The Design Space of LLM-Based AI Coding Assistants: An Analysis of 90 Systems in Academia and Industry

Sam Lau and Philip J. Guo
UC San Diego
La Jolla, CA, USA

Abstract—**Over the past few years, millions of people have been using LLM-based AI tools to aid in programming, data analysis, and software engineering tasks. These AI coding assistants range from specialized tools like GitHub Copilot to general-purpose chatbots like Claude. In parallel, academics have published dozens of papers on forward-looking prototypes to expand our collective thinking beyond present-day industry trends. However, despite rapid advances in both sectors in recent years, we still lack an understanding of how their designs relate to one another and what tradeoffs are commonly made. At this key moment in 2025 when design patterns are starting to emerge, it is important to zoom out to see the forest instead of the trees. To do so, we performed the first comprehensive design analysis of 90 LLM-based AI coding assistants. We categorized the feature sets of 58 industry products and 32 academic projects, then formulated a design space that captures key variations in their user experiences. Our design space covers 10 dimensions related to UI modalities, system inputs, capabilities, and outputs. We use this design space to reveal trends in both industry and academic projects across three eras ranging from autocomplete to chat to agent-based interfaces. Lastly, to address the question of who the target users of these tools are, we present six user personas whose preferences lie in different regions of our design space: professional software engineers, HCI researchers and hobbyist programmers, UX designers, conversational programmers (e.g., product managers and marketers), data scientists, and students.**

Index Terms—**AI coding assistant, LLM, design space**

I. Introduction

Researchers in programming languages, software engineering, and AI have been working for decades on techniques like programming-by-demonstration, program synthesis, and semantic code search, with the ultimate goal of having the computer write more code for humans [1]–[3]. These technologies started to get widespread adoption in 2021 with the launch of GitHub Copilot [4] and, soon afterward, ChatGPT along with dozens of *AI coding assistants* based on LLMs (Large Language Models). In the ensuing four years, there has been a frenzy of activity at large companies, startups, and academic research labs on using LLMs for programming.

However, as both researchers and practitioners in this field, we noticed a gap in how people have been discussing AI coding assistants so far: On one hand, we see many teams across industry and academia creating new systems and writing about their low-level technical details in isolation. On the other, we see 'thought leaders' and journalists making high-level proclamations like how AI means "The End of Programming" [5] or maybe "The End of Programming as We Know It" [6]. There are also many articles on job displacement and economic implications, with titles like "Are Coders' Jobs At Risk? AI's Impact On The Future Of Programming" [7].

But despite all this chatter in recent years, our field still lacks a comprehensive analysis of AI coding assistants *that is both technical and broad*. We feel the timing is now right for such an analysis since these tools have started to converge in form after the first four years of divergent experimentation (2021–2025). Thus, in mid-2025 we conducted the first design survey of AI coding assistants by categorizing the features of 90 systems: 58 industry products and 32 academic prototypes. We aim to provide an archival snapshot of this critical moment in time for future researchers and practitioners to refer to as these tools continue evolving in the coming years.

The main contribution of this paper is a *design space* that captures the most significant dimensions of variation in system features among AI coding assistants in mid-2025. Our approach follows the long-standing HCI tradition of formulating design spaces for technical systems in domains including end-user programming [8], [9], information visualization [10], [11], tangible user interfaces [12], livestreaming tools [13], and computational notebook technologies [14], [15].

Figure 1 shows the ten dimensions of our design space, grouped into four themes: user interface (development environment, user actions, initiative), system inputs (input format, semantic context, personalization), capabilities (autonomy, system actions), and outputs (output format, explainability). This design space lets tool builders see the range of possibilities and weigh trade-offs; and it enables researchers to have a shared vocabulary with which to discuss design variations.

Using this design space as a framework, we discovered overarching trends in both industry and academic systems in this space. For instance, both have progressed through three UI eras so far: 1) tab autocomplete interfaces in 2021–2022, 2) chat-based interactions in 2023–2024, 3) toward autonomous AI agents in 2024–2025. We noticed that industry products often prioritize raw speed (with slogans like "Code at the speed of thought") [16] while academic projects more deeply explored other design dimensions such as scaffolding to help metacognition and self-reflection. We also saw indus-

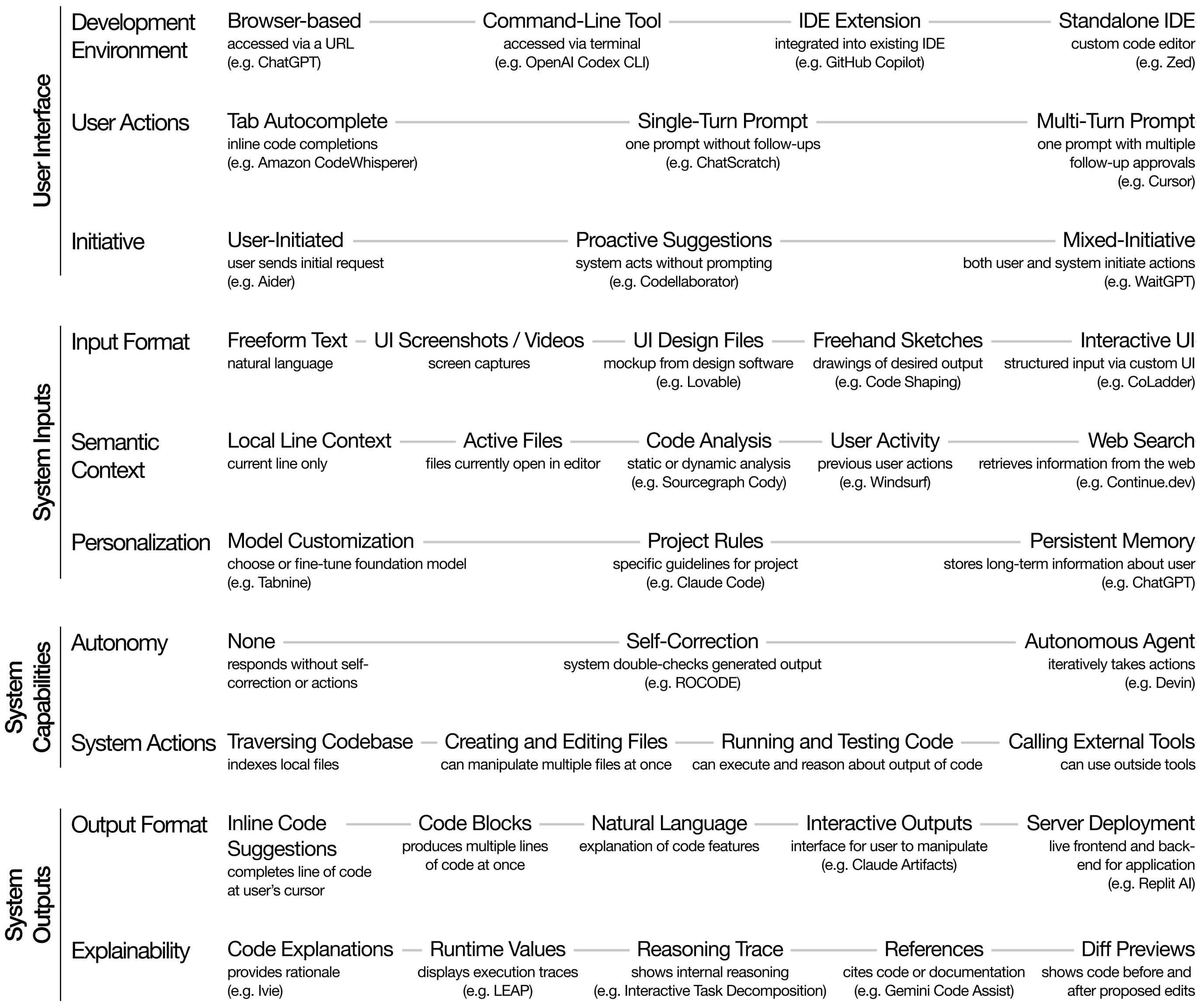
User Interface
Development Environment
Browser-based
accessed via a URL (e.g. ChatGPT)
Command-Line Tool
accessed via terminal (e.g. OpenAI Codex CLI)
IDE Extension
integrated into existing IDE (e.g. GitHub Copilot)
Standalone IDE
custom code editor (e.g. Zed)
User Actions
Tab Autocomplete
inline code completions (e.g. Amazon CodeWhisperer)
Single-Turn Prompt
one prompt without follow-ups (e.g. ChatScratch)
Multi-Turn Prompt
one prompt with multiple follow-up approvals (e.g. Cursor)
Initiative
User-Initiated
user sends initial request (e.g. Aider)
Proactive Suggestions
system acts without prompting (e.g. Codellaborator)
Mixed-Initiative
both user and system initiate actions (e.g. WaitGPT)
System Inputs
Input Format
Freeform Text
natural language
UI Screenshots / Videos
screen captures
UI Design Files
mockup from design software (e.g. Lovable)
Freehand Sketches
drawings of desired output (e.g. Code Shaping)
Interactive UI
structured input via custom UI (e.g. CoLadder)
Semantic Context
Local Line Context
current line only
Active Files
files currently open in editor
Code Analysis
static or dynamic analysis (e.g. Sourcegraph Cody)
User Activity
previous user actions (e.g. Windsurf)
Web Search
retrieves information from the web (e.g. Continue.dev)
Personalization
Model Customization
choose or fine-tune foundation model (e.g. Tabnine)
Project Rules
specific guidelines for project (e.g. Claude Code)
Persistent Memory
stores long-term information about user (e.g. ChatGPT)
System Capabilities
Autonomy
None
responds without self-correction or actions
Self-Correction
system double-checks generated output (e.g. ROCODE)
Autonomous Agent
iteratively takes actions (e.g. Devin)
System Actions
Traversing Codebase
indexes local files
Creating and Editing Files
can manipulate multiple files at once
Running and Testing Code
can execute and reason about output of code
Calling External Tools
can use outside tools
System Outputs
Output Format
Inline Code Suggestions
completes line of code at user's cursor
Code Blocks
produces multiple lines of code at once
Natural Language
explanation of code features
Interactive Outputs
interface for user to manipulate (e.g. Claude Artifacts)
Server Deployment
live frontend and back-end for application (e.g. Replit AI)
Explainability
Code Explanations
provides rationale (e.g. Ivie)
Runtime Values
displays execution traces (e.g. LEAP)
Reasoning Trace
shows internal reasoning (e.g. Interactive Task Decomposition)
References
cites code or documentation (e.g. Gemini Code Assist)
Diff Previews
shows code before and after proposed edits

Fig. 1. The design space of LLM-based AI coding assistants, which we formulated by analyzing the features of 90 projects across academia and industry.

try products converging toward a common feature set while academic projects diverged more in form.

