International Journal of Education Studies and Social Science Research

Volume: 01, Issue: 02 March - April 2026

WHILE ARTIFICIAL INTELLIGENCE WRITES THE CODE, WHAT DO WE DO? THE NEW ROLE OF THE DEVELOPER IN THE ERA OF AUTONOMOUS CODING AGENTS

VERÓNICA CHÁVEZ TORRES[1] AXEL VELASCO CHÁVEZ[2] & JOSÉ GERARDO DE LA VEGA MENESES[3]

[1] School of Business — UPAEP
[2] Software Engineering — UPAEP
[3] School of Business — UPAEP
UPAEP, Puebla, Mexico — 2026

Received: 26/03/2026 Accepted: 31/03/2026 Published: 03/04/2026

ABSTRACT

The emergence of autonomous AI coding agents profoundly reconfigures the role of the software engineer. This paper examines the evolution of AI coding assistants, from syntactic autocomplete to extended-reasoning agents such as Claude Code, and proposes the AIDA Methodology (Mey's AI-Driven Development Architecture): five structured phases for AI-assisted software development. Drawing on empirical evidence of skill-differentiated productivity gains (Dell'Acqua et al., 2023) and performance benchmarks including SWE-bench Verified (72.5 %) and Terminal-bench (43.2 %), we argue that the critical competency of the modern developer is not writing code, but designing architectures, contextualizing problems, and interpreting AI-generated solutions. We also examine the emerging phenomenon of native multimodal processing, curricular implications for university education, the competitive positioning of the professional developer, and the normative gap arising from the absence of formal standards for AI-assisted development.

Keywords: artificial intelligence, software engineering, Claude Code, autonomous agents, AIDA methodology, productivity, university education, multimodal processing, frontier models.

1.0 INTRODUCTION

Software development has undergone profound transformations since the first compilers. However, none of those transformations had so directly questioned the very essence of the programmer's work as the current generation of Artificial Intelligence (AI)-based tools. For the first time in the history of the discipline, a computing system can receive a natural language description and autonomously produce functional code, unit tests, documentation, and version control commits, without a human writing a single line (Anthropic, 2025; OpenAI, 2025).

This reality raises a question that is both uncomfortable and necessary: while Artificial Intelligence writes the code, what do we do? The superficial—and incorrect—answer would be to claim that the developer becomes obsolete. The honest answer requires an analysis that distinguishes between the task of writing code and that of solving problems. This confusion has led many professionals to adopt a defensive stance toward AI tools, while others, more strategically minded, have turned them into an unprecedented competitive advantage (Eloundou et al., 2023; World Economic Forum, 2025).

The empirical context is compelling. According to SQ Magazine (2025), ChatGPT concentrates more than 730 million users on its free tier, compared to just 13.5 million on paid plans. The frontier AI access funnel is even more revealing: of the entire US population, only 34 % have ever used ChatGPT, 15 % use it weekly, 4 % pay for a subscription, and barely 1 % access frontier models (Backlinko, 2025; First Page Sage, 2026). This gap is not technological: it is one of knowledge and strategic adoption.

At the same time, productivity indicators are unequivocal. New websites, iOS applications, and activity in GitHub repositories show year-over-year growth exceeding 20-40 % starting in 2024-2025 (Financial Times, 2026; GitHub, 2025). This paper has four objectives: (1) trace the technological trajectory of code assistants up to autonomous agents; (2) propose the AIDA Methodology as a structured framework; (3) examine native multimodal processing as a new dimension of frontier capabilities; and (4) reflect on the implications for university education in software engineering.

2.0 THEORETICAL FRAMEWORK AND BACKGROUND

2.1 The Iceberg Illusion: Beyond the Chat Interface

When Artificial Intelligence is mentioned in educational or professional contexts, the instinctive reference tends to be ChatGPT, Gemini, or Microsoft Copilot in their conversational mode. These tools represent, metaphorically, the visible tip of the iceberg. Their chat interface normalizes AI as an oracle of answers, hiding beneath it an entire architecture of autonomous capabilities: system integration, chained task execution, complex reasoning, and real agency (Bommasani et al., 2021; Wiles, 2023).

