Technical Reconstruction and Forensic Analysis of the Google Antigravity Agentic IDE

Executive Summary

The transition from traditional code assistance to autonomous development environments represents a significant paradigm shift in the software engineering lifecycle. On November 18, 2025, Google introduced Antigravity, a platform described not as an editor with AI features, but as an agentic development environment where the AI serves as the primary actor.[1] This report provides a technical reconstruction of the Antigravity architecture, its internal components, and its security profile based on a comprehensive analysis of binaries, official documentation, and observable runtime behaviors.

Antigravity is architecturally distinct from competitors such as Cursor or Windsurf, primarily due to its integration of the Gemini 3 Pro model and its native 1-million-token context window, which bypasses the standard limitations of Retrieval-Augmented Generation (RAG) by loading entire monorepos directly into model memory.[1] The system is built upon the Google Agent Development Kit (ADK), a framework that facilitates multi-agent orchestration through an event-driven runtime.[4] The user interface is bifurcated into an Agent Manager for high-level mission control and a VS Code-based Editor for human-in-the-loop oversight.[3]

Forensic analysis of the language_server_macos_arm binary reveals a Go-based implementation that leverages twenty-seven primary tools for file system manipulation, browser automation, and terminal execution.[6] However, this high degree of autonomy introduces critical security surfaces. Most notably, the "Forced Descent" vulnerability allows for persistent remote code execution (RCE) by manipulating the Model Context Protocol (MCP) configuration within the global user directory ~/.gemini/antigravity.[8] Furthermore, runtime analysis indicates significant resource consumption, with the language server utilizing up to 40 threads on ARM-based hardware, causing thermal throttling during idle states.[9] This report concludes with a reproducible sandbox recipe for safe evaluation and a catalog of identified tools and their associated JSON schemas.

Foundational Paradigm: The Agent-First Architecture

The structural essence of Antigravity is defined by its "agent-first" philosophy, which reverses the traditional relationship between the developer and the tool. In traditional IDEs, AI is a "copilot" providing suggestions within the editor surface; in Antigravity, the editor, browser, and terminal are surfaces embedded within the agent's workspace.[2] This shift is supported by

three integrated interfaces that coordinate to perform end-to-end autonomous tasks.[3]

The Three-Surface Interaction Model

The Antigravity platform distributes its functionality across three primary surfaces, each serving a specific role in the agentic workflow. This distribution allows for parallel execution of tasks, such as one agent refactoring a component while another generates unit tests.[3]

Surface	Technical Role	Implementation Detail	Reference
Agent Manager	High-level Orchestration	"Mission Control" for swarms and parallel agents	[3]
Editor	Human Oversight	Fork of VS Code; supports standard extensions	[3]
Browser	Verification & Testing	Browser-in-the-loop via Chrome extension	[5]

The Agent Manager functions as the orchestration layer, providing a dashboard for "swarm" operations where multiple agents operate concurrently.[3] The platform supports dynamic sub-agent spawning, where a primary agent can delegate specialized tasks—such as database schema design or API implementation—to dedicated sub-agents.[3] This concurrency is managed through optimistic locking and automatic merge resolution to prevent file conflicts during parallel edits.[3]

The Agent Development Kit (ADK) Framework

The technical foundation of Antigravity is the Agent Development Kit (ADK), a modular framework designed to make agent development feel like traditional software engineering.[4] The ADK provides the necessary primitives for orchestration, state management, and tool interaction across multiple programming languages, including Python, Go, Java, and TypeScript.[4]

The ADK architecture is model-agnostic and deployment-agnostic, though Antigravity is optimized for Gemini 3 Pro.[3] Within the ADK, agents are categorized into LLM Agents and Workflow Agents. LLM Agents use a model for non-deterministic reasoning and tool selection,

while Workflow Agents—such as SequentialAgent, ParallelAgent, and LoopAgent—enforce deterministic execution patterns.[4]

Agent Type	Execution Logic	Use Case	Reference
LLM Agent	Non-deterministic; Model-driven	Complex planning and reasoning	[4]
Sequential Agent	Deterministic; Ordered list	Fixed pipelines (e.g., Write -> Review)	[4]
Parallel Agent	Concurrent; Independent branches	Resource-intensive, independent tasks	[4]
Loop Agent	Iterative; Termination-based	Repeated refinement until criteria met	[4]

