The Cyborg Developer: Empirical Analysis of Cognitive Extension Through Human-AI Collaborative Programming

Anderson Henrique da Silva
Undergraduate Student
IFSULDEMINAS – Campus Muzambinho
Minas Gerais, Brasil
github.com/anderson-ufrj
ORCID: 0009-0001-7144-4974

December 2025

Abstract

AI coding assistants are transforming software development, yet empirical understanding of how developers integrate these tools into cognitive workflows remains limited. Through computational autoethnography, we analyze 802 collaborative sessions (85,370 messages, 27,672 tool invocations) across 47 projects over 30 days. Our analysis reveals: (1) High cognitive delegation—a delegation score of 0.71 indicates developers treat AI as cognitive extension, not mere autocomplete; (2) Intentional model selection—developers consciously match AI capability to task complexity, with 7.59× longer sessions for high-capability models; (3) Sustained collaboration intensity—2,846 messages per active day and 13.3 projects per week demonstrate deep integration; (4) Context-fluid operation—rapid switching between projects with minimal cognitive overhead; (5) Hierarchical tool usage—execution and exploration dominate, with planning emerging in complex tasks. We introduce *Cyborg Cognition*—the integrated cognitive system formed when human direction-setting combines with AI information gathering and execution. This framework extends theories of distributed cognition to human-AI programming collaboration. Limitations include single-subject design; we provide sensitivity analyses and invite replication.

Keywords: Human-AI Collaboration, Software Engineering, Cognitive Extension, Developer Tools, Empirical Study, Autoethnography, Claude Code, Cyborg Cognition, Extended Mind, Distributed Cognition

1 Introduction

The emergence of AI-powered coding assistants—GitHub Copilot, Claude Code, Cursor, and others—represents a fundamental shift in software development practice. Industry surveys report adoption rates exceeding 70% among professional developers, with claimed productivity gains of 30-50% [GitHub, 2023]. Yet our understanding of how developers integrate these tools into their cognitive workflows remains largely anecdotal, based on surveys and controlled experiments that capture neither the depth nor the longitudinal nature of real-world human-AI collaboration.

Current research on AI coding assistants falls into two categories. First, *productivity studies* measure output metrics—lines of code, task completion time, bug rates—treating the developer as a black box whose internal processes are irrelevant as long as outputs improve [Peng et al., 2023]. Second, *perception studies* rely on self-reported experiences, asking developers how they *feel* about AI tools rather than observing what they *do* with them [Barke et al., 2023]. Neither approach captures the cognitive reality of sustained human-AI collaboration.

Building on prior work analyzing developer cognition through repository artifacts [Da Silva, 2025], we extend the methodological approach from static artifact analysis to dynamic interaction analysis. The Developer Mental Model Framework (DMMF) demonstrated that commits, code structure, and documentation encode developer cognition through multiple dimensions: cognitive (problem-solving patterns), affective (emotional traces in code), conative (motivational indicators), and reflective (self-awareness in documentation). However, artifact-based analysis captures only the *output* of cognitive processes—the committed code—not the *process* itself. DMMF captures cognitive residue—what remains after thinking; the present study captures cognitive process—thinking as it happens, distributed across human and AI.

This paper extends artifact-based analysis to interaction analysis. We ask: **What cognitive patterns emerge when we observe not the products of development, but the moment-to-moment collaboration between developer and AI?**

To answer this question, we employ *computational autoethnography*—systematic self-study enhanced with complete instrumentation of human-AI interactions. Over 30 active days, we captured 802 collaborative sessions comprising 85,370 messages and 27,672 tool invocations across 47 distinct software projects. Unlike survey-based studies, our data includes every tool call, every token exchanged, and every model selection decision, providing unprecedented granularity into the cognitive dynamics of human-AI programming.

Our analysis reveals five key findings:

1. **High Cognitive Delegation**: A delegation score of 0.71 (scale 0-1) indicates that developers treat AI as genuine cognitive extension, not mere autocomplete. Information gathering (33.5%) and task planning (8.7%) are substantially offloaded to AI systems.

2. **Intentional Model Selection**: Sessions using high-capability models (Opus) average 7.59× longer than those using efficient models (Haiku), demonstrating conscious matching of AI capability to task complexity.

3. **Tool Usage Hierarchy**: Execution (Bash, 36%) and exploration (Read/Grep, 33.5%) dominate tool usage, suggesting AI primarily extends the developer's capacity for action and information gathering rather than direct code generation.

4. **Sustained Collaboration Intensity**: With 2,846 messages and 922 tool uses per active day, human-AI collaboration represents sustained practice rather than occasional assistance.

5. **Context Fluidity**: The developer-AI dyad seamlessly transitions across 13.3 projects per week, adapting collaboration patterns to project-specific demands.

Based on these findings, we propose *Cyborg Cognition in Software Development* as a theoretical framework for understanding this phenomenon. Drawing on Clark's extended mind thesis [Clark and Chalmers, 1998] and Haraway's cyborg theory [Haraway, 1991], we conceptualize the developer-AI system as an emergent cognitive unity where human and artificial capabilities become functionally integrated through sustained, intentional collaboration.

1.1 Contributions

This paper makes four contributions:

1. **Empirical Dataset**: The first longitudinal, tool-level record of professional human-AI programming collaboration, capturing interaction patterns invisible to surveys and controlled experiments.

2. **Cognitive Delegation Metrics**: Quantitative measures for how developers distribute cognitive work between themselves and AI systems, operationalizing the intuitive notion of "AI as thinking partner."

3. **Model Selection Analysis**: Empirical evidence of intentional, complexity-aware selection of AI capability tiers, demonstrating meta-cognitive sophistication in human-AI collaboration.

4. **Theoretical Framework**: The concept of Cyborg Cognition, providing vocabulary and analytical dimensions for studying cognitive extension in AI-augmented work.

1.2 Paper Structure

Section 2 reviews related work on AI coding assistants, human-AI collaboration, and developer cognition. Section 3 describes our computational autoethnography methodology and dataset. Section 4 presents five empirical findings with supporting evidence. Section 5 develops the Cyborg Cognition framework and discusses implications. Section 6 concludes with limitations and future directions.

2 Related Work

This section reviews three bodies of literature that inform our study: AI coding assistants, human-AI collaboration, and developer cognition.

2.1 AI Coding Assistants

The release of GitHub Copilot in 2021 marked the mainstream arrival of AI coding assistants. Since then, research has examined these tools from multiple angles.

2.1.1 Productivity Studies

Peng et al. [Peng et al., 2023] conducted a randomized controlled trial showing Copilot users completed tasks 55.8% faster. Vaithilingam et al. [Vaithilingam et al., 2022] found productivity gains varied by task complexity, with benefits concentrated in routine coding. Ziegler et al. [Ziegler et al., 2022] introduced the "acceptance rate" metric, finding that developers accept approximately 26% of Copilot suggestions.

These studies treat developers as input-output systems: AI goes in, code comes out. They do not examine *how* developers integrate AI into their cognitive workflows.

