software

MDPI

Article

The Democratization of Computational Thinking: Education, Practice, and Our AI-Augmented Future

Douglas Schmidt [1] and Dan Runfola [2,*]

[1] Department of Computer Science, William & Mary, 540 Landrum Drive, Williamsburg, VA 23188, USA; dcschmidt@wm.edu
[2] Department of Data Science, William & Mary, 540 Landrum Drive, Williamsburg, VA 23188, USA
* Correspondence: dsmillerrunfol@wm.edu; Tel.: +1-(508)-316-9109

Abstract

This paper advances a theoretical argument that generative AI is accelerating the democratization of computational thinking and, in turn, reshaping education, professional practice, and the nature of computing itself. Traditionally, computational thinking has been closely tied to learning to program, thereby limiting who could effectively employ it. The emergence of large language models (LLMs) challenges this linkage by decoupling many forms of computational problem solving from direct programming. In response to this shift, the paper explores the implications for curriculum design and workforce roles through a theoretical and interpretive lens. Drawing on prior literature, historical context, and illustrative examples from recent scholarship and practice, we develop a conceptual account of AI-augmented computing. We argue that LLMs lower barriers to entry by abstracting away much of manual coding and reallocating effort toward problem framing, prompt engineering, oversight, and validation. We further argue that this transition is redistributing computational skills across disciplines, positioning prompt engineering as an emerging engineering practice, and increasing pressure on universities to redesign curricula around AI literacy, fluency, and mastery.

Keywords: computational thinking; generative AI; prompt engineering; AI-augmented education; software engineering practice; computer and data science education; human–AI collaboration

1. Introduction

The coming democratization of computational thinking: Undergraduate computing enrollments in the United States have historically been cyclical, with notable growth periods in the early 1980s, the late 1990s, and again beginning in the mid-2000s [1–3]. Prior scholarship and national analyses suggest that these waves reflected a mix of labor-market demand ("If I don't have a computer science degree, I won't get a job"), technological change (i.e., the dot-com boom), and institutional capacity constraints. Around 2015, another boom led growth that now extends across a broader computing ecosystem, including data science and other computing-related programs (i.e., computational tracks in engineering, social sciences, humanities, and many other domains) [4].

Today, another structural shift is underway: generative AI is transforming the nature of coding itself, creating new pressures and opportunities for Computer Science (CS) and Data Science (DS) programs to rethink their curricula and outcomes. This shift is manifesting in AI assistants and other tools that generate code and analyze information, blurring the

Academic Editor: Daniel Port

Received: 16 February 2026
Revised: 27 April 2026
Accepted: 7 May 2026
Published: 13 May 2026

line between program writing and program using. For example, non-experts can now rapidly generate non-trivial software programs, including web user interfaces and common data modeling and visualization solutions [5]. These advances compel reexamining long-standing assumptions in computing—how we define it, teach it, and use it.

This paper is structured around three guiding questions. First, how are the roles of developers and other knowledge workers changing as AI systems take on a greater share of coding, analysis, and routine implementation? Second, which capabilities are likely to define success in an AI-augmented future, particularly with respect to prompt engineering, oversight, validation, and human judgment? Third, how should computing education and the broader field evolve as generative AI decouples many forms of computational problem solving from direct programming? This paper examines these questions through a theoretical and interpretive analysis of changes in programming practice, workforce roles, curricular design, and the continuing importance of human oversight.

Paper organization: The remainder of the paper proceeds in five steps. Section 2 establishes the conceptual shift from traditional coding to AI-augmented computational problem solving. Section 3 then examines how this shift is changing workforce roles, while Section 4 identifies the capabilities likely to define success in that new environment. Section 5 turns to pedagogy, asking how CS and DS education should evolve in response. Section 6 argues that human judgment remains the cross-cutting condition for success across all three domains, and Section 7 concludes by drawing these strands together.

2. From Computational Thinking to AI-Augmented Problem Solving

This section establishes the paper's conceptual foundation by explaining how generative AI is changing what computational thinking is, who can practice it, and why that shift creates new questions for workforce roles, pedagogy, and professional success. Jeannette Wing [6] popularized the term "computational thinking" to describe this problem-solving approach grounded in logic, abstraction, decomposition, pattern recognition, generalization, and automation. Historically, applying computational thinking required learning to program by translating ideas into code using languages like Python, Java, or C++. Despite the popularity and impact of this approach to problem solving, the requirement to "learn how to code" has long presented a daunting hurdle for many people.

For example, mastery of programming languages meant grappling with myriad incidental complexities, including arcane syntax and semantics, debugging errors, memory management, and other low-level issues unrelated to the core domain problem. These barriers have long constrained who could transform ideas into functioning software applications. They have also limited how broadly computational thinking could be taught, since students were required to master the intricacies of programming alongside the conceptual practices of the field.

A useful analogy may come from the evolution of automobiles. Over time, operating a car has required progressively less knowledge of how the machine itself works. What once demanded substantial mechanical familiarity for many users now often requires only the ability to specify a goal, monitor performance, and intervene when necessary. As vehicles move toward greater autonomy, the key issue is not whether abstraction reduces the technical knowledge required of users but how far that reduction can go.

Computing appears to be moving along a similar trajectory. In earlier eras, effective use often required working close to the machine through hardware configuration, low-level programming, or manual coding in conventional languages. Today, libraries, frameworks, and other IDE tools have dramatically reduced that burden. Generative AI extends this trend by making natural language an increasingly viable programming abstraction: users can state objectives in ordinary language, and AI systems can generate and execute solutions

while absorbing more of the underlying implementation details. On this view, the long-term direction of computing is toward broader capability with less required knowledge of the underlying models, assumptions, and algorithms.