The AI iceberg can be conceptualized across four levels of increasing depth:

- Surface level: conversational chat, text generation, translation, and grammar correction. This is what the 34 % who have ever used ChatGPT experience.
- Intermediate level: code generation with autocomplete, data analysis, integrated web search, and document processing. This is where the 15 % of weekly users operate.

- Deep level: real code execution in sandboxes, external API integration, autonomous web-browsing agents, and automatic test generation. The territory of the 4 % who pay for a premium subscription.
- Frontier level: chained autonomous agents, multi-agent orchestration, Computer Use—full operating system control—and development pipelines without human intervention. The 1 % who access frontier models.

The developer who believes they already use AI because they ask ChatGPT how to write an SQL query is functionally equivalent to someone who thinks they already use the internet because they read emails. Superficial use of a deep technology does not constitute strategic adoption (Nielsen, 2024).

2.2 The Evolution of Code Assistants: Four Generations

The history of the code assistant can be divided into four clearly differentiated generations (Chen et al., 2021; Ziegler et al., 2022):

- Generation 1: Classic autocomplete (2000-2020). Tools like IntelliSense completed method names based on the immediate context. The assistance was syntactic, not semantic.
- Generation 2: Generative autocomplete (2021-2022). GitHub Copilot introduced the generation of complete code blocks from natural language comments, based on OpenAI's Codex model (Chen et al., 2021).
- Generation 3: Intelligent IDEs and vibe-coding (2023-2024). Platforms such as Cursor, Replit, and Bolt brought AI to the center of the development environment, even for users without programming knowledge. The term vibe-coding was introduced by Karpathy (2025) to describe this paradigm.
- Generation 4: Autonomous agents with extended reasoning (2025-present). Tools such as Claude Code execute tasks autonomously by chaining actions, managing context, and making architectural decisions (Anthropic, 2025).

Vibe-coding democratized software construction, but it revealed that the true bottleneck of development was never writing code, but understanding the problem, designing the solution, and ensuring its quality. Those who delegate these three tasks to AI without understanding them obtain code that works in demos but fails in production (Karpathy, 2025; Finnie-Ansley et al., 2022).

2.3 Reasoning Tokens: The Generational Leap

A token is the minimum processing unit of a language model: approximately 100 tokens equal 75 words in English (OpenAI, 2023). Reasoning models introduce internal thinking tokens: before responding, the model dedicates tokens to a reflection process that is not visible in the

output. This paradigm was introduced by OpenAI with its o1 model in 2024, demonstrating that dedicating computational time to prior reasoning produced significant gains on complex tasks (OpenAI, 2024; Snell et al., 2024).

Anthropic implemented this concept under the Adaptive Thinking philosophy in Claude. Unlike o1, which always reasons in depth, Claude automatically calibrates its level of thinking based on the intrinsic complexity of the query (Anthropic, 2025). This dynamic calibration represents a significant practical advantage in development environments where tasks vary enormously in complexity.

2.4 Native Multimodal Processing: The New Frontier Dimension

One of the most significant transformations in current frontier models is the ability to simultaneously process multiple information modalities. This capability radically redefines what a developer can delegate to AI (Driess et al., 2023; Google DeepMind, 2023).

Modality	Input	Output
Text	Natural language, code, structured prompts	Text, code, documentation
Modality	Input	Output
Image	Screenshots, diagrams, visual errors, OCR	Image and diagram generation
Audio	Real-time transcription, tone analysis	Text-to-Speech, voice cloning
Video	Recording analysis, video-to-code	Video generation from text
Documents	PDFs, spreadsheets, scanned forms	Structured extraction, synthesis
Code	Full repositories, semantic bug detection	Refactoring, automatic tests
Computer Use	Complete graphical interface of the OS	Clicks, typing, autonomous navigation

Table 1. Native modalities of frontier models (2025-2026). Own elaboration.

Computer Use, introduced by Anthropic in October 2024, represents the most radical leap: the model operates at the full operating system level, taking control of the mouse, the keyboard, and any installed application (Anthropic, 2024). The practical consequence is that a frontier model can simultaneously receive the source code, a screenshot of a production error, a recording of a user reporting the bug, and the original requirements PDF—and reason over all those sources in a single session (Google DeepMind, 2023).