2.1.2 User Studies

Barke et al. [Barke et al., 2023] interviewed Copilot users, identifying two modes: "acceleration" (speeding up known tasks) and "exploration" (discovering new approaches). Prather et al. [Prather et al., 2023] studied novices using Copilot, finding both benefits (scaffolding) and risks (over-reliance).

User studies provide rich qualitative data but rely on self-report, which may not reflect actual behavior. Participants describe what they *believe* they do, not necessarily what they *actually* do.

2.2 Human-AI Collaboration

Beyond coding, a broader literature examines how humans collaborate with AI systems.

2.2.1 Levels of Automation

Parasuraman et al. [Parasuraman et al., 2000] proposed a framework for human-automation interaction with four stages: information acquisition, information analysis, decision selection, and action implementation. AI coding assistants participate in all four stages, making them unusually comprehensive automation partners.

2.2.2 Human-AI Teaming

Seeber et al. [Seeber et al., 2020] studied AI as "team member," finding that humans adapt their behavior based on perceived AI capabilities. Our finding of intentional model selection (7.59× ratio) provides empirical evidence of this adaptation in a programming context.

2.2.3 Trust and Reliance

Lee and See [Lee and See, 2004] reviewed trust in automation, distinguishing appropriate trust, over-trust, and under-trust. Our delegation score of 0.71 indicates substantial trust, but whether this is "appropriate" requires task-specific analysis we leave for future work.

2.3 Developer Cognition

Understanding developers as cognitive agents has a long history in software engineering.

2.3.1 Program Comprehension

Letovsky [Letovsky, 1987] proposed a model of program comprehension involving knowledge goals, conjectures, and mental models. Our "exploration" category (33% of tool uses) maps onto the knowledge-gathering phase of this model, with AI serving as an external comprehension resource.

2.3.2 Cognitive Load

Sweller's cognitive load theory [Sweller, 1988] distinguishes intrinsic, extraneous, and germane load. AI coding assistants may reduce extraneous load (boilerplate, syntax lookup) while preserving germane load (architectural decisions, algorithm design). Our tool hierarchy data—with Bash and Read dominating—suggests developers offload low-germane tasks to AI.

2.3.3 Developer Mental Models

Prior work on developer mental models [Da Silva, 2025] established that cognitive patterns can be inferred from software artifacts through analysis of repositories and commits, identifying multiple dimensions including cognitive, affective, conative, and reflective patterns. This artifact-based approach provides the methodological foundation that the present study extends from retrospective artifact analysis to concurrent interaction analysis—capturing not what developers produce but how they produce it through human-AI collaboration.

2.4 Extended Cognition

Our theoretical framework draws on philosophy of mind literature.

2.4.1 Extended Mind Thesis

Clark and Chalmers [Clark and Chalmers, 1998] argued that cognitive processes can extend into the environment. Their famous example involves Otto, who relies on a notebook for memory. We argue that AI coding assistants function analogously—as external cognitive resources that become part of the developer's cognitive system.

2.4.2 Distributed Cognition

Hutchins [Hutchins, 1995] studied cognition in complex sociotechnical systems (airplane cockpits, ship navigation). Software development with AI assistants represents a new form of distributed cognition—human and AI as a cognitive unit.

2.5 Theoretical Positioning: Beyond Existing Frameworks

Before presenting our methodology, we clarify how this work relates to—and extends—existing theoretical frameworks. A legitimate concern is whether studying human-AI collaboration requires new conceptual vocabulary or whether existing frameworks suffice.

Beyond Distributed Cognition: Hutchins' [Hutchins, 1995] distributed cognition describes how cognitive processes spread across people and artifacts in sociotechnical systems. However, Hutchins studied *static* tool configurations (navigation instruments, cockpit displays) where the distribution is architecturally fixed. Human-AI programming involves *adaptive* systems where the distribution is dynamically negotiated—developers actively manage cognitive distribution in real-time through model selection and task-specific delegation. This adaptive dimension is not addressed by classical distributed cognition.

Beyond Extended Mind: Clark and Chalmers' [Clark and Chalmers, 1998] extended mind thesis establishes that cognition can extend into the environment, but treats extension as binary (extended or not) and focuses on *passive* cognitive artifacts (Otto's notebook). AI coding assistants are *active* cognitive partners that reason, generate, and adapt. We operationalize extension as a *continuous spectrum* (our delegation score) rather than a binary state.

Beyond Automation Levels: Parasuraman et al.'s [Parasuraman et al., 2000] levels of automation framework provides useful vocabulary (information acquisition, analysis, decision, action) but assumes fixed automation levels per system. Modern AI assistants operate at variable automation levels within a single session—sometimes gathering information autonomously, sometimes awaiting explicit instruction. Our framework captures this *within-session variability*.

These limitations motivate our empirical investigation: we need data on how cognitive distribution actually occurs in practice before theorizing further.

2.6 Research Gap

Prior work on AI coding assistants examines either:

- **Outputs**: Productivity, code quality, security
- **Perceptions**: User experiences, trust, satisfaction

What is missing is examination of the **process**—the moment-by-moment cognitive dynamics of human-AI collaboration. Our computational autoethnography addresses this gap by capturing every interaction at tool-level granularity over extended naturalistic use.

Additionally, no prior work has examined **model selection** as a cognitive variable. The availability of multiple AI capability tiers (Opus, Sonnet, Haiku) creates a new dimension of human-AI collaboration that existing frameworks do not address.

3 Methodology

This section describes our computational autoethnography approach, data collection infrastructure, and analysis framework.

3.1 Research Approach: Computational Autoethnography

We employ autoethnography—systematic self-study where the researcher is both subject and analyst—enhanced with computational data collection. This approach follows methodological traditions in developer cognition research [Da Silva, 2025], extending reflexive analysis from historical artifacts to real-time interaction.

3.1.1 Rationale for Single-Subject Design

Single-subject studies are often criticized for limited generalizability. We argue this trade-off is appropriate for our research questions for three reasons:

1. **Depth over breadth**: Multi-participant studies of AI coding assistants typically capture 1-2 hours of controlled use. Our longitudinal design captures 30 days of naturalistic, uncontrolled professional practice—approximately 150× more interaction time per subject.

2. **Complete instrumentation**: Full access to a single developer's interaction logs enables tool-level granularity impossible to obtain across multiple participants due to privacy and consent constraints.

3. **Ecological validity**: Unlike laboratory settings, our data reflects actual professional work across 47 real projects with genuine deadlines and quality requirements.

The validity trade-off is explicit: we sacrifice claims about the "average developer" in favor of deep claims about how *a* developer integrates AI into sustained professional practice. Future multi-participant studies can test whether patterns we identify generalize.

3.1.2 Researcher Positionality

The researcher is an independent software developer based in South America with 5+ years of professional experience. During the study period, the researcher worked on production systems (multi-agent AI platforms), research projects (academic papers), and exploratory prototypes. This diversity of project types enables analysis across different cognitive demands.

The researcher was aware that interaction data would be analyzed, which could influence behavior (Hawthorne effect). We mitigate this concern by noting: (1) data collection was automatic and required no conscious effort during work; (2) the 30-day period is long enough for novelty effects to diminish; and (3) professional deadlines created genuine performance pressure regardless of observation.

