

# Code acceleration: Rethinking modern programming practices

GEORGIOS DIAPOULIS, University of Gothenburg and Chalmers University of Technology, Sweden

Using large language models to write code is a common practice among both novice programmers and experts. Multiple prompts often produce large amounts of text and code. For novice programmers, such practices can result in tangled and overly long scripts that are incomprehensible to beginners and unintuitive for experienced developers. Is there a sweet spot for the amount of generated code, or should we rethink and redesign these user practices? Here, I explore speculative futures of programming through turn-based text generation using large language models, and I advocate for a material-centered approach to designing interactions.

CCS Concepts: • **Human-centered computing** → **Interaction design; Text input.**

Additional Key Words and Phrases: Live Coding, Live Writing, Co-coding

## ACM Reference Format:

Georgios Diapoulis. 2025. Code acceleration: Rethinking modern programming practices. *xx, y, Article zzz* (2025), 3 pages. <https://doi.org/XXXXXXXX.XXXXXXX>

## 1 Introduction

The generated code has been accelerated since the advent of large language models (LLMs). Many programmers and the industry of programming are embracing the use of LLMs for writing code at unprecedented rates. I can imagine a scenario where future programmers use mostly generated code from LLMs that the code they authored. An attempt to compare modern programming practices with the early days of programming (e.g., punch cards) demonstrates a shift from objects that exist in physical space to virtual objects that occupy an ever-expanding space. In a nutshell, the activity of writing code becomes more accessible and less diversified [8, 14], and at the same time coding becomes less of a craft and more an issue of accepting suggested changes.

It is evident that the generation of code using LLMs contributes to a technological acceleration that has a direct impact on the means of production. The selling point for code assistants is that they can improve productivity; as a consequence, one can imagine having more free time to pursue their own interests outside of the work environment. In practice, there is a feedback loop between technological acceleration, means of production, and acceleration of social change driven by capitalism and realized using alienated white collar workers.

The ideas I presented here are inspired by the concept of social acceleration by Harmunt Rosa [13]. In the rest of the article, I will examine how the concept of acceleration applies to modern programming practices. Insights will be drawn out from the artistic practice of live coding, which presents an exemplary case of writing code in front of the audience, while the computer program is running. I advocate for an approach to write code differently [7].

---

Author's Contact Information: Georgios Diapoulis, University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden, [georgios.diapoulis@chalmers.se](mailto:georgios.diapoulis@chalmers.se).

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM XXXX-XXXX/2025/0-ARTzzz

<https://doi.org/XXXXXXXX.XXXXXXX>

## 2 The act of (live) writing

Writing is a technology, and coding and writing are activities that unfold over time. Live writing (e.g., writing in a shared document, writing in public) presents a speculative relationship between the two activities of coding and writing, and it is a defamiliarization technique for writing and thinking about writing [2]. Under this prism, live writing shares some similarities with a design approach of using the material as a method, known as material-centered design [15]. In live writing, "writerly concepts are abstracted and emerged" [2][p. 3].

The present article is not written in a lively manner instead I used a top-down approach of iterative refinement. Writing, editing, deleting, and rewriting, were the main ingredients in my writing approach. Contrary, live writing is a bottom-up approach [3]. Imagine for a moment that I was writing this article using an LLM – which I did not, it was mostly typed on Emacs Orgmode using incremental completions from Helm and then imported to Overleaf's ACM template. How could I have used an LLM to live write the article? If I had followed Blackwell's approach for live writing the Live Coding Manual [3], I could have made a list of prompts, asking the LLM to write me a certain amount of text. Each generated text chunk, would be a lemma in a wiki, and later on I could have used different design techniques to linearize this (imaginary) wiki. What happened in practice, is that the present article was written in an unstructured manner, maybe even chaotic, trying to put my thoughts into order and on a (pixelized) paper.

## 3 What can we learn from live coding practices

Live coding is an unconventional approach to programming, where the programmer applies on-the-fly modifications to the running program [4]. It is a lively and active research community holding its own conference since 2015, but most importantly there is an increased interest from artists and practitioners who bring novelty and creativity to the table. Collaborative and networked live coding is a common practice since the early days of live coding [6] and currently there are numerous projects on web interfaces that enable spontaneous collaborations to happen.

Live coding shows a resistance to LLMs [11]. Whereas different projects have been demonstrating various approaches to incorporate text generation during a live session [10, 16, 17], the craft of writing code on-the-fly is likely more important to the live coder. This becomes evident in collaborative live coding sessions, where uncertainty [12] and mixed initiatives [9] are core components of the process. Mistakes are more than welcome, and the attitude is that we are coding for fun [1].