2.5 The Unknown Model Problem

An underestimated consequence of the mass adoption of AI is that most users do not know which model they are using. The interfaces of the most popular platforms are deliberately designed to hide this information: the user sees "ChatGPT" or "Copilot" without knowing whether they are interacting with GPT-4o mini (free model, limited capabilities) or GPT-o3 Pro (frontier model, advanced capabilities).

This opacity has three direct professional consequences: (1) the developer cannot correctly diagnose whether a failure stems from their prompt or from model limitations; (2) they cannot build the judgment to choose the right tool for the task; and (3) they may fall into the −19 % scenario described by Dell'Acqua et al. (2023) without understanding why. Not knowing which model you are using is the clearest signal that you are at the base of the adoption funnel, not at the frontier.

2.6 AI Improving AI

One of the most significant phenomena of the current era is that Artificial Intelligence is being used to improve AI itself: generation of synthetic training data, use of agents to evaluate and refine responses from other models through RLHF, and Constitutional AI—where the models themselves correct each other by applying predefined ethical principles (Bai et al., 2022; Christiano et al., 2017). METR (2025) documented that the length of software engineering tasks that leading models can complete with at least 50 % success was doubling every seven months. The developer who does not keep their knowledge of available tools up to date is effectively working with technology from the past.

2.7 Empirical Evidence: Skill-Differentiated Productivity

Dell'Acqua et al. (2023) measured productivity gains among consultants with different skill levels when using GPT-4. The results revealed four patterns:

- Low skill: +35 %. AI compensates for the professional's prior limitations.
- Average skill: +14 %. AI helps at the margins; the professional was already competent.
- High skill (within the frontier): +42.5 %. The best performers know exactly what to ask the model and can validate its responses.
- Outside the model's frontier: −19 %. When the task exceeds the model's real capabilities, actively using it damages the final result.

The conclusion is clear: the greatest risk for developers is not being replaced by AI, but using it poorly. This is a direct consequence of not knowing the real capabilities of the model being used (Eloundou et al., 2023).

3.0 CLAUDE CODE: A FRONTIER TOOL FOR DEVELOPERS

3.1 Differentiators from Vibe-Coding

Claude Code is not a vibe-coding tool. Tools like Bolt, Lovable, or Replit Agent are designed for users without programming knowledge. Claude Code, on the other hand, is designed specifically for developers: it operates from the terminal, integrates with the project's real file system, executes commands, manages dependencies, reads existing code, and makes architectural decisions (Anthropic, 2025; Willison, 2025). The difference is not one of interface: it is philosophical. Claude Code assumes there is an engineer on the other side who understands what is happening and can validate it.

3.2 How Claude Code Works: The Operational Flow

The operational flow of Claude Code can be described in six iterative steps:

- The developer writes a natural language instruction in the terminal.
- Claude reads the full codebase or the relevant files based on the available context.
- The model plans the necessary changes and communicates them before executing.
- Claude writes the code, executes commands, and internally validates the result.
- The developer reviews the result and approves, corrects, or rejects it.
- The cycle repeats until the defined atomic task is complete.

This flow contrasts radically with that of vibe-coding, where the user vaguely describes what they want and the model generates a complete application without any possibility of structured intervention at each step (Anthropic, 2025).

3.3 Performance Benchmarks

Model	SWE-bench Verified (%)	Terminal-bench (%)	Year
Claude Opus 4	72.5 %	43.2 %	2025
Model	SWE-bench Verified (%)	Terminal-bench (%)	Year
OpenAI o3 Pro	71.7 %	N/A	2025
Gemini 2.5 Pro	63.8 %	N/A	2025
DeepSeek R1	49.2 %	N/A	2025
GPT-4o	33.2 %	N/A	2024

Table 2. SWE-bench Verified and Terminal-bench comparison (Anthropic, 2025; Jimenez et al., 2023).