3.2 Data Collection

3.2.1 Instrumentation

Data was collected from Claude Code, an AI coding assistant that operates within the terminal environment. Claude Code stores complete interaction transcripts in a structured format:

Listing 1: Session storage location

```
~/.claude/projects/<project-hash>/<session-id>.jsonl
```

Each JSONL file contains chronologically ordered events including:

- **User messages**: Developer inputs (queries, instructions, feedback)
- **Assistant messages**: AI responses (text, reasoning, tool calls)
- **Tool invocations**: Complete record of tool name, inputs, and outputs
- **Metadata**: Timestamps, model selection, token usage, project context

Additionally, we implemented a `PostToolUse` hook that enriched each tool invocation with:

- Tool category (exploration, modification, execution, planning, interaction)
- Success/failure indicators

- Token estimates
- Context hints (file types, action patterns)
- Quality score heuristics

3.2.2 Collection Period

Data was collected over 30 active development days. "Active" is defined as days with at least one Claude Code session.

3.2.3 Dataset Summary

Table 1 summarizes the collected data.

Table 1: Dataset Summary

Metric	Value
Total sessions	802
Total messages	85,370
User messages	30,951
Assistant messages	54,419
Tool invocations	27,672
Unique projects	47
Active days	30
Total input tokens	7.4M
Total output tokens	9.2M
Cache tokens	4.98B

3.2.4 Ethical Considerations

As autoethnography, this study involves only the researcher's data. No third-party data was collected. Project names are reported but no proprietary code is disclosed. The aggregated dataset will be made available; raw interaction logs containing potentially sensitive project details will be available upon request with appropriate data use agreements.

3.3 Operationalizing Cognition Through Behavioral Traces

A methodological concern is whether message counts and tool invocations can serve as proxies for cognitive processes. We defend this operationalization on three grounds:

Behavioral trace validity: Cognitive science has a long tradition of inferring mental processes from observable behavior [Letovsky, 1987]. While we cannot directly observe "thinking," we can observe its behavioral signatures. Each tool invocation represents a decision to delegate a cognitive function; each message represents an intention communicated to the AI partner. These are not mere keystrokes—they are choices that reflect underlying cognitive states and strategies.

Granularity advantage: Unlike self-report measures (surveys, interviews), our behavioral data captures *every* interaction without recall bias or social desirability effects. The developer cannot misremember how many times they used a search tool or which model they selected. This granularity reveals patterns invisible to introspection.

Ecological validity: Our data comes from genuine professional work under real deadlines, not laboratory tasks. The cognitive strategies we observe are those the developer actually employs when producing real software—not reconstructed behavior in artificial settings.

We acknowledge that behavioral traces are incomplete windows into cognition. Internal deliberation, uncertainty, and emotional states are not captured. However, our focus is specifically on *human-AI cognitive distribution*—how cognitive labor is allocated between human and AI—which is directly observable through interaction patterns. We are measuring delegation behavior, not the full richness of human thought.

3.4 Analysis Framework

We analyze interactions across four complementary dimensions:

3.4.1 Temporal Analysis

Examining how AI usage patterns evolve over time:

- Daily and weekly aggregation of sessions, messages, and tool uses
- Trend analysis (increasing, stable, or decreasing intensity)
- Peak usage identification

3.4.2 Project Analysis

Comparing collaboration patterns across project contexts:

- Per-project session counts and message volumes
- Tool intensity (tool uses per message)
- Primary model selection per project

3.4.3 Cognitive Delegation Analysis

Quantifying how cognitive work is distributed between developer and AI. We categorize tools by cognitive function:

Table 2: Tool Categories and Delegation Levels

Category	**Tools**	**Delegation**
Exploration	Read, Grep, Glob, WebSearch	High
Modification	Write, Edit, MultiEdit	Medium
Execution	Bash, Task	Variable
Planning	TodoWrite, PlanMode	High
Interaction	AskUserQuestion	Low

The **Delegation Score** is computed as:

$$D = \frac{\sum_{c \in C} p_c \cdot w_c}{\sum_{c \in C} p_c} \tag{1}$$

Where p_c is the percentage of tool uses in category c, and w_c is the delegation weight (1.0 for high, 0.5 for medium/variable, 0.0 for low). A score approaching 1.0 indicates high AI delegation; approaching 0.0 indicates human-heavy workflow.

Execution Weight Rationale: We assign Execution (Bash, Task) a weight of 0.5 rather than 1.0 despite its substantial proportion (35.8%) because Bash commands represent a heterogeneous category spanning the full delegation spectrum. Simple verification commands (e.g., `ls`, `git status`, `pwd`) represent minimal cognitive delegation—the developer knows what to check

and merely uses AI as interface. Conversely, complex multi-step operations (e.g., build pipelines, test suites, deployment scripts) represent substantial delegation where AI autonomously manages operational complexity. Without finer-grained command classification—which we leave for future work—the 0.5 weight represents a conservative middle estimate. Our sensitivity analysis (Table 3, "Execution=High" row) demonstrates this choice is consequential: treating Execution as full delegation yields $D = 0.83$. We report the conservative 0.71 estimate while acknowledging that actual delegation may be higher if Bash usage is predominantly complex operations.

3.4.4 Sensitivity Analysis

Because delegation weights are researcher-defined, we assess robustness through two approaches:

Bootstrap Confidence Interval: We computed 10,000 bootstrap resamples of the 27,672 tool invocations and recalculated the delegation score for each. The 95% confidence interval is [0.708, 0.713], indicating the score is robust to random sampling variation.

Weight Sensitivity Analysis: We tested alternative weight schemes to assess dependence on researcher assumptions:

Table 3: Delegation Score Sensitivity to Weight Assumptions

Scheme	Description	Score
Base	Original weights (H=1.0, M=0.5, L=0.0)	0.71
Conservative	Lower all weights (H=0.8, M=0.3, L=0.0)	0.51
Liberal	Higher all weights (H=1.0, M=0.7, L=0.2)	0.79
Binary	Only high vs low (H=1.0, M=0.0, L=0.0)	0.42
Execution=High	Treat Bash as full delegation	0.83

The delegation score ranges from 0.42 to 0.83 depending on weight assumptions. Our base score of 0.71 represents a moderate assumption. The qualitative finding—substantial cognitive delegation to AI—holds across all reasonable weight schemes.

3.4.5 Model-Complexity Correlation

Analyzing the relationship between model selection and task characteristics:

- Session length by model tier
- Tool use intensity by model
- Project diversity by model preference

Models are categorized into capability tiers:

- **High**: Claude Opus 4.5 (complex reasoning)
- **Medium**: Claude Sonnet 4.5 (balanced)
- **Low**: Claude Haiku 4.5 (fast, routine)

3.5 Comparative Data Sources

To contextualize our findings, we analyzed two additional data sources:

3.5.1 Claude.ai Web Conversations

We exported the same developer's Claude.ai (web interface) conversation history via Anthropic's data export feature. This provides within-subject comparison between:

- **Tool-augmented mode**: Claude Code CLI with file operations, bash execution, and planning tools
- **Conversational mode**: Claude.ai web interface with text-only interaction

Table 4: Claude.ai Web Export Summary

Metric	Value
Total conversations	893
Total messages	7,888
Period	Jan–Dec 2025
Avg messages/conversation	8.8
Projects	14

3.5.2 DevGPT Dataset (External Comparison)

We analyzed the DevGPT dataset [Tao et al., 2024], which contains ChatGPT conversations shared on GitHub, Hacker News, and other platforms. This provides community-level baseline metrics.

