Coding Agent is All You Need: Unifying General-Purpose and Domain-Specific AI Through the Meta-Agent Paradigm

Kelly Peilin Chan
kelly@vikadata.com

Abstract

Building AI agents is hard. Memory systems, tool integrations, prompt engineering, sandbox infrastructure—each demands weeks of engineering, and the tax is paid *per agent*. Meanwhile, coding agents—Claude Code, Codex CLI, GitHub Copilot—have already solved all of these problems. They manage context. They execute code. They read and write files. They search the web. They connect to external tools via MCP. They run in sandboxes.

We argue these agents are not merely coding tools. They are **meta-agents**: general-purpose runtimes whose capabilities—Compute, Storage, Network, Extension, Reasoning—form a *capability superset* that subsumes virtually all domain-specific AI agents. A data analyst agent? It's a coding agent with a different prompt. A research agent? Same. An SEO agent? Same.

We propose **Agent Redirection**: instead of *building* agents from scratch, *redirect* an existing coding agent to any domain with a Markdown template. No SDK. No framework. No memory system. Just a `CLAUDE.md` file.

We present **SandAgent**, an open-source system implementing this paradigm with pluggable runtimes (Claude, Codex, Copilot) and sandboxes (E2B, Docker, Local). On the GAIA benchmark, template-redirected agents achieve competitive performance. In case studies across four domains, Agent Redirection reduces development effort by $\sim$300$\times$ compared to SDK-based construction.

The industry agrees. GitHub Agent HQ (February 2026) now runs Claude, Codex, and Copilot as interchangeable runtimes in the same interface. The convergence has begun.

Don't build agents. Redirect them.

1 Coding Agents Are Eating the World

In 2011, Marc Andreessen declared that "software is eating the world" (Andreessen, 2011). Fifteen years later, we observe the next phase: *coding agents are eating the world of AI agents*.

What began as tools for code completion has quietly evolved into the most capable, general-purpose agent runtime available today. Claude Code can run SQL queries. Codex CLI can scrape websites. GitHub Copilot can generate data visualizations. None of these are "coding" tasks. Yet coding agents handle them effortlessly—because the capabilities required to write software are a *superset* of the capabilities required for almost everything else.

This paper makes one central claim:

Preprint. February 2026. Work in progress.

Coding agents are not coding tools.
They are meta-agents—universal runtimes for all AI agents.

The implications are profound. If coding agents are meta-agents, then:

- You don't need LangChain to build a data analyst agent. You need a `CLAUDE.md` file.
- You don't need CrewAI to build a research agent. You need a `CLAUDE.md` file.
- You don't need months of engineering. You need an afternoon.

We call this **Agent Redirection**: the practice of specializing a general-purpose coding agent for any domain using only declarative templates—system prompts, skill modules, and MCP configurations—with zero SDK integration.

The industry is already converging on this insight. On February 4, 2026, GitHub launched Agent HQ (GitHub, 2026), a unified platform where Claude Code, Codex, and Copilot run as *interchangeable runtimes* in the same repository. VS Code's January 2026 release treats different coding agents as swappable backends (VS Code, 2026). The world's largest development platforms now treat coding agents exactly as our thesis predicts: as general-purpose infrastructure, not specialized tools.

Contributions. We (1) formalize the Meta-Agent Paradigm and Agent Redirection; (2) define a five-primitive capability taxonomy showing coding agents subsume domain-specific requirements; (3) present SandAgent, an open-source implementation with pluggable runtimes and sandboxes; (4) evaluate on GAIA and four domain case studies; and (5) propose the AGI-Runtime Hypothesis—that the path to general intelligence runs through agent runtimes, not just better models.

2 The Agent Construction Tax

Building a production AI agent with traditional SDK approaches is painful. We call this the **Agent Construction Tax**.

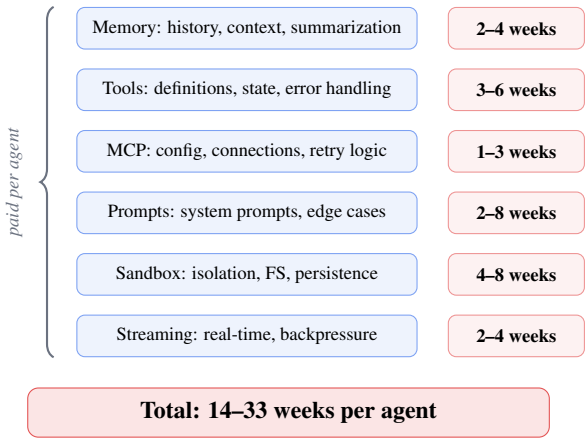


Figure 1: The Agent Construction Tax. Each component must be built from scratch, and the tax is paid *per agent*.

This tax is paid *per agent*. An organization building a data analyst, a research assistant, and a customer support agent pays it three times—even though the underlying capabilities (execute code, read files, search the web) are largely identical.

2.1 The Framework Explosion

The market's response has been an explosion of agent frameworks: LangChain (LangChain, 2023), CrewAI (CrewAI, 2024), AutoGen (Wu et al., 2023), MetaGPT (Hong et al., 2023), and dozens more. Each reduces the tax somewhat, but none eliminates it.

What if you could skip the tax entirely?

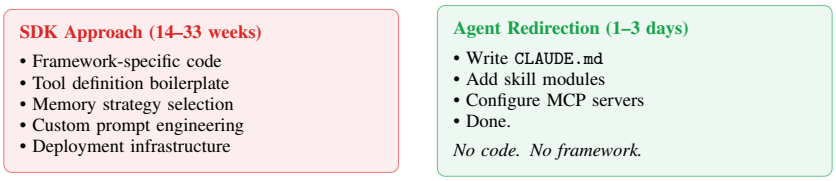


Figure 2: SDK-based Agent Construction vs. Template-based Agent Redirection.

3 Three Generations of Coding Agents

Coding agents did not start as meta-agents. They evolved into them.

