LLMs are not calculators: Why educators should embrace AI (and fear it)

Daniel Jackson
December 26, 2025
This version: January 11, 2026 (v10)

Introduction

In *The Most Human Human* [1], Brian Christian recounts his role as a contestant for the Loebner Prize, in which real people and AI chatbots competed in a version of Turing's imitation game. As a 'human confederate,' Christian's goal was to be deemed the 'most human' respondent by a panel of judges. Fifteen years later, the issues the book raises—what it truly means to be human; whether sounding human is the same as being human; how we are influenced by the machines we use to behave more like them—are as unresolved as ever. But the Turing test is no longer a topic of discussion.

At the close of the last Loebner contest in 2019, a $25,000 prize that had been set aside for the first machine deemed indistinguishable from a human remained unawarded. One wonders what might have happened had the contest still been running when ChatGPT appeared only a few years later. Its many users who today close an interaction with ChatGPT by saying 'thank you' are surely confirming the credibility of its dialog—even as they are unaware that, ironically, their polite closing exacts a highest cost than any previous message (since each input from the user is conveyed to the LLM along with all earlier queries and responses). And yet, at the same time, it is clear that LLMs are far from replacing humans, especially in tasks (such as teaching) that call for emotional engagement with others.

Herein lies a central dilemma facing educators. An LLM can produce simulacra of teacher instruction and student work that are—superficially at least—barely inferior to the real thing, but on closer inspection are often revealed to be a fraud, pretending to hold fresh insights when in fact few are present. In this respect, although calling LLMs 'stochastic parrots' seems unfair, it often feels as if they have mastered the form of communication but little of its true substance.

If instead of judging an LLM's humanity, however, we ask more mundane questions about its utility as a tool, a different picture emerges. LLMs are excellent at coding (at least when closely guided), good at summarizing, and surprisingly effective at brainstorming (within conventional domains). They also offer an alternative to search engines that is often more effective in finding relevant artifacts, especially when it's hard to formulate an effective query.

Observations

Whether to use AI in education is not the question; our students will do so anyway. The question is *how*. Let me disappoint you upfront by revealing that, as yet, I have no answer. This little memo should instead be viewed as an elaboration of the question, in a set of observations that we will need to take into account.

These observations are neither original nor complete. I am writing them down largely for my own benefit. This fall my colleague Mitchell Gordon and I, along with my doctoral student Eagon Meng and a team of teaching assistants, experimented with placing LLMs at the center of our undergraduate software development class. We are about to embark on analyzing the results. I considered delaying writing up these thoughts until the analysis was complete, but came to think that it would be useful for formulate them in advance so they might serve as hypotheses to be tested.

This piece is not a systematic analysis, but more a collection of personal reflections. A report by Andrés Salazar-Gómez and Sanjay Sarma [2] offers a more comprehensive analysis of the ways in which AI might be adopted in education, and the benefits and risks entailed. Its ideas—and Sanjay's educational vision more generally—influenced my thoughts here. Its conclusion is similar to my own: that the advent of AI calls for a renewed focus on pedagogy, and attending more to *how* students learn rather than *what* they learn.

An LLM is not a pocket calculator or a spell checker

It may be reassuring to liken the advent of LLMs to the appearance of pocket calculators in the 1970s, since it would imply that they will replace humans only in trivial skills. Our worries about deskilling would be no different from earlier worries that high schoolers' inability to do long division would threaten their job prospects.

But this is a dangerously misguided view. Calculators could add, multiply and divide big numbers but not much more, and people unable to do these things could still function effectively (especially in England where I grew up, where innumeracy and ignorance about science was often a badge of honor). Having a sense of orders of magnitude and being able to estimate is surely a handy skill—to be able to figure out, for example, that if something doubles thirty times it will grow by a factor of about a billion (if you remember that the base ten log of 2 is about 0.3, so thirty doublings are about 30 * 0.3 = 9 factors of ten). But in their daily lives, most people encounter such questions rarely, and when they do they are likely to be in a context in which computation is available and more accurate answers are needed (for example, calculating compound interest to determine the cost of a mortgage). And anyway estimation was always an ability distinct from computation, and one that calculators never supplanted.