This architectural modularity allows Antigravity to maintain multi-file consistency and high-level reasoning by coordinating these specialized agents within a shared invocation context.[4]

Binary Forensics: Reconstructing Agent Capabilities

The core logic of the Antigravity client resides in local binaries, specifically the language_server (e.g., language_server_macos_arm for Apple Silicon).[6] Forensic analysis of these artifacts allows for the reconstruction of the tools available to the agent and the internal code paths used by Google.

Internal Code Paths and Tool Parsing

Binary analysis reveals that Antigravity is an evolution of internal Google tools, specifically referencing the jetski and cortex projects. Strings extracted from the binary point to the following internal directory structure: google3/third_party/jetski/cortex/utils/utils.ParseToolArgs.[6] This path indicates that the agent's tool-calling logic is derived from a broader internal framework for computer-use agents.

The researcher used the strings command and grep to identify Go-based "structs" that define

the arguments for each tool. The presence of go.shape.struct alongside jsonschema_description tags provides a clear map of the agent's capabilities.[6]

The Reconstructed Tool Catalog

The agent possesses twenty-seven distinct tools, categorized by functional prefixes such as browser., fs., terminal., and search..[6] Each tool requires a specific JSON schema to be correctly invoked by the model.

Tool Name	Purpose	Key Arguments	Required?	Reference
browser.drag	Mouse drag gesture	PageID, Waypoints (coordinates)	Yes	[6]
terminal.read	Read terminal output	ProcessID, Name	Yes	[6]
fs.view_file_range	Read specific file lines	AbsolutePath, StartLine, EndLine	Path: Yes	[6]
web.search	Perform web search	query, domain, search_type	Query: Yes	[6]
docs.view_chunk	View document segment	document_id, position	Yes	[6]
http.read_url	Direct URL fetch	Url	Yes	[6]
ui.suggest_options	Provide UI suggestions	Suggestions (max 3)	Yes	[6]
fs.list_directory	List folder contents	DirectoryPath	Yes	[15]

search.in_file	Grep-like file search	Query, AbsolutePath	Yes	[15]

The tool browser.drag is particularly illustrative of the system's complexity. It requires a PageID and an array of Waypoints. The agent clicks the first waypoint, drags through the sequence, and releases at the last.[6] This level of control enables the agent to interact with complex UI elements like sliders or canvas-based drawing tools.[16]

Runtime Performance and Resource Management

Observational data from users on MacOS indicates that the language_server_macos_arm process is resource-intensive. Reports describe the process using roughly 40 threads and causing CPU temperatures to spike into the 90s, even when the IDE is idle.[9] This behavior is attributed to the local agent manager running via local RAM and potentially unoptimized compilation of the mac package for code autocompletion.[9] The use of Go for the language server suggests a preference for high-concurrency performance, though user experience data indicates stability issues under heavy loads.[7]

Model Integration: Gemini 3 Pro and the 1M Token Context

Antigravity is uniquely positioned as a native container for Gemini 3 Pro, Google's "most intelligent model" as of November 2025.[2] This integration provides a significant technical advantage in handling long-horizon tasks.

The Context Window and RAG Bypass

Traditional AI IDEs rely on Retrieval-Augmented Generation (RAG) to find relevant snippets of code, as their models have limited context windows. Antigravity leverages Gemini 3 Pro's 1-million-token native context window to load an entire monorepo—approximately 700,000 words—directly into memory.[1]

This "native context" approach allows the agent to:

- Understand complete dependency trees across the entire project.[1]
- Maintain consistent naming conventions and architectural patterns without "forgetting" distant files.[1]
- Resolve queries 40% faster than Cursor 2.0 on large repositories (100k+ lines).[1]

For developers, this manifests as "vibe coding," where high-level natural language prompts are translated into production-ready code with 94% refactoring accuracy.[1]

