence that the application remained alive.

5 Implemented Controls and Residual Surfaces

5.1 Platform services

At product level, `vscode/product.json` sets `enableTelemetry=false` and the default `telemetry.telemetryLevel=off`. The first controls product capability; the second controls the initial user setting. At renderer level, the Workbench telemetry facade assigns `NullTelemetryService`. At shared-process level, the registration uses the same null service and a `NullAppender` array; the `OneDataSystemAppender` import is disabled. Retaining the interface while substituting a null object preserves dependency-injection contracts without constructing a network sender.

The default for `workbench.enableExperiments` is `false`. The browser contribution for Settings Sync contains no executable imports or registrations. The extension-recommendation service uses delayed rather than eager instantiation. The last control limits proactive startup work but does not remove the extension gallery; it is therefore not interpreted as evidence that gallery traffic can never occur.

5.2 Assistant analytics and source-to-bundle checking

The assistant requires an independent audit because extension telemetry need not inherit the platform setting [4]. In the audited snapshot, the shared `Telemetry` class in `core/util/posthog.ts` is a no-op: `capture()` and `captureError()` return immediately, `getTelemetryClient()` returns `undefined`, and `setup()` constructs no client. A second `TeamAnalytics` factory returns `undefined`, so retained PostHog and Logstash implementation classes are not selected by the current factory.

The configuration loader retains the upstream-compatible default `allowAnonymousTelemetry=true`. In isolation this would be a conflicting signal. The IDE adapter, however, computes consent as `globalEnabled && xynapseEnabled`; product-level opt-out propagates `false` into loaded configuration. The no-op facade remains the final sender-prevention layer even if a true value were propagated accidentally. Sentry initialization additionally requires both telemetry consent and a team- member identity; the cold-start profile satisfies neither active path.

The audit also inspects the packaged assistant bundle at `resources/app/extensions/xynapse-assistant/out/extension.js`. It contains the corresponding no-op entry point, and its SHA-256 is recorded. This

closes a common source-to-binary gap in which TypeScript is corrected while a release still packages an older JavaScript bundle.

5.3 Residual online-capability boundaries

Five observed properties delimit the claim. Update check, download, installation, and restart actions remain registered. Product configuration contains a GitHub update source. An external URL template exists for webview content. The YAML loader keeps a compatibility telemetry default of true even though later gates and the no-op facade suppress the audited active path. Finally, the assistant declares `onStartupFinished`, making its activation part of the observed window but also a future regression point.

Interpretation rule

These findings do not contradict the cold-start observation. They prevent the broader statements that Xynapse is fully offline, that update functionality is removed, or that no external endpoint remains in the code. Here, controlled egress means default denial for the tested startup state plus an explicit register of functions requiring separate tests.

6 Method

6.1 Research objects and static predicates

The source object is public commit `5f01a9fb0fdebc2a40c7843552c18aad189a0453` on the repository's `master` branch. The `audit_xynapse_source.ps1` script accepts a source root and output path, verifies required files, computes SHA-256 values, and applies predicates to parsed JSON or TypeScript/JavaScript text. It does not read credentials or user configuration.

Each observation has an identifier, category, file, Boolean observation, and interpretation. A `control` is a declared architecture or protective property. A `scope-boundary` is a residual capability that must be found to constrain the claim. Thus, 5/5 boundaries does not mean that five defenses passed; it means that five limitations were detected as expected.

Table 1: Static-audit groups and interpretations.

Group	Predicate	Supported interpretation
Product	`enableTelemetry=false`; default level `off`	Platform telemetry is disabled at capability and default levels.
Renderer	Workbench delegates to `NullTelemetryService`	Renderer callers retain the interface without an active sender.
Shared process	`NullTelemetryService` and `NullAppender`	The shared telemetry path has no active appender.
Experiments	`workbench.enableExperiments=false` by default	Online experiments are not enabled by default.
Settings Sync	no executable contribution registrations	The browser contribution does not launch a sync workflow.
Recommendations	delayed service instantiation	The service is not created eagerly; gallery removal is not claimed.
Providers	`yandex_gpt` and `gigachat` classes and index entries	The two model adapters exist and are registered.
Assistant source	no-op capture/client and undefined provider factory	Primary analytics entry points construct no sender.