LLMs, in contrast, can summarize and write on behalf of their users. Even if an LLM's best efforts tend to be clunky and staid, they may be good enough for many tasks (and superior to those of children learning to write). Hand wringing about a generation unable to multiply 23 by 17 seems quaint when we consider the prospect of a future in which most people cannot express themselves in full sentences. And if, as some claim,'writing is thinking' and the formation of ideas and their expression are inseparable, this bodes ill for our future ability to think at all.

It is also possible that the naysayers were right about calculators, and that the loss of any basic skill has profound and unpredictable consequences. On the other hand, we should remember that not all learners are alike. Eliminating barriers to expression might enable more students to flourish intellectually, especially those who (due to dyslexia, for example) have good ideas but have trouble translating them into a communicable form. Although relaxed policies for calculator usage eventually prevailed, and concerns about the loss of a basic skill may have been overwrought, studies suggest that adopting calculators actually enhanced problem solving ability for many students [3].

LLMs excel at expansive tasks

Computers, until now, seemed to be good only at technical tasks that could be precisely specified and that required no judgment in their execution. They excelled in their ability to perform such tasks on a vast scale and with a reliability and precision that humans could not match, and this is what made computers interesting: even if each step was dull and easily performed by a human, the overall effect was something new.

LLMs, however, seem to be more like humans than their computational predecessors. They are poor at sharp-edged tasks that call for exact reasoning, and shockingly good at squishy, expansive tasks even when given vague and incomplete directions.

In software design, for example, prior to LLMs, the available tools would be of little help in brainstorming possible features for your app, but could help you find subtle bugs in a design by exhaustively searching its state space. With LLMs, the very opposite is true. Their training on a vast corpus of knowledge about software designs and applications enables them to enumerate in seconds a wider range of possibilities that most designers could imagine, and yet they are incapable of the kind of systematic analysis that finding subtle bugs often requires[1].

LLMs are not compilers

Because LLMs are so good at generating code from informal specifications, it's tempting to think of them as compilers for a higher-level language. This analogy is misleading. LLMs are notoriously unreliable, and will, for example, invent library functions that don't exist (which would be amusing if hackers hadn't discovered that the names of such 'hallucinated' functions appear repeatedly, allowing rogue versions of such functions to be posted online in the hope that they will be incorporated by lazy programmers). On the other hand, the ability of LLMs to use domain knowledge to fill gaps in informal specifications (and even override explicit constraints) is incredibly useful in practice.

This means reversing a common programming practice, in which a domain-specific problem is 'abstracted' into a generic variant, thereby revealing a mathematically pure form that is more amenable to standard solutions. With LLMs, the domain-specific variant may be preferable.

For example, prior to LLMs, a novice programmer aiming to code a social media friending concept (that restricts access to a user's posts to their friends) would have been advised to restate the problem in terms of connections in a generic graph, enabling the use of standard graph-theoretic reasoning. An LLM, however, will do far better with a specification that talks about users and friends. It will correctly assume, for example, that the friend relation should be symmetric, and that users should have access to their own posts, even if this contradicts a specification given by the user.

---

[1] As an example, I posed the following question to GPT 5.1: *Suppose I have a phone book that stores phone numbers for different users, with actions to add and remove numbers for users. Will adding and removing the same number for a given user always leave the phone book in its previous state?* A tool such as Alloy would immediately generate a counterexample: adding a number that is already present. On one run, GPT failed to find this case; on another, it found it, but only after a paragraph that listed conditions to ensure this property (which it claimed to be sufficient but were in fact not). At the same time, GPT suggested that my property might be broken if adding but not removing normalized the formatting of phone numbers; or that in a distributed setting, the removal request might be arrive at the server before the add—expansions of the context that one would associate more with human than tool intelligence.

In short, the standard computer science strategy of abstracting problems into more generic, application-independent forms is not compatible with LLMs, and may anyway be worth reconsidering (by adopting more design-centric approaches). And because LLMs are so tolerant of incomplete specifications, programmers will need to be keenly aware of how gaps are filled, often incorrectly.

