Tools

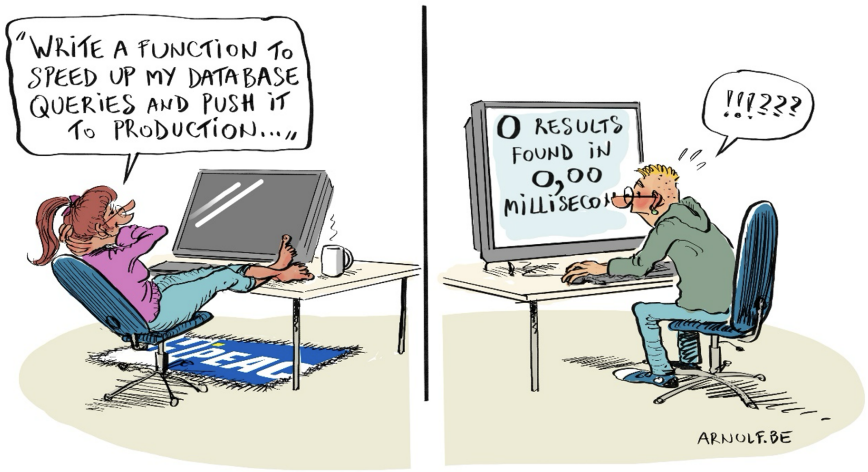
Recommendations for tools

Promote the use of AI in software development

Research, prototype and deploy AI-assisted software development environments, while implementing robust measures to ensure correctness, safety, security, confidentiality, and regulatory compliance. This will help balance the rapid adoption of AI with the need for secure and reliable systems. It should also help non specialists to be able to create efficient software and increase the productivity of developers.

Promote the use of AI in hardware development

Research, prototype and deploy open AI assistants for hardware development, increasing the productivity for designing new, efficient hardware and decreasing the time to market. This is a key element for Europe to stay in the hardware race. The use of AI should be a collaboration between humans and AI systems, as promoted in previous HiPEAC Visions as "centaur"' teams. The focus should be on domains that are still open, like architecture search and exploration, rather than on optimizing the floor-planning, which is already covered by various companies.



Background

The history of hardware and software development has been a progression toward specifying more what is to be done and less how it is to be done – a move from implementation detail towards higher abstraction. The primary factors that have fuelled that

progression have been improvements in processor speed and compiler optimization. It appears that the next technical advance which can drive a new discontinuity in this progression is AI-based hardware and software development tools.

These tools are revolutionary due to their ability to create hardware or software from a natural language description without any human intervention. If the dream to automate the entire development process becomes reality, it would democratize software, allowing anyone to create state-of-the-art software, while potentially eliminating many hardware and software developer jobs. While a similar impact may be seen across many sectors, AI-based tools for hardware and software are distinct from AI in other contexts, due to the strict need for correctness and security, the complexity of integrated co-design hardware/software systems and the limited training data for hardware design and new technologies.

AI-based tools have the potential to disrupt software and hardware development, and missing out on this discontinuity could leave Europe hopelessly behind. The NCP takes for granted the ability for the user to orchestrate and create new software capabilities that would have traditionally required custom software development. These AI tools also streamline the development process, reducing the time and cost to develop the NCP itself. For Europe to successfully lead the NCP, it must have access to the latest technologies, which depends on its universities, research centres and companies being up to date with the forefront of advances in the AI revolution.

State of the art

According to the 2024 Stack Overflow Developer Survey `[StackOverflow2024]`, 62% of software developers were already using AI tools, with an additional 14% planning to adopt them soon. As of the time of writing, these tools can generate functional code from a natural language description, spot likely errors (off-by-one errors or usual code patterns), suggest and apply refactoring, estimate computational complexity, and so on. They leverage vast training data and an understanding of patterns, semantics and context, and are much more powerful and tolerant of ambiguity than earlier tools such as syntax-directed parsers. They can help modernize code to a new environment (e.g. language, major API revision, certified OS), and also generate documentation and test cases, helping maintainability and team onboarding. For a more detailed survey of these capabilities, see `[Metzger]` [KordonZaourar].