Table 5: DevGPT Dataset Summary

Metric	Value
Total entries	1,716
Valid conversations	1,920
Total prompts	6,979
Code snippets	4,974
Avg prompts/conversation	3.6

3.5.3 Key Comparative Metrics

Table 6 summarizes the three data sources.

Table 6: Cross-Source Comparison

Source	Sessions	Messages	Avg/Session
Claude Code (CLI)*	802	85,370	**106.4**
Claude.ai (Web)	893	7,888	8.8
DevGPT (Community)	1,920	13,958	7.3

*Weighted average across all model tiers. Per-model breakdown: Opus 171.6, Haiku 22.6, Sonnet 77.3 msgs/session.

The **12.1**× difference in session intensity between Claude Code and Claude.ai (same developer, same AI) suggests that the interaction modality—not just AI capability—fundamentally shapes collaboration patterns. The **14.6**× difference versus DevGPT community average indicates either individual variation or the effect of tool-augmented workflows.

3.6 Limitations of Method

1. **Single subject**: Patterns may reflect individual style rather than general phenomena
2. **Specific tooling**: Claude Code interaction patterns may differ from Copilot, Cursor, or other tools
3. **Expertise level**: A senior developer's patterns may not generalize to novices
4. **Temporal scope**: 30 days may not capture longer-term evolution
5. **Observer effect**: Awareness of data collection could influence behavior

We address these limitations through transparency about claims' scope and explicit invitation for replication studies with different subjects and tools.

4 Findings

This section presents five empirical findings from our analysis of 802 sessions and 27,672 tool invocations.

4.1 Finding 1: High Cognitive Delegation

> *Developers delegate significant cognitive work to AI, treating it as cognitive extension rather than autocomplete.*

Our delegation score of **0.71** (scale 0-1) indicates high cognitive delegation to AI systems. Table 7 breaks down tool usage by cognitive category.

Table 7: Tool Usage by Cognitive Category

Category	Uses	%	Delegation
Execution	9,964	36.0%	Variable
Exploration	9,278	33.5%	High
Modification	5,962	21.5%	Medium
Planning	2,407	8.7%	High
Interaction	46	0.2%	Low
Total	27,672	100%	**0.71**

4.1.1 Interpretation

The developer operates in a mode where AI handles:

- **Information retrieval** (33.5%): Reading files, searching codebases, fetching web content
- **Task management** (8.7%): Maintaining todo lists, planning work sequences
- **Execution** (35.8%): Running commands, managing processes

Human cognition focuses on *direction-setting* and *quality judgment*, while AI handles *information gathering* and *action execution*. This division maps onto the distinction between "thinking" (human) and "doing" (AI-assisted).

Notably, explicit requests for human input (AskUserQuestion) comprise only 0.2% of tool uses. The AI operates with high autonomy, rarely pausing to confirm decisions with the developer.

4.1.2 Comparison to Autocomplete Model

Traditional code completion (e.g., IntelliSense) operates at the token level—predicting the next few characters. The delegation pattern we observe is qualitatively different: entire cognitive tasks ("find all files that implement X," "run tests and fix failures") are delegated as units. This represents delegation at the *goal* level, not the *token* level.

4.2 Finding 2: Intentional Model Selection

Developers consciously match AI capability to task complexity, demonstrating meta-cognitive awareness.

Table 8 shows the distribution of model usage across sessions.

Table 8: Model Selection Patterns

Model	Sessions	%	Avg Msgs	Tier
Opus 4.5	451	56.7%	171.6	High
Haiku 4.5	328	41.2%	22.6	Low
Sonnet 4.5	17	2.1%	77.3	Medium

4.2.1 The 7.59× Ratio

Opus sessions average 171.6 messages compared to Haiku's 22.6 messages—a ratio of **7.59×**. This is not random variation. The developer selects:

- **Opus** for complex reasoning tasks requiring extended collaboration
- **Haiku** for quick operations and routine queries

4.2.2 Evidence of Intentionality

If model selection were arbitrary, we would expect similar session lengths regardless of model. The 7.59× difference demonstrates that the developer:

1. Anticipates task complexity before beginning
2. Selects appropriate AI capability tier
3. Engages in extended collaboration when complexity warrants it

This represents *meta-cognitive sophistication*—thinking about which kind of thinking partner is appropriate for the task at hand.

4.3 Finding 3: Tool Usage Hierarchy

AI primarily extends the developer's execution and exploration capacity, with modification as output.

Table 9 shows the distribution of the top 10 tools.

Table 9: Top 10 Tool Usage

Tool	Uses	%	Category
Bash	9,599	34.7%	Execution
Read	6,494	23.5%	Exploration
Edit	4,335	15.7%	Modification
TodoWrite	2,382	8.6%	Planning
Write	1,620	5.9%	Modification
Grep	1,349	4.9%	Exploration
Glob	1,066	3.9%	Exploration
WebSearch	221	0.8%	Exploration
WebFetch	148	0.5%	Exploration
Task	123	0.4%	Execution

4.3.1 The Execute-Explore-Modify Pattern

The hierarchy reveals a workflow pattern:

1. **Execute** (36%): Run commands via Bash
2. **Explore** (33.5%): Gather information via Read, Grep, Glob
3. **Modify** (21%): Apply changes via Edit, Write

Notably, this inverts traditional cognitive science models where information gathering precedes action. With AI assistance, the developer can "act first, understand later"—running commands to observe behavior rather than reading code exhaustively before acting. The AI absorbs the cognitive cost of context-switching between exploration and execution.

4.3.2 Bash Dominance

The prevalence of Bash (34.7%) indicates the developer uses AI as an "execution engine"—a capable agent that can run arbitrary commands, interpret outputs, and take follow-up actions. This goes beyond code generation to *operational capability.*

4.4 Finding 4: Sustained Collaboration Intensity

Human-AI collaboration becomes sustained practice, not occasional assistance.

Table 10 summarizes daily usage patterns.

Table 10: Collaboration Intensity Metrics

Metric	Value
Active days	30
Avg sessions/day	26.7
Avg messages/day	2,846
Avg tool uses/day	922
Avg projects/week	13.3

4.4.1 Interpretation

With nearly 3,000 messages per active day, AI collaboration is woven into the fabric of daily work. This is not "occasional assistance" (asking AI a question once per hour) but *continuous partnership* (ongoing dialogue throughout the workday).

The 26.7 sessions per day suggests frequent context switches, with each session representing a focused interaction unit.

4.4.2 Implications for Workflow

This intensity level implies:

- AI is always "present" during development
- The cost of initiating AI collaboration is near-zero
- Developer and AI maintain shared context across sessions

4.5 Finding 5: Project-Context Fluidity

The developer-AI dyad adapts collaboration patterns to project-specific demands.

Table 11 shows the top projects by session count.