Group	Predicate	Supported interpretation
IDE/config gate	global AND extension consent; false propagation	The assistant inherits product opt-out in the audited path.
Packaged bundle	matching no-op entry point and recorded bundle hash	The release bundle contains the audited assistant behavior.
Boundaries	update, webview, compatibility default, startup activation	Residual online surfaces are identified rather than hidden.

The static script evaluates 19 control predicates and five boundary predicates. Predicates cover the product JSON, renderer and shared-process telemetry services, experiment defaults, Settings Sync contribution, recommendation lifecycle, provider classes and registration, assistant telemetry source, IDE-to-assistant consent propagation, Sentry gating, and the packaged bundle. The public Git commit identifies the source snapshot; the binary and bundle hashes identify the tested artifacts. This does not prove a fully reproducible build, because the study did not rebuild the executable from a clean machine and compare byte-for-byte identity.

6.2 Dynamic instrument and environment

The dynamic audit uses Microsoft Sysmon 15.21 and Event ID 3, documented as a network-connection event for TCP/UDP connections [7, 8]. The XML configuration filters images ending in `Xynapse.exe`. Process-attributed events reduce unrelated host noise compared with a host-wide packet capture, but do not reveal HTTP payloads and do not cover differently named helper executables or Event ID 22 DNS.

The experiment ran on August 13, 2026, on Microsoft Windows 10 Pro 10.0.19045. The tested portable executable was Xynapse 1.108.0, 210,825,216 bytes, SHA-256 `20e1208038440b7795ecd9d4365a42ae961a87b20f8ff84f167306b2f50a447b`.

Three independent runs used newly created user-data and extension directories. The application started without a workspace. The operator did not open chat, select a model, invoke Git, open the extension gallery, activate a command, follow a webview link, or invoke the updater. Each observation window lasted 60 seconds. The processes were then stopped and events exported to CSV and JSON.

Once per second, the script queried for processes named Xynapse. Because the CIM query itself consumed time, each run yielded 49 samples. For run r,

$$L_r = \frac{n_r^{\text{present}}}{n_r^{\text{sample}}}, \qquad N_r = |\mathcal{E}_r|,$$

where L_r is the fraction of samples with at least one live Xynapse process and N_r is the number of filtered Event ID 3 observations. Acceptance was defined before interpretation as

$$\forall r \in \{1, 2, 3\} : L_r = 1 \wedge N_r = 0.$$

The liveness term is necessary because zero network events from an application that exited immediately would not support autonomous startup. Unique process identifiers were also counted to document multiprocess execution.

6.3 Positive control and data minimization

A zero result can be caused by a faulty Sysmon rule. For a positive control, the system `curl.exe` was copied under the name `Xynapse.exe` and invoked against `https://example.com/`. This is not a product test; it is a test of the instrument path “Image ends with Xynapse.exe.” Success required exit code zero and at least one Event ID 3 observation.

The host used a local proxy, so Sysmon recorded a pair of TCP events involving `127.0.0.1` and `kubernetes.docker.internal`. An external destination is unnecessary for validating the image filter; the

relevant fact is that the network call by a process with the tested name was observed. The proxy limits interpretation of destinations but does not change the zero count in the product runs.

The procedure captured no API keys, prompts, user settings, or packet payloads. The supplement includes XML configuration, PowerShell scripts, aggregate JSON/CSV, positive-control data, source-audit output, and checksums. Temporary user-data directories are excluded.

7 Results

7.1 Static audit

All 19 control predicates returned `observed=true`; all five expected boundaries were also found. The current code is stronger than the early thesis prototype description in several places: renderer and shared-process telemetry use null implementations, and the assistant no-op facade appears in the packaged bundle. Conversely, a broad statement that update contributions were removed is not supported: manual update actions and a GitHub source remain.