The paradigm of translating a natural-language problem description into a working solution without traditional coding is already emerging in practice [7]. For many tasks, users can now obtain initial results by crafting prompts rather than hand-coding algorithms. Researchers and professionals in non-computing domains who once had to learn programming to analyze data or build tools can increasingly leverage conversational AI assistants to generate draft analyses, visualizations, or simulations. For example, a biologist or sociologist may be able to pose questions to an LLM and receive computational outputs without directly writing Python or R, though these outputs still require careful validation, interpretation, and refinement by domain experts.

Figure 1 shows a conceptual model of how the barrier-to-entry inversion enabled by LLMs democratizes computational thinking by abstracting away programming expertise, putting powerful tools into the hands of non-programmers. This figure captures the shift in who can engage meaningfully in computational thinking tasks like big-data statistical analysis before and after the emergence of LLMs. In the "Before LLMs" era on the left of the figure, the barriers to entry were steep: only computer scientists and data scientists had relatively easy access to such capabilities, while domain experts and the general public faced high technical thresholds. In the "After LLMs" era on the right of the figure, however, the barrier to entry is lowered across the board, especially for domain experts who already have the requisite knowledge to pose meaningful questions and can now perform once-esoteric tasks by submitting natural language prompts to modern LLMs.

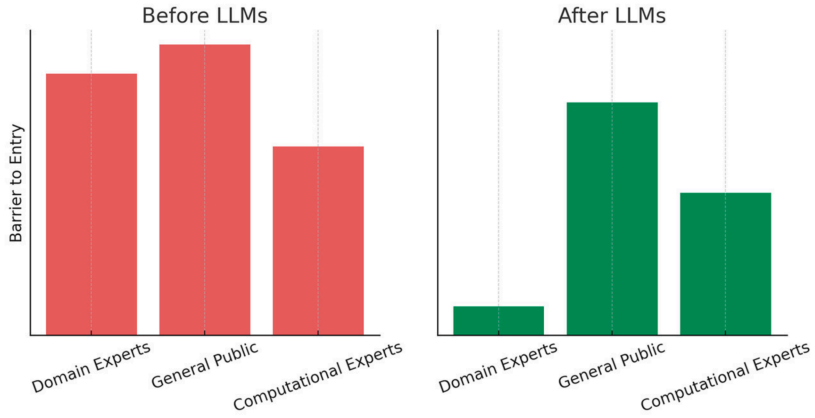

Figure 1. Barrier-to-entry inversion: a conceptual representation of the difficulty of big-data modeling before and after LLMs.

Figure 2 visualizes another aspect of this democratization: LLMs are beginning to compress the path from problem formulation to output generation, giving non-specialists access to capabilities that previously required substantially more technical skill. Recent studies illustrate this trend but also its present limits. In visualization, LLMs can often generate code for simpler charts and support basic visualization understanding tasks [8], while agentic systems such as MatPlotAgent show improved performance on scientific plotting workflows [9]; at the same time, these studies also indicate weaker performance on more complex visualizations and more nuanced interpretive tasks. In biology, spaLLM reports improved spatial-domain analysis in multiomics datasets through LLM integration [10], though such systems remain bound by task and modality constraints. In economics, EconAgent suggests that LLM-empowered agents can produce more realistic macroeconomic behavior within simulated environments [11]. This productivity acceleration underscores

how dramatically the barrier to entry for computational problem-solving has been lowered. Domains that once required custom programming for analysis or simulation can now benefit from this new mode of computational thinking. Computational thinking is thus no longer scarce; it's everywhere.

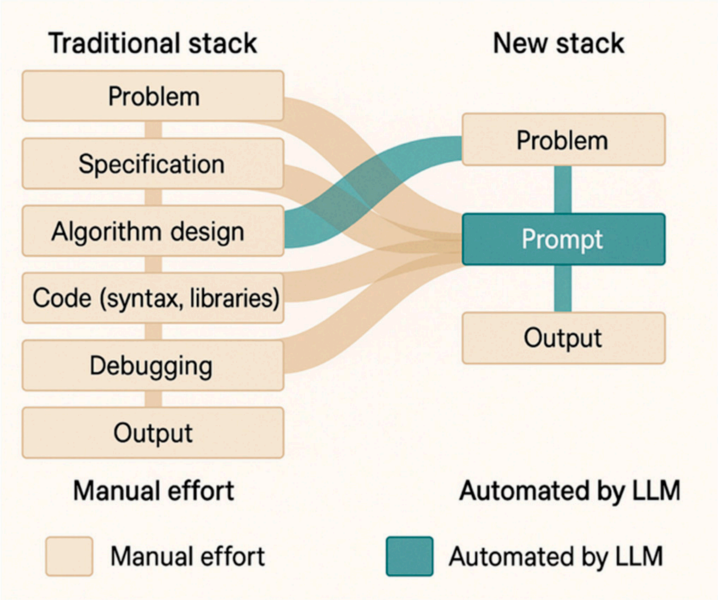

Figure 2. Collapsing the computational thinking stack.

3. Changing Roles in the AI-Augmented Workforce

This section examines our first guiding question, focusing on how AI is reshaping knowledge work and transforming the roles of developers and other computational professionals. Generative AI, along with related tools, is already reshaping the job market. Tasks once associated with knowledge work, such as analysis, coding, and writing, are increasingly automatable at scale. This trend marks a historic inflection point; earlier waves of automation displaced factory labor and service work; today, AI is encroaching on roles long considered secure behind a desk or workstation [12,13].