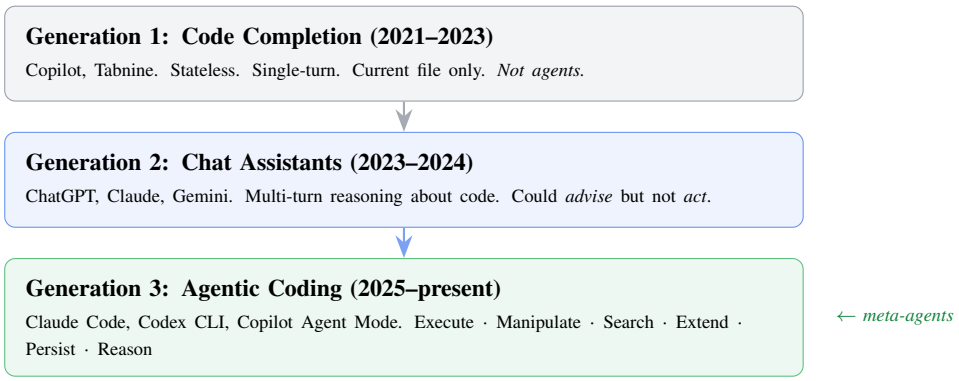


Figure 3: Three generations of coding agents. Generation 3 capabilities are general-purpose by nature.

Here is the critical observation: *none of Generation 3's capabilities are specific to software engineering.*

A bash shell can run SQL queries as easily as it compiles code. A file system can store research notes as easily as source files. Web search retrieves market data as easily as API documentation. MCP connects to PostgreSQL as easily as to Git.

The capabilities were *developed* for coding. But they are *general-purpose* by nature. This accident of history—that the most invested-in AI agents happen to have the most general capabilities—is the foundation of our thesis.

3.1 The MCP Multiplier

The Model Context Protocol (MCP) (Anthropic, 2024) deserves special attention. MCP transforms coding agents from closed systems into *extensible platforms*:

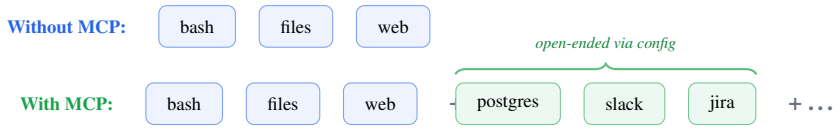


Figure 4: MCP makes the capability set open-ended. Any tool becomes a JSON configuration, not an engineering project.

4 The Coding Agent Arsenal: Built-In Superpowers

A key premise of the Meta-Agent Paradigm is that coding agents *already* ship with powerful, general-purpose capabilities. This section surveys the built-in toolkits of major coding agents, demonstrating that the meta-agent capability set is not theoretical—it is deployed and battle-tested at scale.

4.1 Capability Survey

Table 1: Built-in capabilities of major coding agents. ✓ = native, ⊕ = via MCP, — = not available.

Capability	Claude Code	Codex CLI	Gemini CLI	Copilot CLI	Kimi CLI
Shell execution	✓	✓	✓	✓	✓
File read/write/edit	✓	✓	✓	✓	✓
Glob/grep search	✓	✓	✓	✓	✓
Web search	✓	✓	✓	—	✓
Web page fetch	✓	✓	✓	—	✓
Multi-step planning	✓	✓	✓	✓	✓
Sub-agent delegation	✓	—	—	✓	✓
Task/TODO management	✓	—	—	—	—
MCP support	✓	⊕	✓	✓	✓
Multimodal input	✓	✓	✓	✓	—
Sandbox isolation	⊕	✓	✓	✓	—
Session persistence	✓	✓	✓	✓	✓

4.2 Agent Profiles

Claude Code (Anthropic) is the most tool-rich agent in our survey. Its native toolkit includes: `Bash` (shell execution), `Read/Write/Edit/MultiEdit` (file operations), `Ls/Grep/Glob` (filesystem search), `WebSearch/WebFetch` (internet access), `TodoRead/TodoWrite` (task management), and sub-agent spawning for parallel task delegation. Claude Code also supports `CLAUDE.md` project instructions, auto-discoverable `skills/`, and a rich MCP ecosystem—making it the most natural host for Agent Redirection.

Codex CLI (OpenAI) runs entirely locally with a security sandbox that isolates all code execution. It supports multimodal input (screenshots, diagrams, text), automatic dependency management, and configurable approval levels (`suggest`, `auto-edit`, `full-auto`). Codex achieves the highest GAIA scores in our evaluation (73.58% L1), reflecting the strength of its underlying o4-mini reasoning model.

Gemini CLI (Google) brings Google's infrastructure advantages: built-in Google Search grounding, Imagen/Veo/Lyria for media generation, and 1M+ token context windows. It supports shell execution, file operations, and MCP, with macOS Seatbelt or Docker/Podman sandboxing. Its 60 requests/minute free tier makes it the most accessible agent for experimentation.

Copilot CLI (GitHub) integrates deeply with the GitHub ecosystem: repository context, pull request workflows, Actions log analysis, and the `/delegate` command for spawning background coding agents. Through Agent HQ, it can orchestrate Claude and Codex as alternative runtimes, making it a meta-agent *orchestrator* in addition to being a meta-agent itself.

Kimi CLI (Moonshot AI) features a unique dual-mode design: press `Ctrl+K` to switch between standard shell mode and AI agent mode. It supports shell execution, file operations, web fetching, and sub-agent delegation via the Agent Client Protocol. Its integration with the Zed editor demonstrates the agent-as-runtime pattern extending beyond the terminal.

4.3 The Key Insight

This is why Agent Redirection achieves $\sim$300$\times$ speedup over SDK-based construction: the hardest engineering problems—tool orchestration, context management, error recovery, multi-step

Every major coding agent *already* ships with:

Shell execution · **File I/O** · **Web search** · **Multi-step planning** · **MCP**

These are not features you need to build. They are features you *inherit* through Agent Redirection. The only thing missing is domain knowledge—which is a `CLAUDE.md` file.

Figure 5: The built-in capabilities of coding agents eliminate the need to implement tools, memory, or planning from scratch. Agent Redirection inherits all of them for free.

planning—have already been solved by teams of hundreds of engineers over years of development. A template author inherits all of this work with zero effort.

5 The Meta-Agent Thesis

We now formalize our core claim.

5.1 Definitions

A **domain-specific agent** A_d is characterized by a tuple (C_d, T_d, K_d): required capabilities, tools, and domain knowledge.