Lastly, we use our design space to illustrate diverse use cases for AI coding assistants (Figure 2). We identified six user personas whose needs occupy different segments of the design space: 1) professional software engineers need power tools that can work within large existing codebases. 2) HCI researchers and hobbyist programmers prefer a combination of AI assistance and hand-coding to create prototypes from scratch in order to validate new ideas. 3) UX designers can use AI to turn mockups from Figma and other design tools into working interactive code prototypes. 4) People such as product managers, sales, marketing, and executives (called 'conversational programmers' [17]) can use AI in a 'vibe coding' [18] way to make simple demos without looking at the underlying code. 5) Data scientists may value explainability from AI tools to better validate outputs. 6) Students learning to code are best served by AI tools that make them reflect on their code and provide pedagogically-appropriate explanations.

Like prior design space analyses, our dimensions and personas come from our own interpretations of the data, and we want to encourage broader community-wide validation efforts.

In sum, this paper's contributions are:

- A design analysis of 90 LLM-based AI coding assistants in academia and industry up to mid-2025.
- A ten-dimensional design space that covers variations in user-facing features across all of these systems.
- A comparative analysis of industry products and academic prototypes to reveal emerging trends across both.
- Six personas representing typical types of users of AI coding assistants, and how their preferences for tool features differ along our design space dimensions.

II. Related Work

There have been many papers published recently on specific LLM technologies applied to programming which we cite throughout this paper. However, to our knowledge, there has not yet been a survey paper that comprehensively reviews the literature in this field. Our work is novel since it contributes a survey of both the academic literature and industry products.

A nearby cluster of related work is empirical studies of the user experience of specific AI coding assistants, done via controlled lab studies, user interviews, chat log analysis, or online surveys [19]–[24]. These projects focus on testing a single tool (mostly GitHub Copilot for earlier work, followed by ChatGPT) or a small group of tools. The goal of these studies is to characterize usability rather than system features.

Methodologically, the closest related work are HCI survey papers that formulate design spaces in other domains such as end-user programming [8], [9], information visualization [10], [11], tangible user interfaces [12], livestreaming tools [13], and computational notebook technologies [14], [15]. Most recently, Morris et al. described a design space of general-purpose GenAI interactions (not specific to coding) in a 2023 arXiv report [25]; and Wang et al. surveyed the space of human-AI interactions in online social learning [26]. We extend these lines of work to characterize LLM-based AI coding assistants.

III. Methods

A. Defining "AI Coding Assistant" and Scoping Our Study

To realistically scope our study we first formed a definition of "AI coding assistant" based on both our experiences as software tools researchers and our observations of industry developments. ***We define an AI coding assistant as an LLM-based tool that generates code based on user inputs, which can consist of human-written code snippets, natural language requests via text or voice, direct-manipulation gestures, or visual inputs such as sketches or screenshots.***

Based on our definition, a tool does not need to *exclusively* generate code in order to qualify as an AI coding assistant. For instance, general-purpose LLM-based chatbots such as ChatGPT and Claude qualify since users can prompt them to generate code (in addition to many other tasks). Note that even coding-specific tools like GitHub Copilot perform other tasks too, such as explaining code or helping to debug.

We restrict our definition to *LLM-based tools* in order to scope our study to modern AI coding assistants that began around 2021 with the launch of GitHub Copilot. We acknowledge that researchers have been working for decades on several foundational lines of work that serve similar roles as LLM AI coding assistants, such as PBD (Programming-by-Demonstration), PBE (Programming-by-Example), program synthesis tools [1], and earlier work on training neural networks on text-to-code embeddings [2]. These are out of scope for our study; we refer readers to the 2020 workshop report *Deep Learning & Software Engineering: State of Research and Future Directions* [3] for an overview of pre-LLM research.

While we acknowledge that coding is only one part of the entire software development lifecycle, we focus our study on AI tools that directly help programmers to write new code. Thus, *we did not include the broader ecosystem of AI software engineering tools* that are specifically made for tasks such as static analysis, log/trace analysis, automated debugging [27]–[29], program repair [30], [31], DevOps, CI/CD, test suite management, documentation, code reviews, or security audits. We also excluded 'no-code' products that directly generate apps without ever exposing the underlying code to users.

B. Gathering AI Coding Assistants in Academia and Industry

We looked for both academic and industry projects that fit our definition at the time of writing in May 2025.

For academic projects we looked through the entire proceedings of major HCI and SE (software engineering) research conferences from Jan 2021 to May 2025 and followed relevant citations from those seed papers. These conferences were CHI, UIST, CSCW, DIS, IUI, ICSE, OOPSLA, FSE, ASE, ISSTA, and VL/HCC. We included only system papers that described prototypes of new AI coding assistant tools, not empirical studies of how people used existing tools like GitHub Copilot. We also did not include purely algorithmic or 'backend' papers such as those that trained custom neural nets for code retrieval [32], since they do not describe new user-facing tools.

For industry projects we searched online for relevant company webpages, product announcements, technical documentation, blog posts, tech news items, industry conference talks, YouTube videos, podcasts, and discussion forum posts (e.g., on Hacker News) for any released products that fit our definition. We also consulted with colleagues familiar with the latest work in LLMs and software developer tools to augment our list.

C. Data Overview and Analysis

For each of the 90 systems that met our criteria, the research team analyzed its features by reading relevant papers and user guides/documentation, watching demo and talk videos, and trying out selected tools that were publicly available. We focused our qualitative analysis on technical features rather than business-oriented ones such as pricing, licensing models, or target markets. The research team met multiple times to merge our notes, categorize them into themes using an inductive analysis approach [33], and iterate until we could not find additional features. To resolve inconsistencies in initial rounds of classification, we discussed results iteratively in team meetings with one researcher taking the lead on industry systems and the other on academic systems, due to each having more experience with those respective areas.

Then to formulate a design space from these features, we followed a similar methodology as Segel and Heer [10], who created a design space of narrative visualizations from content analysis of interactive webpages, and Lau et al., who did so for computational notebook systems [14]. Specifically, we sought to distill generalizable concepts from specific AI coding assistant features and surface which concepts were shared across multiple instances. We made several iterations as a team before finalizing our 10 dimensions, which are grounded in user-facing features.

D. Study Design Limitations

We analyzed user-facing features of AI coding assistants, but we did not formally evaluate their usability or performance via user testing, benchmarks, or controlled experiments. We did not cover pre-LLM coding assistants, so we may have missed interaction modes that are unique to earlier systems. We did not analyze differences in underlying LLMs (e.g., GPT-4 vs. Deepseek) since they do not affect UI features. And while we tried our best to be comprehensive, we cannot guarantee that we found *all* available systems at the time of writing.

There is subjectivity in how we picked design space dimensions, so other researchers may choose different ones. We did not empirically validate our choices with the broader community of researchers or practitioners. Note that prior design space analyses [10], [14] also relied solely on the expertise of their authors to interpret the data. Both authors of our study have relevant domain expertise: we have published papers on AI coding tools, used them to build software systems, and previously worked as software engineers in industry.

IV. Overview of 90 AI Coding Assistants

A. Industry Products

We first summarize the 58 industry products that met our selection criteria. Note that industry products often evolve over time to incorporate new capabilities. Since we focus on the evolution of user interaction with AI coding tools, we not only included initial product releases but also updates that added substantially different interaction paradigms (e.g., we count regular ChatGPT and ChatGPT Canvas as separate systems).

Early LLM research from 2016–2020 developed models capable of completing natural language for tasks such as text summarization and reading comprehension. An emergent property of these LLMs was their unexpected proficiency at writing code, which was documented at least as early as OpenAI GPT-2's release in 2019 [34]. The earliest LLM-based AI coding tool we found was Tabnine, which incorporated GPT-2 in 2019 to auto-complete code fragments [35].

In 2020, GPT-3 was released [36], followed by Codex, a version of GPT-3 specifically trained on code [37]. Codex became the foundation for GitHub Copilot, released in 2021 [4]. Copilot quickly gained popularity, partly due to its direct integration into the Visual Studio Code (VS Code) IDE, and acquired over one million paying users in its first two years [38]. Copilot's popularity inspired several other industry products released soon after, including Amazon CodeWhisperer [39] and Replit AI [40], both launched in mid-2022.

The release and immediate popularity of ChatGPT in late 2022 sparked an explosion of similar products [41]. ChatGPT was the first to feature a conversational UI, which significantly aided its adoption. It was also capable of writing and debugging code without requiring additional fine-tuning. Between 2023 and 2025 many companies launched their own foundation models with similar chatbot UIs, including Google [42], Anthropic [43], Meta [44], Mistral [45], Qwen [46], DeepSeek [47], and Databricks [48]. All these chatbots could process code as input prompts and produce code as outputs. (Note that we included only the most popular LLM-based chatbots in this list, but there are many others.)