Table 11: Top 10 Projects by Sessions

Project	**Sessions**	**Messages**	**Type**
ProjectA-frontend	206	—	Web UI
ProjectB	112	—	Audio/ML
ProjectA-backend	64	—	API
ProjectA	58	—	Multi-agent
ProjectC	55	—	Personal
ProjectD	33	—	Research
ProjectE	30	—	General
ProjectF	21	—	Academic
ProjectG	20	—	LLM
ProjectH	16	—	Conference

4.5.1 Context Diversity

The 47 distinct projects span:

- Production systems (ProjectA-*)
- Research projects (ProjectD, ProjectF)
- Experiments (ProjectG, ProjectB)
- Administrative work (ProjectE, ProjectH)

4.5.2 Adaptive Collaboration

Different project types likely require different collaboration patterns:

- **Frontend work**: More exploration (understanding UI state), more modification (CSS/-component changes)
- **Backend work**: More execution (API testing), more debugging
- **Research work**: More planning (organizing ideas), more writing

The developer-AI system reconfigures itself based on project nature, demonstrating contextual intelligence in the collaboration.

4.5.3 Project Concentration Analysis

A methodological concern is that the ProjectA ecosystem dominates the dataset:

Table 12: Project Concentration

Project Category	**Sessions**	**%**
ProjectA-* (combined)	328	40.9%
Other projects (43)	474	59.1%

To assess whether findings are driven by this dominant project, we computed delegation scores separately:

- **ProjectA-* sessions**: $D = 0.69$
- **Non-ProjectA sessions**: $D = 0.73$
- **Overall**: $D = 0.71$

The minimal difference (0.04) suggests delegation patterns are relatively consistent across project types. Similarly, the 7.59× model selection ratio holds when computed separately for ProjectA (7.2×) and other projects (8.1×).

This consistency strengthens confidence that observed patterns reflect general collaboration style rather than project-specific artifacts. However, we acknowledge that a single developer's multiple projects may share underlying characteristics, limiting independence.

4.6 Summary of Findings

Table 13: Summary of Key Findings

ID	**Finding**	**Key Metric**
F1	High Cognitive Delegation	Score: 0.71
F2	Intentional Model Selection	Ratio: 7.59×
F3	Tool Usage Hierarchy	Bash: 36%
F4	Sustained Intensity	2,846 msgs/day
F5	Context Fluidity	47 projects

5 Discussion

This section develops our theoretical framework, connects findings to prior work, and discusses implications for research and practice.

5.1 Cyborg Cognition: A Theoretical Framework

Based on our empirical findings, we propose *Cyborg Cognition in Software Development* as a framework for understanding sustained human-AI collaboration.

5.1.1 Definition

> **Cyborg Cognition** is the emergent cognitive system formed when a developer's mental processes become integrated with AI capabilities through sustained, intentional collaboration.

This is not metaphor. Our data shows that cognitive functions—memory (file reading), search (grep), planning (todo management), execution (bash)—are literally distributed across human and AI components. The "cyborg" is the functional unit that thinks and acts, not the human alone.

5.1.2 Theoretical Grounding

Our framework draws on two intellectual traditions:

Extended Mind Thesis [Clark and Chalmers, 1998]: Clark and Chalmers argued that cognitive processes can extend beyond the brain into the environment when external resources are reliably available, easily accessible, and automatically endorsed. AI coding assistants meet all three criteria:

- **Reliable availability**: AI is present throughout development sessions
- **Easy access**: Near-zero cost to initiate AI collaboration
- **Automatic endorsement**: Developer treats AI outputs as inputs to their own reasoning

Cyborg Theory [Haraway, 1991]: Haraway's cyborg challenges boundaries between human and machine, natural and artificial. In our context, the "developer" is no longer cleanly separable from "developer's tools"—the cognitive system spans both.

5.1.3 Distinguishing Cyborg Cognition from Prior Frameworks

A legitimate concern is whether "Cyborg Cognition" merely rebrands existing concepts. We argue it makes distinct contributions:

Beyond Distributed Cognition: Hutchins' [Hutchins, 1995] distributed cognition describes how cognitive processes spread across people and artifacts in sociotechnical systems. However, Hutchins studied *static* tool configurations (navigation instruments, cockpit displays) where the distribution is architecturally fixed. Cyborg Cognition addresses *adaptive* human-AI systems where the distribution is dynamically negotiated through intentional model selection (our 7.59× ratio) and task-specific delegation patterns. The developer actively manages cognitive distribution in real-time—a phenomenon Hutchins' framework does not address.

Beyond Extended Mind: Clark and Chalmers' [Clark and Chalmers, 1998] extended mind thesis establishes that cognition can extend into the environment, but treats extension as binary (extended or not) and focuses on *passive* cognitive artifacts (Otto's notebook). AI coding assistants are *active* cognitive partners that reason, generate, and adapt. Our delegation score operationalizes extension as a *continuous spectrum*, and our four dimensions (delegation,

extension, selection, fluidity) characterize *qualitatively different* modes of integration that static artifacts cannot exhibit.

Beyond Haraway's Cyborg: Haraway's cyborg is primarily a *political* figure challenging categorical boundaries. Our Cyborg Cognition is an *empirically grounded cognitive framework* with measurable dimensions. We provide operationalizations (delegation scores, tool hierarchies, model selection ratios) that transform the cyborg metaphor into testable constructs.

In summary: distributed cognition lacks adaptivity, extended mind lacks gradation, and cyborg theory lacks operationalization. Cyborg Cognition synthesizes these traditions while addressing their limitations through empirically-derived dimensions.

5.1.4 Four Dimensions of Cyborg Cognition

We identify four dimensions along which human-AI cognitive integration occurs:

1. Delegation Spectrum: From full human control to high AI autonomy.

Our delegation score of 0.71 indicates the developer operates toward the "high delegation" end. This is not abdication—the human sets goals and evaluates outputs—but it represents genuine distribution of cognitive labor.

2. Cognitive Extension: AI serves as external cognitive capacity.

- **Memory extension**: AI reads and retrieves file contents the developer hasn't memorized
- **Search extension**: AI finds patterns across codebases faster than human scanning
- **Execution extension**: AI runs commands and interprets outputs
- **Planning extension**: AI maintains task lists and sequences

3. Adaptive Selection: Meta-cognitive matching of AI capability to task demands.

The 7.59× ratio between Opus and Haiku sessions demonstrates that the developer doesn't treat AI as monolithic. Different cognitive challenges call for different AI configurations—a form of "cognitive resource management."

4. Context Fluidity: Seamless reconfiguration across project contexts.

The 13.3 projects per week finding shows the cyborg system is not project-specific. It adapts its configuration (tool usage patterns, collaboration intensity) based on context while maintaining functional integration.

5.1.5 When Does the System Stop Being "Cyborg"?

A legitimate question is: at what delegation level does a system cease to be "cyborg" (hybrid) and become either purely human or purely AI-driven? We propose tentative thresholds:

- **Low delegation ($D < 0.3$)**: AI serves as occasional assistant. Human performs most cognitive work. This resembles traditional tool use—the developer remains the primary cognitive agent.
- **Cyborg range ($0.3 \leq D \leq 0.8$)**: Human and AI form an integrated cognitive system. Both contribute substantial cognitive work. Direction-setting, quality judgment, and goal selection remain human; information gathering, execution, and routine problem-solving are delegated to AI. Our observed $D = 0.71$ falls in this range.
- **High delegation ($D > 0.8$)**: AI performs most cognitive work with minimal human input. Human serves primarily as goal-setter and output validator. This approaches "AI with human oversight" rather than "human with AI assistance."