## 4 How to move forward without hitting a wall?

How all abovementioned practices may be applied to co-coding with LLMs? Materiality matters in coding, and unless one wants to just *play code*, using ChatGPT style programming is a recipe for reducing diversity and creativity, alongside the meditative and fun aspects of writing code. When people are live writing code during a performance, the fun aspects are absolutely important. Current practices of co-coding are, to the best of my knowledge, single-initiative practices. We can certainly experiment more on mixed-initiative interactions. Other aspects related to deskilling have to be eliminated when designing for co-coding. It is unnecessary, and maybe boring, to read through the full response of a prompt when you are asking a trivial question. Also, the printout typing speed of an LLM may be a factor for increasing engagement, although it likely depends on user's reading skills. We must think outside of the box to find compelling designs, and that depends on experimentation and practice. Designing LLMs for co-coding should be compassionate to the effort a programmer brings to the table, as the effort put in is rewarding in a live coding context.

Here, I present a list of ideas for further experimentation, adhering to a material-centered design approach:

- Paper printout for every prompt

- on the same page so that it becomes unreadable
- on different pages and organized as Blackwell’s wiki
- A bracelet that vibrates for every character on the LLM’s output
- Git commit on every compile cycle to be send to LLM
- LLM’s denial to respond
- If copy/paste is enabled then typing is not allowed. If typing is enabled then copy/paste is not allowed <sup>1</sup>
- Plain-text printout of every prompt
- One-liner print out of every prompt
- Mechanically activated keyboard that types the prompts output
- LLM to send you email of every prompt
- LLM to make post on your social network (FB, Twitter, Instagram)
- Co-coding without asking to accept suggested changes

## 5 Conclusion

Digital writing has increased the amount of text production, while reducing the effort of production. Since the advent of LLMs we notice an acceleration in the amount of generated text. We witness an era where a skillful craft evaporates to a decision-making compliance issue. Given that writing systems enabled the flourishing of arts and science [5] it is important to rethink the design of text interfaces, so that are not capitalized and driven by a motivation to increase sales and satisfy the user. Deceleration is likely not a solution, instead we need to find ways to engage the users in a skillful and effortful manner.

## References

- [1] Joanne Louise Armitage. 2018. Spaces to fail in: negotiating gender, community and Technology in Algarve. *Dancecult: Journal of Electronic Dance Music Culture* 10, 1 (2018).
- [2] Alan Blackwell, Geoff Cox, and Sang Won Lee. 2016. Writing the Live Coding Book. In *International Conference on Live Coding 2016*. International Conference on Live Coding.
- [3] Alan F Blackwell, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. 2022. *Live coding: a user’s manual*. MIT Press.
- [4] Nick Collins, Alex McLean, Julian Rohrerhuber, and Adrian Ward. 2003. Live coding in laptop performance. *Organised sound* 8, 3 (2003), 321–330.
- [5] Peter T Daniels and William Bright. 1996. *The world’s writing systems*. Oxford University Press.
- [6] Alberto De Campo. 2014. 6.7 Republic: Collaborative Live Coding 2003–2013. *Collaboration and learning through live coding* (2014), 152.
- [7] Eric Deibel. 2024. The Life Sciences and the Future Imperfect. (2024). doi:10.5281/zenodo.14500975
- [8] Anil R Doshi and Oliver P Hauser. 2023. Generative artificial intelligence enhances creativity but reduces the diversity of novel content. *arXiv preprint arXiv:2312.00506* (2023).
- [9] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 159–166.
- [10] Luis Navarro and David Ogborn. 2017. Cacharpo: Co-performing Cumbia Sonidera with Deep Abstractions. In *Proceedings of the International Conference on Live Coding*.
- [11] Iván Paz. 2025. Grammars of Live and Death: Resistance, Emergence and Community. doi:10.5281/zenodo.15526903
- [12] Julian Rohrerhuber and Alberto de Campo. 2004. Waiting and uncertainty in computer music networks. In *ICMC*.
- [13] Hartmut Rosa. 2003. Social acceleration: ethical and political consequences of a desynchronized high-speed society. *Constellations: An International Journal of Critical & Democratic Theory* 10, 1 (2003).
- [14] Advait Sarkar. 2024. Intention Is All You Need. *arXiv preprint arXiv:2410.18851* (2024).
- [15] Mikael Wiberg. 2018. *The materiality of interaction: Notes on the materials of interaction design*. MIT press.
- [16] Elizabeth Wilson, Gy  srgy Fazekas, and Geraint Wiggins. 2024. Tidal MerzA: Combining affective modelling and autonomous code generation through Reinforcement Learning. *arXiv preprint arXiv:2409.07918* (2024).
- [17] E Wilson, S Lawson, A McLean, J Stewart, et al. 2021. Autonomous Creation of Musical Pattern from Types and Models in Live Coding. (2021).

<sup>1</sup>This is a feature of <https://nudel.cc> and was presented by Lu Wilson in the ICLC2025 keynote.