Checking LLM answers

A beautiful idea in computer science contrasts the cost of finding an answer to a problem with the cost of checking one. For many classes of problems, the costs differ dramatically, making finding intractable but checking easy. For example, finding values for boolean variables that makes a formula such as *P and (not P or Q)* true[2] generally requires trying every possible combination of values (of which there are 2^N for N variables), but checking whether a particular assignment of values is valid is trivial, involving no more than plugging in the values and evaluating the formula.

One wonders then if LLMs might be solving problems which, if intractable in the general case, are at least checkable. This would align with the obvious advice to carefully check an LLM's responses. For some kinds of problem, this is a reasonable approach. Code generated by an LLM can be tested, and some researchers believe LLMs might one day generate proofs of correctness, which could then be confirmed step by step. Citations can be checked and links can be followed; claimed facts can be looked up in Wikipedia.

In practice, however, the situation is not so rosy. First, many users simply will not take the trouble to evaluate LLM responses. It's just too tempting, especially in the face of a pressing deadline, to accept the output blindly. Second, the validity of an LLM response may depend as much on what is *not* said as what is said, and errors of omission are not readily checkable. When these two factors come together, the results are especially unfortunate. Students in our class insisted, for example, that the framework we told them to use lacked certain features, even though a Google search would have found these features immediately in publicly available documentation.

Context selection

Every query to an LLM involves a single query and produces a response. There is no memory of any prior queries or responses. Indeed, this is one respect in which some people believe LLMs are inherently inferior to humans, lacking the ability to learn from their interactions. The illusion of memory is produced in tools like ChatGPT by simply adding to any query a context comprising all the previous queries and responses. This augmentation is implicit and even extends beyond a single chat; ChatGPT's memory feature allows users to decide whether previous conversations are included. Additionally, a ChatGPT query can be augmented with an uploaded file. By all these means, the application gives the user some limited control over the *context* of a query.

The choice of the context for a query has become a primary question, with 'context engineering' now replacing 'prompt engineering' as the vogue term for the key skill required when using an LLM [4]. In coding tools such as Cursor and Copilot, control of context is surprisingly rudimenta-

---

2 This problem, known as SAT, has a particularly interesting history because it was the first to be shown to be NP-complete, and so became a yardstick for hard problems. Theoreticians assessing a problem could translate SAT into it, thereby showing that if it could be solved easily, SAT could too—thus proving intractability by contradiction. By the 1990s, however, SAT solvers has become so effective for the problems that actually arose in practice that people started translating problems into SAT. So the archetypal hard problem became the archetypal easy problem.

ry, and often results in modifying files that should have been beyond the scope of a request. In one assignment we gave our students, we recommended that they use a tool that the course staff had developed that offered precise control of context, and warned them against using more automated tools such as Cursor. The temptation to skip the work of identifying the context proved irresistible, and almost 80% of the students seem to have succumbed and in so doing broke the underlying framework (which Cursor unwisely decided could be modified to incorporate problem-specific details). It was interesting that these students apparently didn't even notice that Cursor had messed up.

Especially in programming, therefore, students should use tools that require them to specify query context explicitly. Ideally, tools should preserve a complete provenance chain, showing which inputs led to which outputs. This would also support a more incremental and exploratory way of working in which a user can try different prompts and go down different paths, and go back to earlier points, revise prompts and move forward again.

Intentionality in LLM usage

LLMs are extremely flexible tools, and for any given task can be used in a wide variety of creative ways. Given a problem to solve, asking the LLM for a solution is only the most obvious, and perhaps the least useful. One could ask the LLM to critique a solution prepared by hand; to critique its own solution; to find a range of solutions and compare them; to build a refutation to a proposed solution; to analyze the tone and style of a proposal, and evaluate it for symptoms of cognitive bias or prejudice; and so on.

Taking full advantage of these opportunities not only will produce better results, but also preserve (and maybe even enhance) the user's sense of personal agency. Intentional usage won't come naturally to students, however, and will need to be encouraged and inculcated explicitly and systematically. In our class, even though we provided students with rubrics and suggested they be used to evaluate ongoing project work, very few students took advantage of this—which is especially surprising given that the grading depended on the same rubrics. The urge to minimize work is overwhelming, and most of us are led towards tool usage that will confirm rather than contradict our assumptions.

