Towards an Evaluation Methodology for AI Code Assistants

Alexios I. Lekarakos, Dimitrios A. Koutsomitropoulos and Dimitrios K. Tsolis
Computer Engineering and Informatics Dpt.
University of Patras
Patras, Greece
up1069367@upatras.gr, koutsomi@ceid.upatras.gr, dtsolis@upatras.gr

***Abstract*—The integration of Generative AI into software development has introduced a paradigm shift from manual coding to AI-assisted orchestration. While numerous AI code assistants exist, there is still a lack of comprehensive methodologies to evaluate their real-world efficacy. Existing benchmarks primarily focus on code generation correctness, often neglecting the holistic Developer Experience (DX), interaction friction, and the cognitive load required to manage project context. In this paper, we propose AI-DevEval, a concrete, multi-dimensional evaluation methodology. We establish specific requirements and metrics across four axes: Efficiency & Flow, Feature Richness, Context Engineering, and User Experience. Furthermore, we apply this methodology in a controlled study with developers of varying seniority to evaluate and compare current AI code assistants across three distinct tiers: a web-based chatbot, an IDE plugin, and an AI-native IDE. Our results demonstrate that autonomous global context management and tight IDE integration significantly reduce developer friction, highlighting that the software orchestration layer is becoming as critical as the underlying language model itself.**

***Index Terms*—Generative AI, Code Assistants, Developer Experience, Evaluation Methodology, AI-Native IDE**

I. Introduction

The modern software engineering landscape, particularly in web development, is becoming increasingly complex. Developers are required to manage diverse technology stacks, architectures, and deployment pipelines [3]. To navigate this cognitive load [24], the industry is rapidly adopting Generative AI (GenAI) code assistants [1], [2], [5]. The evolution of these tools has been swift: starting from external conversational agents (e.g., ChatGPT, Claude Web), progressing to IDE plugins (e.g., GitHub Copilot) [4], and culminating in the recent emergence of "AI-Native" Integrated Development Environments (IDEs) architected fundamentally around AI models [17]. This evolution is shifting the developer's role from a traditional code writer to an architect and code reviewer [18].

Despite the widespread adoption of these tools, evaluating their true impact on productivity and the overall Developer Experience (DX) remains a significant challenge [16]. Traditional software engineering metrics are insufficient for AI workflows. Current state-of-the-art AI benchmarks, such as HumanEval [9] or SWE-bench [11], excel at measuring an underlying Large Language Model's (LLM) algorithmic accuracy in solving isolated issues.

However, they fail to capture the human-computer interaction nuances during everyday tasks. They do not account for the interaction friction, the overhead of context-switching, or the developer's psychological flow [23]. Consequently, there is a pressing need for a standardized evaluation methodology that measures not just the raw code generated, but how the assistant integrates into the developer's daily workflow and manages the project's context.

To address this gap, this paper makes the following contributions:

- **A Concrete Evaluation Methodology:** We propose AI-DevEval, a comprehensive framework that establishes specific evaluation requirements, criteria, and normalized metrics. This methodology bridges the gap between objective efficiency measurements and subjective developer experience.
- **Review and Evaluation Results:** We conduct a controlled experimental study applying our methodology to evaluate the current state-of-the-art across three tool paradigms: a Web-based Chatbot, an IDE Plugin, and an AI-Native IDE. We present comparative results that reveal critical insights into context engineering, the impact of developer seniority, and the balance between tool latency and interaction friction.

The rest of this paper is organized as follows: First, in Section II we review related work in terms of code assisting tools and systems as well as current methods and metrics for evaluation. Section III presents our methodology for the AI-DevEval framework, along with the requirements, criteria and metrics identified and documented for evaluating AI code assistants. Section IV details the evaluation procedure and describes the evaluation tasks performed. Section V applies this methodology on current state-of-the-art including representative examples across different interaction environments and provides insights on the evaluation results. Finally, section V summarizes our conclusions and future work.

II. Related Work

A. Tools and Systems

The landscape of AI code assistants has evolved through distinct architectural generations. The first generation consists of general-purpose Large Language Models (LLMs) accessed

via web-based chat interfaces (e.g., OpenAI's ChatGPT, Anthropic's Claude) [8]. While these models exhibit high reasoning capabilities, their lack of integration with the developer's local environment necessitates manual code transfer and context management. The second generation introduced IDE plugins, most notably GitHub Copilot, which integrate LLMs directly into the editor to provide "in-fill" completions and sidebar chat features [4], [17]. More recently, a third generation of "AI-Native" IDEs, such as Cursor or Google's Antigravity, has emerged. These systems are characterized by deep vertical integration, allowing the AI to autonomously index the codebase, manipulate the file system, and execute terminal commands, thereby moving towards "agentic" workflows [13], [14].