Aside from powering general-purpose chatbots, advances in foundation models inspired a wave of products specifically targeting software developers. Many resembled Copilot as they were implemented as IDE extensions with autocomplete and chat interfaces, such as Cline [49], Sourcegraph Cody [50], the JetBrains AI Assistant [51], Continue.dev [52], chattr [53], Tabby [54], and Gemini Code Assist [55].

Several products were designed to run within the terminal on a user's machine, including Warp [56], Aider [57], Claude Code [58], and OpenAI Codex CLI [59] (which is, confusingly, a different product than the original Codex from 2021).

Other products were built as standalone IDEs rather than extensions to existing environments in order to implement custom UI flows, most notably Cursor [60], Devin [61], Zed [16], and Windsurf [62].

Also, AI coding assistants targeting data scientists became integrated into existing computational notebooks, including Colab [63], Deepnote [64], Observable [65], and Marimo [66].

Starting around 2024, industry also began exploring alternative forms of output beyond text and code. For example, Claude Artifacts [67], ChatGPT Canvas [68], and Gemini Canvas [69] generate code, execute it, and display the results to users, which often consists of data visualizations or interactive webpages. Some newer products went further by adding the capability to generate and deploy full-stack web applications from user prompts, UI screenshots, or Figma design mockups – e.g., Lovable [70], Firebase Studio [71], Vercel v0 [72], Townie [73], Bolt [74], and HeyBoss [75].

Updates since submission: In just the three months between when we submitted this paper (May 2025) and when we finalized it for publication (Aug 2025), many new industry products have been released. These include command-line interface tools (OpenAI Codex [76], Gemini CLI [77], qwen-coder [78], Amp [79], OpenCode [80]), standalone IDEs (nao [81], Void [82]), IDE extensions (Kiro [83]), agentic tools (Jules [84], Factory [85], codename goose [86], Claude Artifacts [87], Copilot Agent [88]), and tools for designers to generate web UI code (Figma Make [89], Google Stitch [90]). Most of these fall into the same categories as existing tools we analyzed. But one new trend is the emergence of tools that orchestrate other tools: For example, Claude Squad [91] and Conductor [92] enable users to run many instances of Claude Code in parallel to act like a 'fleet' of software engineers.

B. Academic Prototype Systems

We now summarize the 32 systems described in academic research papers in HCI and software engineering. Whereas industry systems are end-to-end products meant for direct consumer use, academic systems are often prototypes that show a proof-of-concept of a single innovation.

The earliest paper we found was from 2022: GenLine [93] from Google Research was developed around 2020–2021 at the same time as GitHub Copilot, and it supported similar goals of autocomplete-style code generation within an IDE.

We found only 3 papers from 2023, and incidentally all three were about adding AI to data science environments. ColDeco [94] improves explainability for AI-generated code that operates on tabular data. Grounded Abstraction Matching [95] adds editable natural-language explanations of AI-generated Python code for data science. And McNutt et al. designed a range of UI mock-ups to augment computational notebooks with AI code completions [15]. We speculate that this early work (done in 2020–2022, all before ChatGPT's release[1]) built upon the growing momentum of data science tooling research in HCI and software engineering leading up to 2020 [14].

ChatGPT launched at the end of 2022, followed by rapid updates to LLM APIs (e.g., GPT-3.5, GPT-4) that researchers could use to build prototypes. Thus, 2023 was when many research groups started entering this space, culminating in 13 papers published in 2024. BISCUIT [96] and DynaVis [97] continued the prior year's trajectory of data science tooling by using AI to synthesize bespoke analysis UIs; Interactive Task Decomposition [98] and WaitGPT [99] also targeted data scientists but focused more on helping them understand and steer AI code generation. Ivie [100] and LEAP [101] improve AI code autocompletion by adding inline explanations using static and run-time information, respectively. ClarifyGPT [102], CoLadder [103], and Language-Oriented Code Sketching [104] augment plain-text prompts with scaffolding to improve code generation quality. ChatScratch [105] and NetLogo Chat [106] built specialized AI chatbots for educational programming environments. Lastly, CodeCompose [107] and StackSpot [108] presented case studies of developing production-ready AI coding assistants within large companies.

Momentum continued through 2025, where we found 15 papers at the time of writing. Some work continued the 'first era' of Copilot-style AI code autocomplete, while others pushed toward multimodal inputs, autonomous agents, and emerging trends of 'vibe coding' (i.e., creating prototypes by only prompting the AI but not looking at the code) [18]. For instance, CodeA11y [109] is like Copilot but ensures that AI-generated code follows web accessibility guidelines. Codellaborator [110] and Chen et al. [111] made proactive AI code suggestions. Multimodal input systems such as DCGen [112] generate UI code from screenshots, while Code Shaping [113] and Gomes et al. [114] take freehand sketches and synthesize data science code, and Dango [115] detects multimodal inputs on data tables. DBox [116] and Kazemitabaar et al. [117] augment coding assistants for CS education by adding UI affordances to make learners engage more thoughtfully with AI-generated code. PAIL [118] and intention-based code refinement [119] bridge the gap between users specifying high-level software designs and getting an AI to effectively generate code that is well-aligned with designer intentions. PInG [120] augments prompts with editable code comments to help users steer LLM code generation, while ROCODE [121] uses program analysis to steer the LLM toward generating error-free code. Lastly, DynEx [122] and Dream Garden [123] enable 'vibe coding' [18] by letting users prototype an end-to-end app using high-level text prompts.

[1]We assume a publication lag time of 1.5 to 2 years for building and validating system prototypes and then getting peer-review papers published.

V. The Design Space of AI Coding Assistants

We grouped the 10 design space dimensions from Figure 1 into four themes: user interface, system inputs, capabilities, and outputs. We now present each dimension in turn.

A. User Interface

1) Development Environment: At first glance the most visible property of an AI coding assistant is where it resides.

General-purpose chatbots like ChatGPT are *browser-based*, so users must copy-paste AI-generated code from their web browser into their own project files [41]. However, some industry systems like Bolt [74], Townie [73], and Replit [40] are full browser-based IDEs where code is run in-browser as well. This is also common in systems that support data science workflows, such as Deepnote [64], Observable [65], Marimo [66], and Colab AI [63], which allow users to access AI tools within web-based computational notebooks.

Others are *command-line tools*, including Aider [57], Claude Code [58], OpenAI Codex CLI [59], and Warp [56]. Users chat with them in the terminal, where they can autonomously read, edit, and create files on the user's machine.

Many systems in both academia and industry are implemented as *IDE extensions* or plug-ins, such as GitHub Copilot for Visual Studio Code [38] and chattr for RStudio [53]. Integrating into an IDE allows for interactions like inline code autocomplete, explanatory pop-ups, and chat sidebars.

Lastly, some industry products like Cursor [60], Windsurf [62], Tabby [54], and Zed [16] are *standalone IDEs* that can integrate AI more deeply into their user interfaces without the restrictions that IDEs typically impose on extensions.

2) User Actions: Next, systems differ in what kinds of user actions invoke the AI assistant.

The simplest action, popularized by the original GitHub Copilot in 2021, is *tab autocomplete* where the user sees grayed-out ghost text of an AI code suggestion under their cursor and can press Tab to accept it [124]. Newer versions of Cursor, Windsurf, and Copilot take this idea farther by automatically moving the cursor to the AI-predicted 'next edit location' and generating related suggestions when the user hits Tab multiple times in rapid succession [125]–[127].

Most systems support a *single-turn prompt* where the user constructs a prompt to send to the AI assistant and directly receives code. Most often, this prompt consists of text instructions, but multimodal LLMs support voice, images, or even videos as prompt inputs. In addition, academic systems like CoLadder [103] and DBox [116] allow users to construct prompts using structured UIs, which the system frontend compiles into a single text prompt to send to the LLM.

Systems that support *multi-turn prompts* extend single-turn prompting by sometimes asking users for extra input rather than generating the requested code immediately. For example, tools such as ClarifyGPT [102], Windsurf [62], and Replit AI [40] will ask the user clarification questions to better understand their requirements. Additionally, systems like Cursor decompose a single prompt into sequential implementation steps, requiring explicit user approval at each stage [60]. This incremental approach aims to increase user control, especially for more complex tasks.

3) Initiative: Most systems can be triggered by *user-initiated* actions such as writing and submitting a text prompt. Some academic prototypes such as Codellaborator [110] and Chen et al. [111] implement *proactive suggestions* where the system generates code or writes documentation in the background without user initiation; Codellaborator also shows a second edit cursor to make users aware of what the AI is proactively doing, akin to seeing real-time edits on Google Docs.

A few systems are *mixed-initiative* – intertwining user- and system-initiated actions. These include 'steerable' AI assistants like WaitGPT [99] and Interactive Task Decomposition [98] where the user can see what the LLM generates in an interactive UI and make adjustments on-the-fly. Many industry systems support a narrow form of mixed-initiative interaction by having user-initiated features alongside system-initiated features. Most commonly, systems will provide a chat interface that is entirely user-initiated while suggesting code completions that are entirely system-initiated, similar to early versions of GitHub Copilot [124] and JetBrains AI Assistant [51]. However, we are unaware of industry systems that provide mixed-initiative interactions within the *same* feature.

B. System Inputs

4) Input Format: What does the AI assistant take as input from the user? All systems support *freeform text*, which can be a mix of code and natural language instructions. Some programmers use speech-to-text features of ChatGPT and similar tools to talk to the AI, but that speech is simply transcribed to text. ChatScratch [105] prioritizes voice input in the UI since its target audience is young kids with limited typing skills. Other academic systems like Language-Oriented Code Sketching [104] and PInG [120] augment the user's text input with scaffolding to help the LLM generate better code.