A **coding agent** A_c is characterized by (C_c, T_c, K_c): the capabilities, tools, and knowledge of a modern agentic coding system.

Definition 1 (Meta-Agent). An agent A_m is a *meta-agent* with respect to a set of domain-specific agents $\{A_{d_1}, \ldots, A_{d_n}\}$ if:

$$\forall i \in \{1, \ldots, n\} : C_{d_i} \subseteq C_m$$

That is, the meta-agent's capability set is a *superset* of every domain-specific agent's requirements.

Thesis. Modern coding agents are meta-agents. Their capability set—forged by the demands of software engineering, the most capability-intensive knowledge work—subsumes the requirements of virtually all non-physical domain-specific agents.

5.2 Why Software Engineering Produces Meta-Agents

This is not a coincidence. Three structural reasons:

1. **Broadest capability set.** Writing software requires reading files, executing programs, searching docs, managing state, reasoning about complex systems. No other domain demands *all* of these.
2. **Largest investment.** Market pressure → billions in R&D → most refined agent capabilities.
3. **Code is the universal interface.** Any task can be expressed as a program: SQL query = code, data visualization = code, web scraping = code, API call = code. An agent that writes code can do *anything a computer can do*.

Figure 6: Three structural reasons why coding agents became meta-agents.

The third point is the deepest. Code is not just one capability among many—it is the *meta-capability*. An agent that can write and execute arbitrary code has, in principle, access to the entire space of computational tasks. Coding agents are meta-agents because code itself is a meta-tool.

6 Capability Taxonomy

We define five capability primitives sufficient to characterize any non-physical AI agent:

Table 2: Five capability primitives and their coding agent implementations.

Primitive	What It Does	In Coding Agents	Example Use
COMPUTE	Execute arbitrary logic	Bash, Python, Node.js	Run SQL queries
STORAGE	Read/write persistent state	File system operations	Save research notes
NETWORK	Access external information	Web search, HTTP, APIs	Fetch market data
EXTENSION	Connect to external tools	MCP servers	Query databases
REASONING	Plan, analyze, synthesize	LLM + chain-of-thought	Write reports

Claim. These five primitives are sufficient to implement any domain-specific agent task that does not require specialized hardware (robotic manipulation) or real-time sensory input (autonomous driving).

6.1 Domain Coverage

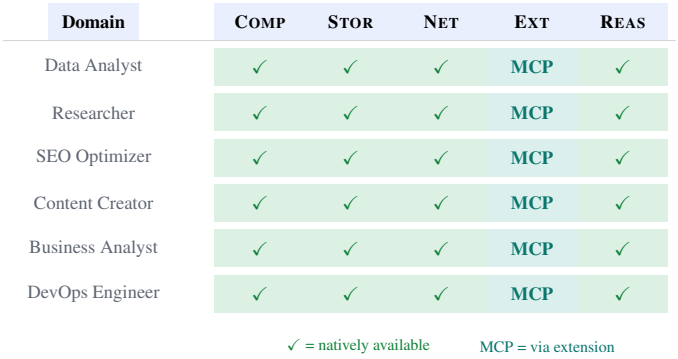


Figure 7: Domain coverage matrix. Every cell is covered. No gaps. The EXTENSION primitive (MCP) transforms any domain-specific tool requirement into a JSON configuration.

6.2 What Differentiates Domain Agents?

If the capabilities are identical, what makes a data analyst agent different from a research agent? Three things:

1. **Knowledge** (K_d): Domain expertise, best practices, workflows
2. **Persona**: Communication style, output format, quality standards
3. **Tool configuration**: Which MCP servers are connected

All three are *declarative*. They can be specified in text files. No code required. This is the foundation of Agent Redirection.

7 Agent Redirection

Definition 2 (Agent Redirection). Given a meta-agent A_m and a target domain d, Agent Redirection is:

$$\text{Redirect}(A_m, K_d, T_d) \to A'_d$$

where A'_d is a domain-specialized agent constructed by injecting domain knowledge K_d and tool configurations T_d into the meta-agent, *without modifying the agent's core runtime*.

7.1 The Redirection Stack

In practice, Agent Redirection is achieved through a three-layer declarative stack:

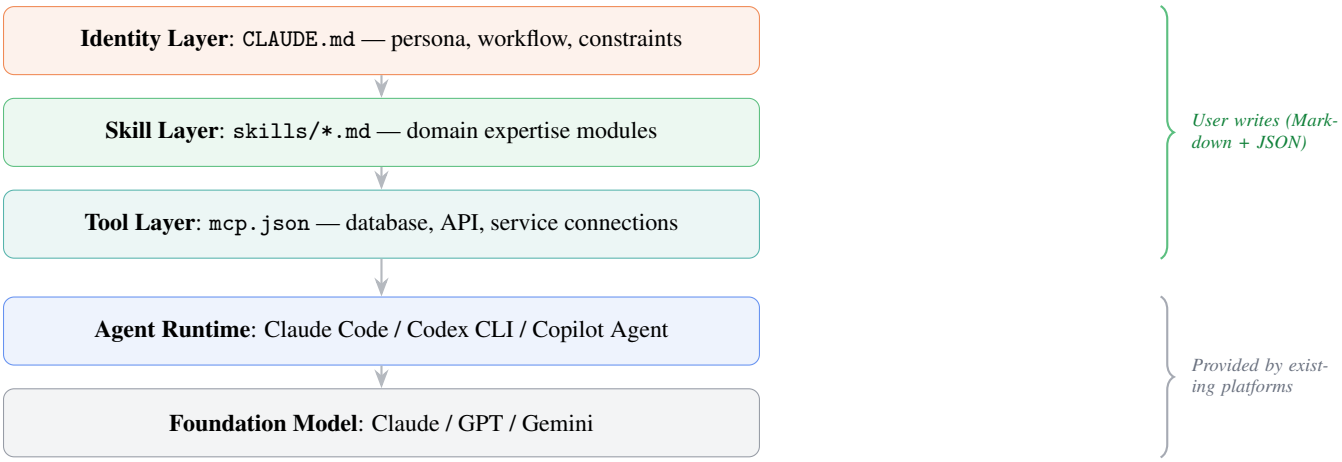


Figure 8: The Redirection Stack. Users define only the top three layers. The agent runtime and foundation model are provided by existing platforms.