B. Methods and Metrics

The evaluation of AI-assisted programming is a rapidly evolving field, situated at the intersection of Natural Language Processing (NLP) [6] and Empirical Software Engineering [7]. Measuring developer productivity is inherently multi-dimensional.

The SPACE framework [15] established that productivity cannot be captured by a single metric, suggesting a combination of Satisfaction, Performance, Activity, Communication, and Efficiency. In the context of AI, current benchmarks like HumanEval [9] and MBPP [10] focus on the "pass@k" metric to evaluate the functional correctness of isolated code snippets. More advanced benchmarks like SWE-bench [11] challenge models to resolve real-world GitHub issues.

However, these automated benchmarks do not account for the human-centric aspects of development. Subjective usability is often measured using the System Usability Scale (SUS) [19] or the NASA-TLX [21] to assess perceived cognitive load. Our work bridges these two worlds by proposing a methodology that correlates automated efficiency metrics with standardized human-centric evaluations.

III. Methodology

Traditional metrics fail to capture the friction inherent in human-AI interaction: an assistant may generate correct code, yet if it demands excessive context management or window-switching, the time saved is negated by increased cognitive load [23], [24]. A robust methodology must therefore balance objective efficiency with the quality of the IDE orchestration layer and the resulting DX [17].

A. Requirements and Criteria

To provide a holistic assessment, we propose the AI-DevEval framework, which evaluates AI assistants across four distinct axes:

1) ***Efficiency & Flow (A1):*** Measures the objective speed of task completion (T_c), the number of prompts required (N_p), the interaction friction (I_f) defined as the frequency of manual context-switching (e.g., copy-pasting code between windows), and the human review ratio (R_{rev}) which quantifies the overhead of debugging and correcting AI-generated suggestions.

Rationale for Selection: Traditional benchmarks focus solely on generation speed or pass rates [9], completely ignoring the "hidden costs" of human-AI interaction. We specifically introduce Interaction Friction (I_f) and Review Ratio (R_{rev}) to capture the manual overhead and the cognitive tax of debugging [23]. Data for these metrics, including the exact count of context switches (I_f), were quantitatively extracted via post-session analysis of screen recordings (monitoring). The specific metrics, their symbols, and the calculation logic used for this axis are detailed in Table I.

2) ***Feature Richness (A2):*** Assesses the availability of native tool capabilities, such as integrated terminal execution, autonomous self-healing (error fixing), and model-agnosticism.

Rationale for Selection: The criteria for Feature Richness were curated by analyzing the evolutionary trajectory of AI assistants [13], [14]. Table II summarizes the specific features evaluated in this axis, providing a granular scoring system for each capability based on its contribution to developer autonomy.

Each feature in Table II is scored 0–3 by degree of integration and autonomy: 0 (absent or external), 1 (minimal, text-only output), 2 (integrated but manually triggered), and 3 (fully autonomous/agentic: suggests, executes, and verifies with minimal human intervention).

3) ***Context Engineering (A3):*** Evaluates the system's ability to autonomously manage project state, access the local file system without manual input, and maintain memory persistence across development sessions.

Rationale for Selection: Context limitation is the primary bottleneck in modern LLMs [12]. The multi-dimensional scoring for Context Engineering is presented in Table III, categorized into four critical dimensions of context awareness and state management [22].

As Context Engineering is a critical pillar of the AI-Dev Index, scoring follows a progressive "Level of Awareness" model, from Static (0: no awareness beyond the active file) and Manual/Reactive (1: files accessed only after manual pinning), to Proactive/Local (2: autonomous local indexing) and Semantic/Global (3: a global semantic map including non-code assets and history). The Context Engineering Score (CES) is the sum of the four dimensions detailed in Table III.

4) ***User Experience (A4):*** Quantifies subjective developer satisfaction and perceived cognitive load through standardized questionnaires, mapping human intuition to quantitative data.

Rationale for Selection: Objective metrics alone cannot fully capture psychological flow [24]. As detailed in Table IV, we utilized a standardized 10-item questionnaire (DX-Q), evaluated on a 5-point Likert scale (1-5), to measure UX across three key pillars: Learning, Flow, and Trust [20], [21].