Some systems support visual inputs such as *screenshots/videos* since they use multimodal LLMs (e.g., GPT-4o [128]). The most relevant use case here is passing in a UI design mock-up image and having it generate HTML/CSS code that instantiates that design in code. These types of inputs are supported by industry tools like Claude Code [58], and recent versions of Gemini can even generate code based off of videos that show users interacting with an application [129]. To improve code generation quality, academic systems like DCGen [112] segment a screenshot into constituent UI components before passing it to the LLM.

Going beyond screenshots, designer-oriented tools such as v0, Bolt, and Lovable can take *UI design files* as input, most commonly created by tools like Figma [70], [72], [74]. This allows the AI to more reliably generate HTML/CSS/JavaScript frontend code that instantiates a UI design mockup.

Another type of visual input is *freehand sketches* where the user can draw a UI and have the system generate skeleton HTML/CSS code that matches the drawing; with additional text prompting, the system can fill in color palettes and other aesthetic customizations. Academic systems like Code Shaping [113] take this idea further by connecting pen strokes to on-screen elements like code and console output.

Some systems have custom *interactive UIs* for users to precisely specify their inputs. For instance, industry tools like Lovable [70] and HeyBoss [75] generate interactive webpages as output and let users select a portion of that page to reference in their follow-up prompts. Academic systems like CoLadder [103] and Interactive Task Decomposition [98] have UIs that scaffold more structured text inputs. And data science oriented systems like ColDeco [94] and Dango [115] let users select portions of data tables to prompt the AI assistant.

5) Semantic Context: Aside from direct user inputs, systems automatically collect context from the user's codebase and surrounding artifacts when constructing an LLM prompt.

Early systems like the original GitHub Copilot had only *local line context* where it would copy the lines of code surrounding the user's cursor into the prompt [124]. Since older LLMs had small context windows (roughly 4000 'words'), an entire source file may not fit into one query. As context windows grew larger, systems were able to pull in entire *active files* that the user opened or edited recently in their IDE.

However, these approaches still treat source files as plain text, so other systems improved on this by doing *code analysis* to surface semantically meaningful context for the LLM. For instance, Sourcegraph Cody and Windsurf do static analysis to summarize the structure of large codebases that are too big to fit in the context window [50], [126]. ROCODE [121] integrates static analysis into the LLM's token generation algorithm to allow it to error-correct. And tools like ClarifyGPT [102] do automatic test case generation, mutation testing, and dynamic analysis to enrich LLM prompts.

Another form of context involves tracking *user activity* within the IDE. For instance, agentic IDEs like Windsurf track what the user has edited recently and what terminal commands they have run to include as context when the agent operates on the user's behalf [130]. Proactive AI assistants like Codellaborator [110] track when the user has momentarily paused editing in order to pop up timely proactive suggestions.

Systems can also grab context outside the user's codebase by calling out to tools like *web search*. Most commonly, this is used to fetch technical documentation and usage examples for imported libraries. For example, Continue.dev can automatically retrieve public library documentation [52], and Sourcegraph Cody can search for internal documentation by integrating with common industry tools like Notion for notes and Prometheus for system monitoring [50].

6) Personalization: AI coding assistants are frontends to underlying foundation models such as GPT-4, so many support *model customization* by letting the user select an LLM and inputting an API key for payment. In addition, products such as Tabnine allow enterprise users to fine-tune their own model on their company's internal codebase [131].

In addition, systems like Cursor, Windsurf, and Claude Code support *project rules* written as Markdown text files where the user specifies project-wide coding conventions, domain knowledge, and library versions; those systems can also analyze the user's codebase to help them draft those rules [58], [130], [132]. These project rules get automatically included in the LLM prompts and guide the system to generate new code that is consistent with the existing codebase.

Systems such as ChatGPT, Gemini, and Windsurf have *persistent memory* features that remember past conversations so they can better personalize suggestions [130], [133]. These features take advantage of growing context windows in modern LLMs: For reference, GPT-3.5, which powered the original ChatGPT release in late 2022, has a context of 4096 tokens, while Gemini can take up to 2 million input tokens as of 2025.

C. System Capabilities

7) Autonomy: Both academic and industry systems have trended toward more autonomy over time, with the baseline being *none* – the user initiates an action and the system constructs a single LLM prompt in response.

A partial form of autonomy is *self-correction* where the system iteratively applies an AI-produced code edit, runs the code, and if there is an error, tries to correct it by passing the error message to the LLM and seeing if it can generate a fix. One notable example is Cline, which can double-check generated code by opening the user's web browser and interacting with the web application to detect mistakes [134]. Academic systems like ROCODE [121] implement self-correction by altering the LLM's token generation algorithm so that it can backtrack and error-correct within a single invocation.

Recent academic papers like DynEx [122] and Dream Garden [123], along with the latest updates from industry products like Devin [61], Windsurf [62], Cursor [60], and Claude Code [58], have been working toward *autonomous agents* where a single user prompt can trigger the assistant to make multiple LLM queries, analyze their responses, edit different parts of the codebase, run terminal commands, and modify the filesystem on the user's behalf. As a safety net, the user can review and approve agent actions, or they can let it run fully autonomously. Agents can spawn their own sub-agents that run in background processes. The user experience challenge here is supporting the human in reviewing edits made by multiple agents that touch different parts of the codebase and finish at unpredictable times. There is a risk that these agents' edits may overlap or clash with each other.

8) System Actions: In addition to generating code suggestions for users to consider, AI assistants can also perform a variety of actions. These range from *traversing the entire codebase* to point out what other related locations the user may consider editing to automatically *creating and editing files* on the user's behalf. Assistants can also be *running and testing code* in the background, report their results to the user, and (if given permission) automatically patch the code to fix runtime errors. This pattern is especially common for command-line tools such as Codex CLI [59] and Aider [57]. Lastly, assistants are now *calling external tools* such as web search (e.g., Continue.dev [52]), static analysis (e.g., Sourcegraph Cody [50]), and a growing ecosystem of developer tools connected via the emerging standard of MCP (Model Context Protocol) [135].

D. System Outputs

9) Output Format: AI coding assistants present their outputs in diverse formats ranging from *inline code suggestions* at the user's cursor position in an IDE (e.g., Copilot [124], Windsurf [62], Zed [16]) to *generating code blocks* that either display standalone (e.g., ChatGPT web interface) or in an IDE. They may augment code with *natural language* explanations in either an inline format like Ivie [100] or in a separate explanation pane like Grounded Abstraction Matching [95]. And some generate *interactive outputs*: for example, industry products like ChatGPT Canvas [68], Gemini Canvas [69] and Claude Artifacts [58] generate interactive web frontends that users can test directly in their browser. Academic systems like DynaVis explored bespoke output visualization widgets [97] and UI overlays that encourage users to critically engage with the generated code rather than quickly accepting the AI's suggestions [117]. Lastly, a growing number of industry tools produce production-ready *server deployments* that not only have interactive frontends but also have backends that enable data persistence, such as Townie [73], Replit AI [40], Bolt [74], Lovable [70], and v0 [72].

10) Explainability: Our final dimension include ways for the AI assistant to explain its actions. The most straightforward form is static *code explanations* first implemented by industry tools like GitHub Copilot [4] that provide explanations for generated code, then extended by academic systems like Ivie [100] and Grounded Abstraction Matching [95].

Some academic systems go further by running the AI-generated code and presenting dynamic visualizations of *runtime values* (e.g., ColDeco [94], LEAP [101], WaitGPT [99]).

Chat-based systems can give *reasoning traces* to show their internal step-by-step 'thought' process to make it clearer to users why they chose to generate a certain piece of code, which is common for systems that use reasoning models like Deepseek R1 [47]. Academic systems like Interactive Task Decomposition [98] augment these traces with a lightweight UI to allow users to edit those traces and re-run to refine their original query. Relatedly, McNutt et al. [15] prototyped other forms of reasoning traces like showing LLM confidence levels and alternative token choices in computational notebooks.

LLM-generated explanations may also include *references* to primary sources (e.g., documentation webpages) or other parts of the user's codebase. One notable example is a recent update to GitHub Copilot which notifies the user when AI-generated code matches public open-source code on GitHub [136].

Lastly, AI-enabled IDEs like Windsurf and Zed show *diff previews* of proposed edits that agents are about to make, often spread throughout different parts of the codebase, so that users can review and approve changes, much like doing a code review [16], [126].

VI. Discussion

High-Level Observations: One recurring observation as we performed this research was how hard it could be to find out precisely what some industry products did. Product webpages can be filled with marketing hype and superlative language, and articles about these products often copy that overhyped style from company press releases. For instance, the homepage of one of the most popular ones, Windsurf, describes its IDE as "where the work of developers and AI truly flow together, allowing for a coding experience that feels like literal magic." When clicking through to the Features page, we were met by the headline: "So what can you do with this tool? It's so powerful that you'll wonder what you *can't* do." Thus, to cut through all the hype we needed to dig into technical documentation, watch tutorial videos made by outside users, and try using some of the tools ourselves.

In contrast, academic papers strive to present a more balanced assessment of the systems they describe and then situate those systems within the landscape of related work. Also, each paper describes a particular targeted innovation along one dimension of our design space, rather than being an end-to-end product that innovates along multiple dimensions.

Both industry and academic work progressed through the same three eras so far: 1) tab autocomplete in 2021–2022, 2) chat-based interactions in 2023–2024, 3) toward autonomous agents in 2024–2025. Major industry products like GitHub Copilot, Cursor, and Windsurf now incorporate all three modes into a single general-purpose IDE. In contrast, academic papers prototype a single interaction in more detail or explore a paradigm for a specific domain (e.g., DreamGarden implements agents for game programming [123]).