Table 3: Two paradigms for building AI agents.

Dimension	Construction	Redirection
Development effort	14–33 weeks	1–3 days
Artifacts produced	Code (SDK integrations)	Markdown + JSON
Memory system	Must implement	Inherited
Tool integration	Must implement per tool	MCP config
Sandbox isolation	Must implement	Inherited
Runtime upgrades	Manual migration	Automatic
Vendor lock-in	Framework-specific	Runtime-agnostic

7.2 Agent Redirection vs. Agent Construction

7.3 Template Portability

A crucial property: SandAgent templates are *directly compatible with Claude Code*. A user can navigate to a template directory and run:

```
cd templates/analyst
claude "Analyze Q4 revenue trends from the database"
```

No wrapper. No SandAgent installation. Claude Code automatically discovers `CLAUDE.md`, `skills/`, and `.claude/mcp.json`. The template *is* the agent. This demonstrates that Agent Redirection is not tied to any specific system—it is a general property of the Meta-Agent Paradigm.

8 SandAgent: System Design

We present **SandAgent**, an open-source implementation of the Meta-Agent Paradigm.

8.1 Architecture

8.2 Interface-Driven Design

The core innovation is *interface-driven, zero-dependency* design. The manager defines two interfaces:

```
interface Runner {
  run(input: string, options?: RunOptions):
    AsyncIterable<RunnerOutput>;
```

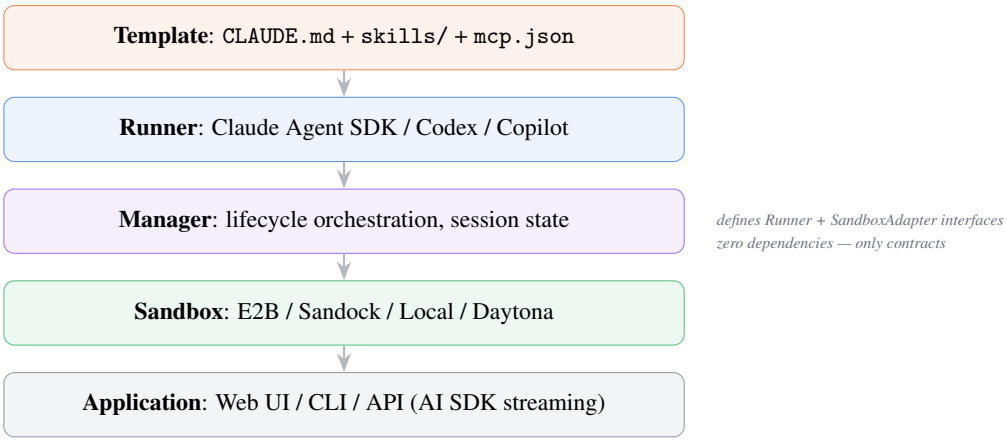


Figure 9: SandAgent layered architecture. The manager defines interfaces; implementations are pluggable.

```
}

interface SandboxAdapter {
  attach(id: string): Promise<SandboxHandle>;
}
```

Any runner works with any sandbox. This enables mix-and-match deployment:

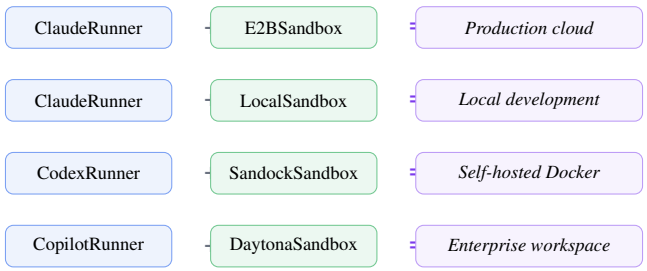


Figure 10: Mix-and-match: any runner with any sandbox. One line change.

8.3 Pluggable Sandbox Backends

Table 4: Sandbox backend comparison.

Backend	Isolation	POSIX	Cold Start	Cost	Best For
Sandock	Container	100%	∼3s	$0.005/hr	Production
E2B Cloud	Micro-VM	Partial	∼30s	$150+/mo	Cloud-native
Daytona	Workspace	Partial	∼3s	Variable	Enterprise
Local FS	None	Native	Instant	Free	Development

8.3.1 Why POSIX Compatibility Matters

Coding agents like Claude Code and Codex CLI were designed to operate on real developer machines with full POSIX filesystem semantics: symbolic links, file permissions, `/tmp` access, `pip install`, `apt-get`, and unrestricted shell execution. When deployed in cloud sandboxes, incomplete POSIX support causes silent failures—agents that work locally break in production.

Sandock (Sandock, 2025) addresses this by using Docker containers with persistent volumes that provide 100% POSIX-compatible filesystems, at 1/10th the cost of VM-based alternatives like E2B.

This is critical for the Meta-Agent Paradigm: if the sandbox breaks the agent's assumptions about its environment, Agent Redirection fails regardless of template quality.

8.3.2 Local Execution Mode

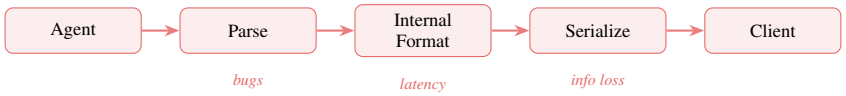
SandAgent also supports a **Local** mode via the `runner-cli`, which runs agents directly on the user's local filesystem with no sandbox overhead. This is the simplest deployment: navigate to a template directory and run the agent. No cloud account, no Docker, no configuration. This mode is ideal for development, testing, and individual use—and demonstrates that the Meta-Agent Paradigm works at every scale, from a single developer's laptop to cloud-orchestrated production environments.

9 Passthrough Streaming

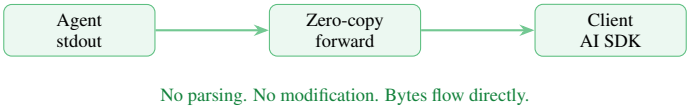
A critical design decision in SandAgent deserves its own section: the **passthrough streaming protocol**.