Many knowledge workers are understandably concerned about what AI portends for their careers. The familiar adage—often attributed to Richard Baldwin [14]—holds: "AI won't take your job, but someone who knows how to use AI might." In other words, those who skillfully leverage AI tools can outperform peers who do not.

In computational fields, generative AI appears to be redistributing developer effort rather than eliminating it, as highlighted notionally in Figure 3. Even before the rise of AI, developers spent substantial time on maintenance, debugging, and technical debt rather than on new implementation alone [5]. More recent studies of AI-assisted programming suggest that effort is increasingly shifting from manual code production toward prompting, reviewing, integrating, and validating AI-generated outputs, though the available evidence remains preliminary, task-dependent, and not yet representative of developer workflows in the wild [15]. If these controlled findings hold, it suggests that the developer role will evolve from producing code to managing and validating an AI-driven workflow.

This transformation raises a provocative question: if working solutions can be produced via strategic dialogue with an AI assistant, does that not also qualify as programming? The implication is that computational thinkers, i.e., those adept at framing problems and directing AI, may soon outnumber—and in some domains outperform—traditional software engineers in the workforce, even without traditional programming instruction.

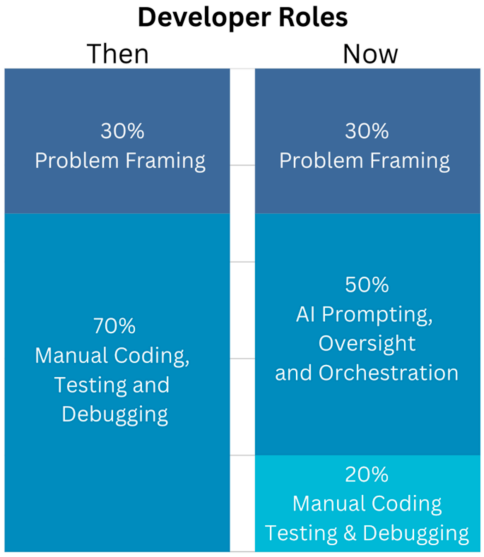

Figure 3. Redefining developer roles.

In a workforce paradigm dominated by computational thinkers, humans increasingly serve as the architects, guides, and editors of AI-produced work—not as traditional programmers. This shift is already redefining how both students and working professionals view their identities and career trajectories in technology and, in turn, forcing employers and universities to reassess how they define "career readiness" in the age of generative AI. Those best positioned for long-term success may be those who combine the speed and agility of AI-augmented learning with the rigorous thinking and domain mastery traditionally gained through formal study. That blend of fast, adaptive skill acquisition built on a bedrock of core principles will likely define the most resilient and versatile tech professionals of the coming decade.

4. Skills for Success: Prompt Engineering and AI-Oriented Practice

This section addresses our second guiding question by considering which capabilities are becoming more important as AI systems take on a greater share of implementation. The practice of prompting has advanced through distinct phases over the past few years, as shown in Figure 4. In the early days of the current AI boom—less than two years ago—prompting was often dismissed as a gimmick, a fleeting "flash in the pan" for extracting useful answers from LLMs [16]. At that stage, prompting was largely an art, i.e., a solo, trial-and-error process guided by intuition rather than disciplined methods. Over time, however, practitioners began to realize that carefully structured natural language inputs could reliably shape model outputs. Prompt engineering came to be understood as programming *through* the AI, rather than programming the AI itself, with wording, context, and sequencing exerting significant influence on the quality and reliability of responses [17,18].

The middle stage, shown in Figure 4 as *Prompting as Craft*, reflects the current maturation of this skill into a repeatable methodology. Seasoned users now leverage reusable prompt patterns, which are tested phrasing techniques that consistently elicit higher-quality or structured outputs [17,18]. Techniques like "chain-of-thought reasoning" prompts (where an LLM is directed to articulate its step-by-step logic before answering), "few-shot" prompting (where examples precede a new request), and "planning" (where explicit sub-goals are articulated by the LLM to reference during inference) allow users to consistently guide LLMs toward more logical, context-aware, consistent, and transparent results. At this

stage, prompting becomes less about improvisation and more about applying accumulated expertise to achieve predictable and accurate outcomes.

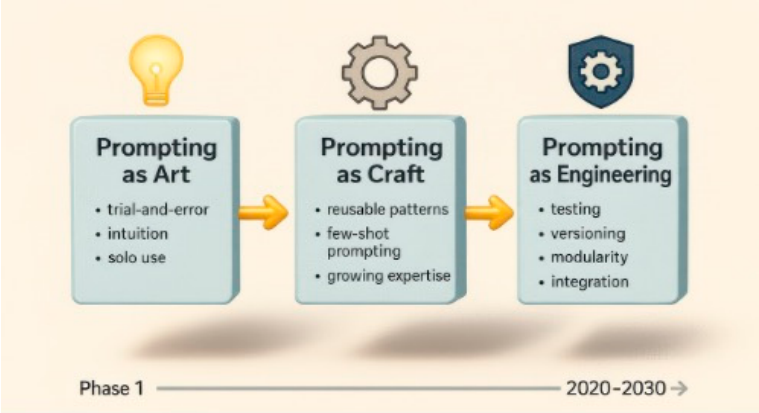

Figure 4. From promoting as art to prompting as engineering.