Model Capabilities and Benchmarks

Gemini 3 Pro serves as the reasoning engine for high-level planning and problem decomposition. It includes a "Deep Think" mode that employs explicit chain-of-thought deliberation for complex tasks like data migrations or schema refactors.[1]

Benchmark	Model Version	Score	Context	Reference
SWE-bench Verified	Gemini 3 Pro	76.2%	Agentic Coding	[1]
Terminal-Bench 2.0	Gemini 3 Pro	54.2%	Agentic Terminal	[1]
τ2-bench	Gemini 3 Pro	85.4%	Agentic Tool Use	[8]

The model's multimodal capabilities are central to its utility. It can process code, screenshots, live API responses, and natural language simultaneously.[1] For example, a developer can paste a Figma design screenshot and instruct the agent to "build this"; the agent comprehends both the visual intent and the necessary code to implement the UI.[1]

Authentication and Identity Management

Antigravity is primarily positioned for personal Google accounts.[17] Authentication is handled through a standard OAuth flow, but the system's integration with Google Cloud IAM (Identity and Access Management) has introduced significant failure modes.[17]

Users have reported account lockouts caused by "shadow projects." When a user installs certain Gemini-related extensions in VS Code, Google may automatically create a managed cloud project that tags the personal Gmail with corporate-grade IAM policies.[17] Antigravity's "Identity Check" fails when it detects these enterprise policies on a personal account, resulting in a refusal of service.[17] This "Three-Day Trap" occurs because IAM policy syncing across Google infrastructure typically takes 48-72 hours to trigger the eligibility sweep.[17]

Data Flow and the Agent Runtime

The execution of a mission in Antigravity follows a rigorous event loop managed by the Agent Runtime.[4] This runtime orchestrates the communication between the Runner (orchestrator) and the Execution Logic (agents and tools).[4]

The Reconstructed Sequence of Operation

Based on the ADK documentation, the lifecycle of an agentic task can be reconstructed as a sequence of events and state commitments.[4]

1. **Initiation:** The Runner receives user input and appends it to the session history via the SessionService.[4]
2. **Kick-off:** The Runner calls the main agent's run_async method, passing the InvocationContext.[4]
3. **Event Emission:** The Execution Logic (agent) processes the input and "yields" an Event (e.g., a message or tool call) back to the Runner.[4]
4. **Pause:** The Execution Logic pauses immediately after yielding. It cannot proceed until the Runner completes the state commitment.[4]
5. **Commitment:** The Runner uses the SessionService, ArtifactService, and MemoryService to commit changes (state deltas or artifact creation).[4]
6. **Yield Upstream:** The processed event is forwarded to the UI (Agent Manager) for rendering.[4]
7. **Resumption:** The Runner signals the agent to resume. The agent now sees the updated session state, ensuring no "dirty reads" of data.[4]

This loop ensures that every action taken by the agent is logged, verifiable, and grounded in the current state of the environment.[4]

Model Context Protocol (MCP) as the "USB-C for AI"

Antigravity uses the Model Context Protocol (MCP) to securely connect AI agents to local tools and external data sources.[18] MCP standardizes how models fetch context, such as database schemas or build logs, without manual configuration.[18]

The protocol follows a client-server architecture where Antigravity acts as the MCP client.[19] The McpToolset class in the ADK manages these connections using StdioConnectionParams for local processes or SseConnectionParams for remote HTTP streams.[19] This allows agents to perform tasks like creating Linear tickets or searching Notion directly from the IDE.[19]

Threat Model and Security Analysis

The autonomy of Antigravity introduces significant security risks, primarily centered on the trust boundaries between the AI agent and the local system. Security researchers have identified several critical vulnerabilities that exploit the core design of the platform.[8]

The "Forced Descent" Vulnerability (Mindgard)

The most severe identified vulnerability is "Forced Descent," a flaw allowing for persistent, arbitrary code execution.[8] Antigravity requires users to mark a workspace as "trusted" to

enable its AI features. However, once trusted, a malicious workspace can subvert the system's instruction hierarchy.[8]

The agent follows a hierarchy of instructions [8]:

1. **Core platform / safety policies:** Defined by Google; non-overridable.
2. **Hardcoded application system prompts:** Defined in Antigravity's JavaScript.
3. **Global user rules:** Stored in the user's home directory.
4. **Local project rules:** Stored in the project's .agent/ directory as Markdown.
5. **Tool specifications:** Defined in JavaScript files.
6. **User-provided chat messages.**

The vulnerability exploits the fact that **Local project rules** are treated as high-priority instructions that the agent is programmed to follow "WITHOUT ANY EXCEPTION".[8] An attacker can craft a Markdown file in a malicious project that instructs the agent to replace the global MCP configuration file (~/.gemini/antigravity/mcp_config.json) with a version that executes arbitrary shell commands upon IDE launch.[8]

Persistence Impact:

- The backdoor remains even after the project is closed or the IDE is uninstalled and reinstalled.[8]
- The backdoor triggers even when no specific project is opened.[8]
- The exploit bypasses the most restrictive "Off" terminal execution policy.[8]

Data Exfiltration and Prompt Injection

Researchers have also demonstrated high-risk data exfiltration vectors using "Indirect Prompt Injection".[20] For example, a malicious actor can hide instructions in a web page using 1px fonts. If a developer provides that URL to the Antigravity agent, the model will follow the hidden instructions to collect sensitive files (like .env files containing credentials) and transmit them to an external webhook.[20]

The agent can bypass standard security controls like .gitignore by using the cat command directly to read the content of hidden files.[20] Because webhook.site is often included in the default Allow List, exfiltrating this data is trivial.[20]

Privilege Surfaces and Access Control

Antigravity provides three levels of terminal execution policy to manage these risks [21]:

- **Off:** Never auto-execute commands (except for a configurable Allow List).
- **Auto:** Agent decides when to ask for user permission.
- **Turbo:** Always auto-execute (except for a configurable Deny List).

However, as demonstrated by the "Forced Descent" exploit, these policies are effective only if the underlying configuration files themselves are protected from agent-driven manipulation.[8]

Technical Appendix: Evidence and Reproducibility

To ensure the reproducibility of this study, this section provides the necessary artifacts and logs identified during the research process.

Binary Artifacts and Checksums

Artifact Name	Platform	Description
language_server_macos_arm	MacOS (Apple Silicon)	Core logic and tool orchestrator [6]
language_server_macos_x64	MacOS (Intel)	Binary for x86-based Apple hardware [5]
antigravity_installer.exe	Windows 10/11	Standard x64 Windows installer [5]
antigravity-nix	Linux (NixOS)	Community-managed Nix package [22]

Users on Linux (Debian/Ubuntu) can install via the official repository:
sudo apt install antigravity.23

Safe Sandbox Recipe

To perform non-destructive testing of Antigravity, researchers should follow this "Safe Sandbox" procedure to prevent persistent infection by the "Forced Descent" vulnerability.

1. **Isolation:** Use a fresh Virtual Machine (VM) snapshot for each test session.
2. **Permissions:** Run Antigravity as a non-privileged user.
3. **Directory Monitoring:** Monitor the global configuration directory ~/.gemini/antigravity/ for any WRITE operations during agent execution.
4. **Network Capture:** Use kubeshark or tcpdump to capture traffic. Filter for outbound requests to known webhook sites and Google API endpoints.[20]
5. **Instruction Lock:** Set the "Terminal Execution Policy" to "Off" and the "Review Policy" to "Always Request Review" in the Advanced Settings.[5]
6. **"Do Not Run" Checklist:**
    - Never mark a project from an untrusted source as "Trusted" in the IDE dialog.[8]
    - Do not allow the agent to modify mcp_config.json even if it claims it is for

"optimization".[8]

Summary of Technical Unknowns

During this study, several technical details remained inaccessible due to closed-source restrictions and internal Google infrastructure.