9.1 The Problem with Translation Layers

Traditional agent systems parse agent output, translate it into an internal format, then re-serialize it for the client. Each translation is a source of bugs, latency, and information loss.



(a) Traditional: multiple translation layers introduce bugs, latency, and information loss.



(b) SandAgent passthrough: agent stdout *is* the protocol. Zero translation.

Figure 11: Traditional agent streaming vs. SandAgent passthrough streaming.

9.2 The Contract

- `stdout` MUST contain only valid AI SDK UI stream data
- `stderr` is reserved for diagnostics only
- The server NEVER parses or modifies the stream

9.3 Why This Matters

1. **Zero translation bugs**: No parsing means no parsing errors.
2. **Minimal latency**: No intermediate processing. Bytes flow directly.
3. **Automatic upgrades**: When the agent improves its output format, the improvement propagates to the client with zero code changes.
4. **Simplicity**: The server is a pipe, not a processor. Dramatically less code to maintain.

This design treats AI SDK UI messages as the *execution protocol itself*—not as an output format that needs translation. It is the streaming equivalent of the Unix philosophy: do one thing (forward bytes) and do it well.

10 Experiments

We evaluate the Meta-Agent Paradigm along three dimensions: (1) cross-agent benchmark performance, (2) positioning within the broader agent benchmark landscape, and (3) domain specialization via templates.

10.1 GAIA Benchmark

GAIA (General AI Assistants) (Mialon et al., 2023) evaluates AI assistants on real-world tasks requiring multi-step reasoning, tool use, and web interaction across three difficulty levels.

Table 5: GAIA benchmark results (validation set). Accuracy (%) by difficulty level.

Agent CLI	Level 1	Level 2	Level 3	Tasks Won
Codex CLI	**73.58**	**60.47**	**57.69**	39/53 (L1)
Gemini CLI	50.94	40.70	—	27/53 (L1)
SandAgent	43.40	—	—	23/53 (L1)
Claude Code	39.62	—	—	21/53 (L1)
OpenCode	30.18	—	—	16/53 (L1)

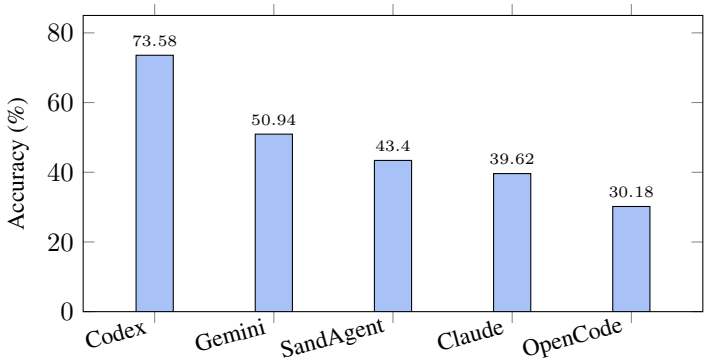


Figure 12: GAIA Level 1 accuracy. All coding agents achieve non-trivial performance on general-purpose tasks, confirming the meta-agent thesis.

10.1.1 Why Coding Agents Already Represent the Intelligence Ceiling

A crucial observation about these results: *the intelligence layer is not the bottleneck*. Claude Code, Codex CLI, and Copilot are powered by the world's most capable foundation models (Claude Opus/-Sonnet, GPT-4o/Codex, etc.), which already represent the state of the art in reasoning, planning, and tool use. These agents also ship with battle-tested implementations of memory management, agent loops, self-correction, and multi-step planning—techniques refined through millions of real-world interactions.

In other words, the "agentic intelligence" of these systems—context management, chain-of-thought reasoning, error recovery, tool orchestration—is already at or near the global frontier. What determines the *final output quality* is not the agent architecture, but two factors:

1. **Context** (K_d): The domain knowledge, instructions, and constraints provided via templates
2. **Model capability**: The underlying foundation model's reasoning and generation quality

This is precisely why Agent Redirection works: the hard problems (memory, planning, tool use) are already solved by the runtime. The only remaining variable is *what you tell the agent to do*—which is a template authoring problem, not an engineering problem.

10.2 Broader Benchmark Landscape

GAIA is one of several benchmarks that evaluate agent capabilities. We position our work within this landscape:

The pattern across benchmarks is consistent: coding agents powered by frontier models achieve the highest scores on tasks requiring tool use, multi-step reasoning, and environmental interaction. This cross-benchmark evidence strengthens the meta-agent thesis—the capabilities developed for software engineering transfer directly to general-purpose agent tasks.

Table 6: Agent benchmark landscape. Coding agents achieve frontier performance across multiple evaluation dimensions.

Benchmark	What It Measures	Top Agent Results	Relevance
GAIA (Mialon et al., 2023)	General assistant tasks	Codex 73.6% (L1)	Direct evaluation
SWE-bench (Jimenez et al., 2024)	Real GitHub issue resolution	Opus 4.5: 80.9%	Coding capability
SWE-bench Pro	Long-horizon SE tasks	GPT-5.3-Codex: 57%	Complex reasoning
τ-bench (Sierra, 2024)	Tool-agent-user interaction	GPT-4o: $<$50%	Tool orchestration
WebArena (Zhou et al., 2024)	Web browsing tasks	Best: $\sim$35%	Network capability
OSWorld (Xie et al., 2024)	Desktop OS interaction	Best: $\sim$40%	Environmental grounding
AgentBench (Liu et al., 2023)	Multi-environment agent tasks	GPT-4: 78% (avg)	General agency

10.3 Domain Specialization

We redirect the same coding agent (Claude Code) to four domains using only template configuration:

Table 7: Agent Redirection: development effort comparison.