GitHub Copilot `[GitHubCopilot]` is a state-of-the art AI-powered coding assistant, which integrates OpenAI's Codex model `[OpenAICodex]` into Microsoft's development environments such as Visual Studio Code and GitHub. It provides developers with real-time code suggestions, completions, and contextual guidance, across a wide range of programming languages and frameworks, streamlining tasks from boilerplate generation to debugging. By analysing surrounding code and comments, it predicts and generates relevant code snippets, enabling faster development and reducing repetitive tasks. Microsoft has positioned Copilot as not just a tool for writing code but as an intelligent collaborator that enhances productivity, encourages best practices, and lowers the barrier to entry for complex programming tasks. As such, it is a complement to their broader Microsoft Copilot AI companion `[MicrosoftCopilot]`, which is integrated across multiple Microsoft products, including Word, Excel, PowerPoint, Outlook and Teams.

Google's Gemini Code Assist `[GoogleCodeAssist]` is another prominent AI coding assistant, which integrates with integrated development environments (IDEs) such as Visual Studio Code, JetBrains IDE and others, supporting major languages such as Java, JavaScript, Python, C and C++. In December 2024, Google announced Gemini 2.0, which includes Jules, a more powerful but experimental AI-powered coding agent for Python and Javascript, which integrates with developers' GitHub workflows, handling code development such as bug fixes, and preparing pull requests to land fixes directly back into GitHub `[GoogleJules]`.

Meta's Code Llama [CodeLlama] was released in 2024 as an extension of their Llama 2 language model that is fine-tuned for software development across multiple programming languages. Unlike most of the alternatives, the model weights and inference source-code for Code Llama are freely available under Meta's strategy of fostering open innovation in the AI ecosystem [CodeLlamaLicence]. As such, it offers the possibility for fine-tuning and customization.

Overall, AI-driven coding assistants have amassed nearly $1 billion of funding since the start of 2023, with the vast majority, such as Microsoft's Copilot, Google's Gemini, and tools from startups such as Replit and Magic, being controlled by US-based companies [FTAug2024].

Mistral AI, a Paris-based startup, is a notable European success that has made significant strides in the AI ecosystem, releasing several AI language models and raising substantial funding. Their Codestral model [Codestral] has been specialized for code development, and it targets 80+ programming languages, including Python, Java, C, C++, Javascript and Bash. With its context window of 32K tokens, Codestral outperforms other models in RepoBench, a benchmark for code generation.

In the high-performance computing (HPC) space, the LLM4HPC project at Oak Ridge National Laboratory has developed a number of tools for HPC software development [LLM4HPC]. This includes ChatBLAS, an AI-generated BLAS (Basic Linear Algebra Subprograms) library for linear algebra, automatic parallelization with large language models (LLMs), F2XLLM for Fortran modernization, and is developing ChatHPC, an AI assistant for HPC programmers.

There are also several efforts to develop AI-based tools and platforms to assist with hardware design, although most are either proprietary or not widely available (see [ALSAQER] for a recent survey). ChipNeMo [ChipNeMo] is an LLM developed by NVIDIA, specifically tailored for the semiconductor industry. By employing domain adaptation techniques—such as custom tokenization, domain-specific pretraining, and supervised fine-tuning—ChipNeMo enhances performance in chip design tasks. It excels in applications like engineering assistant chatbots, electronic design automation (EDA) script generation, and bug summarization and analysis, often surpassing general-purpose models. As of now, ChipNeMo is not publicly available. NVIDIA has detailed its development and capabilities in research publications, but the model itself remains proprietary and is not accessible for public use.

Other important activities include ChipGPT [ChipGPT], which generates and optimizes Verilog code from a natural language specification. ChatEDA [ChatEDA] is an AI-based assistant that helps engineers orchestrate a complex EDA workflow using natural language. Additionally, LLMs have been employed to assist in the writing of architecture specifications (e.g. SpecLLM [SpecLLM]) and to explain error messages from synthesis tools [Qiu24].