This illustrates why a machine-readable audit should be designed to narrow claims as well as confirm them. A search limited to expected `NullTelemetryService` strings would miss update and webview surfaces and could create a false impression of full network isolation.

7.2 Cold-start observations

Table 2: Results of the three cold-start runs.

Run	Window (s)	Samples	Present	Unique PIDs	Event ID 3
1	60	49	49	11	0
2	60	49	49	11	0
3	60	49	49	11	0
Total	180	147	147	–	0

For every run, $L_r = 1$. Each run observed 11 unique Xynapse process IDs, consistent with a multiprocess Electron application. The counts of all, initiated, and initiated-external events were zero. The acceptance criterion was therefore met in all three windows.

The positive-control command exited with code zero and the filter recorded two TCP events. The zero result in the main series is therefore not explained by Sysmon Event ID 3 being globally disabled or by a completely nonfunctional image-name condition. This control does not rule out missed event types and is not a substitute for independent packet capture.

Exact empirical statement

On Windows 10 Pro 10.0.19045, the identified Xynapse 1.108.0 executable remained present throughout each of three independent 60-second windows with fresh user-data and extension directories, no workspace, and no user action. No Sysmon Event ID 3 event was observed for an image named `Xynapse.exe`. The same image filter recorded two events in a separately labelled positive control.

8 Discussion

8.1 Value of combining static and dynamic evidence

The static audit explains why a quiet cold start is expected: null platform telemetry services, a no-op assistant facade, experiments disabled by default, and no executable Settings Sync contribution. The dynamic audit shows that, in the tested build and scenario, these mechanisms were not bypassed by another observed initializer. Each form of evidence is weaker alone.

Source inspection cannot detect a stale bundle, an unexpected build flag, or every runtime path. A runtime trace does not locate the controlling mechanism and does not transfer automatically to another release. Combining them creates the evidence chain

$$\text{requirement} \rightarrow \text{source predicate} \rightarrow \text{bundle check} \rightarrow \text{binary hash} \rightarrow \text{runtime event}.$$

This chain can serve as a release gate: a changed audited file, packaged bundle, or executable hash invalidates the prior result and triggers both stages again.

8.2 Why “controlled egress” is more accurate than “offline”

An AI IDE may intentionally call a remote model. Users also expect Git, external links, extension installation, and sometimes updates. Prohibiting all network access either removes those functions or requires local models and a closed supply chain. A practical goal is therefore not universal zero networking, but the absence of unrequested background activity and the ability to associate each allowed endpoint with an action and policy.

The term controlled egress requires disclosure of residual capabilities. A GitHub update configuration calls for a dedicated update-command test covering endpoint, payload, and consent. An external webview template calls for a webview-opening scenario. Provider adapters call for synthetic repositories and test credentials so that actual payload structure can be compared with selected context without exposing secrets.

8.3 Operational use

Organizations handling nonpublic repositories need to answer four operational questions: which process established a connection; which action initiated it; which endpoint class it reached; and what context could have been transmitted. The present cold-start protocol answers the first and partially links it to the second. Even at that scope, it creates a release control that a declarative setting without a runtime trace does not provide.

A production matrix should define allowlist profiles such as `cold-start`, `model-yandex`, `model-gigachat`, `git`, `extensions`, and `update`. Each profile should specify actions, expected endpoints, and forbidden classes. Controlled egress then becomes a regression specification rather than a one-time demonstration.

Supporting YandexGPT and GigaChat reduces dependence on one foreign provider, but it is not equivalent to local processing. Both audited adapters call cloud APIs. Provider choice may address contractual, availability, or jurisdictional constraints; actual retention and logging behavior must be checked against the current service terms. Provider endpoints also evolve, so an allowlist should be derived from versioned adapter configuration rather than copied permanently from a paper.