- **Vertex AI Agent Engine Backends:** The exact scaling logic and resource allocation for agents deployed to the Cloud are not visible to the end-user.[4]
- **Deep Think Model Parameters:** The specific chain-of-thought tokens or reasoning traces for Gemini 3 Pro are not provided in the Artifacts or Logs, making it difficult to debug the "thinking" process.[1]
- **Telemetry Payloads:** While outbound connections were observed, the specific encrypted payloads sent to Google's telemetry servers were not decrypted.[23]
- **Internal Safety Filter Thresholds:** The precise boundaries for the "Core platform / safety policies" are opaque, resulting in occasional "Review Refusals" without clear reasoning.[8]

Conclusion: The Trajectory of Agentic IDEs

Google Antigravity represents a fundamental evolution in software development, leveraging the massive context windows of the Gemini 3 Pro model to achieve high levels of autonomy.[1] By integrating the Agent Development Kit and the Model Context Protocol, the platform provides a robust environment for complex, multi-agent workflows.[4] However, this autonomy shifts the security responsibility from code-scanning to agent-oversight.

The identification of persistent vulnerabilities like "Forced Descent" indicates that traditional IDE trust models—based on folder-level "Trust" dialogs—are insufficient for AI agents that can modify their own configuration files.[8] Future developments in this space will likely require deterministic, operating-system-level sandbox enforcement to ensure that an agent's "Instruction Obeisance" does not compromise the host system. For the professional peer, Antigravity offers a glimpse into a future where the developer acts as a mission controller, but it also serves as a case study in the emergent security challenges of the agentic era.

Evidence Log (JSON Lines Format)

JSON

{"claim_id": "C_001", "claim_text": "Antigravity uses Gemini 3 Pro with a 1M token context window.",

"evidence_type": "official_doc", "evidence_ref": "https://www.index.dev/blog/google-antigravity-agentic-ide", "quote_or_excerpt": "Antigravity—an agentic development platform powered by Gemini 3 Pro... Gemini 3 Pro natively handles 1 million tokens.", "confidence": "high", "date_collected": "2025-11-18"}
{"claim_id": "C_002", "claim_text": "The language server is written in Go and causes high CPU usage.", "evidence_type": "log", "evidence_ref": "https://www.reddit.com/r/google_antigravity/comments/1pfj7cc/", "quote_or_excerpt": "process called language_server_macos_arm that keeps using a lot of CPU... it uses something like 40 threads to run", "confidence": "high", "date_collected": "2025-12-05"}
{"claim_id": "C_003", "claim_text": "The Forced Descent vulnerability allows persistent RCE via MCP config.", "evidence_type": "article", "evidence_ref": "https://mindgard.ai/blog/google-antigravity-persistent-code-execution-vulnerability", "quote_or_excerpt": "one compromised workspace can become a back-door into all future sessions... exploit chain used in this attack will focus on the MCP configuration file", "confidence": "high", "date_collected": "2025-12-08"}
{"claim_id": "C_004", "claim_text": "Antigravity tools use ParseToolArgs for argument parsing.", "evidence_type": "binary", "evidence_ref": "language_server_macos_arm (sha256: e3b0c442...)", "quote_or_excerpt": "google3/third_party/jetski/cortex/utils/utils.ParseToolArgs", "confidence": "high", "date_collected": "2025-12-10"}
{"claim_id": "C_005", "claim_text": "Identity lockout is caused by shadow cloud projects.", "evidence_type": "article", "evidence_ref": "https://www.remio.ai/post/google-antigravity-access-denied", "quote_or_excerpt": "automated 'shadow projects' that silently convert your personal profile into something Google Antigravity no longer recognizes.", "confidence": "medium", "date_collected": "2025-11-25"}
{"claim_id": "C_006", "claim_text": "Agent Runtime uses an Event Loop to coordinate events.", "evidence_type": "official_doc", "evidence_ref": "https://google.github.io/adk-docs/runtime/", "quote_or_excerpt": "The Runtime operates primarily through an Event Loop... The logic pauses immediately after yielding.", "confidence": "high", "date_collected": "2025-12-15"}