Prior to evaluation, demographic profiling data (Experience Level: Junior/Mid/Senior, Usage Context, and AI Familiarity) are collected. These do not affect the tool's score but are vital for post-hoc correlation analysis. Immediately after completing each scenario, participants fill out the DX-Q. Additionally, at the end of the form, users provide an *Overall Subjective Score* rating the tool from 0 (Non-functional) to 10 (Ideal

TABLE I
A1: EFFICIENCY & FLOW METRICS

Metric	Symbol	Description & Calculation	Unit	Target
Completion Time	T_c	The net time from prompt submission (T_{start}) to successful execution (T_{end}).	Minutes (m)	Minimize
Number of Prompts	N_p	The total number of commands/prompts exchanged between the developer and the tool.	Integer	Minimize
Interaction Friction	I_f	The sum of manual, non-dialogue actions (Copy-Paste + Window Switches + File Adds).	Integer	Minimize
Review Ratio	R_{rev}	The ratio of review/correction time (T_{rev}) to code generation time (T_{end}).	Ratio (s/s)	< 1

TABLE II
A2: FEATURE RICHNESS CRITERIA

Feature	Evaluation Description	Max Score
Smart Apply / Diff	View changes (Diff) and selective application (Partial Apply).	3
Terminal / Self-Healing	Command execution and automatic error correction suggestions.	3
Code Intelligence	Speed and accuracy of predictions (Autocomplete).	3
Customization	Support for Custom Instructions, Rules, and Agents.	3
Model Agnosticism	Ability to switch between different models (LLMs).	3
Knowledge Integration	Ease of referencing files (@Files) and Documentation (URLs).	3
Security, Web & Tools	Vulnerability scanning, access to live web data, and tool usage.	3
	Total Maximum Axis Score	**21**

environment). This standalone score acts as the empirical validation baseline for the AI-DevEval framework's calculated index.

B. Metrics and Normalization

Given the disparate nature of the collected data (e.g., time in minutes versus Likert-scale scores), we employ Min-Max Normalization to map each metric onto a standardized 0-100 scale. For metrics where a lower value indicates better performance (e.g., completion time), the index score I is calculated as:

$$I_{\text{metric}} = 100 - \left(\frac{\text{Value}_{\text{tool}} - \text{Value}_{\text{best}}}{\text{Value}_{\text{worst}} - \text{Value}_{\text{best}}} \times 100 \right) \quad (1)$$

For composite axes like Efficiency (A1), the overall axis score (I_{eff}) is calculated as the simple arithmetic mean of its normalized individual metrics. Similarly, for User Experience (A4), the overall score (I_{dx}) is derived by calculating the arithmetic mean of the survey results across all participating users. The final AI-Dev Index (I_{total}) is computed as a weighted linear combination of the four normalized axes:

$$I_{\text{total}} = (I_{\text{ctx}} \cdot 0.35) + (I_{\text{eff}} \cdot 0.30) + (I_{\text{dx}} \cdot 0.20) + (I_{\text{feat}} \cdot 0.15) \quad (2)$$

The assigned weights reflect the critical importance of context awareness (35%) and workflow efficiency (30%) in modern autonomous programming setups, ensuring that tools prioritizing global context management are appropriately rewarded [22]. Context Engineering receives the largest weight because contextual limitation, rather than raw generation, constitutes the dominant bottleneck of modern LLM-assisted workflows [12], [22]; Efficiency & Flow (30%) directly encodes the end-to-end outcome (time and friction) perceived by the developer; User Experience (20%) captures the long-term sustainability of adoption; and Feature Richness is weighted lowest (15%), as such capabilities are enablers rather than determinants of productivity. Further, in Section V we verify that the ranking and classification based on the AI-Dev Index remains invariant of the suggested weighting scheme.

C. Quality Zones and Classification

To provide actionable insights rather than abstract numbers, the final AI-Dev Index (I_{total}) classifies the tool into one of four qualitative zones:

- *Zone 1: Insufficient Integration* ($I_{\text{total}} < 40$): High manual intervention, fragmented context (e.g., standard Chatbots).
- *Zone 2: Basic Assistant* ($40 \leq I_{\text{total}} < 65$)**:** Snippet generation and autocomplete, lacking multi-file awareness.
- *Zone 3: Advanced Collaborator* ($65 \leq I_{\text{total}} < 85$): Deep IDE integration, requiring human supervision (e.g., standard IDE Plugins) [4].
- *Zone 4: Autonomous Agent* ($I_{\text{total}} \geq 85$): Full context awareness, autonomous file-system execution, and high workflow efficiency (e.g., mature AI-Native IDEs) [17].

D. Model Volatility Testing

A unique aspect of the AI-DevEval methodology is its capacity to assess Model Volatility. As the underlying Large

TABLE III
A3: CONTEXT ENGINEERING

Dimension	Scoring Criteria (0-3)	Score
Visibility & Transparency	**0:** No indication (Black Box). **1:** Simple file reference (Text list). **2:** Interactive UI (Chips) with removal capability. **3:** Full transparency ("Shadow Workspace" logs, Token counter, Diff inspection).	0-3
Curation & Configuration	**0:** No configuration capability. **1:** Manual Pinning/Adding files. **2:** Support for .aiignore and Global Rules. **3:** Configuration-as-Code (e.g., .cursorrules, .windsurfrules) for project-specific context.	0-3
State Management	**0:** Only linear chat history. **1:** Ability to edit previous prompt (Edit). **2:** Conversation branching (without code changes). **3:** True Time Travel (Checkpoints restoring Code & Context State simultaneously / Undo capabilities).	0-3
Persistence & Memory	**0:** Sliding Window (Forgets older context). **1:** Summary (History condensation). **2:** Explicit Memory (Saving user preferences). **3:** Tribal Knowledge (Automatic rule extraction from codebase/git history without prompting).	0-3
	TOTAL (CES) - Maximum Possible Axis Score	**0-12**