Looking ahead, Figure 4 proposes a possible third phase: Prompting as Engineering [16], or 'prompt engineering in-the-large.' We frame this as a prospective trajectory rather than a settled description of current practice. In this phase, prompting would move beyond individual experimentation toward a more scalable professional discipline involving testing, version control, modularity, and system-wide integration, mirroring earlier shifts in software development from ad hoc coding to structured engineering. Whether this transition occurs broadly, however, will depend on overcoming substantial barriers, including limited standardization, model instability, weak reproducibility, and the absence of mature certification and governance practices.

Scaling prompt engineering requires adopting many of the disciplined practices that matured in traditional software engineering, as shown in Figure 5. Version control, testing, documentation, code reuse, and modular integration are all relevant, but the analogy is not exact. Like source code, prompts are shaped by their execution environment; however, prompt behavior often depends more heavily on model versions, system settings, and surrounding context, with greater volatility and less standardization. Prompt versioning therefore requires tracking not just prompt text but also the model, configuration, and evaluation conditions under which behavior was observed. Prompt engineering thus resembles software engineering while confronting a more fluid development environment, requiring deliberate attention to the following practices:

- Quality assurance: Verifying that AI outputs are accurate, relevant, and appropriate for their intended context, not merely accepting the first response generated.
- Testing and validation: Exploring alternate phrasings and input scenarios to assess consistency and uncover potential failure cases.
- Maintenance: Updating prompts as models evolve or task requirements shift and tracking version history, which is akin to maintaining traditional source code.
- Integration: Ensuring AI-generated components integrate smoothly with each other and traditional systems, including error handling, prompt chaining, and context continuity.

These practices are nascent but will become foundational to reliable AI-driven development [16]. Without these practices, teams building complex systems through prompting risk accumulating significant technical debt. As Morris argues, the core concern is that excessive reliance on prompting as a substitute for stronger interface design and system design may leave AI applications built on unstable and insufficiently reproducible foundations [19].

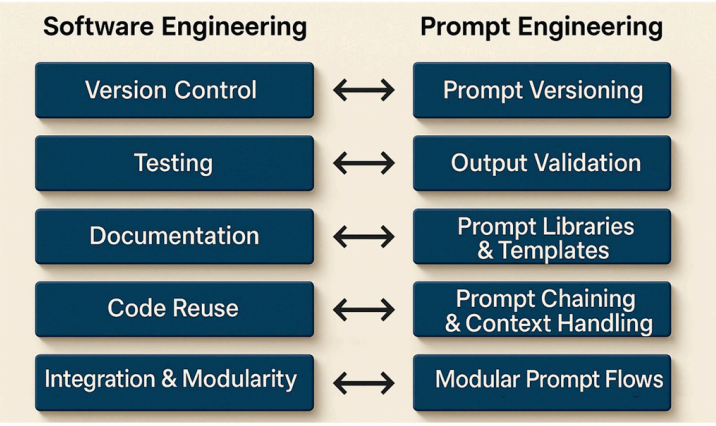

Figure 5. Comparing software and prompt engineering.

Conversely, approaching prompt engineering with a scalable engineering mindset unlocks significant potential. For instance, teams can curate libraries of validated prompts and model templates—analogous to software libraries—for recurring tasks like data extraction, code generation, or style transformation. Large-scale projects may institute prompt design reviews akin to code reviews or even unit tests to ensure AI components are reliable, interpretable, and aligned with fairness goals.

Over time, we expect AI tooling to mature, along with emerging patterns for orchestrating multiple AI agents within a single application. Integrated development environments (IDEs) are already offering features to debug, trace, and optimize prompt flows. In short, prompt engineering is evolving from ad hoc experimentation into a rigorous discipline poised to become central to future software-reliant application and system development.

5. Rethinking Pedagogy for an AI-Augmented Era

We now examine our final guiding question by examining how computing education should change in response to these shifts in practice, roles, and required skills. Few domains feel the impact of democratized computational thinking more acutely than higher education. Faculty across the computational sciences now face the challenge of preparing students for a world where conventional coding is no longer the primary means of, or barrier to, implementation. This challenge raises a foundational question: Should everyone still learn to code? For now, we argue the answer remains 'yes,' but in a qualified and likely transitional sense. Learning to code still matters today, not simply because students will write every solution by hand, but because coding remains one of the most effective available ways to teach core habits of computational reasoning, including precision, decomposition, debugging, and verification. At present, these habits are not yet consistently cultivated through AI interaction alone. At the same time, we do not assume that coding will remain the only or even primary pathway for developing such habits. As AI-augmented pedagogies mature, some of these competencies may increasingly be taught through other forms of structured problem formulation, model supervision, and solution validation. In this sense, coding remains valuable today, even if its pedagogical role may narrow over time.

5.1. Impact of AI on Computer Science Education

In an era of AI copilots and conversational IDEs, traditional software development skills are no longer sufficient. Instead, students must learn to collaborate with AI by leveraging tools like Codex, Copilot, Cursor, Zed, or Windsurf to design solutions, craft effective prompts, and critically evaluate the results. CS curricula must therefore expand

to teach prompt engineering and AI-augmented problem solving alongside coding with conventional programming languages.

Introductory CS courses have historically (and reasonably) emphasized writing programs from scratch in languages like Java, C++, or Python and debugging them manually, reflecting an era when coding was the primary pathway into computational thinking. Today, however, focusing solely on manual coding risks leaving the curriculum outdated. Students recognize that many tasks once considered essential can now be performed more quickly and reliably by AI tools and AI-augmented IDEs.

Universities that persist in teaching CS as they did decades ago will struggle to attract and retain students in an era defined by instant, AI-generated solutions. To remain relevant and prepare graduates for the workforce, therefore, educators must integrate AI-driven tools into their pedagogy. Rather than dismiss these technologies as shortcuts or "cheating," forward-looking instructors are framing natural language programming and prompt engineering as core components of modern computing practice [20].