Lorena Barba provides an admirably candid account of the impact of introducing LLMs in a beginner programming class [5]. It's a gruesome but insightful and instructive tale. Her students used LLMs simply to avoid work, copy-pasting problems in and solutions out. They resisted her advice on more intentional usage, and became angry when she tweaked the autograder to accept only single submissions (so that repeated attempts from aimless repetition were no longer possible).

The temptation to complete an assignment more quickly may simply be so great that when students have access to an LLM they will tend to use it in whatever way minimizes their effort, even if at the same time it minimizes their learning. It is almost as if students become drugged, not only unable to resist the LLM's siren call, but blind to the slop that they submit, somehow imagining that their teachers won't notice.

Barba wisely observes that in some respects this is nothing new. AI is exacerbating an old problem: the 'illusion of competence,' a cognitive distortion in which students believe they are working pro-

ductively and progressing well when in fact they are not learning at all. The only antidote, she suggests (in line with Salazar-Gómez and Sarma [2]) is active learning and reflective practices.

LLM influence on reading and writing

Interacting with a chatbot like ChatGPT can feel rewarding and engaging (sycophancy aside). This suggests a productive way to work: making emerging thoughts explicit, refining plans, brainstorming directions—all in a back-and-forth with an AI partner. I have been amazed at how effectively I can use ChatGPT to help me understand a research paper (asking it to provide concrete examples, compare to other work, refute my objections, and so on). I also use it to tighten new research ideas, and to generate cases that challenge my generalizations. One software engineer I know who works alone spends his entire day in conversation with an LLM, laying out ideas progressively and getting incremental feedback and suggestions along the way.

This style of work may be good in the way in which it encourages *writing*. Articulating, recording and questioning emerging design ideas is invariably better than brainstorming[3] alone. Moreover, the centrality of LLM prompts in software development suggests that they should be saved as essential development artifacts, along with the causal chains showing what preceded each piece of code or design or analysis. This will surely reverse the troubling tendency, elevated to a principle by the advocates of agile methods, to view informal artifacts such as specifications and designs as suspect and inferior to the code, the 'real' product.

On the other hand, engaging with an LLM as a trusted and monogamous partner has a serious downside: a reduction in *reading*. I asked the teaching assistants in my class what in their view distinguished the students who were most successful in their project work. They told me that it was prior experience with complex software development. When I asked what particular skill they had that less prepared students lacked they surprised me by pointing to rather a basic one: the ability to locate and read relevant documentation. Novice software developers equipped with LLMs, it seems, are not inclined to obtain background understanding in advance of a design or coding task, and instead call upon an LLM on demand, when the need arises. As a result, they never gain a big picture view, and find themselves tacking from side to side, uncertain of their direction, and following advice that often turns out to be mistaken.

Used intentionally, however, LLMs may help us become better readers. I've found it very helpful when trying to understand a research paper to ask an LLM to explain aspects of it, to construct additional examples for me, and perhaps most usefully, to rebut my initial criticisms. We bring our own cultural assumptions and cognitive biases to everything we read, so—especially when reading something from a different community whose assumptions and use of language differ from our own—it can be helpful to have an LLM confirm that one's interpretation makes sense, or even just to ask 'This seems obvious. What am I missing?'.

LLMs are objective

My last point touches on an intriguing advantage of LLMs. While the tendency of LLMs to produce output that is formulaic and flat is rightly criticized (and tends to make them poor writers),

---

[3] Skeptics of design thinking, amongst which I count myself, have noted that its greatest beneficiary may have been 3M, the maker of sticky notes. Perhaps LLMs are the sticky notes of the future.

the flipside of this weakness is a strength. Lacking a personal voice and a subjective take, an LLM may provide more objective feedback than friends and colleagues. You can have an LLM respond to your own writing, finding weaknesses in your arguments, identifying cognitive biases, or highlighting places in which your tone might come across differently from what you intended (and you can even specify the audience you have in mind).