The Need for Speed* vs. *The Wisdom of Waiting: A major theme we saw across many industry products was their emphasis on improving coding speed. For instance, the features webpage for Cursor has the headline "Features: Build software faster" and one of its most prominent features is described as "Tab, tab, tab. Cursor lets you breeze through changes by predicting your next edit." Similarly, Zed's tagline is "Code at the speed of thought." With speed as an overarching goal, industry products push our design space toward more autonomy via agents and lower friction in user interactions (i.e., tab tab tab). This likely aligns with commercial interests, since potential customers and investors are attracted to the promise of helping programmers to get more work done faster.

But is faster always better? Academic projects can challenge this assumption since they do not need to optimize for commercial goals. Most notably, Kazemitabaar et al. prototype AI coding assistants that *slow down the user* by introducing "Friction-Induced AI, wherein the AI does not allow users to use its generated solution immediately. Instead, it engages the user in the process of generation, while challenging them and providing opportunities for reflection. [...] it refers to temporarily slowing the user's interaction, ensuring they engage critically, rather than passively using AI's output." [117] And projects like PAIL [118], DynEx [122] and intention-based code refinement [119] scaffold users to take their time to critically reflect on and refine their designs rather than blasting them with AI-generated code as fast as possible. Thus, we believe academia is better-positioned to innovate along these dimensions due to lack of direct market pressures.

Feature Convergence (industry) vs. Divergence (academia): In the first four years of LLM-based AI coding assistants (2021–2025), companies have invested large amounts of money and personnel into creating products that compete for market share. We noticed a 'convergent evolution' in design ideas where products built by different companies end up with similar feature sets over time. Most notably, starting in 2024 many products added support for AI agents with multi-prompt reasoning and external tool calling. Perhaps this phenomenon is partly due to user needs being similar, so these companies all get the same kinds of user testing results and customer feedback that inform their development plans. Another reason may be them deciding to purposely copy popular features from competitors using a 'fast-follow' strategy that is common in tech, as summarized by the creator of Townie [73]. Industry is well-positioned to hone in on immediate user needs and put in the engineering effort necessary to polish the user experience.

In contrast, academic publications by definition need to be clearly differentiated from prior work, so we saw more diversity in interaction modalities across our ten design space dimensions. For instance, systems like CoLadder [103] and Interactive Task Decomposition [98] experimented with more elaborate interactive UIs to help users refine their prompts. Relatedly, since researchers are not constrained by adding features that appeal to a mass market of customers, they can target niche domains (e.g., ChatScratch [105] for kids) or prototype more esoteric features. This freedom enables them to prototype more divergent interfaces than what companies are willing to invest in, which in turn may inspire the next generation of AI coding assistants that go beyond today's user needs. That said, skeptics can argue that academic ideas may not be based on compelling user needs and may lead to user experiences that are too complex for widespread adoption.

Who Are AI Coding Assistants For? Six User Personas

The term "AI coding assistant" is very broad. But who exactly is being assisted by the AI? In other words, what kinds of people use these tools, and what UI affordances might they want? To spark discussion here we can use our design space in Figure 1 to map selected dimensions to user preferences. Figure 2 shows six user personas that we propose:

1) **Professional Software Engineers** are the primary target users for many industry products. They likely prefer advanced development environments consisting of command-line and IDE-based tools (see blue highlights in Figure 2). They need tools to provide a lot of semantic context since they are working within large established codebases in their jobs (rather

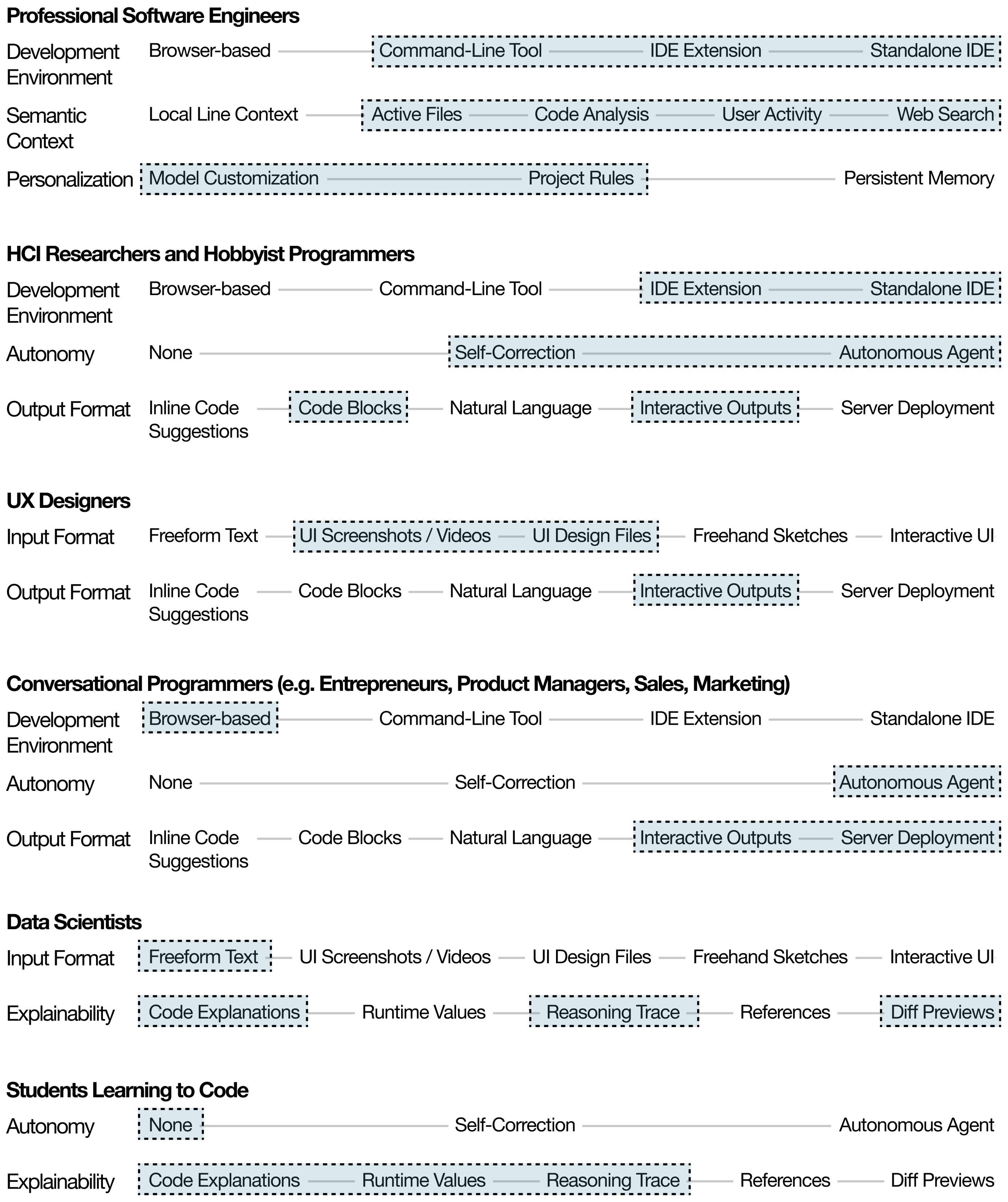

Fig. 2. Six example user personas that represent common users of AI coding assistants. For each one we show selected dimensions from the design space of Figure 1 with highlights to represent what kinds of system features they might prefer. (To save space we do not show all 10 dimensions for each persona.)

than making apps from scratch). And they want a high degree of personalization using custom models and project rules.

2) **HCI Researchers and Hobbyist Programmers** are similar to software engineers in that they want control over the code, except they are working on prototypes of new ideas rather than extending an existing production codebase. They may prefer having AI generate blocks of code for them to tweak and produce interactive outputs such as demo web apps.

3) **UX Designers** have also been finding these tools useful for their work, especially designer-focused systems like Bolt, v0, and Lovable. They are able to use them to make interactive code prototypes that are more bespoke and high-performance than what is possible in design tools like Figma (e.g., involving advanced WebGL or HTML Canvas animations). These working prototypes enable them to communicate better with software engineers when discussing how to implement real product ideas. As such, Figure 2 shows that designers value being able to input UI screenshots, animation videos, and UI design files from Figma into AI coding assistants; they also likewise want interactive outputs such as dynamic webpages.

4) **Conversational Programmers (e.g., entrepreneurs, product managers, executives, sales, marketing)** is a term coined by Chilana et al. [17] to refer to professionals who want to communicate better with programmers but who do not themselves know how to code. These people can now prompt AI to create demos of their visions without reading the code (i.e., 'vibe coding' [18]). Thus, they may prioritize browser-based AI agents that can generate interactive web outputs and deploy full-stack apps without any coding.

5) **Data Scientists** and other computational scientists value correctness and explainability since they want to ensure that their analyses are robust. They want to be able to easily read AI-generated code and understand how each line works so that they can gain some confidence that the code is processing their data in the intended way. Unlike UX designers and conversational programmers, they may not care as much about elaborate user interfaces or multimedia inputs/outputs.

6) **Students Learning to Code** are best served by projects like ChatScratch [105], DBox [116], prototypes by Kazemitabaar et al. [117], and other hint-generation and scaffolding tools for CS Education [137]. These tools should *not* be autonomous since students must learn to write and understand code themselves. Explainability is also important for students, since the pedagogical goal is to use code as a means for learning computational concepts rather than to produce professional-quality software.