8.4 Engineering lessons

First, a source comment is not evidence: an update file may contain a comment about disabled contributions while executable actions remain registered. Second, domain-name search in a bundle overreports risk when inactive dependency code is packaged behind a no-op facade; entry-point analysis and runtime observation are required. Third, a zero trace without a positive control says little about the instrument. Fourth, a fresh profile prevents contamination by user settings, but simultaneously narrows external validity because installed extensions and a real workspace are absent.

9 Threats to Validity and Limitations

9.1 Internal validity

The study covers one machine, one operating-system build, and one executable. Three runs reduce sensitivity to a single transient condition but do not give a statistical guarantee against a rare timer. Sixty seconds may be shorter than a delayed update or recommendation job. System time, proxy state, DNS cache, and network conditions were not randomized.

Sysmon Event ID 3 records TCP/UDP connections rather than HTTP semantics. An image-name filter misses a helper with a different executable name. Event ID 22 DNS was not enabled, and a system resolver may perform work in another process. The positive control validates the chosen filter path, not the completeness of host telemetry.

9.2 Construct validity

The broad phrase “without tracking” is not used as a measured variable. The study measures the absence of one event class in one state. Local logs, crash dumps, file access, IPC, in-memory identifiers, and transmission after user interaction are outside the metric. Likewise, 19/19 checks means that the declared predicates hold; it does not mean 100% security coverage.

9.3 External validity and researcher bias

The result cannot be transferred automatically to Linux or macOS, an installer build, another commit, a profile containing extensions, an open workspace, or an authenticated account. It does not characterize upstream Visual Studio Code or Continue. Every release requires a new artifact hash and rerun.

The project author also designed the architecture, protocol, and interpretation, creating a risk of confirmation bias. Public source, machine-readable results, negative scope boundaries, and a positive control reduce but do not remove this risk. Stronger evidence requires independent replication on another machine and a second observation mechanism such as Windows Filtering Platform/ETW or packet capture with process correlation.

10 Verification Roadmap

The next phase should expand the scenario matrix rather than broaden the claim:

1. independently clone and build the pinned commit in a clean Windows VM;
2. observe idle windows of 5, 15, and 60 minutes for delayed jobs;
3. collect Sysmon Event IDs 1, 3, and 22 and correlate ProcessGuid and parent relationships to cover differently named helpers;
4. use a synthetic secret-free repository for separate YandexGPT and GigaChat runs and record endpoints and payload structure without credentials;
5. test update, webview, extension, and Git actions as separate allowlist profiles;
6. run source predicates on every commit and a network smoke test for each release candidate; and
7. change the compatibility telemetry default to false, remove unreachable analytics implementations from release bundles or prove tree shaking, and isolate manual update from startup paths.

11 Conclusion

This paper presented a controlled-egress architecture and reproducible cold-start assurance case for Xynapse IDE. The distribution combines Code-OSS/Electron, an adapted Continue runtime, null/no-op telemetry paths, and YandexGPT and GigaChat provider adapters. For public commit `5f01a9fb0fdebc2a40c7843552c18aad189a0453`, the static audit confirmed 19 target predicates while preserving five residual online capabilities as explicit boundaries. For one identified Windows binary, three 60-second runs produced zero Sysmon TCP/UDP events under the image filter while the application remained observable; a positive control produced two events.

The result does not establish that the IDE never uses a network. It establishes a narrower, release-specific property: a normal cold start on a fresh profile completed without an observed connection attributable to the Xynapse image before a user action. Publishing the boundaries next to the result makes this claim reproducible and provides a foundation for a broader action-to-endpoint regression matrix.

Artifact Availability

The Xynapse source is publicly available at `https://github.com/jabrailkhalil/xynapse`, pinned to commit `5f01a9fb0fdebc2a40c7843552c18aad189a0453`. The review supplement contains the source-audit script and JSON output, Sysmon configuration and scripts, three-run aggregate, positive control, minimized event exports, and a SHA-256 manifest. It contains no user-data directory, credential, prompt, or packet payload. A permanent public archive identifier will be added before publication; until then, the checksum-verified supplement is supplied privately to editors and reviewers.

Declarations

Funding. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Competing interests. The author is the developer of Xynapse IDE and therefore has a non-financial interest in the system evaluated. No other competing interest is declared.