For this to work, you might have to overcome the LLM's default sycophancy and instruct it to be adversarial. You should be skeptical of supportive responses that confirm your presuppositions. You also need to be careful about context, and make sure that its analysis is based solely on the artifact at hand. This may require, for example, turning off the feature in ChatGPT that allows one chat to be used as a context for another.

How LLMs will get us organized whether we like it or not

In our class this past fall, we taught our students a novel code structuring technique that offers better separation of concerns than traditional approaches [6]. The behavior of the system is described as a collection of *concepts*, each of which encapsulates a coherent unit of functionality and has no dependencies on other concepts. Concept specifications and implementations can then be generated, by hand or LLM, entirely independently; each concept can be treated almost as if it were an application in its own right.

Independence of concepts is essential for teamwork. Without it, different members of team are likely to tread on each other's toes, especially if they are using agentic coding tools which are not designed to respect modularity boundaries. Independence also reduces the need for context: implementing a concept requires the specification only of that concept, and of no other.

For this approach to work smoothly, we gave our students instructions for how to develop their applications incrementally and systematically. We also gave them a collection of background documents to be used as context for LLM prompts, to give the LLM guidance about concept design and implementation.

I've argued elsewhere [6] that the power of LLMs in software development will be amplified dramatically if we bring more structure to our software products and processes. Without such structure, not only are the advantages of division of labor lost, but the LLM itself must work harder to perform development tasks, and undesirable outcomes (such as straying beyond module boundaries) are more likely.

In this respect, software development is no different from any other complex activity. If we want an LLM to grade student work, it will surely do better if given comprehensive, clear and well structured rubrics. Likewise, in a business setting, I cannot imagine that LLM performance won't be improved by streamlining processes and documenting them carefully.

I recently read about a startup that offers an LLM coding tool that augments any explicit specification from users with goals that it finds by reading team email threads and Slack channels. Apparently the founders believed that this would allow (and even encourage) developers to continue to work in a haphazard manner, sharing their ideas not only in structured, official documentation but also in unstructured exchanges on Slack and email.

My prediction is that such a tool would have exactly the opposite effect. As the developers learned how much impact their Slack and email comments had on the code being generated, they would alter their habits, becoming more careful and structured in their conversations, deleting or demoting references to bad ideas, and so on. The human participants thus end up conditioned by their environment, like Pavlov's dog. Rather than us training the LLM, it ends up training us.

The dangers of partial automation

When self-driving cars first appeared, the Society of Automotive Engineers defined six levels of automation [7]. Below full automation (at levels 4 and 5), the scale distinguished crash avoidance and driver assistance features (such as automatic emergency braking at level 0 and adaptive cruise control at level 1) from partial automation (at levels 2 and 3), as found in Tesla's autopilot.

Experts worry that partial automation creates a false sense of security, induces boredom and then —when it fails—leaves the driver in a challenging situation without adequate contextual awareness. And indeed studies suggest that, unlike crash avoidance and driver assistance, partial automation may not reduce crash risk at all (and a correlation between reduced partial automation and collision rates may be due to better equipped cars having brighter headlights) [8].

The risks of partial automation apply to LLMs too. When they are used to generate complex artifacts, and worse to iterate on them, users may be left behind and put in an untenable position, having lost control of their work. This seems to be what often happens with coding tools such as Cursor and Copilot if used without tight oversight: they rapidly corrupt the codebase, undermining module boundaries and introducing subtle bugs and security vulnerabilities, so that the programmer is soon demoted from collaborator to confused bystander. If this becomes a widespread practice in industry, we can expect a steadily growing 'technical debt' that will result in more brittle, insecure and inflexible software.

Psychological costs

In considering the impact of LLMs on the products of human work, educators have a special concern. While the immediate work product may be the assigned essay, analysis or design, the ultimate product of education is of course the learners themselves. So the ways in which LLMs shape the mental skills and habits of their users becomes central.

Little is known about this, but there are reasons for concern. A recent EEG study of students writing essays found that neural connectivity was lowered by use of web search and lowered yet further by use of LLMs—and that 'brain only' participants reported higher satisfaction, and produced more diverse work with fewer factual errors [9]. Most worrying are possible long term impacts, such as a loss of critical thinking skills [10] and more general cognitive atrophy.