For instance, an introductory programming course might incorporate AI-augmented IDEs, where students generate code with AI assistance but must still explain, refine, and validate the results. This approach, reflected in the "Modern CS Curriculum" side of Figure 6, shifts the emphasis from the conventional focus on syntax and hand-implementation to problem definition, logic, refinement, and verification. In doing so, students can build the skills necessary to thrive in an AI-augmented workforce where computational thinking increasingly takes the form of design and prompting rather than manual coding [21,22].

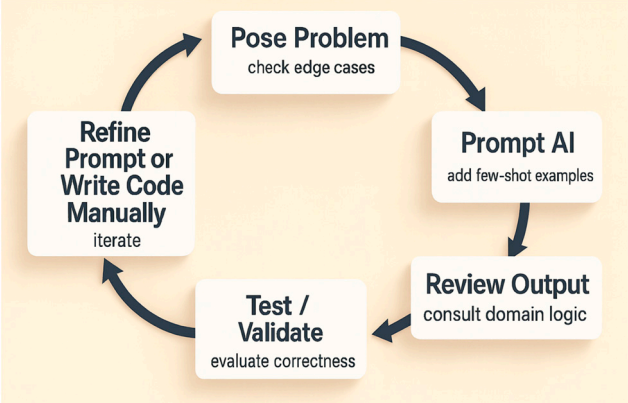

Figure 6. AI-augmented development needs iteration.

Educators should avoid encouraging blind reliance on AI tools [19]. Instead, they must develop discerning programmers who use AI to accelerate their work while still ensuring quality [20]. Effective practice follows an iterative loop shown in Figure 6: framing the problem, prompting the AI, reviewing outputs with domain expertise, and testing for correctness before refining prompts or writing code manually. This mirrors long-standing "computational thinking," but now the challenge is teaching it without the traditional anchor of syntax and programming—an unfamiliar paradigm that demands new instructional approaches.

Students trained in this iterative "trust but verify" approach that tests AI outputs, checks edge cases, and refines solutions will be much better prepared than those who simply accept an AI tool's initial output. Developing these habits requires metacognition to monitor how AI reasons, identify flaws or gaps, and intervene when needed. In essence, computational thinking must be redefined for the AI era. For the time being, graduates often still require robust mental models of algorithms and data structures. Increasingly,

however, they will apply those models in dialogue with intelligent systems, rather than solely through manual coding in traditional IDEs.

5.2. Impact of AI on Data Science Education

Data Science and related applied computational disciplines are grounded in statistical reasoning and computational methods, so they too face major disruption [23]. AI can increasingly assist with—and in some cases partially automate—many traditionally time-intensive steps, including ETL pipelines, data cleaning, exploratory analysis, and initial model construction, though current performance remains uneven, especially with real-world, domain-specific, or messy data, as shown in Figure 7. Rather than scripting in matplotlib or tweaking model syntax, data scientists can now issue natural-language instructions and receive draft analyses, visualizations, and models. As a result, success in data science tasks will depend less on coding mechanics alone and more on the ability to frame problems in ways AI systems can operationalize and extend and humans can critically validate, refine, and interpret.

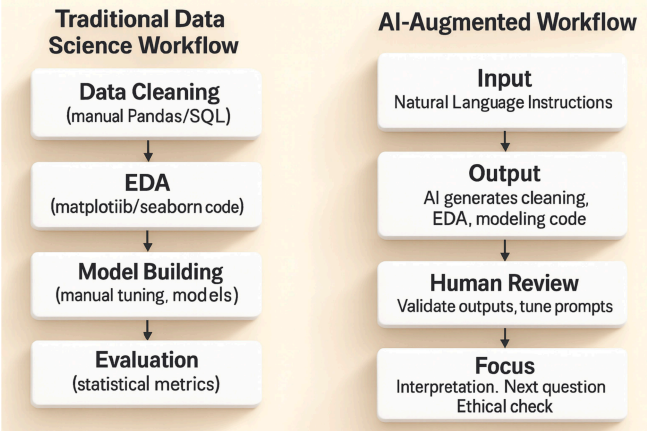

Figure 7. Yesterday's data science vs. tomorrow's.

This shift also makes teaching computational thinking central. With routine engineering tasks being offloaded to automation, the differentiating skill for new data scientists is the capacity to abstract business or research problems into data-driven forms that interface effectively with large language models and then to validate and interpret the results. Senior data scientists of tomorrow will need to define what kinds of data are required to solve a given problem, how to collect them at a reasonable cost, how to verify that AI-driven pipelines operate as intended, and how to analyze outputs downstream to generate meaningful insights. As highlighted in Figure 7, the discipline is evolving from "How do I build the pipeline?" to "How do I think about the problem so the pipeline and the AI behind it produce something I can trust?"

Rather than diminishing the value of fundamentals, the advent of AI in data science makes them even more essential. AI can generate code, models, and visualizations from natural-language prompts, but humans must still review and refine those outputs using their domain knowledge—and, for generated knowledge to be useful, a human must be accountable for and decisions predicated on the outputs. For example, students should ask: *Is the model valid? Are the assumptions sound? Is the analysis unbiased, cost-effective, and timely?* These questions rely on core statistical reasoning and operational awareness. By integrating AI-augmented tools into the curriculum, educators can train graduates not merely to accept AI results but to interrogate, validate, and improve them. In this new workflow, success shifts from hand-coding to curating analyses with judgment, context, and rigor.