Claude Opus 4 leads with 72.5 % on SWE-bench and 43.2 % on Terminal-bench, being the first model to surpass the 40 % threshold on the latter benchmark (Anthropic, 2025). These numbers matter because they measure tasks that a real developer would perform in their daily

work: fixing reported bugs, navigating unfamiliar codebases, and validating results in the terminal (Jimenez et al., 2023).

3.4 Ralph Mode: Total Chained Autonomy

Claude Code can operate at different levels of autonomy. At the far end of the spectrum lies what the community colloquially calls Ralph mode: granting Claude Code total autonomy to execute chained tasks without interruption. The model reads files, writes code, runs tests, installs dependencies, and makes commits in a single session without requiring approval at each step. This mode is extremely powerful for well-defined tasks with a clear architecture, and potentially destructive for ambiguous tasks with insufficient context (Anthropic, 2025).

4.0 PROMPT ENGINEERING AND TOKEN OPTIMIZATION

4.1 From Vague Prompts to Architectural Prompts

The difference between a vague prompt and an architectural one is not in length, but in the specificity of context (White et al., 2023; Sahoo et al., 2024). The five elements of an architectural prompt are: exact location, observed behavior, technology stack, contract constraints, and expected result.

Vague prompt — Avoid: "Fix the error in the authentication file." This prompt forces the model to infer which error, in which exact file, with which stack, and under which constraints. The result will be a generic solution that likely does not respect the project's architecture.

Architectural prompt — Prefer: "In src/auth/jwt.service.ts, the validateToken() function throws TypeError: Cannot read property 'exp' when the token arrives expired. The project uses NestJS with Passport-JWT. Do not modify the public interface. Add handling for the case where payload is null and return false in that scenario."

The architecture of the prompt reflects the architecture of the developer's thinking. An engineer who cannot describe a problem precisely will also be unable to validate the solution that AI generates.

4.2 Token Optimization in Claude Code Sessions

In Claude Code, tokens have a dual cost: economic (price per million in the API) and functional (the available context is finite). The main optimization strategies are:

- clear at the end of each feature: clears the full history and restarts the context.
- compact in long sessions: compresses the history while retaining only the relevant semantic summary, reducing token usage by up to 84 %.

- Atomic tasks: limiting each task to a maximum of three affected files reduces context and improves precision.
- CLAUDE.md as persistent context: documenting the architecture avoids repeating instructions every session.

5.0 AIDA METHODOLOGY: MEY'S AI-DRIVEN DEVELOPMENT ARCHITECTURE

Despite the proliferation of AI tools for development, no formal standard—ISO, IEEE, or otherwise—exists that establishes how AI-assisted software should be developed. Existing methodologies were designed under the assumption that code is written exclusively by humans (Sommerville, 2016; Pressman & Maxim, 2020). The AIDA Methodology emerges from practical experience in real-world projects with Claude Code and seeks to fill this gap with a structured, reproducible, quality-oriented framework.

Phase	Name	Principle	Key Actions
F1	ARCHITECTURE	Think before coding	Define problem and scope. Design architecture (may be AI-assisted; final decision belongs to the developer). Choose stack with technical justification. Define infrastructure.
F2	CONTEXT	Give AI memory	Create CLAUDE.md. Document architecture, stack, conventions, and folder structure. Allowed and prohibited libraries. Business rules (no credentials).
F3	EXECUTION	Delegate with precision	Create TODO.md with atomic backlog. Max. 3 files per session. Prioritize: DB → models → logic → UI. One atomic task per Claude Code session.
F4	VALIDATION	You are the architect	Review and approve every change before committing. Validate against F1 architecture. Run tests after each task. Manage context: /clear and /compact.
F5	REUSE	Build for the future	Document successful patterns. Create reusable CLAUDE.md templates by project type. Generate the final CLAUDE.md as a stack template. Update technical decisions.

Table 3. AIDA Methodology: phases, principles, and key actions. Own elaboration.

Phase 2 is what makes the entire process coherent: without a well-documented CLAUDE.md, every session starts from scratch. Phases 3 and 4 implement a delegation-validation cycle that ensures AI generates code within the project's parameters. Phase 5 converts individual experience into reusable organizational capital.