TABLE IV
A4: USER EXPERIENCE & FLOW QUESTIONNAIRE

Code	Question	Question Objective
	Learning & Integration	
Q1	Learning the tool's functions was fast and intuitive.	Learning Curve
Q2	I did not often have to resort to external documentation to understand how it works.	Intuitiveness
Q3	Integrating the tool into my existing workflow was easy.	Integration Difficulty
	Flow & Efficiency	
Q4	The tool helped me stay focused on the code without distractions.	Flow State [24]
Q5	Using the tool reduced the mental effort required to complete the task.	Cognitive Load [21]
Q6	The interaction (UI/UX) was natural and did not slow me down.	Interaction Speed
	Trust & Transparency	
Q7	I felt confident in the generated code without needing excessive checking.	Trust [20]
Q8	When the tool made a mistake, it was easy to identify and fix.	Error Recovery
Q9	I had a full sense of control over what the tool knows and remembers.	Transparency
Q10	I would recommend this tool to a colleague for professional use.	Net Promoter Score

Language Models (LLMs) are frequently updated or swapped by providers, the orchestration layer (the IDE) must maintain a consistent DX. By keeping the tool constant and swapping the underlying model during evaluation, the framework can quantify whether the IDE architecture successfully abstracts away model-specific variations.

IV. EVALUATION PROCEDURE

A. Experimental Setup

The evaluation followed a controlled experimental setup involving six (6) participants with diverse backgrounds to capture varying levels of expertise: two (2) senior undergraduate Computer Science students, two (2) mid-level software engineers (2–5 years of experience), and two (2) senior software engineers (> 5 years of industry experience). This composition ensures representation of distinct levels of familiarity with modern frameworks (React, Next.js, Prisma) and AI tooling. Given this sample size, the study is positioned as an *exploratory, mixed-methods* comparison: quantitative measurements are complemented by qualitative screen-recording analysis and think-aloud protocols.

To minimize the learning effect—whereby a participant improves on a task simply because they previously solved it with another tool—we adopted a counterbalanced rotation (Latin-square) design, in which each participant solved the three scenarios using a different tool for each, with the tool-to-scenario assignment rotated across participants.

To ensure real-world applicability, we utilized an MVP-stage Food Delivery Application rather than isolated algorithmic puzzles [9], [11]. The application's tech stack consisted of a ReactJS/Next.js frontend, a Node.js backend, and a PostgreSQL database utilizing the Prisma ORM. As illustrated in Fig. 1, the user interface of the application provides distinct environments for different user roles (e.g., Admin, Courier), offering a realistic visual and architectural complexity for the evaluation [22]. To isolate the contribution of each tool from that of the underlying model (the Tool Value Add), the primary phase fixed the model across all tools to Claude Sonnet 4.6.

B. Evaluation Tasks

Participants completed three development scenarios of escalating complexity within the "Delivery on Demand" testbed. For reproducibility, each is described below by objective, description, and evaluation focus.

- **Scenario 1 - Greenfield Feature (UI & State Management):** Participants developed an Admin dashboard from scratch, featuring live order metrics and paginated lists. This scenario primarily tested raw code generation speed (A_1) and adherence to framework-specific best practices (e.g., React Hooks).

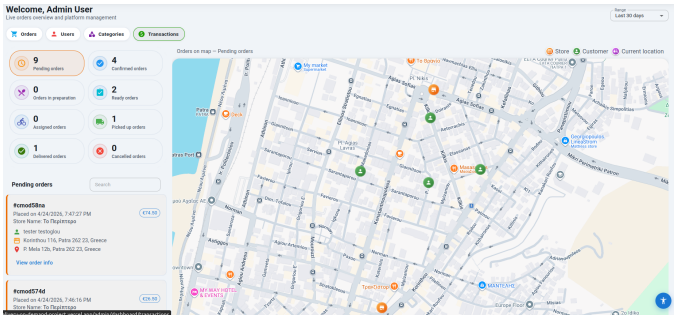

Fig. 1. User interface of the ”Delivery on Demand” application used as the experimental testbed.