Tradeoffs in Design Space Dimensions: Aside from clarifying user personas, our design space also lets AI tool developers debate tradeoffs in both user experience and engineering effort. Here are some examples for each dimension:

1) Development Environment: there is a tradeoff between ease-of-access (browser-based tools like Val Town's Townie) and power/customizability (standalone IDEs like Cursor). 2) User Actions: tab autocomplete can keep the user in flow longer while they are coding, but multi-turn prompts can give the AI more context albeit at the cost of greater user effort. 3) Initiative: tradeoffs in the user's feeling of control and understanding about what the tool is doing for them – proactive and mixed-initiative tools give up some user control for more serendipity. 4) Input Format: supporting non-text inputs requires more powerful multimodal models and engineering custom UIs atop them. 5) Semantic Context: supporting more context requires more powerful models and the engineering effort to integrate those models into products. 6) Personalization: tradeoffs between ease-of-use for novices and customizability for power users. 7) Autonomy: greater AI autonomy leads to less user control and reduced ability for a human to debug when the AI makes mistakes. 8) System Actions: similarly, the more actions an AI can take, the more that things can go wrong behind-the-scenes without the user even noticing. 9) Output Format: more elaborate outputs could be overwhelming to novice users. 10) Explainability: more advanced explanation formats may cause cognitive overload due to too many UI elements shown on-screen while coding.

Broadly speaking, even across dimensions, the more that AI does for the human user, the harder it can be for novices to develop the critical thinking skills [138] necessary to become more advanced users of AI. Experts may benefit a lot from greater automation, but novices may be denied the opportunity or motivation to develop deeper expertise in the first place.

VII. Conclusion: Can we bridge *The Two Cultures* of Academia and Industry?

We presented the first survey of 90 LLM-based AI coding assistants drawn from both industry (58 products) and academia (32 papers). Using this design space, we identified three UI eras so far (autocomplete, chat, agents) and overarching trends of industry products converging in feature sets around speed while academic prototypes pursue more diverse goals. Lastly, we propose six user personas to address "Who are AI coding assistants for?" and discussed how each may prioritize tools along different regions of our design space.

In 1959 novelist and scientist C. P. Snow warned that the rift of *The Two Cultures* between the science and humanities could hinder progress in solving important problems [139]. In doing this research we noticed a similar 'Two Cultures' split between academic and industry work in AI coding assistants where both march along in parallel without much acknowledgment of the other side's contributions. Lau et al. noted the same split in 2020 with regard to computational notebook research and products [14]. The career incentives, level of resources, and working structures differ so much that perhaps cross-dissemination of ideas is not feasible: Academia lacks the tremendous engineering resources and money required to build and experiment on production-scale systems, while industry lacks the market incentives to deploy research innovations that are too critical of AI's capabilities. However, we feel that each has much to offer the other side, and closer integration can move the field forward. The open question is: how can we bridge the two cultures gap in the years to come?

Acknowledgments

Thanks to Irene Hou for help with proofreading.

References

[1] S. Gulwani, O. Polozov, R. Singh *et al.*, "Program synthesis," *Foundations and Trends® in Programming Languages*, vol. 4, no. 1-2, pp. 1–119, 2017.

[2] X. Rong, S. Yan, S. Oney, M. Dontcheva, and E. Adar, "Codemend: Assisting interactive programming with bimodal embedding," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 247–258. [Online]. Available: https://doi.org/10.1145/2984511.2984544

[3] P. Devanbu, M. Dwyer, S. Elbaum, M. Lowry, K. Moran, D. Poshyvanyk, B. Ray, R. Singh, and X. Zhang, "Deep learning & software engineering: State of research and future directions," *arXiv preprint arXiv:2009.08525*, 2020.

[4] D. Gershgorn, "GitHub and OpenAI launch a new AI tool that generates its own code," https://www.theverge.com/2021/6/29/22555777/github-openai-ai-tool-autocomplete-code, Jun. 2021.

[5] M. Welsh, "The end of programming," *Communications of the ACM*, vol. 66, no. 1, pp. 34–35, 2022.

[6] T. O'Reilly. (2025) The end of programming as we know it. [Online]. Available: https://www.oreilly.com/radar/the-end-of-programming-as-we-know-it/

[7] S. Duranton. (2024) Are coders' jobs at risk? ai's impact on the future of programming. [Online]. Available: https://www.forbes.com/sites/sylvainduranton/2024/04/15/are-coders-jobs-at-risk-ais-impact-on-the-future-of-programming/

[8] B. Hartmann, L. Yu, A. Allison, Y. Yang, and S. R. Klemmer, "Design as exploration: Creating interface alternatives through parallel authoring and runtime tuning," in *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 91–100. [Online]. Available: https://doi.org/10.1145/1449715.1449732

[9] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: Integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 513–522. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753402

[10] E. Segel and J. Heer, "Narrative visualization: Telling stories with data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1139–1148, Nov 2010.

[11] T. Horak, A. Mathisen, C. N. Klokmose, R. Dachselt, and N. Elmqvist, "Vistribute: Distributing interactive visualizations in dynamic multi-device setups," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3290605.3300846

[12] G. W. Fitzmaurice, H. Ishii, and W. A. S. Buxton, "Bricks: Laying the foundations for graspable user interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '95. USA: ACM Press/Addison-Wesley Publishing Co., 1995, p. 442–449. [Online]. Available: https://doi.org/10.1145/223904.223964

[13] I. Drosos and P. J. Guo, "The design space of livestreaming equipment setups: Tradeoffs, challenges, and opportunities," in *Proceedings of the 2022 ACM Designing Interactive Systems Conference*, ser. DIS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 835–848. [Online]. Available: https://doi.org/10.1145/3532106.3533489

[14] S. Lau, I. Drosos, J. M. Markel, and P. J. Guo, "The design space of computational notebooks: An analysis of 60 systems in academia and industry," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2020, pp. 1–11.

[15] A. M. McNutt, C. Wang, R. A. DeLine, and S. M. Drucker, "On the Design of AI-Powered Code Assistants for Notebooks," in *CHI '23: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1–16.

[16] N. Sobo, "Introducing Zed AI - Zed Blog," https://zed.dev/blog/zed-ai, Aug. 2024.

[17] P. K. Chilana, R. Singh, and P. J. Guo, "Understanding conversational programmers: A perspective from the software industry," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1462–1472. [Online]. Available: https://doi.org/10.1145/2858036.2858323

[18] S. Willison. (2025) Not all AI-assisted programming is vibe coding (but vibe coding rocks). [Online]. Available: https://simonwillison.net/2025/Mar/19/vibe-coding/

[19] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools," *Queue*, vol. 20, no. 6, p. 35–57, Jan. 2023. [Online]. Available: https://doi.org/10.1145/3582083

[20] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491101.3519665

[21] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, Apr. 2023. [Online]. Available: https://doi.org/10.1145/3586030

[22] R. Khojah, M. Mohamad, P. Leitner, and F. G. de Oliveira Neto, "Beyond code generation: An observational study of chatgpt usage in software engineering practice," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3660788

[23] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of ai programming assistants: Successes and challenges," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3608128

[24] R. Wang, R. Cheng, D. Ford, and T. Zimmermann, "Investigating and designing for trust in ai-powered code generation tools," in *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1475–1493. [Online]. Available: https://doi.org/10.1145/3630106.3658984

[25] M. R. Morris, C. J. Cai, J. Holbrook, C. Kulkarni, and M. Terry, "The design space of generative models," 2023. [Online]. Available: https://arxiv.org/abs/2304.10547

[26] Q. Wang, I. Camacho, S. Jing, and A. K. Goel, "Understanding the design space of AI-mediated social interaction in online learning: Challenges and opportunities," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. CSCW1, Apr. 2022. [Online]. Available: https://doi.org/10.1145/3512977

[27] K. H. Levin, N. van Kempen, E. D. Berger, and S. N. Freund, "Chatdbg: Augmenting debugging with large language models," in *Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, 2025. [Online]. Available: https://arxiv.org/abs/2403.16354

[28] Y. Bajpai, B. Chopra, P. Biyani, C. Aslan, D. Coleman, S. Gulwani, C. Parnin, A. Radhakrishna, and G. Soares, "Let's Fix this Together: Conversational Debugging with GitHub Copilot," in *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Los Alamitos, CA, USA: IEEE Computer Society, Sep. 2024, pp. 1–12. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/VL/HCC60511.2024.00011

[29] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Agentless: Demystifying LLM-based Software Engineering Agents," in *Proceedings of the 2025 ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE '25)*, 2025.

[30] I. Bouzenia, P. T. Devanbu, and M. Pradel, "RepairAgent: An Autonomous, LLM-Based Agent for Program Repair," in *ICSE '25: Proceedings of the 47th International Conference on Software Engineering*, 2025.

[31] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, "AutoCodeRover: Autonomous Program Improvement," in *ISSTA '24: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 13–24.

[32] Y. Wang, Y. Wang, D. Guo, J. Chen, R. Zhang, Y. Ma, and Z. Zheng, "RLCoder: Reinforcement Learning for Repository-Level Code Completion," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 165–177. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00014

[33] J. M. Corbin and A. L. Strauss, *Basics of qualitative research: techniques and procedures for developing grounded theory*. SAGE Publications, Inc., 2008.

[34] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[35] J. Vincent, "This AI-powered autocompletion software is Gmail's Smart Compose for coders," https://www.theverge.com/2019/7/24/20708542/coding-autocompleter-deep-tabnine-ai-deep-learning-smart-compose, Jul. 2019.

[36] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[37] OpenAI, "OpenAI Codex," https://openai.com/index/openai-codex/, Aug. 2021.

[38] T. Ray, "Microsoft has over a million paying Github Copilot users: CEO Nadella," https://www.zdnet.com/article/microsoft-has-over-a-million-paying-github-copilot-users-ceo-nadella/, Oct. 2023.

[39] F. Lardinois, "Amazon launches CodeWhisperer, a GitHub Copilot-like AI pair programming tool," Jun. 2022.

[40] A. Masad, "Replit — Ghostwriter AI & Complete Code Beta," https://blog.replit.com/ai, Sep. 2022.