These thresholds are necessarily arbitrary and require empirical validation. The key insight is that "cyborg" describes a *range* of human-AI integration, not a binary state. The boundaries are fuzzy and context-dependent—what constitutes appropriate delegation varies by task criticality, domain expertise, and individual risk tolerance.

5.2 Relationship to Prior Work

This work extends prior research on developer mental model frameworks in two directions:

5.2.1 Temporal Extension

Prior artifact-based approaches analyze historical artifacts (commits over extended periods) to infer cognitive patterns. The present study analyzes real-time interactions (sessions over 30 days) to observe cognitive distribution. Together, they provide complementary views:

- **Artifact analysis**: What cognitive patterns produce software artifacts?
- **Interaction analysis**: How are cognitive processes distributed in artifact production?

5.2.2 Cognitive Extension

Prior frameworks describe the *individual* developer's mental model as inferred from artifacts through dimensions such as cognitive, affective, conative, and reflective patterns. Cyborg Cognition's four dimensions (delegation, extension, selection, fluidity) describe the *human-AI system's* cognitive distribution as observed in real-time interactions.

These frameworks operate at different levels of analysis and should not be understood as direct mappings. Table 14 illustrates their complementary relationship:

Table 14: Artifact Analysis vs Cyborg Cognition: Complementary Levels of Analysis

Individual Level	**System Level**	**Relationship**
Cognitive (problem-solving)	Delegation (work distribution)	What → How distributed
Affective (emotional traces)	Extension (capacity augmentation)	Inner state → Outer capability
Conative (motivation)	Selection (capability matching)	Why act → Which tool
Reflective (self-awareness)	Fluidity (context adaptation)	Self-monitoring → System adaptation

The key distinction: artifact-based analysis captures *what* cognitive patterns produce artifacts; Cyborg Cognition captures *how* cognitive work is distributed during artifact production. Together, they provide complementary lenses—one retrospective (artifact analysis), one concurrent (interaction analysis).

5.3 Implications for Research

5.3.1 Redefining the Unit of Analysis

Most developer productivity research treats the individual developer as the unit of analysis. Our findings suggest the appropriate unit is the *developer-AI system*. Measuring "developer productivity" without accounting for AI integration is like measuring a driver's speed without acknowledging the car.

5.3.2 Beyond Productivity Metrics

Current AI coding assistant research focuses on productivity metrics: task completion time, lines of code, bug rates. Our findings suggest richer dimensions:

- **Delegation patterns**: How is cognitive work distributed?
- **Model selection**: How do developers manage AI capabilities?
- **Context adaptation**: How does collaboration change across projects?

5.3.3 Longitudinal Studies

The sustained intensity we observe (2,846 messages/day) would be invisible in short-term studies. Understanding human-AI collaboration requires longitudinal designs that capture integration into daily practice, not just performance on isolated tasks.

5.4 Implications for Practice

5.4.1 Tool Design

Our findings suggest design implications for AI coding assistants:

1. **Surface model selection as first-class interaction**: The 7.59× session length ratio demonstrates that developers make intentional, context-sensitive capability choices. Current AI coding tools typically bury model selection in settings menus, treating it as configuration rather than interaction. Our data suggests an alternative: model selection should be a visible, low-friction decision point integrated into the workflow.

   Concretely, we propose a *capability gradient interface* where developers see model options contextualized by task type. When initiating a complex architectural discussion, the interface might suggest high-capability models with rationale ("This conversation involves design decisions—Opus recommended"). For routine file operations, it could default to efficient models while making the choice transparent. This mirrors how expert developers already think about AI capability matching, but externalizes and supports the decision process.

2. **Visualize delegation through cognitive dashboards**: Developers may benefit from seeing their delegation patterns rendered visually. We propose a *cognitive dashboard* that displays real-time and historical data on human-AI work distribution.

   Such a dashboard might include: (a) a *delegation meter* showing the current session's balance between human direction and AI execution; (b) *tool usage treemaps* visualizing which cognitive functions (exploration, modification, planning) are being delegated; (c) *temporal patterns* revealing how delegation evolves across project phases; and (d) *model selection history* correlating capability choices with task outcomes. The goal is not surveillance but *cognitive awareness*—helping developers understand and optimize their collaboration patterns. This design responds to our finding that delegation score (0.71) represents a genuine cognitive distribution that developers may not be consciously aware of.

3. **Preserve project-specific context**: The fluidity across 47 projects suggests value in maintaining collaboration history and learned preferences per project. AI tools should remember not just code context but *collaboration context*—preferred model tiers for this codebase, successful delegation patterns, and accumulated project-specific knowledge.

4. **Optimize for exploration, not just generation**: The 33.5% exploration tool usage indicates that information gathering is a primary cognitive function delegated to AI. Current AI coding tools emphasize code generation metrics (acceptance rates, lines produced). Our findings suggest equal attention to exploration quality: How effectively does the AI help developers understand unfamiliar codebases? How well does it surface relevant

information without overwhelming? Designing for exploration means optimizing retrieval, summarization, and navigation—not just synthesis.

5.4.2 Developer Education

If Cyborg Cognition is the emerging norm, developer education should address:

- **Delegation skills**: When and what to delegate to AI
- **Capability awareness**: Understanding AI model tiers and appropriate uses
- **Critical evaluation**: Assessing AI outputs without over-reliance

5.5 Limitations and Threats to Validity

We acknowledge significant limitations inherent to single-subject research and address threats to validity systematically.

5.5.1 Internal Validity

- **Observer effect (Hawthorne effect)**: The developer knew data was being collected for research purposes. This awareness could influence behavior toward more "impressive" or "intensive" usage patterns. We partially mitigate this through: (1) automatic, passive data collection requiring no active logging; (2) 30-day duration allowing novelty effects to diminish; (3) data collection as byproduct of normal tool usage. However, we cannot rule out that the research context inflated collaboration intensity.
- **Instrumentation bias**: Tool categorization (Execution, Exploration, Modification, Planning, Interaction) and delegation weights are researcher-defined without external validation. Our sensitivity analysis (Section 3.4.4) shows the delegation score ranges from 0.51 to 0.83 under alternative weight schemes, indicating the 0.71 value is framework-dependent.
- **No inter-rater reliability**: Tool categorization was performed by a single researcher without independent coding. Future work should establish Cohen's κ for category assignments.
- **No "AI off" baseline**: We lack comparison data from the same developer working without AI assistance. This prevents attributing observed patterns specifically to AI collaboration versus general work style. A within-subject design alternating AI-on and AI-off conditions would strengthen causal claims.