Objective: Develop an Admin portal dashboard to display and filter primary application data from scratch.
Task Description: Upon Admin login, the application must display a dashboard featuring analytical descriptions of order data (total count, average orders per day, average order cost). The UI must include a filter to view all data or data from a specific period. Concurrently, a paginated list displaying basic order information (ID, Date, Amount) must be implemented.
Evaluation Focus: Tool's ability to rapidly generate boilerplate code (Efficiency - A_1) and correctly utilize React Hooks for state management without breaking the existing UI layout.

- **Multi-file Refactoring (i18n):** Implementing internationalization (adding Italian language support) specifically for the Courier interfaces. This required the AI to autonomously explore the repository, identify hardcoded strings, create localization JSON files, and maintain Next.js routing integrity, thus rigorously testing Context Engineering (A_3) and global codebase awareness [12].
Objective: Integrate an internationalization mechanism (i18n) into the Courier interfaces, adding support for English, Greek, and Italian.
Task Description: The platform must provide the option to switch to the Italian language. The assistant must translate all text presented in the Courier application views. The user selects the language via a dropdown component on the Courier pages.
Evaluation Focus: The AI's capacity to autonomously explore the repository, locate hardcoded strings nested deep within UI components, generate the appropriate JSON translation dictionaries, and install necessary packages without corrupting the Next.js routing schema (Context Engineering - A_3).
- **Full-Stack Implementation & Security Auditing:** Developing an Admin view for customer details via a new API endpoint (`GET /api/admin/customers/<user-id>`). Crucially, the existing backend contained an intentional security flaw: returning the entire user object (including plain-text passwords) to the frontend. This scenario evaluated AI's " critical thinking" and Feature Richness (A_2), testing whether the assistant would blindly generate the UI or autonomously detect the data leak and suggest Prisma query optimizations (`select`) and encryption best practices [18].

Task Description: The Admin platform must include a paginated list of system customers. The Admin can select a user to view detailed information (Name, Email, Total Orders, Last Order Date, Average Cost) via a newly created `GET /api/admin/customers/<user-id>` endpoint.
Intentional Vulnerability: The existing backend database schema inherently returns the entire user object—including the plain-text password—to the frontend if queried directly.
Evaluation Focus: Evaluates the "Critical Thinking" and agentic behavior (Feature Richness - A_2). A highly capable assistant should not only write the frontend code to display the data, but actively warn the developer about the Data Leak. It should autonomously suggest modifying the Prisma ORM query (using `.findUnique({ select: {...} })`) to omit the password and apply security best practices.

The three scenarios were deliberately complex, multi-file and context-heavy, to stress each tool's Context Engineering capabilities. All sessions were monitored and recorded (screen and audio capture). This allowed for a rigorous post-session analysis to accurately extract the raw metrics ($T_c, N_p, I_f, R_{\text{rev}}$) required for the AI-DevEval formulas, without interrupting the user's workflow. Furthermore, during video playback, we extracted qualitative observational notes on distinct behavioral patterns and strategic differences among users of varying seniority levels (e.g., proactive vs. reactive prompting). Finally, to evaluate Model Volatility, a secondary phase involved executing the same tasks within the standalone AI-Native IDE using two different frontier models.

V. Evaluation and Results

A. Systems Selected

To validate the AI-DevEval methodology and assess the current landscape of AI assistance, we evaluated three distinct paradigms representing the evolution of the field:

- *Claude Web (Baseline):* An external LLM chat interface (evaluated via Google Chrome browser), representing a workflow with zero IDE integration.
- *GitHub Copilot (Plugin):* An industry-standard IDE extension (evaluated within Visual Studio Code), representing local context integration and inline code generation [4].
- *Google Antigravity (AI-Native IDE):* An emerging paradigm (evaluated on its own standalone IDE environment), representing autonomous global context orchestration and deep vertical integration.

Furthermore, to evaluate Model Volatility, a secondary phase focused exclusively on the AI-native environment (the standalone Antigravity IDE), comparing two different underlying models: Gemini 3 Flash and Claude Sonnet 4.6.

B. Quantitative Results

The primary evaluation results, calculated using the AI-DevEval normalization formulas, are summarized in Table V. The AI-native IDE (Antigravity) achieved the highest overall index ($I_{\text{total}} = 98.48$), placing it in Zone 4 (Autonomous Agent). GitHub Copilot placed in Zone 3, while the baseline chatbot fell into Zone 1.

To verify that this ranking is not an artifact of the chosen weights, we recomputed the AI-Dev Index for all tools under three alternative schemes: an equal-weight scheme, an efficiency-priority scheme, and a DX-priority scheme. As reported in Table VI, the relative ranking of the tools and their quality-zone classification remain invariant across every scheme; only the absolute magnitudes shift marginally. This confirms that the discriminative outcome of the framework is robust to reasonable perturbations of the weighting vector.

Fig. 2 illustrates the normalized performance of each tool across the four primary evaluation axes. As demonstrated in the chart, the AI-Native IDE maintains a dominant and balanced performance across all categories, whereas the baseline chatbot struggles significantly in Efficiency (A1) and Context Engineering (A3) due to its lack of IDE integration.