5.3. Rethinking Curricula for an AI-Augmented Era

Academic institutions are beginning to revise their programs in response to generative AI, but the pace and depth of change vary widely. Demand now extends well beyond traditional CS pathways: students in the humanities, social sciences, natural sciences, and professional fields increasingly want to incorporate computational thinking and AI into their studies. One visible example is our own institution's minor in AI Ethics, which is designed to be accessible to all majors and spans courses across multiple disciplines. William & Mary should be viewed here as illustrative rather than exceptional. Recent national survey data suggest that this kind of experimentation is part of a much broader pattern: among 337 higher-education leaders surveyed in late 2024, 44% reported creating new AI-focused classes, 19% reported creating an AI major or minor, and 14% reported adopting AI literacy as an institutional or general-education learning outcome [24].

The three-tier curricular model in Figure 8 provides a conceptual model of how AI-related learning is expanding across the university. It should be understood not as a replacement for existing work on digital literacy, educational technology, or computing curricula, but as a way of organizing those concerns for the specific context of generative AI. At the foundation is AI Literacy, the widest and fastest-growing layer, which includes students from every discipline who are learning to read, question, and responsibly interpret AI-generated content. These students—whether in biology, economics, sociology, the arts, or professional fields—may never write code, but they are developing the ability to use AI tools to explore ideas, evaluate evidence, and support domain-specific analysis.

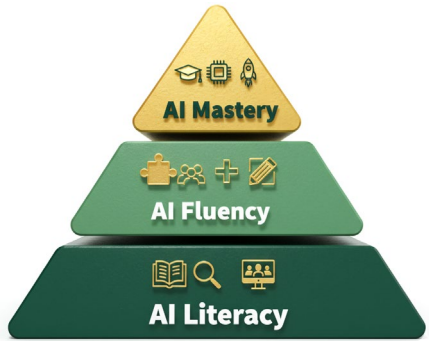

Figure 8. Hierarchy of AI proficiency.

The middle layer, AI Fluency, includes students who move beyond basic use to actively shape AI-assisted workflows. These learners combine computational thinking, data skills, and domain expertise. They can design prompts, critique model behavior, integrate analytic tools, and collaborate with AI systems to build and refine solutions. This group includes many students in emerging majors and minors that combine computer science, data science, and discipline-specific computation.

At the apex is AI Mastery, a smaller but vital group of students who develop deep technical competence—traditional programmers, computer and data engineers, and other specialists who build, optimize, and audit the systems everyone else relies on. They advance the underlying models, ensure reliability and security, and design new architectures and tools.

Together, these three layers describe a future in which computing education is no longer a single pipeline aimed only at producing programmers. Instead, it is becoming a broader educational ecosystem in which students participate at different levels and in different ways. The contribution of this model is to make those levels more explicit for an AI-augmented university: some students need the ability to use AI critically, others need to direct and integrate it in professional practice, and a smaller number need to

build and govern the systems themselves. This view carries significant implications for higher education. Universities must move beyond designing curricula solely for future programmers and instead support a much broader population whose advantage will come from AI-augmented reasoning within their own disciplines.

This shift also requires teaching students to frame problems for AI systems, critically assess machine-generated outputs, and refine results through domain expertise. Universities must embed AI tools throughout disciplinary curricula while cultivating adaptability alongside technical depth.

6. Navigating Limitations: Why Humans Remain Central

Across all three questions—workforce change, skills, and pedagogy—one theme recurs: AI increases the value of human judgment rather than eliminating it. AI-augmented development offers major benefits, but its limitations are real. LLMs and AI tools still carry notable risks as natural-language programming partners, especially oversimplification and hallucination. Early systems often produced confident but fabricated answers when used without careful prompting [25]. Even today's models can still confabulate under ambiguity or when pushed beyond their scope—sometimes inventing details or citations that appear credible but are entirely false.

Fortunately, generative AI tools are improving rapidly. Newer LLMs employ techniques that reduce the likelihood of hallucinations. Yet their outputs remain largely a black box that is fast but opaque, lacking the transparency needed for verification. While human thought is slower, it is more explainable and—critically, to many societal functions—accountable.

The lesson for education is unmistakable: LLMs and chatbots are no substitute for a solid foundation in domain knowledge and logical reasoning, any more than a junior associate in a law firm would be a substitute for a seasoned partner. Students must be trained not only to use AI tools but also to interrogate them so they learn when to trust, when to challenge, and when to disengage. Only then can AI serve as a reliable partner rather than an unaccountable oracle.

Another growing concern is how ubiquitous AI assistance may reshape human cognitive development. As shown conceptually in Figure 9, there may be an 'optimal zone' in which AI-augmented learning enhances engagement and productivity without displacing the productive struggle needed for durable learning [26,27]. Existing research suggests that over-reliance on AI may weaken critical thinking, decision-making, and analytical reasoning [28], while other experimental and meta-analytic work indicates that generative AI can improve learning performance and higher-order thinking under some conditions [29]. The pedagogical challenge, therefore, is not simply whether to use AI, but how to structure its use so that it augments rather than replaces core reasoning and schema formation [26,27].

These educational challenges intersect with broader ethical and societal concerns. Preserving academic integrity becomes harder when AI can complete entire assignments with minimal oversight. Authorship and attribution grow murky when content is generated collaboratively between humans and machines [30]. Bias in training data raises the risk of perpetuating harmful stereotypes or flawed conclusions, while the environmental costs of training large-scale AI models demand accountability and sustainability. Further complicating the picture are questions of misuse, ranging from disinformation campaigns to malicious automation. We do not attempt to resolve these issues comprehensively here; rather, we highlight them as a set of interrelated concerns that point to a broader research, pedagogical, and policy agenda. These issues are now at the forefront of academic and policy debates as institutions grapple with how to ensure AI empowers human potential without undermining it.