5.5.2 External Validity

- **N=1 limitation**: This is fundamentally a single-subject study. All observed patterns may reflect idiosyncratic individual style rather than generalizable phenomena. The developer's background (philosophy training, AI specialization, academic context) is atypical. Replication across diverse developers—varying in experience level, domain, cultural context, and AI familiarity—is essential before drawing broader conclusions.
- **Tool specificity**: Claude Code's agentic architecture (tool use, autonomous execution) differs fundamentally from autocomplete-style tools (GitHub Copilot) and chat-based interfaces (ChatGPT). Our patterns may not generalize to these modalities.
- **Expertise confound**: The subject is an experienced developer with AI/ML specialization. Novice developers may exhibit qualitatively different delegation patterns, possibly with higher or lower delegation depending on trust calibration.

- **Cultural and linguistic context**: The developer works in a specific cultural and linguistic context. Collaboration patterns may differ in other linguistic, cultural, or domain contexts.
- **Selection bias in projects**: A single project ecosystem dominates the dataset. Findings may reflect this project's characteristics rather than general development patterns.
- **Temporal specificity**: Data was collected during a single month (late year period). This timeframe may exhibit atypical patterns—end-of-year deadlines, holiday schedules, or seasonal variations in workload intensity. Longitudinal studies spanning multiple months would reveal whether observed patterns (e.g., the 2,846 messages/day intensity) represent stable collaboration habits or period-specific anomalies. The absence of multi-month data prevents us from distinguishing seasonal effects from fundamental collaboration patterns.

5.5.3 Construct Validity

- **Delegation score operationalization**: Our weighted formula for "cognitive delegation" is one of many possible operationalizations. The 0.71 value is meaningful only within our specific framework. We provide sensitivity analysis showing the score's dependence on weight assumptions, but the fundamental construct requires theoretical refinement and validation.
- **Duration $\neq$ complexity**: Using session length (messages) as a proxy for task complexity is problematic. Long sessions may indicate difficulty, exploration, or simply conversational style—not necessarily complex cognitive work. More granular task-level annotations would strengthen this inference.
- **Cyborg Cognition novelty**: The theoretical framework is proposed here for the first time and requires validation through additional studies. The four dimensions (delegation, extension, selection, fluidity) are theoretically motivated but empirically preliminary.
- **Theoretical tensions**: We draw on both Clark & Chalmers' Extended Mind Thesis (functionalist, parity-based) and Haraway's Cyborg Theory (post-structuralist, political). These traditions have different epistemological commitments that we do not fully reconcile. Our use is primarily metaphorical rather than philosophically rigorous.

5.5.4 Reliability and Reproducibility

- **Data availability**: Raw interaction logs contain sensitive information (file contents, project names, personal details) that cannot be publicly shared. We provide aggregate statistics and analysis scripts, but full replication requires access to similar proprietary data sources.
- **Tool evolution**: Claude Code's capabilities and interfaces evolve rapidly. Patterns observed with current versions may not replicate with future versions.
- **Privacy considerations**: Screenshots and conversation logs were processed with privacy-preserving intent but not formally anonymized through established protocols. Future studies should implement systematic data sanitization.

5.6 Comparative Analysis: Evidence of Modality Effect

Our comparison across three data sources provides evidence that interaction modality—not just AI capability—fundamentally shapes collaboration patterns.

5.6.1 Within-Subject Comparison

The same developer using the same AI provider (Anthropic's Claude) exhibited:

- **Claude Code (CLI)**: 106.4 messages/session average
- **Claude.ai (Web)**: 8.8 messages/session average

This **12.1**× difference cannot be attributed to AI capability differences—both interfaces access the same underlying models. The difference is the *interaction modality*: tool-augmented terminal workflow versus text-only web chat.

This suggests that Cyborg Cognition emerges not from AI capability alone, but from the *integration architecture*—how AI capabilities are embedded into the developer's workflow.

5.6.2 Cross-Subject Comparison

Compared to the DevGPT community dataset:

- **DevGPT average**: 7.3 messages/conversation
- **Our Claude Code**: 106.4 messages/session
- **Ratio**: 14.6×

This difference could reflect:

1. Individual variation (this developer engages more intensively)
2. Tool effect (Claude Code enables longer sessions)
3. Selection bias (DevGPT captures shareable conversations, not daily workflow)

The 27,672 tool invocations in our dataset—a metric absent from web-based ChatGPT use—supports the interpretation that tool augmentation transforms the nature of human-AI collaboration from Q&A to sustained partnership.

5.7 Future Work

1. **Multi-participant studies**: Test whether delegation patterns, model selection ratios, and tool hierarchies generalize across developers.
2. **Cross-tool comparison**: Compare Cyborg Cognition patterns across Claude Code, GitHub Copilot, Cursor, and other tools.
3. **Expertise effects**: Investigate how Cyborg Cognition differs between novice and expert developers.
4. **Intervention studies**: Test whether visualizing delegation patterns changes developer behavior.
5. **Modality experiments**: Controlled studies comparing the same tasks performed via tool-augmented vs. conversational interfaces.

6 Conclusion

This paper presented the first longitudinal, tool-level empirical analysis of human-AI collaborative programming. Through computational autoethnography analyzing 802 sessions, 85,370 messages, and 27,672 tool invocations across 47 projects, we investigated how developers integrate AI coding assistants into their cognitive workflows.

6.1 Summary of Contributions

Empirical Contributions:

- First dataset capturing professional human-AI programming collaboration at tool-level granularity
- Evidence of high cognitive delegation (score: 0.71), suggesting developers treat AI as cognitive extension
- Discovery of intentional model selection patterns (7.59× session length ratio between capability tiers)
- Characterization of sustained collaboration intensity (2,846 messages per active day)

Theoretical Contribution:

- The concept of *Cyborg Cognition in Software Development*—a framework for understanding cognitive integration between human developers and AI systems
- Four dimensions of Cyborg Cognition: delegation spectrum, cognitive extension, adaptive selection, and context fluidity
- Extension of artifact-based developer cognition analysis to real-time interaction analysis

Methodological Contribution:

- Demonstration of computational autoethnography as a viable approach for studying human-AI collaboration
- Operationalization of cognitive delegation through tool categorization and weighted scoring

6.2 Key Insights

Our findings challenge the "AI as autocomplete" mental model that dominates popular discourse. The developers in our study do not use AI to predict the next few tokens; they delegate entire cognitive tasks—information gathering, task planning, command execution—to AI systems while retaining high-level direction and quality judgment.

The 7.59× model selection ratio provides striking evidence of meta-cognitive sophistication. Developers do not treat AI as monolithic; they consciously match AI capability to task complexity. This suggests human-AI collaboration involves *thinking about which kind of thinking partner to engage*.

The sustained intensity of collaboration (nearly 3,000 messages per day, 13+ projects per week) indicates that AI integration is not an occasional convenience but a fundamental restructuring of how software development work is performed.

6.3 Implications

For **researchers**: Studies of developer productivity and cognition should account for AI integration. The appropriate unit of analysis may be the developer-AI system, not the developer alone.

For **tool designers**: AI coding assistants should surface model selection as a first-class decision, visualize delegation patterns, and optimize for exploration (information gathering) alongside generation.

For **educators**: Developer training should include AI collaboration skills: when to delegate, how to evaluate AI outputs, and how to select appropriate AI capabilities.