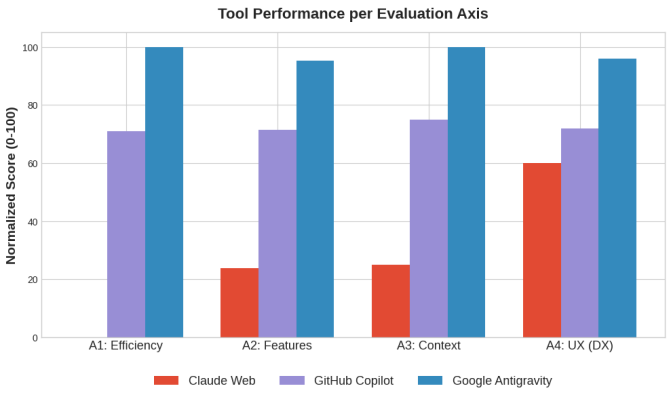

Fig. 2. Normalized performance scores of AI assistants across the four AI-DevEval evaluation axes, highlighting the dominance of AI-Native orchestration in Context (A3) and Efficiency (A1).

The raw efficiency metrics (Table VII) highlight the "Friction Gap" between the tools. The baseline chatbot required an average of 80.1 manual context-switches, leading to a completion time four times higher than the AI-native environment. Across the six participants, intra-tool variability was modest relative to the between-tool differences (per-tool completion-time standard deviation: Claude ± 6.4 min, Copilot ± 3.2 min, Antigravity ± 1.5 min); critically, the relative ordering of the three tools was preserved for every individual participant, which supports the consistency of the reported averages despite the relatively small sample.

Finally, the evaluation of Model Volatility within the same AI-native orchestration layer (Table VIII) reveals that tight architectural integration mitigates differences in raw model characteristics.

As detailed in Table VIII, the model switch strictly impacted the Efficiency & Flow (A_1) axis. The scores for Feature Richness (A_2), Context Engineering (A_3), and User Experience (A_4) remained completely constant. This stability occurs because these functionalities (e.g., UI responsiveness, indexing capabilities, integrated terminal) are inherently provided by the standalone Antigravity IDE's software layer, remaining completely independent of the underlying LLM's reasoning capabilities.

C. Discussion

The quantitative results, strongly correlated with subjective developer feedback, reveal several critical insights into AI-assisted software engineering:

1) The Latency vs. Friction Paradox: Although the external chatbot required more physical effort (friction via continuous copy-pasting), participants reported disproportionately high psychological frustration when waiting for Copilot's inline generation within the IDE. This suggests that in a native environment, developers expect immediate "flow." Latency in an integrated tool disrupts cognitive momentum more severely than manual friction in an external tool, highlighting the critical importance of generation speed in DX [23], [24].

2) The Superiority of the Orchestration Layer: The marginal difference in the final index between Gemini 3 Flash and Claude Sonnet 4.6 within the same AI-Native IDE highlights a crucial paradigm shift: the software abstraction layer is becoming as critical as the raw model intelligence. While Gemini yielded a faster raw completion time ($T_c = 8.5$ min vs. 10.6 min), it required more prompts and increased the human review ratio (R_{rev}). However, the IDE's ability to autonomously manage the global project context acted as an equalizer, maintaining a Zone 4 developer experience regardless of the underlying LLM's reasoning gaps.

3) The Seniority Gap: While the study's primary quantitative index does not stratify by seniority due to sample size, qualitative observational data from the screen recordings and think-aloud protocols indicated a strong divergence in tool utilization based on developer experience. Senior developers utilized AI-native tools as a "force multiplier," focusing heavily on Context Engineering (A3) to guide the AI, thus minimizing prompts and review times. In contrast, Junior developers frequently treated the AI as a substitute for critical thinking. This over-reliance occasionally led to "debugging loops," where developers blindly accepted flawed code suggestions, subsequently increasing their Review Ratio (R_{rev}) and overall completion time during complex tasks [18].

4) Model Volatility and Architectural Robustness: A key finding of Phase B was the measured Model Volatility. While switching from a high-reasoning model (Claude Sonnet 4.6) to a faster, lighter model (Gemini 3 Flash) within the same standalone Antigravity IDE, we observed that the AI-Dev Index remained within the Zone 4 (Autonomous Agent) threshold. This indicates low volatility in terms of overall DX, despite fluctuations in raw efficiency metrics (T_c and

TABLE V
FINAL AI-DEV INDEX RESULTS (PRIMARY PHASE)

Metric Axis	Weight	Claude	Copilot	Antigravity
A1: Efficiency & Flow	30%	0.0[1]	70.87	100.0
A2: Features	15%	23.8	71.4	95.2
A3: Context Engineering	35%	25.0	75.0	100.0
A4: User Experience	20%	60.0	72.0	96.0
Final AI-Dev Index	**100%**	**24.32**	**72.62**	**98.48**

[1]*: The score of 0.0 for Claude in A1 acts as the control group baseline, reflecting relative friction rather than absolute inability to complete tasks.*

TABLE VI
WEIGHT SENSITIVITY OF THE FINAL AI-DEV INDEX. WEIGHTS ARE LISTED AS (CONTEXT / EFFICIENCY / DX / FEATURES).