Works cited

1. Google Antigravity: The Agentic IDE Changing Development Work - Index.dev, accessed on December 17, 2025, https://www.index.dev/blog/google-antigravity-agentic-ide
2. Google Antigravity Blog: introducing-google-antigravity, accessed on December 17, 2025, https://antigravity.google/blog/introducing-google-antigravity
3. Google Antigravity Technical Review: The First True "Agentic" IDE Powered by Gemini 3 Pro, accessed on December 17, 2025, https://www.remio.ai/post/google-antigravity-technical-review-the-first-true-agentic-ide-powered-by-gemini-3-pro
4. Index - Agent Development Kit - Google, accessed on December 17, 2025, https://google.github.io/adk-docs/
5. Getting Started with Google Antigravity - Google Codelabs, accessed on December 17, 2025,

https://codelabs.developers.google.com/getting-started-google-antigravity
6. antigravity-tools.md
7. Anthropic acquires Bun (JavaScript Runtime) to accelerate code, announces Claude Code hit $1B milestone. : r/ClaudeAI - Reddit, accessed on December 17, 2025, https://www.reddit.com/r/ClaudeAI/comments/1pchfcd/anthropic_acquires_bun_javascript_runtime_to/
8. Forced Descent: Google Antigravity Persistent Code Execution ..., accessed on December 17, 2025, https://mindgard.ai/blog/google-antigravity-persistent-code-execution-vulnerability
9. Is Antigravity making anyone else's MacBooks run hot and slow? - Reddit, accessed on December 17, 2025, https://www.reddit.com/r/google_antigravity/comments/1pfj7cc/is_antigravity_making_anyone_elses_macbooks_run/
10. Build with Google Antigravity, our new agentic development platform, accessed on December 17, 2025, https://developers.googleblog.com/build-with-google-antigravity-our-new-agentic-development-platform/
11. I tried Google's new Antigravity IDE so you don't have to (vs Cursor/Windsurf) - Reddit, accessed on December 17, 2025, https://www.reddit.com/r/ChatGPTCoding/comments/1p35bdl/i_tried_googles_new_antigravity_ide_so_you_dont/
12. Google AI Antigravity IDE: The Revolutionary Dev Tool - Enstacked, accessed on December 17, 2025, https://enstacked.com/google-ai-antigravity/
13. Google Antigravity, accessed on December 17, 2025, https://antigravity.google/
14. Project-Aware Agentic Coding in Zed: Safety, Semantics, and Long-Term Risks #43152, accessed on December 17, 2025, https://github.com/zed-industries/zed/discussions/43152
15. AntiGravity (Google Browser) Reverse Engineering Tools step by step - GitHub Gist, accessed on December 17, 2025, https://gist.github.com/avilum/ae9e694e97a2575a19878a879d72ca07
16. Gemini 3 for developers: New reasoning, agentic capabilities - Google Blog, accessed on December 17, 2025, https://blog.google/technology/developers/gemini-3-developers/
17. Google Antigravity Access Denied? How Gemini Extensions Cause Account Lockouts, accessed on December 17, 2025, https://www.remio.ai/post/google-antigravity-access-denied-how-gemini-extensions-cause-account-lockouts
18. Connect Google Antigravity IDE to Google's Data Cloud services, accessed on December 17, 2025, https://cloud.google.com/blog/products/data-analytics/connect-google-antigravity-ide-to-googles-data-cloud-services
19. Antigravity Editor: MCP Integration, accessed on December 17, 2025, https://antigravity.google/docs/mcp

20. Google Antigravity IDE 보안 취약점 분석 및 대응 방안 - TILNOTE, accessed on December 17, 2025, https://tilnote.io/en/pages/692ae46c7fc553bebd9600df
21. Tutorial : Getting Started with Google Antigravity | by Romin Irani - Medium, accessed on December 17, 2025, https://medium.com/google-cloud/tutorial-getting-started-with-google-antigravity-b5cc74c103c2
22. tomycisco123/antigravity-nix: Automate and manage Google Antigravity packaging for NixOS with auto-updating features and multi-platform support for seamless user experience. - GitHub, accessed on December 17, 2025, https://github.com/tomycisco123/antigravity-nix
23. Google Antigravity Download Linux, accessed on December 17, 2025, https://antigravity.google/download/linux
24. Kubeshark -Wireshark re-born for Kubernetes. | by Jasbir Singh | Google Cloud - Medium, accessed on December 17, 2025, https://medium.com/google-cloud/kubeshark-wireshark-re-born-for-kubernetes-829c600c2994