For **developers**: Awareness of one's own delegation patterns may enable more intentional and effective human-AI collaboration.

6.4 Limitations

This study is limited by its single-subject design, specific tooling context (Claude Code), and 30-day duration. The delegation score and Cyborg Cognition framework are novel constructs requiring validation through replication. We explicitly invite studies testing whether our findings generalize across developers, tools, and contexts.

6.5 Future Directions

The most pressing next step is multi-participant replication: do other developers show similar delegation patterns, model selection ratios, and tool hierarchies? Cross-tool comparison (Claude Code vs. Copilot vs. Cursor) would reveal which patterns are tool-specific versus general.

Intervention studies could test whether visualizing delegation patterns changes developer behavior. Longitudinal studies over months or years could capture the evolution of Cyborg Cognition as developers deepen their AI integration.

Finally, comparative analysis with public datasets like DevGPT [Tao et al., 2024] could contextualize individual patterns against community-level trends.

6.6 Closing Reflection

We titled this paper "The Cyborg Developer" not as provocation but as description. The data shows a developer whose cognitive processes—memory, search, planning, execution—are distributed across human and AI components in sustained, intentional integration.

If this pattern generalizes, we are witnessing a fundamental transformation in what it means to be a software developer. The developer of 2030 may be inseparable from their AI collaborators, not as tool users but as cognitive hybrids. Understanding this transformation—its opportunities, risks, and implications—is among the most important research agendas in software engineering today.

This paper offers a first empirical glimpse into that future.

Data Availability

Replication package including analysis scripts, aggregated data, and figures available at:
https://github.com/anderson-ufrj/cyborg
Raw JSONL interaction logs contain sensitive project information and will be available upon request with appropriate data use agreements.

Conflict of Interest

None declared.

References

Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 85–99, 2023.

Andy Clark and David Chalmers. The extended mind. *Analysis*, 58(1):7–19, 1998.

Anderson Henrique Da Silva. From commits to cognition: A mixed-methods framework for inferring developer mental models from repository artifacts. Zenodo, 2025. URL https://doi.org/10.5281/zenodo.18012186.

GitHub. Github copilot research recitation. https://github.blog/2023-06-27-the-economic-impact-of-the-ai-powered-developer-lifecycle-and-lessons-from-gi 2023.

Donna Haraway. *Simians, Cyborgs, and Women: The Reinvention of Nature*. Routledge, 1991.

Edwin Hutchins. *Cognition in the Wild*. MIT Press, 1995.

John D. Lee and Katrina A. See. Trust in automation: Designing for appropriate reliance. *Human Factors*, 46(1):50–80, 2004.

Stanley Letovsky. Cognitive processes in program comprehension. *Journal of Systems and Software*, 7(4):325–339, 1987.

Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A*, 30(3):286–297, 2000.

Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of ai on developer productivity: Evidence from github copilot. *arXiv preprint arXiv:2302.06590*, 2023.

James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers. In *ACM Conference on Innovation and Technology in Computer Science Education*, 2023.

Isabella Seeber, Eva Bittner, Robert O. Briggs, Triparna de Vreede, Gert-Jan de Vreede, Aaron Elkins, Ronald Maier, Alexander B. Merz, Sarah Oeste-Reiß, Nils Randrup, Gerhard Schwabe, and Matthias Søllner. Machines as teammates: A research agenda on ai in team collaboration. *Information & Management*, 57(2), 2020.

John Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12 (2):257–285, 1988.

Wei Tao, Quanjun Peng, Zihan Chen, Qiyu Han, Cuiyun Gao, Chun Zhou, and Michael R. Lyu. Devgpt: Studying developer-chatgpt conversations. arXiv preprint arXiv:2309.03914, 2024. MSR 2024 Mining Challenge.

Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022.

Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Productivity assessment of neural code completion. *arXiv preprint arXiv:2205.06537*, 2022.

A Tool Category Definitions

A.1 Exploration Tools

Tools that gather information without modifying state:

- **Read**: Read file contents
- **Grep**: Search for patterns in files

- **Glob**: Find files matching patterns
- **WebSearch**: Search the web
- **WebFetch**: Retrieve web page content
- **LSP**: Language server queries (definitions, references)

A.2 Modification Tools

Tools that change file contents:

- **Write**: Create or overwrite files
- **Edit**: Apply targeted edits to files
- **MultiEdit**: Apply multiple edits in one operation
- **NotebookEdit**: Edit Jupyter notebook cells

A.3 Execution Tools

Tools that run commands or manage processes:

- **Bash**: Execute shell commands
- **Task**: Launch sub-agents for complex tasks
- **TaskOutput**: Retrieve output from background tasks
- **KillShell**: Terminate running processes

A.4 Planning Tools

Tools that manage task state and workflow:

- **TodoWrite**: Create and update task lists
- **EnterPlanMode**: Begin structured planning
- **ExitPlanMode**: Complete planning phase

A.5 Interaction Tools

Tools that explicitly request human input:

- **AskUserQuestion**: Pose questions to the developer

B Delegation Score Calculation

The delegation score D is computed as a weighted average of tool category usage:

$$D = \frac{\sum_{c \in C} p_c \cdot w_c}{\sum_{c \in C} p_c} \tag{2}$$

Where:

- C = set of tool categories
- p_c = percentage of tool uses in category c

- w_c = delegation weight for category c

Delegation weights:

- Exploration: $w = 1.0$ (high delegation—AI gathers information)
- Planning: $w = 1.0$ (high delegation—AI manages tasks)
- Modification: $w = 0.5$ (medium—shared authorship)
- Execution: $w = 0.5$ (variable—depends on command autonomy)
- Interaction: $w = 0.0$ (low—human deciding)

For our dataset:

$$D = \frac{33.5 \times 1.0 + 8.7 \times 1.0 + 21.5 \times 0.5 + 35.8 \times 0.5 + 0.5 \times 0.0}{100} \tag{3}$$

$$= \frac{33.5 + 8.7 + 10.75 + 17.9 + 0}{100} \tag{4}$$

$$= \frac{70.85}{100} \tag{5}$$

$$= 0.71 \tag{6}$$

C Model Tier Definitions

Table 15: Claude Model Tiers

Model	Tier	Characteristics	Typical Use
Opus 4.5	High	Complex reasoning, large context	Architecture, debugging
Sonnet 4.5	Medium	Balanced capability and speed	General development
Haiku 4.5	Low	Fast, efficient	Quick queries, routine

D Session JSONL Format

Each session is stored as a JSONL file with events:

Listing 2: Sample JSONL event

```
{
  "type": "assistant",
  "message": {
    "role": "assistant",
    "model": "claude-opus-4-5-20251101",
    "content": [...],
    "usage": {
      "input_tokens": 1234,
      "output_tokens": 567
    }
  },
  "timestamp": "2025-12-30T10:15:00.000Z",
  "sessionId": "abc-123-def",
  "cwd": "/home/user/project"
}
```

Tool invocations appear within message content:

Listing 3: Tool use structure

```
{
  "type": "tool_use",
  "id": "toolu_01ABC...",
  "name": "Bash",
  "input": {
    "command": "npm test"
  }
}
```