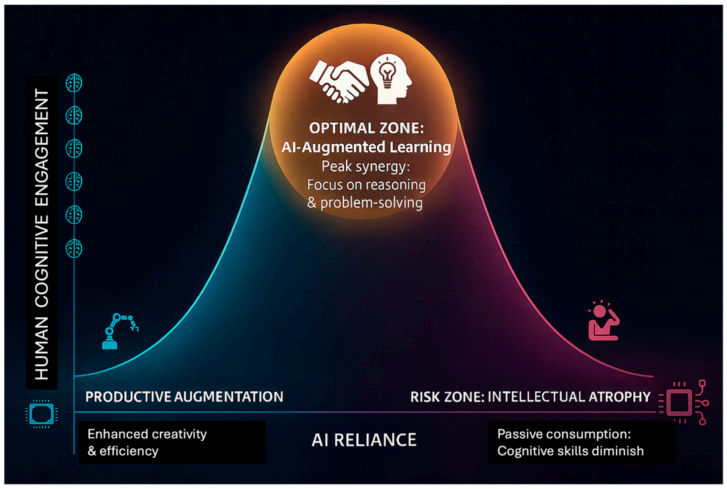

Figure 9. The sweet spot of AI reliance.

7. Conclusions: Reconnecting Pedagogy, Work, and Success in an AI-Augmented Future

We conclude by bringing the paper's three guiding questions back together and arguing that the future of computational thinking depends on aligning educational reform, evolving professional roles, and the cultivation of new forms of human–AI collaboration. The rapid spread of generative AI marks not an isolated disruption but the latest chapter in a long history of democratization in computing. What began with punch cards and mainframes accessible only to specialists moved to personal computers, graphical interfaces, the web, smartphones, and cloud platforms—each wave lowering technical barriers and expanding who could participate. Over time, tasks that once required deep programming expertise became routine activities embedded in user-friendly tools. Generative AI continues this arc by democratizing not just computation, but aspects of reasoning, synthesis, and problem formulation itself.

To resist this shift would be like King Canute commanding the tide to retreat—an instructive reminder that even authority has limits in the face of structural change. The diffusion of AI capabilities across society is driven by economic incentives, usability gains, and competitive pressures that cannot simply be legislated or wished away. Attempts to halt adoption outright are unlikely to succeed and risk leaving institutions unprepared. The more constructive response is to acknowledge the inevitability of this tide and focus on shaping how it is integrated—guiding its use toward responsible, equitable, and intellectually rigorous ends.

The AI-augmented tide is rising; our task is to learn how to surf its turbulent waters, rather than being left behind or overwhelmed by it. By embracing this change—and guiding it responsibly—we can ensure that the next generation of computational thinkers uses AI not to replace human creativity and judgment, but to amplify them. This emerging paradigm holds the potential to democratize computing but also raises urgent questions about the future of the computing profession and how we prepare the next generation of technologists.

Limitations: This paper is a theoretical and interpretive synthesis rather than an empirical study, and its claims should be read in that light. It draws together prior literature, historical comparison, and illustrative examples to advance a conceptual argument about the democratization of computational thinking, but it does not test that argument through original data collection, formal hypothesis evaluation, or systematic review methods. As a result, some of the claims advanced here are necessarily provisional, selective, and

dependent on the current state of a fast-moving literature. In addition, several of the examples discussed are intended to be illustrative rather than exhaustive, and the paper does not attempt to resolve all the important empirical, pedagogical, ethical, or labor-market questions raised by generative AI. A further limitation is the pace of change in the field itself. Generative AI tools, model capabilities, institutional responses, and workforce practices are evolving rapidly, which means that some claims made here may age quickly or require revision as new evidence emerges. We therefore offer this paper not as a final account of an already settled transition, but as a framework for ongoing debate, refinement, and empirical testing.