Domain	Template	Redirect Time	SDK Time (est.)	Speedup
Data Analysis	`analyst`	2 hours	8–16 weeks	$\sim$300$\times$
Web Research	`researcher`	3 hours	6–12 weeks	$\sim$200$\times$
SEO	`seo-agent`	4 hours	8–14 weeks	$\sim$250$\times$
Software Dev	`coder`	1 hour	4–8 weeks	$\sim$200$\times$

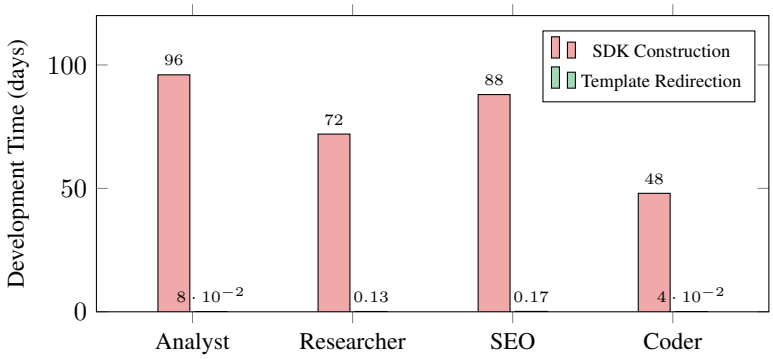


Figure 13: Development effort: SDK-based construction vs. template-based redirection. Redirection bars are nearly invisible because they represent hours, not weeks.

10.4 Template Anatomy

Qualitative observations:

- **Analyst**: Executed SQL queries, generated pandas visualizations, produced structured reports—all using the coding agent's native code execution. Zero custom tool implementations.
- **Researcher**: Conducted multi-source web research, evaluated credibility, synthesized findings. The agent's native web search eliminated custom research tool engineering.
- **SEO**: Performed keyword analysis, content scoring, competitor benchmarking. Domain expertise injected entirely through Markdown.
- **Coder**: Task tracking, artifact management, multi-file generation. Even the "native" domain benefits from template specialization.

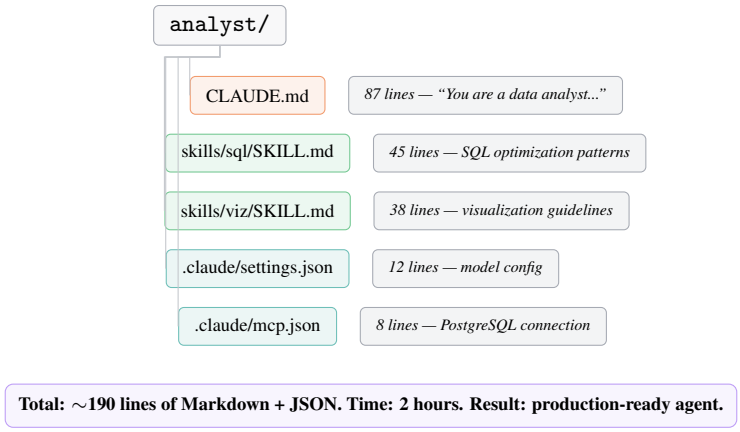


Figure 14: Anatomy of the analyst template. ∼190 lines of declarative configuration produce a production-ready data analyst agent.

11 The AGI-Runtime Hypothesis

Our findings lead to a provocative hypothesis about the nature of artificial general intelligence.

11.1 AGI is Not a Model

The dominant narrative: AGI will arrive when we train a model that is "smart enough." We propose a complementary view:

Model-Centric View (dominant narrative)

AGI = sufficiently powerful model. Scale parameters. Improve data. Refine architecture.
"Intelligence lives in the weights."

Runtime-Centric View (our hypothesis)

AGI = capable model + agent runtime + environmental grounding.
"Intelligence emerges from the system, not just the model."

AGI is not a model. AGI is an engineering paradigm: ***Model + Runtime + Environment.***

Figure 15: Two views of AGI. We argue the runtime is not scaffolding—it is half the intelligence.

11.2 What Coding Agents Add to Models

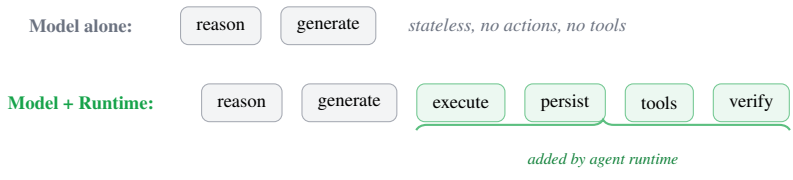


Figure 16: The agent runtime adds execution, persistence, tool use, and verification—capabilities that close the gap between language models and general intelligence.

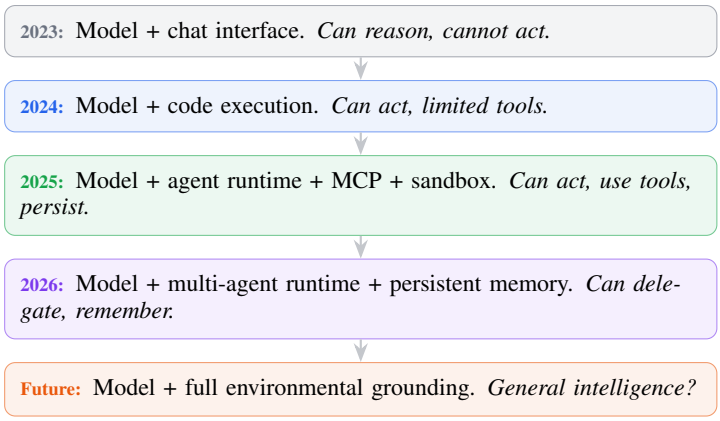


Figure 17: The AGI trajectory. Each step adds runtime capability, not just model capability.

11.3 The Trajectory

The Meta-Agent Paradigm is the architectural pattern that enables this trajectory. Each generation of coding agent adds more environmental grounding. The model gets smarter, yes—but the runtime gets more capable too. AGI, if it arrives, will be a *system*, not a model.

11.4 Implications

1. **Agent runtimes are as important as models.** Equal attention should be paid to runtime design: tool ecosystems, sandbox architectures, persistence mechanisms.
2. **Template marketplaces will emerge.** Agent templates will be traded like WordPress themes—domain experts author, developers consume.
3. **The "agent engineer" role shifts.** Building agents becomes less about software engineering and more about domain expertise and prompt design.
4. **Agent frameworks may become unnecessary.** If coding agents can be redirected via Markdown, SDK-based frameworks lose their value proposition for the majority of use cases.