Author contributions. Dzhabrail Khalilov: conceptualization, software, methodology, investigation, validation, data curation, visualization, writing–original draft, and writing–review and editing.

Declaration of Generative AI and AI-Assisted Technologies in the Manuscript Preparation Process

During the preparation of this work, the author used OpenAI Codex to assist translation from the author's Russian manuscript, improve English readability, and prepare LaTeX. After using this tool, the author reviewed and edited the content against the source code, machine-readable audit artifacts, and cited sources and takes full responsibility for the content of the manuscript.

References

[1] Khalilov, D.E. (2026). *Design and Development of the Xynapse Intelligent Programming Environment with an Integrated AI Assistant*. Master's thesis, HSE University, Moscow. [In Russian].

[2] Khalilov, D.E. (2026). Xynapse IDE source repository. GitHub. `https://github.com/jabrailkhalil/xynapse` (accessed August 13, 2026).

[3] Microsoft. Source Code Organization. Visual Studio Code Wiki. `https://github.com/microsoft/vscode/wiki/source-code-organization` (accessed August 13, 2026).

[4] Microsoft (2026). Telemetry. Visual Studio Code Documentation. https://code.visualstudio.com/docs/configure/telemetry (accessed August 13, 2026).

[5] Microsoft. Differences between the repository and Visual Studio Code. Visual Studio Code Wiki. https://github.com/microsoft/vscode/wiki/Differences-between-the-repository-and-Visual-Studio-Code (accessed August 13, 2026).

[6] Continue Dev, Inc. Continue: open-source AI developer tooling. GitHub. https://github.com/continuedev/continue (accessed August 13, 2026).

[7] Russinovich, M., Garnier, T. (2026). Sysmon v15.21. Microsoft Sysinternals. https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon (accessed August 13, 2026).

[8] Microsoft (2026). Sysmon events. Windows Security Documentation. https://learn.microsoft.com/en-us/windows/security/operating-system-security/sysmon/sysmon-events (accessed August 13, 2026).

[9] Yandex Cloud (2026). Available generative models. AI Studio Documentation. https://aistudio.yandex.ru/docs/en/ai-studio/concepts/generation/models.html (accessed August 13, 2026).

[10] Yandex Cloud (2026). Sending a structured request. AI Studio Documentation. https://aistudio.yandex.ru/docs/en/ai-studio/operations/generation/completions-structured.html (accessed August 13, 2026).

[11] Sber (2026). GigaChat API request authorization. Developer Documentation. https://developers.sber.ru/docs/ru/gigachat/api/reference/rest/gigachat-api (accessed August 13, 2026). [In Russian].

[12] Niu, L., Mirza, S., Maradni, Z., Pöpper, C. (2023). CodexLeaks: Privacy leaks from code generation language models in GitHub Copilot. In *32nd USENIX Security Symposium*, pp. 2133–2150. https://www.usenix.org/conference/usenixsecurity23/presentation/niu

[13] Weiss, R., Ayzenshteyn, D., Amit, G., Mirsky, Y. (2024). What was your prompt? A remote keylogging attack on AI assistants. In *33rd USENIX Security Symposium*, pp. 3367–3384. https://www.usenix.org/conference/usenixsecurity24/presentation/weiss

[14] Vekaria, Y., Canino, A.L., Levitsky, J., et al. (2025). Big help or big brother? Auditing tracking, profiling, and personalization in generative AI assistants. In *34th USENIX Security Symposium*, pp. 8115–8134. https://www.usenix.org/conference/usenixsecurity25/presentation/vekaria

[15] Perry, N., Srivastava, M., Kumar, D., Boneh, D. (2023). Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2785–2799. doi:10.1145/3576915.3623157.

[16] Pineau, J., Vincent-Lamarre, P., Sinha, K., et al. (2021). Improving reproducibility in machine learning research. *Journal of Machine Learning Research*, 22(164), 1–20. https://jmlr.org/papers/v22/20-303.html