[41] OpenAI, "Introducing ChatGPT," https://openai.com/index/chatgpt/, Nov. 2022.

[42] N. Grant and C. Metz, "Google Releases Bard, Its Competitor in the Race to Create A.I. Chatbots," *The New York Times*, Mar. 2023.

[43] S. Goldman, "OpenAI rival Anthropic introduces Claude, an AI assistant to take on ChatGPT," Mar. 2023.

[44] K. Leswing, "Mark Zuckerberg announces Meta's new large language model as A.I. race heats up," https://www.cnbc.com/2023/02/24/mark-zuckerberg-announces-meta-llama-large-language-model.html, Feb. 2023.

[45] D. Coldewey, "Mistral AI makes its first large language model free for everyone," Sep. 2023.

[46] T. Qu, "Alibaba opens Tongyi Qianwen model to public as new CEO embraces AI," https://www.scmp.com/tech/big-tech/article/3234385/alibaba-opens-ai-model-tongyi-qianwen-public-competition-baidu-tencent-and-other-chinese-big-tech, Sep. 2023.

[47] DeepSeek-AI, X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, H. Gao, K. Gao, W. Gao, R. Ge, K. Guan, D. Guo, J. Guo, G. Hao, Z. Hao, Y. He, W. Hu, P. Huang, E. Li, G. Li, J. Li, Y. Li, Y. K. Li, W. Liang, F. Lin, A. X. Liu, B. Liu, W. Liu, X. Liu, X. Liu, Y. Liu, H. Lu, S. Lu, F. Luo, S. Ma, X. Nie, T. Pei, Y. Piao, J. Qiu, H. Qu, T. Ren, Z. Ren, C. Ruan, Z. Sha, Z. Shao, J. Song, X. Su, J. Sun, Y. Sun, M. Tang, B. Wang, P. Wang, S. Wang, Y. Wang, Y. Wang, T. Wu, Y. Wu, X. Xie, Z. Xie, Z. Xie, Y. Xiong, H. Xu, R. X. Xu, Y. Xu, D. Yang, Y. You, S. Yu, X. Yu, B. Zhang, H. Zhang, L. Zhang, L. Zhang, M. Zhang, M. Zhang, W. Zhang, Y. Zhang, C. Zhao, Y. Zhao, S. Zhou, S. Zhou, Q. Zhu, and Y. Zou, "DeepSeek LLM: Scaling Open-Source Language Models with Longtermism," Jan. 2024.

[48] Databricks, "Introducing DBRX: A New State-of-the-Art Open LLM," https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, Wed, 03/27/2024 - 11:34.

[49] S. Rizwan, "My submission to Anthropic's Build with Claude June 2024 hackathon: Claude Dev [later renamed to Cline], an autonomous software engineer right in your IDE. Open Source and Available on VSCode Marketplace Now! Reddit Post in r/ClaudeAI," Jul. 2024.

[50] B. Liu, "Open sourcing Cody | Sourcegraph Blog," https://sourcegraph.com/blog/open-sourcing-cody, Mar. 2023.

[51] D. Jemerov, "AI Assistant in JetBrains IDEs | The IntelliJ IDEA Blog," https://blog.jetbrains.com/idea/2023/06/ai-assistant-in-jetbrains-ides/, Jun. 2023.

[52] T. Dunn, "Show HN: Continue – Open-source coding autopilot | Hacker News," https://news.ycombinator.com/item?id=36882146, Jul. 2023.

[53] E. Ruiz, "Posit AI Blog: Chat with AI in RStudio," https://blogs.rstudio.com/tensorflow/posts/2024-04-04-chat-with-llms-using-chattr/, Apr. 2024.

[54] M. Zhang, "Introducing First Stable Release: V0.0.1 | Tabby AI coding assistant," https://www.tabbyml.com/blog/first-stable-release, Aug. 2023.

[55] R. Salva, "Get coding help from Gemini Code Assist — now for free," https://blog.google/technology/developers/gemini-code-assist-free/, Feb. 2025.

[56] F. Lardinois, "Warp brings an AI bot to its terminal," Mar. 2023.

[57] Aider, "Aider Release history," https://aider.chat/HISTORY.html, Jun. 2023.

[58] Anthropic, "Claude 3.7 Sonnet and Claude Code," https://www.anthropic.com/news/claude-3-7-sonnet, Feb. 2025.

[59] OpenAI, "OpenAI Codex CLI," Apr. 2025.

[60] K. Wiggers, "Anysphere raises $8M from OpenAI to build an AI-powered IDE," Oct. 2023.

[61] S. Wu, "Introducing Devin, the first AI software engineer," https://cognition.ai/blog/introducing-devin, Mar. 2024.

[62] Windsurf, "Windsurf Launch," https://windsurf.com/blog/windsurf-launch, Nov. 2024.

[63] C. Perry, "AI-powered coding, free of charge with Colab," https://blog.google/technology/developers/google-colab-ai-coding-features/, May 2023.

[64] G. Szalai, "Introducing Deepnote AI," https://deepnote.com/blog/introducing-deepnote-ai, Jun. 2023.

[65] H. Woodburn, "Give your data work a boost with AI Assist," https://observablehq.com/blog/ai-assist, Aug. 2023.

[66] A. Agrawal, "Marimo Notebook: Newsletter 13 (Generate notebooks with LLMs — now from the command line)," https://marimo.io/blog/newsletter-13, Apr. 2025.

[67] Anthropic, "Introducing Claude 3.5 Sonnet," https://www.anthropic.com/news/claude-3-5-sonnet, Jun. 2024.

[68] S. Willison, "ChatGPT Canvas can make API requests now, but it's complicated," https://simonwillison.net/2024/Dec/10/chatgpt-canvas/, Dec. 2024.

[69] D. Citron, "New ways to collaborate and get creative with Gemini," https://blog.google/products/gemini/gemini-collaboration-features/, Mar. 2025.

[70] Lovable, "GPT Engineer is now Lovable," https://lovable.dev, Jun. 2023.

[71] F. Lardinois, "Google launches Project IDX, a new AI-enabled browser-based development environment," Aug. 2023.

[72] J. Palmer, "Announcing v0: Generative UI," https://vercel.com/blog/announcing-v0-generative-ui, Oct. 2023.

[73] S. Krouse, "Introducing Townie AI," https://blog.val.town/blog/townie/, Sep. 2024.

[74] W. A. Ghazaleh, "StackBlitz (Bolt.new): 0 to $20M ARR in 2 months. The fastest growing startup ever?" https://www.todayin-ai.com/p/stackblitz, Jan. 2025.

[75] X. Qu, "Introducing HeyBoss AI," https://www.linkedin.com/posts/xiaoyinqu_today-im-thrilled-to-introduce-heybossxyzthe-activity-7287893469660946435-XCmP/, Feb. 2025.

[76] OpenAI, "Introducing Codex," https://openai.com/index/introducing-codex/, May 2025.

[77] T. Mullen and R. J. Salva, "Gemini CLI: Your open-source AI agent," https://blog.google/technology/developers/introducing-gemini-cli-open-source-ai-agent/, Jun. 2025.

[78] Q. Team, "Qwen3-Coder: Agentic Coding in the World," https://qwenlm.github.io/blog/qwen3-coder/, Jul. 2025.

[79] T. Ball, "Amp is now available. Here's how I use it." https://ampcode.com/how-i-use-amp, May 2025.

[80] R. Kovács, "OpenCode: Open Source Claude Code Alternative is Here:," https://apidog.com/blog/opencode/, Jun. 2025.

[81] C. Gouze, "Launch YC: Nao Labs - Cursor for data," https://www.ycombinator.com/launches/NW1-nao-labs-cursor-for-data, May 2025.

[82] B. Couriol, "The Void IDE, Open-Source Alternative to Cursor, Released in Beta," https://www.infoq.com/news/2025/06/void-ide-beta-release/, Jun. 2025.

[83] N. Swaminathan and D. Singh, "Introducing Kiro - Kiro," https://kiro.dev/blog/introducing-kiro/, Jul. 2025.

[84] K. Korevec, "Build with Jules, your asynchronous coding agent," https://blog.google/technology/google-labs/jules/, May 2025.

[85] M. Grinberg and E. Reyes, "Factory: The Command Center for Software Development," https://www.factory.ai, Feb. 2025.

[86] A. Abati, "Introducing codename goose | codename goose," https://block.github.io/goose/blog/2025/01/28/introducing-codename-goose, Jan. 2025.

[87] Anthropic, "Create AI-Powered Apps with Claude Artifacts - No Coding Required," https://www.anthropic.com/news/build-artifacts, Jun. 2025.

[88] Github, "GitHub Copilot coding agent in public preview - GitHub Changelog," May 2025.

[89] P. Ng, R. Chouhan, and T. Duncalf, "Introducing Figma Make: A New Way to Test, Edit, and Prompt Designs | Figma Blog," https://www.figma.com/blog/introducing-figma-make/, May 2025.

[90] V. Nallatamby, A. Benard, and S. El-Husseini, "From idea to app: Introducing Stitch, a new way to design UIs- Google Developers Blog," https://developers.googleblog.com/en/stitch-a-new-way-to-design-uis/, May 2025.

[91] M. Amjad, "Smtg-ai/claude-squad: Manage multiple AI terminal agents like Claude Code, Aider, Codex, OpenCode, and Amp." https://github.com/smtg-ai/claude-squad, Apr. 2025.

[92] Conductor, "Conductor," https://conductor.build/, Jul. 2025.

[93] E. Jiang, E. Toh, A. Molina, K. Olson, C. Kayacik, A. Donsbach, C. J. Cai, and M. Terry, "Discovering the syntax and strategies of natural language programming with generative language models," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491102.3501870

[94] K. Ferdowsi, J. Williams, I. Drosos, A. D. Gordon, C. Negreanu, N. Polikarpova, and A. Sarkar, "ColDeco: An End User Spreadsheet Inspection Tool for AI-Generated Code," in *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2023, pp. 82–91.