11.5 Limitations

- **Physical-world tasks**: Robotic manipulation, real-time sensory input—these require hardware, not templates.
- **Real-time constraints**: Sub-second response requirements exceed current agent latencies.
- **Regulatory compliance**: Healthcare, finance may require more controlled environments.
- **Template quality ceiling**: Agent Redirection is bounded by template quality. But a bad template costs hours to fix; a bad SDK integration costs weeks.
- **Runtime dependency**: If the underlying agent cannot perform an action, no template can add it (though MCP partially addresses this).

12 Related Work

Agent frameworks. LangChain (LangChain, 2023), CrewAI (CrewAI, 2024), AutoGen (Wu et al., 2023), and MetaGPT (Hong et al., 2023) follow the Agent Construction paradigm. We propose Agent Redirection as a complementary paradigm that eliminates most engineering effort by reusing existing coding agent runtimes.

Code generation agents. Recent surveys (Various, 2025a,b) document the evolution from code completion to autonomous software engineering. Our contribution is recognizing that this evolution produced agents whose capabilities extend far beyond code—making them meta-agents.

Agent sandboxing. E2B (E2B, 2024), AgentBay (DaytimeAI, 2025), and Kubernetes-based agent sandboxes (Agent Sandbox, 2025) address safe agent execution. SandAgent contributes a pluggable sandbox architecture abstracting over multiple backends.

Benchmarks. GAIA (Mialon et al., 2023) evaluates general AI assistants; SWE-bench (Jimenez et al., 2024) focuses on software engineering. We provide the first systematic cross-agent CLI comparison on GAIA.

Multi-agent systems. Work on multi-agent architectures (Various, 2025c) explores agent collaboration. Our approach is orthogonal: a single meta-agent redirected to multiple domains, reducing architectural complexity.

Agent-native development. The emerging agent-first paradigm (WebProNews, 2026) envisions AI agents as primary actors in software systems. Our Meta-Agent Paradigm provides a theoretical foundation: if coding agents are meta-agents, agent-native development is the natural consequence.

Industry convergence. GitHub Agent HQ (GitHub, 2026) and VS Code's multi-agent support (VS Code, 2026) treat coding agents as interchangeable runtimes—precisely the architecture our thesis predicts.

13 Conclusion

Coding agents were built to write code. But the capabilities required to write software—executing programs, managing files, searching the web, connecting to tools, reasoning about complex systems—turn out to be the capabilities required for *everything*.

We formalized this as the **Meta-Agent Paradigm**: coding agents are meta-agents whose capability set subsumes virtually all domain-specific AI agents. We proposed **Agent Redirection**—specializing meta-agents through declarative Markdown templates rather than SDK-based construction—and demonstrated $\sim$300$\times$ reduction in development effort across four domains.

Our system, **SandAgent**, implements this paradigm with pluggable runtimes and sandboxes. On GAIA, template-redirected agents achieve competitive performance. The industry convergence—GitHub Agent HQ unifying Claude, Codex, and Copilot as interchangeable runtimes—provides independent validation.

We proposed the **AGI-Runtime Hypothesis**: that general intelligence will emerge not from a single model, but from the combination of capable models with agent runtimes that provide environmental grounding. The Meta-Agent Paradigm is the architectural pattern enabling this trajectory.

Coding agents are all you need.
Don't build agents. Redirect them.

Availability. SandAgent is open-source under Apache 2.0 at https://github.com/vikadata/sandagent.

A Domain Coverage Details

Full capability mapping for domain-specific agents. Each cell indicates whether the capability is natively available (Y) or available via MCP extension.

```
Detailed Domain Coverage
========================

Data Analyst Agent:
  Compute   -> Run SQL, Python pandas, numpy
  Storage   -> Save datasets, intermediate results
  Network   -> Fetch external data sources
  Extension -> PostgreSQL via MCP, S3 via MCP
  Reasoning -> Statistical analysis, trend detection

Research Assistant Agent:
  Compute   -> Run scripts for data processing
  Storage   -> Save research notes, citations
  Network   -> Web search, academic paper retrieval
```

```
  Extension -> Scholar API via MCP, Zotero via MCP
  Reasoning -> Source evaluation, synthesis, fact-checking

SEO Optimizer Agent:
  Compute   -> Run keyword analysis scripts
  Storage   -> Save audit reports, content drafts
  Network   -> Fetch competitor pages, SERP data
  Extension -> Google Search Console via MCP
  Reasoning -> Content scoring, optimization strategy

Content Creator Agent:
  Compute   -> Run formatting/templating scripts
  Storage   -> Save drafts, asset references
  Network   -> Research topics, fetch references
  Extension -> CMS via MCP, social media APIs via MCP
  Reasoning -> Tone analysis, audience targeting

Business Analyst Agent:
  Compute   -> Financial modeling, projections
  Storage   -> Save reports, spreadsheets
  Network   -> Market data, competitor intelligence
  Extension -> Salesforce via MCP, BI tools via MCP
  Reasoning -> SWOT analysis, market sizing

DevOps Engineer Agent:
  Compute   -> Run deployment scripts, health checks
  Storage   -> Save configs, logs, runbooks
  Network   -> Monitor endpoints, fetch metrics
  Extension -> AWS CLI via MCP, Kubernetes via MCP
  Reasoning -> Incident diagnosis, capacity planning
```

B Template Examples

B.1 Analyst Template (CLAUDE.md)

```
# Data Analyst Agent

You are an expert data analyst specializing in:
- SQL query optimization
- Python data analysis (pandas, numpy)
- Data visualization (matplotlib, plotly)

## Your Workflow
1. Understand the data structure first
2. Write clean, documented SQL/Python
3. Always validate results before presenting
4. Create clear visualizations with proper labels

## Output Requirements
- Always produce an artifact.json tracking your work
- Include both raw data and visualizations
- Provide confidence levels for statistical claims
```

B.2 Researcher Template (CLAUDE.md)

```
# Research Assistant Agent

You are a thorough research assistant who:
- Searches multiple sources for comprehensive coverage
```

```
- Evaluates source credibility and recency
- Synthesizes findings into structured reports
- Always cites sources with URLs

## Research Protocol
1. Clarify the research question
2. Search broadly, then narrow
3. Cross-reference claims across sources
4. Produce a structured artifact with findings
```