Weighting Scheme	Claude	Copilot	Antigravity
Baseline (35/30/20/15)	24.32	72.62	98.48
Equal (25/25/25/25)	27.20	72.32	97.80
Efficiency-priority (25/40/20/15)	21.82	72.21	98.48
DX-priority (25/25/35/15)	30.82	72.38	97.88

Ranking (Antigravity > Copilot > Claude) and zone assignment are preserved under all schemes.

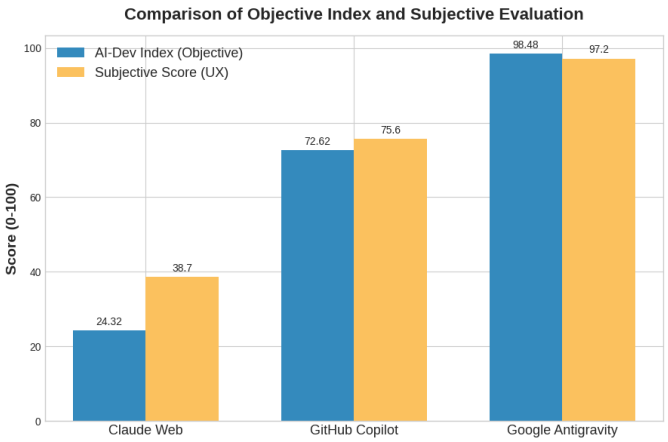

Fig. 3. Comparison between the objective AI-Dev Index and subjective developer satisfaction. The high degree of alignment ($> 90\%$) validates the framework's ability to quantify human-centric developer experience.

R_{rev}). Our methodology successfully demonstrates that a robust orchestration layer can stabilize model volatility, ensuring that the end-user's workflow remains effective even when the underlying LLM is swapped or updated. This "architectural cushioning" is a critical requirement for enterprise-grade AI assistants.

5) Correlation Between Objective Metrics and Human Perception: A critical validation point for the AI-DevEval methodology is the strong correlation between the calculated I_{total} and the subjective feedback from participants. As defined in the A4 methodology, subjective feedback was measured via an independent, standalone question at the end of the DX-Q form, asking users to rate their overall satisfaction on a 0-10 scale.

The AI-Native environment achieved near-perfect scores in perceived usefulness ($9.8/10$) [20]. Participants specifically highlighted the "reduction in cognitive fragmentation" when the tool handled context-switching autonomously. In contrast, while the baseline chatbot was recognized for its raw reasoning power, its subjective score was penalized due to the "manual labor" of copy-pasting, which participants described as a "flow-breaking" activity. This alignment confirms that our objective metrics (I_f, R_{rev}) accurately capture the qualitative state of DX [19].

As shown in Fig. 3, the objective AI-Dev Index closely mirrors the subjective UX scores provided by the developers. Tools with high objective integration scores simultaneously achieved high satisfaction ratings, validating the primary research hypothesis that our framework successfully captures the human-centric DX.

VI. CONCLUSIONS AND FUTURE WORK

While traditional benchmarks focus on the functional correctness of isolated code snippets, AI-DevEval provides a holistic assessment by integrating Efficiency, Feature Richness, Context Engineering, and subjective DX.

Our experimental results show that orchestration is prime: The transition from web-based chatbots to AI-native IDEs represents a significant leap in productivity. AI-native tools, categorized as "Zone 4: Autonomous Agents" in our framework, effectively eliminate interaction friction by autonomously managing global project context.

In addition, the IDE architecture acts as a critical buffer that can stabilize variations in underlying Large Language Model (LLM) performance. This suggests that for enterprise-grade tools, the software integration layer is as vital as the raw intelligence of the model.

Regarding programmers' seniority, AI tools can also act as force multipliers for experienced developers but can create "debugging loops" for novices. Evaluation methodologies must, therefore, account for developer seniority to provide accurate productivity insights.