Author Contributions: Conceptualization, D.S. and D.R.; Writing, D.S. and D.R.; Visualization, D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created for this piece.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. National Academies of Sciences, Engineering, and Medicine. Historical Degree Production in Computing. In *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments*; The National Academies Press: Washington, DC, USA, 2018. [CrossRef]
2. CRA Enrollment Committee Institution Subgroup. *Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006*; Computing Research Association: Washington, DC, USA, 2017.
3. Lehman, K.J.; Karpicz, J.R.; Rozhenkova, V.; Harris, J.; Nakajima, T.M. Growing Enrollments Require Us to Do More: Perspectives on Broadening Participation During an Undergraduate Computing Enrollment Boom. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 809–815.
4. Tims, J.L.; Tucker, C.S.; Weiss, M.A.; Zweben, S.H. Student Enrollment and Retention in 2022–23 U.S. Undergraduate Computing Programs. *ACM Inroads* **2024**, *15*, 20–46. [CrossRef]
5. Stripe/Harris Poll. The Developer Coefficient: How Software Engineering Productivity Fuels Growth. Stripe, 2018. Available online: https://stripe.com/reports/developer-coefficient-2018 (accessed on 1 December 2025).
6. Wing, J.M. Computational Thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
7. Michaelsen, G.A.; dos Santos, R.P. Is English the New Programming Language? How About Pseudo-code Engineering? *Acta Sci.* **2024**, *26*, 157–204.
8. Khan, S.R.; Chandak, V.; Mukherjea, S. Evaluating LLMs for Visualization Generation and Understanding. *Discov. Data* **2025**, *3*, 15. [CrossRef]
9. Yang, Z.; Zhou, Z.; Wang, S.; Cong, X.; Han, X.; Yan, Y.; Liu, Z.; Tan, Z.; Liu, P.; Yu, D.; et al. MatPlotAgent: Method and Evaluation for LLM-Based Agentic Scientific Data Visualization. *arXiv* **2024**, arXiv:2402.11453. Available online: https://arxiv.org/abs/2402.11453 (accessed on 1 December 2025).
10. Li, L.; Dong, L.; Zhang, H.; Xu, D.; Li, Y. spaLLM: Enhancing Spatial Domain Analysis in Multi-omics Data Through Large Language Model Integration. *Brief. Bioinform.* **2025**, *26*, Bbaf304. [CrossRef] [PubMed]
11. Nian, L.; Gao, C.; Li, M.; Li, Y.; Liao, Q. EconAgent: Large Language Model-Empowered Agents for Simulating Macroeconomic Activities. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, Bangkok, Thailand, 11–16 August 2024; pp. 15523–15536.
12. Eloundou, T.; Manning, S.; Mishkin, P.; Rock, D. GPTs are GPTs: Labor Market Impact Potential of Large Language Models. *Science* **2024**, *384*, 1306–1308. [CrossRef] [PubMed]
13. Autor, D.H.; Levy, F.; Murnane, R.J. The Skill Content of Recent Technological Change: An Empirical Exploration. *Q. J. Econ.* **2003**, *118*, 1279–1333. [CrossRef]
14. World Economic Forum. Growth Summit 2023: Job Creation and Reskilling Must Be Central to Growth in the Age of Uncertainty, Advancing AI and the Green Transition. Available online: https://www.weforum.org/press/2023/05/growth-summit-2023-job-creation-and-reskilling-must-be-central-to-growth-in-the-age-of-uncertainty-advancing-ai-and-the-green-transition (accessed on 3 May 2026).

15. Mozannar, H.; Li, Z.; Wu, C.; Chen, J.; Wang, S.; Zhang, H. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*; ACM: New York, NY, USA, 2024; pp. 1–14. [CrossRef]
16. Chen, Z.; Wang, C.; Sun, W.; Yang, G.; Liu, X.; Zhang, J.; Liu, Y. Promptware Engineering: Software Engineering for LLM Prompt Development. *arXiv* **2025**, arXiv:2503.02400. Available online: https://arxiv.org/abs/2503.02400 (accessed on 1 March 2025).
17. White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; Schmidt, D.C. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. In Proceedings of the 30th Pattern Languages of Programs Conference (PLoP '23), Allerton Park, IL, USA, 22–25 October 2023.
18. White, J.; Hays, S.; Fu, Q.; Spencer-Smith, J.; Schmidt, D.C. ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design. In *Generative AI for Effective Software Development*; Duc, A.N., Abrahamsson, P., Khomh, F., Eds.; Springer Nature: Berlin/Heidelberg, Germany, 2024.
19. Morris, M.R. Prompting Considered Harmful. *Commun. ACM* **2024**, *67*, 28–30. [CrossRef]
20. Porter, L.; Zingaro, D. *Learn AI-Assisted Python Programming: With GitHub Copilot and ChatGPT*, 2nd ed.; Manning Publications: New York, NY, USA, 2024.
21. Delaney, J. Catching the Vibe of Vibe Coding. *Communications of the ACM News*, 6 May 2025.
22. Denny, P.; Prather, J.; Becker, B.B.; Finnie-Ansley, J.; Hellas, A.; Leinonen, J.; Luxton-Reilly, A.; Reeves, B.N.; Santos, E.A.; Sarsa, S. Computing Education in the Era of Generative AI. *Commun. ACM* **2024**, *67*, 56–65. [CrossRef]
23. Tu, X.; Zou, J.; Su, W.; Zhang, L. What Should Data Science Education Do with Large Language Models? *Harv. Data Sci. Rev.* **2024**, *6*, 1. [CrossRef]
24. American Association of Colleges and Universities (AAC&U); Elon University Imagining the Digital Future Center. *Leading Through Disruption: Higher Education Leaders Assess AI's Impacts on Teaching and Learning*; AAC&U: Washington, DC, USA; Elon University: Washington, DC, USA, 2025.
25. Hicks, M.; Humphries, J.; Slater, J. ChatGPT is Bullshit. *Ethics Inf. Technol.* **2024**, *26*, 38. [CrossRef]
26. Sweller, J.; van Merriënboer, J.J.G.; Paas, F. Cognitive Architecture and Instructional Design: 20 Years Later. *Educ. Psychol. Rev.* **2019**, *31*, 261–292. [CrossRef]
27. Kapur, M. Examining Productive Failure, Productive Success, Unproductive Failure, and Unproductive Success in Learning. *Educ. Psychol.* **2016**, *51*, 289–299. [CrossRef]
28. Zhai, C.; Wibowo, S.; Li, L.D. The Effects of Over-Reliance on AI Dialogue Systems on Students' Cognitive Abilities: A Systematic Review. *Smart Learn. Environ.* **2024**, *11*, 28. [CrossRef]
29. Deng, R.; Jiang, M.; Yu, X.; Lu, Y.; Liu, S. Does ChatGPT Enhance Student Learning? A Systematic Review and Meta-Analysis of Experimental Studies. *Comput. Educ.* **2025**, *227*, 105224. [CrossRef]
30. Lemley, M.; Ouellette, L. Plagiarism, Copyright, and AI. University of Chicago Law Review Online, 2025. Available online: https://ssrn.com/abstract=5399463 (accessed on 1 December 2025).