6.0 IMPLICATIONS FOR SOFTWARE ENGINEERING EDUCATION

6.1 The University Curriculum Under Scrutiny

The honest answer to how many courses remain relevant is: all of them. But their purpose changes radically. The Databases course is now more important than ever, because the engineer who does not understand normalization, indexes, and transactions will lack the capacity to validate whether the code AI generates is correct, efficient, and secure (Becker et al., 2023; Finnie-Ansley et al., 2022).

6.2 The Shift in Course Orientation

- Databases: from learning to write queries to learning to design schemas that AI cannot model incorrectly.
- Web Development: from learning HTML/CSS/JS syntax to architecting scalable applications and evaluating generated code.
- Algorithms and Data Structures: from memorizing implementations to understanding computational complexity in order to detect when AI's solution is suboptimal.
- Requirements Engineering: from documenting for the human team to documenting with enough precision for an AI agent to execute correctly. CLAUDE.md is the new SRS.
- Software Architecture: from being an advanced, specialized course to being a transversal competency required from the first semesters.

6.3 The Renaissance of the Humanities in Engineering

In a world where AI can generate code and technical documentation, the courses that popular imagination considers secondary—ethics, communication, philosophy, organizational psychology—become radically more valuable (Vallor, 2016; World Economic Forum, 2025). AI has no interests, no awareness of the social consequences of its products, and cannot build the trust necessary for a team to function. Learning to be human in an AI-dominated world is not a romantic metaphor: it is a strategic necessity.

7.0 DISCUSSION

7.1 The Normative Gap

The absence of a formal standard for AI-assisted software development is not a minor detail. No existing methodology contemplates scenarios where the "author" of the code cannot be an identifiable individual, or where the versioning of architectural decisions must include the instructions given to the agent (Sommerville, 2016). The AIDA Methodology is a practical proposal as a starting point; standardization bodies face the challenge of formalizing these practices.

7.2 The Developer's Competitive Positioning

The developer who masters frontier AI tools can deliver projects in days that previously took weeks, at prices that conventional teams cannot match. The real risk is not being replaced by AI, but being replaced by another developer who uses it better (Eloundou et al., 2023; Dell'Acqua et al., 2023). The competitive differential manifests across three dimensions: speed, quality, and scalability.

7.3 Limitations and Future Work

The AIDA Methodology emerges from practical experience in individual projects and has not been validated through controlled studies. The cited benchmarks measure capabilities in English and with predominantly English-language repositories. Future work should include: empirical validation of AIDA with real development teams; adaptation for university educational contexts; and exploration of how coordinated multi-agent systems modify the execution and validation phases.

8.0 CONCLUSIONS

The question that titles this paper is not rhetorical: it is the most important question a software engineer can ask themselves at this moment.

While Artificial Intelligence writes the code, we think, design, decide, and humanize. We take charge of the problem before AI translates it into code. We validate that the generated code respects the architecture we defined. We know which model we are using, what capabilities it has, and which tasks are within and outside its frontier. We leverage native multimodal processing to enrich the context we give the agent.

The AIDA Methodology represents a concrete first response to how to structure this new role. Its five phases are not steps for delegating development to AI: they are the framework that ensures the developer remains the architect of the system, even if they are no longer the one writing every line.

The transformation we are living through does not eliminate software engineering: it elevates it. And, paradoxically, it gives back something that the pressure of writing code manually had taken away: the time to truly think about the problem.

REFERENCES

1. Anthropic. (2024, October). Introducing computer use, a new Claude AI model, and an upgraded Claude.ai. Anthropic Blog. https://www.anthropic.com/news/3-5-models-and- computer-use.