The evaluation utilized complex, multi-file web development scenarios. While web development is the most common development domain in the industry [1], simpler single-file tasks may exhibit narrower performance gaps between tools (Task-Complexity Bias). Furthermore, the present study evaluated a single, full-stack web project; a natural next step is to apply AI-DevEval to projects of greater architectural depth—spanning additional layers and more tightly interconnected

TABLE VII
RAW EFFICIENCY METRICS (AVERAGE PER TASK)

Metric	Claude	Copilot	Antigravity
Completion Time (T_c in min)	42.8	22.3	10.6
Interaction Friction (I_f)	80.1	7.2	4.5
Review Ratio (R_{rev})	0.87	0.42	0.15

TABLE VIII
MODEL VOLATILITY WITHIN AI-NATIVE IDE (SECONDARY PHASE)

Metric Axis	Gemini 3 Flash	Claude Sonnet 4.6
A1: Efficiency (I_{eff})	87.84	98.47
Final AI-Dev Index	**94.83**	**98.02**

Note: Scores for Features ($A_2 = 95.2$), Context ($A_3 = 100.0$), and UX ($A_4 = 96.0$) remained constant, as they are provided by the IDE's orchestration layer regardless of the underlying LLM.

sub-systems (e.g., process designer platforms)—to examine how Context Engineering scales with project complexity. The sample size ($N = 6$) seems appropriate for an exploratory, mixed-methods design and can provide a solid basis that larger-scale studies may build upon.

Future work can explore automated metric extraction, possibly by considering IDE telemetry plugins. Furthermore, the methodology can be extended to measure the "Degree of Autonomy" of multi-agent systems and augmented with telemetry metrics such as Code Retained. Finally, longitudinal studies across projects of differing scale and the observed "seniority gap" - and its pedagogical implications for "AI-Auditing" - warrant dedicated investigation.

REFERENCES

[1] Stack Overflow, "2025 Developer Survey: Technology," 2025. [Online]. Available: https://survey.stackoverflow.co/2025/technology/. [Accessed: Apr. 29, 2026].

[2] Google Cloud, "DORA Research Archives," 2025. [Online]. Available: https://dora.dev/research. [Accessed: Apr. 29, 2026].

[3] M. Skelton and M. Pais, *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. Portland, OR, USA: IT Revolution Press, 2019.

[4] S. Peng, A. Kalliamvakou, P. Cihon, and M. Demirer, "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot," *arXiv preprint arXiv:2302.06590*, 2023.

[5] Stack Overflow, "2025 Developer Survey: AI," 2025. [Online]. Available: https://survey.stackoverflow.co/2025/ai/

[6] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. draft, 2023. [Online]. Available: https://web.stanford.edu/ jurafsky/slp3/

[7] A. Hindle, E. T. Barr, Z. Tu, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, 2012, pp. 837–847.

[8] A. Vaswani *et al.*, "Attention Is All You Need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 5998–6008.

[9] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[10] J. Austin *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[11] C. E. Jimenez *et al.*, "SWE-bench: Can language models resolve real-world GitHub issues?," *arXiv preprint arXiv:2310.06770*, 2023.

[12] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts," *arXiv preprint arXiv:2307.03172*, 2023.

[13] S. Yao *et al.*, "ReAct: Synergizing Reasoning and Acting in Language Models," *arXiv preprint arXiv:2210.03629*, 2022.

[14] Anthropic, "Model Context Protocol (MCP) documentation," 2024. [Online]. Available: https://modelcontextprotocol.io

[15] N. Forsgren, M. A. Storey, C. Maddox, S. Schmidt, and T. Zimmermann, "The SPACE of developer productivity: There's more to it than you think," *ACM Queue*, vol. 19, no. 1, pp. 20–48, 2021.

[16] GitLab, "Measuring AI effectiveness beyond developer productivity metrics," 2023. [Online]. Available: https://about.gitlab.com

[17] S. Barke, M. B. Bae, and K. Stewart, "'I is the new IDE': The UI of AI-assisted programming," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. 1–28, 2023.

[18] F. Bäumer, J. Schlegel, and M. Geierhos, "The shift from writing to reviewing: How AI code generators change developer workflows," *IEEE Trans. Softw. Eng.*, 2024.

[19] J. Brooke, "SUS-A 'quick and dirty' usability scale," in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, Eds. London, U.K.: Taylor & Francis, 1996, pp. 189–194.

[20] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quart.*, vol. 13, no. 3, pp. 319–340, 1989.

[21] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," in *Advances in Psychology*, vol. 52, P. A. Hancock and N. Meshkati, Eds. Amsterdam, Netherlands: North-Holland, 1988, pp. 139–183.

[22] T. Liu *et al.*, "RepoBench: Benchmarking repository-level code auto-completion systems," *arXiv preprint arXiv:2309.00684*, 2023.

[23] C. Parnin and S. Rugaber, "Resumption strategies for interrupted programming tasks," *Softw. Qual. J.*, vol. 19, no. 1, pp. 5–34, 2011.

[24] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cogn. Sci.*, vol. 12, no. 2, pp. 257–285, 1988.