Social isolation is also a risk. Student engagement in our class forum was much lower this year than in previous years. Whether this is due in part to LLM use is unclear. Other, more mundane, impacts of LLMs may require scrutiny too. Some of our students grumbled that using LLMs in coding means spending most of your time *waiting* for the LLM to respond. The consequences of such an environment for motivation and 'vigilance decrement' [11] remain to be seen.

Conclusions[4]

Large language models are not like calculators or reference tools. They embed themselves deeply within the practices—reading, writing, reasoning, and designing—through which learning itself occurs. That is what makes them both powerful and dangerous in educational settings.

The key risk is not cheating or factual error, but the gradual erosion of student agency. When LLMs take over expansive, ill-defined work, they can create an illusion of understanding while displacing the slow, effortful processes through which real insight is formed. But when used deliberately, they can widen the space of ideas students explore and make the normally invisible work of thinking more explicit.

As educators, we cannot leave this tradeoff to chance. LLMs should neither be banned nor treated as neutral tools. They must be integrated intentionally, with explicit attention to context, modes of use, and the preservation of student ownership over both process and product. The goal is not better outputs, but better thinkers. Whether LLMs undermine or strengthen that goal will depend not on the technology itself and what its future might hold, but on the pedagogical choices we make.

Acknowledgments

I learned most—perhaps all—of the ideas in this article from friends and colleagues, including Martha Eddison, Mitchell Gordon, Eric Klopfer, Eagon Meng and Sanjay Sarma.

References

[1] Brian Christian. *The most human human: What artificial intelligence teaches us about being alive*. Anchor Books, 2012.

[2] Andrés F. Salazar-Gómez and Sanjay Sarma. *AI in Education.* Future of Education Series, July 2025. Available at https://openlearning.mit.edu/sites/default/files/2025-10/How to Think of AI in Education_SalazarGomez-Sarma_2025_v2.pdf.

[3] Aimee J. Ellington. A Meta-Analysis of the Effects of Calculators on Students' Achievement and Attitude Levels in Precollege Mathematics Classes. *Journal for Research in Mathematics Education* 34(5). Nov. 2003.

[4] Anthropic. *Effective context engineering for AI agents*. September 29, 2025. Available at https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

[5] Lorena A. Barba. *Experience embracing genAI in an engineering computations course: What went wrong and what next.* Figshare. Journal contribution. Available at https://doi.org/10.6084/m9.figshare.28926647.v1. March 2025.

[6] Eagon Meng and Daniel Jackson. What You See Is What It Does: A Structural Pattern for Legible Software. *Proceedings of the 2025 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '25).*

---

[4] This conclusion was drafted by ChatGPT 5.2. It's better than the one I wrote originally. When I asked GPT to critique the paper, it identified several other flaws, most of which I have chosen not to correct (such as the over-reliance on examples from software engineering). It rightly observed that while 'this essay works best as an internal faculty white paper, a discussion starter for curriculum committees, or a foundation for a more formal empirical or policy-oriented follow-up, to reach a broader audience it would need clearer stakes, a more explicit normative position, and tighter integration of evidence.'

[7] SAE International. *SAE J3016 Recommended Practice: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 2014. Available at https://www.sae.org/news/blog/sae-levels-driving-automation-clarity-refinements.

[8] Insurance Institute for Highway Safety (IIHS). *IIHS-HLDI research finds little evidence that partial automation prevents crashes*. July 11, 2024. https://www.iihs.org/news/detail/iihs-hldi-research-finds-little-evidence-that-partial-automation-prevents-crashes.

[9] Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, Pattie Maes. *Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task*. https://arxiv.org/abs/2506.08872.

[10] Hao-Ping (Hank) Lee et al. The Impact of Generative AI on Critical Thinking: Self-Reported Reductions in Cognitive Effort and Confidence Effects From a Survey of Knowledge Workers. *Conference on Human Factors in Computing Systems*, 2025 (pp. 1-23).

[11] Norman H. Mackworth. The breakdown of vigilance during prolonged visual search. *Quarterly Journal of Experimental Psychology*, Vol. 1, pp.6–21, 1948.