2. Anthropic. (2025). Introducing Claude 4. Anthropic Blog. https://www.anthropic.com/news/claude-4
3. Backlinko. (2025-2026). ChatGPT usage statistics. Backlinko Research. https://backlinko.com/chatgpt-users
4. Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., ... & Kaplan, J. (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback. arXiv preprint arXiv:2204.05862.
5. Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming is hard—or at least it used to be: Educational opportunities and challenges of AI code generation. Proceedings of the 54th ACM Technical Symposium on Computer Science Education, 500–506.
6. Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., ... & Liang, P. (2021). On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258.
7. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.
8. Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. Advances in Neural Information Processing Systems, 30.
9. Dell'Acqua, F., McFowland III, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., & Lakhani, K. R. (2023). Navigating the jagged technological frontier: Field experimental evidence of the effects of AI on knowledge worker productivity and quality. Harvard Business School Working Paper 24-013. NBER.
10. Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Ichter, B., ... & Florence, P. (2023). PaLM-E: An embodied multimodal language model. Proceedings of the 40th International Conference on Machine Learning, 202, 8469–8488.
11. Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2023). GPTs are GPTs: An early look at the labor market impact potential of large language models. arXiv preprint arXiv:2303.10130.
12. Financial Times / Burn-Murdoch, J. (2026). The past year has seen an explosion in coding productivity. Financial Times Data Visualization. https://www.ft.com/content/coding- productivity-2026
13. Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of OpenAI Codex on introductory programming. Proceedings of the 24th Australasian Computing Education Conference, 10–19.
14. First Page Sage. (2026). ChatGPT user statistics 2026. First Page Sage Research. https://firstpagesage.com/reports/chatgpt-statistics

15. GitHub. (2025). Octoverse 2025: The state of open source. GitHub Inc. https://octoverse.github.com
16. Google DeepMind. (2023). Gemini: A family of highly capable multimodal models. Technical Report. https://arxiv.org/abs/2312.11805
17. Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). SWE-bench: Can language models resolve real-world GitHub issues? arXiv preprint arXiv:2310.06770.
18. Karpathy, A. (2025, February). The rise of vibe coding. X (formerly Twitter). https://x.com/karpathy/status/1886192184808149295
19. METR. (2025). Measuring the ability of AI models to complete long-horizon tasks. METR Research. https://metr.org/blog/2025-07-10-measuring-autonomous-ability
20. Minsky, M. (1986). The society of mind. Simon & Schuster.
21. Mollick, E. (2024). Co-intelligence: Living and working with AI. Portfolio/Penguin.
22. Nielsen, J. (2024). AI productivity paradox: Why most users underutilize AI capabilities. Nielsen Norman Group. https://www.nngroup.com/articles/ai-productivity-paradox
23. OpenAI. (2023). Tokenizer documentation. OpenAI Developer Platform. https://platform.openai.com/tokenizer
24. OpenAI. (2024). Learning to reason with LLMs (o1 System Card). OpenAI. https://openai.com/o1
25. OpenAI. (2025a). GPT-5 system card. OpenAI. https://openai.com/gpt-5
26. OpenAI. (2025b). Codex CLI: Agentic coding in the terminal. OpenAI Developer Blog. https://openai.com/blog/codex-cli
27. Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv preprint arXiv:2302.06590.
28. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill Education.
29. Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927.
30. Snell, C., Lee, J., Xu, K., & Kumar, A. (2024). Scaling LLM test-time compute optimally can be more effective than scaling model parameters. arXiv preprint arXiv:2408.03314.
31. Sommerville, I. (2016). Software engineering (10th ed.). Pearson Education.
32. SQ Magazine. (2025). ChatGPT statistics 2025. SQ Magazine Research. https://www.sqmagazine.com/chatgpt-statistics
33. Vallor, S. (2016). Technology and the virtues: A philosophical guide to a future worth wanting. Oxford University Press.

34. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., ... & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. arXiv preprint arXiv:2302.11382.
35. Wiles, J. (2023, November). What is computer use in AI? Gartner Insights. https://www.gartner.com/en/articles/ai-computer-use
36. Willison, S. (2025, September 29). Claude Sonnet 4.5 is probably the "best coding model in the world". Simon Willison's Weblog. https://simonwillison.net/2025/Sep/29/claude-sonnet-4- 5/
37. World Economic Forum. (2025). The future of jobs report 2025. World Economic Forum. https://www.weforum.org/publications/the-future-of-jobs-report-2025
38. Ziegler, A., Larson, E., Fox, C., & Ciesielski, L. (2022). Productivity assessment of neural code completion. Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, 21–29