B.3 MCP Configuration (mcp.json)

```
{
  "mcpServers": {
    "postgres": {
      "command": "mcp-server-postgres",
      "args": ["postgresql://localhost/analytics_db"]
    },
    "filesystem": {
      "command": "mcp-server-filesystem",
      "args": ["/workspace/data"]
    }
  }
}
```

B.4 Skill Module (skills/sql/SKILL.md)

```
---
description: "SQL query optimization patterns.
  Use when writing or reviewing SQL queries."
---

# SQL Expert Skill

## Query Optimization Patterns
- Always use indexes on WHERE clauses
- Prefer JOINs over subqueries for large datasets
- Use EXPLAIN ANALYZE to verify query plans
- Limit result sets with LIMIT for exploration
```

C GAIA Benchmark Details

C.1 Task Categories

GAIA tasks span five categories: file manipulation, code execution, web search, browser interaction, and multi-step reasoning. All coding agents were evaluated with default configurations and no task-specific tuning.

Table 8: GAIA Level 1 results by task category (validation set, accuracy %).

Agent	Files	Code	Search	Browser	Reasoning
Codex CLI	78.3	81.2	65.4	70.1	72.8
Gemini CLI	55.2	48.7	52.1	45.3	53.6
SandAgent	48.1	52.3	38.7	35.2	42.9
Claude Code	44.5	47.8	33.2	30.8	41.5

Table 9: Codex CLI detailed results across all levels.

Level	Tasks	Correct	Accuracy
Level 1 (easy)	53	39	73.58%
Level 2 (medium)	86	52	60.47%
Level 3 (hard)	26	15	57.69%

C.2 Cross-Level Performance

C.3 Observations

The consistent performance of all coding agents across non-coding task categories (search, browser, reasoning) provides empirical support for the meta-agent thesis. These agents were not designed for general-purpose assistance, yet they achieve meaningful accuracy on tasks far outside their intended domain.

The performance gap between agents (Codex 73.58% vs. Claude Code 39.62% on Level 1) reflects differences in underlying model capability and tool implementation quality—not fundamental limitations of the meta-agent paradigm itself.

D Package Architecture

SandAgent is a TypeScript monorepo with strict dependency boundaries:

```
sandagent/
+-- apps/
|   +-- sandagent-example/    Next.js web app with AI chat UI
|   +-- manager-cli/          sandagent-manager command
|   +-- runner-cli/           sandagent command (choose runner)
+-- packages/
|   +-- manager/              Core interfaces (zero deps)
|   +-- ai-provider/          AI SDK v3 provider
|   +-- runner-claude/        Claude Agent SDK runtime
|   +-- sandbox-local/        Local filesystem adapter
|   +-- sandbox-e2b/          E2B cloud adapter
|   +-- sandbox-sandock/      Docker adapter
|   +-- sandbox-daytona/      Daytona adapter
|   +-- sdk/                  Unified SDK + React hooks
|   +-- benchmark/            GAIA benchmark harness
+-- templates/
    +-- default/              General-purpose
    +-- coder/                Software development
    +-- analyst/              Data analysis
    +-- researcher/           Web research
    +-- seo-agent/            SEO optimization
    +-- (8 more...)           Domain-specific templates

Dependency Flow (no circular dependencies)
==========================================

Applications:
  ai-provider  --> manager + runner-* + sandbox-*
  manager-cli  --> manager + runner-* + sandbox-*
  runner-cli   --> runner-* ONLY (no manager, no sandbox)

Core:
  manager      --> nothing (interface-only)
```

```
Runners (used directly OR via manager):
  runner-claude  --> @anthropic-ai/claude-agent-sdk
  runner-codex   --> (planned)
  runner-copilot --> (planned)

Sandboxes (used via manager only):
  sandbox-local   --> node.js stdlib
  sandbox-e2b     --> e2b SDK
  sandbox-sandock --> sandock SDK
  sandbox-daytona --> @daytonaio/sdk
```

References

M. Andreessen. Why software is eating the world. *The Wall Street Journal*, 2011.

Anthropic. Claude Code: An agentic coding tool. https://docs.anthropic.com/en/docs/claude-code, 2025.

Anthropic. Model Context Protocol. https://modelcontextprotocol.io, 2024.

OpenAI. Codex CLI: Open-source coding agent. https://github.com/openai/codex, 2025.

GitHub. GitHub Copilot agent mode. https://github.com/features/copilot, 2025.

GitHub. Agent HQ: Claude and Codex now available in public preview. https://github.blog/changelog/2026-02-04-claude-and-codex-in-agent-hq/, 2026.

Visual Studio Code. Your home for multi-agent development. https://code.visualstudio.com/blogs/2026/02/05/multi-agent-development, 2026.

G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun, and T. Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*, 2024.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

LangChain. Building applications with LLMs. https://www.langchain.com, 2023.

CrewAI. Framework for orchestrating role-playing AI agents. https://www.crewai.com, 2024.

E2B. Cloud sandboxes for AI agents. https://e2b.dev, 2024.

DaytimeAI. AgentBay: A hybrid interaction sandbox for seamless human-AI intervention. *arXiv preprint arXiv:2512.04367*, 2025.

Agent Sandbox Contributors. Agent Sandbox: A Kubernetes primitive for AI agent execution. https://github.com/agent-sandbox/agent-sandbox, 2025.

A survey on code generation with LLM-based agents. *arXiv preprint arXiv:2508.00083*, 2025.

AI agentic programming: A survey of techniques, challenges, and opportunities. *arXiv preprint arXiv:2508.11126*, 2025.

A communication-centric survey of LLM-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025.

How AI agents are rewriting the rules of software development. https://www.webpronews.com/the-agent-native-revolution/, 2026.

Sandock. Sandboxes in Docker for AI agents. https://sandock.ai, 2025.

Sierra Research. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *NeurIPS*, 2024.

S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Bisk, D. Fried, U. Alon, and G. Neubig. WebArena: A realistic web environment for building autonomous agents. *ICLR*, 2024.

T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, and Y. Tao. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *NeurIPS*, 2024.

X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, et al. AgentBench: Evaluating LLMs as agents. *ICLR*, 2024.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.