[95] M. X. Liu, A. Sarkar, C. Negreanu, B. Zorn, J. Williams, N. Toronto, and A. D. Gordon, ""what it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3544548.3580817

[96] R. Cheng, T. Barik, A. Leung, F. Hohman, and J. Nichols, "BISCUIT: Scaffolding LLM-Generated Code with Ephemeral UIs in Computational Notebooks," in *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2024, pp. 13–23.

[97] P. Vaithilingam, E. L. Glassman, J. P. Inala, and C. Wang, "DynaVis: Dynamically Synthesized UI Widgets for Visualization Editing," in *CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–17.

[98] M. Kazemitabaar, J. Williams, I. Drosos, T. Grossman, A. Z. Henley, C. Negreanu, and A. Sarkar, "Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition," in *UIST '24: Proceedings of the 37th ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–12.

[99] L. Xie, C. Zheng, H. Xia, H. Qu, and Z.-T. Chen, "WaitGPT: Monitoring and Steering Conversational LLM Agent in Data Analysis with On-the-Fly Code Visualization," in *UIST '24: Proceedings of the 37th ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 119:1–119:11.

[100] L. Yan, A. Hwang, Z. Wu, and A. Head, "Ivie: Lightweight Anchored Explanations of Just-Generated Code," in *CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–15.

[101] K. Ferdowsi, R. Huang, M. B. James, N. Polikarpova, and S. Lerner, "Validating AI-Generated Code with Live Programming," in *CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–14.

[102] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, and Q. Wang, "ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification," in *Proc. ACM Softw. Eng. 1, FSE (Proceedings of the 32nd ACM SIGSOFT International Symposium on Foundations of Software Engineering)*, 2024, pp. Article 103, 23 pages.

[103] R. Yen, J. S. Zhu, S. Suh, H. Xia, and J. Zhao, "CoLadder: Manipulating Code Generation via Multi-Level Blocks," in *UIST '24: Proceedings of the 37th ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 11:1–11:20.

[104] C. Zhu-Tian, Z. Xiong, X. Yao, and E. Glassman, "Sketch Then Generate: Providing Incremental User Feedback and Guiding LLM Code Generation through Language-Oriented Code Sketches," *arXiv preprint arXiv:2405.03998*, 2024.

[105] L. Chen, S. Xiao, Y. Chen, R. Wu, Y. Song, and L. Sun, "ChatScratch: An AI-Augmented System Toward Autonomous Visual Programming Learning for Children Aged 6–12," in *CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–11.

[106] J. Chen, X. Lu, M. Rejtig, D. Du, R. Bagley, M. S. Horn, and U. Wilensky, "Learning Agent-based Modeling with LLM Companions: Experiences of Novices and Experts Using ChatGPT & NetLogo Chat," in *CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–17.

[107] V. Murali, C. Maddila, I. Ahmad, M. Bolin, D. Cheng, N. Ghorbani, R. Fernandez, N. Nagappan, and P. C. Rigby, "AI-Assisted Code Authoring at Scale: Fine-Tuning, Deploying, and Mixed Methods Evaluation," in *Proc. ACM Softw. Eng. 1, FSE (Proceedings of the 32nd ACM SIGSOFT International Symposium on Foundations of Software Engineering)*, 2024, pp. Article 48, 20 pages.

[108] G. Pinto, C. R. de Souza, J. B. Neto, A. Monteiro, and T. Gotto, "Lessons from Building StackSpot AI: A Contextualized AI Coding Assistant," in *ICSE-SEIP '24: Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 408–417.

[109] P. Mowar, Y.-H. Peng, J. Wu, A. Steinfeld, and J. P. Bigham, "CodeA11y: Making AI Coding Assistants Useful for Accessible Web Development," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[110] K. Pu, D. Lazaro, I. Arawjo, H. Xia, Z. Xiao, T. Grossman, and Y. Chen, "Assistance or Disruption? Exploring and Evaluating the Design and Trade-offs of Proactive AI Programming Support," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[111] V. Chen, A. Zhu, S. Zhao, H. Mozannar, D. Sontag, and A. Talwalkar, "Need Help? Designing Proactive AI Assistants for Programming," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[112] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. R. Lyu, "Divide-and-Conquer: Generating UI Code from Screenshots," in *Proceedings of the 2025 ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE '25)*, 2025.

[113] R. Yen, J. Zhao, and D. Vogel, "Code Shaping: Iterative Code Editing with Free-form AI-Interpreted Sketching," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[114] L. F. Gomes, V. J. Hellendoorn, J. Aldrich, and R. Abreu, "An Exploratory Study of ML Sketches and Visual Code Assistants," in *ICSE '25: Proceedings of the 47th International Conference on Software Engineering*, 2025.

[115] W.-H. Chen, W. Tong, A. Case, and T. Zhang, "Dango: A Mixed-Initiative Data Wrangling System using Large Language Model," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[116] S. Ma, J. Wang, Y. Zhang, X. Ma, and A. Y. Wang, "DBox: Scaffolding Algorithmic Programming Learning through Learner-LLM Co-Decomposition," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[117] M. Kazemitabaar, O. Huang, S. Suh, A. Z. Henley, and T. Grossman, "Exploring the Design Space of Cognitive Engagement Techniques

with AI-Generated Code for Enhanced Learning," in *Proceedings of the 2025 ACM Conference on Intelligent User Interfaces (IUI '25)*, 2025.

[118] J. D. Zamfirescu-Pereira, E. Jun, M. Terry, Q. Yang, and B. Hartmann, "Beyond Code Generation: LLM-Supported Exploration of the Program Design Space," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[119] Q. Guo, X. Xie, S. Liu, M. Hu, X. Li, and L. Bu, "Intention is All You Need: Refining Your Code from Your Intention," in *ICSE '25: Proceedings of the 47th International Conference on Software Engineering*, 2025.

[120] Y. Di and T. Zhang, "Enhancing Code Generation via Bidirectional Comment-Level Mutual Grounding," in *ICSE '25: Proceedings of the 47th International Conference on Software Engineering*, 2025.

[121] X. Jiang, Y. Dong, Y. Tao, H. Liu, Z. Jin, and G. Li, "ROCODE: Integrating Backtracking Mechanism and Program Analysis in Large Language Models for Code Generation," in *ICSE '25: Proceedings of the 47th International Conference on Software Engineering*, 2025.

[122] J. Ma, K. Sreedhar, V. Liu, P. A. Perez, S. Wang, R. Sahni, and L. B. Chilton, "DynEx: Dynamic Code Synthesis with Structured Design Exploration for Accelerated Exploratory Programming," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[123] S. Earle, S. Parajuli, and A. Banburski-Fahey, "DreamGarden: A Designer Assistant for Growing Games from a Single Prompt," in *CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025.

[124] N. Friedman, "Introducing GitHub Copilot: Your AI pair programmer," Jun. 2021.

[125] P. Kravtsov, "A New Tab Model | Cursor - The AI Code Editor," https://www.cursor.com/blog/tab-update, Jan. 2025.

[126] V. Kuka, "Codeium Releases Windsurf Wave 5 with Enhanced Tab Functionality," https://learnprompting.org/blog/windsurf-wave-5, Apr. 2025.

[127] B. Murtaugh and B. Holland, "Copilot Next Edit Suggestions (preview)," https://code.visualstudio.com/blogs/2025/02/12/next-edit-suggestions, Feb. 2025.

[128] OpenAI, "GPT-4o | OpenAI," https://openai.com/index/hello-gpt-4o/, May 2024.

[129] L. Kilpatrick, "Gemini 2.5 Pro Preview: Even better coding performance- Google Developers Blog," https://developers.googleblog.com/en/gemini-2-5-pro-io-improved-coding-performance/, May 2025.

[130] Windsurf, "Wave 8: UX Features + Plugins Update," https://windsurf.com/blog/windsurf-wave-8-ux-features-and-plugins, May 2025.

[131] S. Kedar, "Switchable models come to Tabnine Chat," https://www.tabnine.com/blog/introducing-switchable-models-for-tabnine-chat/, Apr. 2024.

[132] Cursor, "Rules | Cursor - The AI Code Editor," https://www.cursor.com/changelog/improved-copilot-ux-new-gpt-4-model, Apr. 2024.

[133] OpenAI, "Memory and new controls for ChatGPT," https://openai.com/index/memory-and-new-controls-for-chatgpt/, Mar. 2024.

[134] A. Osmani, "Why I use Cline for AI Engineering," Jan. 2025.

[135] Anthropic, "Introducing the Model Context Protocol," https://www.anthropic.com/news/model-context-protocol, Nov. 2024.

[136] R. J. Salva, "Introducing code referencing for GitHub Copilot," Aug. 2023.

[137] J. Prather, J. Leinonen, N. Kiesler, J. Gorson Benario, S. Lau, S. MacNeil, N. Norouzi, S. Opel, V. Pettit, L. Porter, B. N. Reeves, J. Savelka, D. H. Smith, S. Strickroth, and D. Zingaro, "Beyond the hype: A comprehensive review of current trends in generative ai research, teaching practices, and tools," in *2024 Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2024. New York, NY, USA: Association for Computing Machinery, 2025, p. 300–338. [Online]. Available: https://doi.org/10.1145/3689187.3709614

[138] L. Tankelevitch, V. Kewenig, A. Simkute, A. E. Scott, A. Sarkar, A. Sellen, and S. Rintel, "The Metacognitive Demands and Opportunities of Generative AI," in *CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–24.

[139] C. P. Snow, "Two cultures," *Science*, vol. 130, no. 3373, pp. 419–419, 1959.