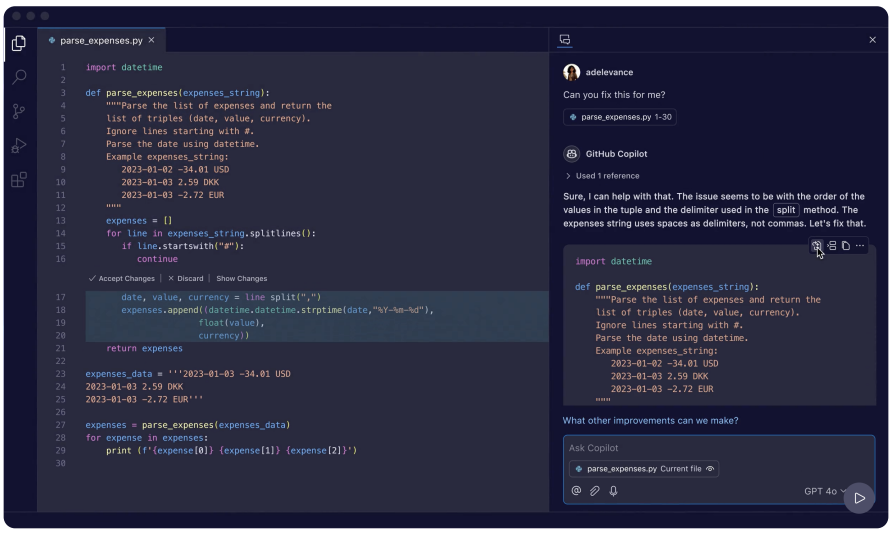


Figure 1: GitHub Copilot, a state-of-the-art AI-powered coding assistant [GitHubCopilot]

Explore the use of AI-based tools to support software and hardware development, but insist on measures to ensure correctness, safety, security, confidentiality and compliance.

The biggest barrier and risk associated with generative AI in development is the inability to fully trust the code it produces. Like people, these tools are prone to confabulation (or “hallucination”), generating incorrect or misleading outputs. Given the opaque nature of AI-based models and likely lack of access to their training data, AI-generated code, whether hardware or software, must be seen as unsafe and insecure. This poses a significant challenge in the context of the NCP, where AI-generated software would be expected to interact with the internet and influence the real world, all without human oversight. Another risk is confidentiality, particularly with online tools provided by third parties, as well as shared tools within an organization that has access to confidential third-party software.

As discussed below, Europe should invest in basic research (including formal methods) and enact sovereignty measures to address these risks. In the meantime, however, development processes must be set up for careful human reviews of AI-generated code, just like code fully written by a human programmer.

Use a combination of LLMs and traditional tools, with the LLM as the user interface and driver to orchestrate

It is likely that all aspects of code development, debugging, optimization and maintenance will shift to using a natural language as the bridge between the human and machine.

The more interesting question is where to place the interface between the LLM and lower levels of the hardware–software stack. Programming languages have traditionally been designed and updated with the expectation that most code will be written by humans. An exception is assembly languages, which were initially written by humans, but for decades have been designed to be targeted by a compiler. In general, programming languages are created to address a specific need, often tailored to an application domain, hardware architecture and performance requirements. Overall, they end up being created to solve a real practical problem created by application/user needs and/or platform capabilities, and they find a corresponding trade-off among many factors, including abstraction,

expressiveness, simplicity, understandability, maintainability, safety, reliability, efficiency, support for parallelism, scalability for large codebases, portability, and support for a robust ecosystem of tools and libraries.

It remains uncertain which languages should be targeted by the AI-based tool, and whether the ambitious vision of an AI-based model directly transforming "natural language to transistors or machine code" will ever be feasible, especially for current million-line plus codebases. Achieving this vision is likely to face significant challenges related to energy efficiency (of the AI-based system generating the code), scalability (to ever larger codebases), precision (resolving the inherent ambiguities of natural language), and understandability (to support human and/or machine verification and facilitate troubleshooting). These challenges suggest that one or more levels of abstraction between natural language and machine-level code will remain necessary. Abstraction not only helps mitigate ambiguity and complexity but also provides modularity and structure, essential for debugging, optimization, and the efficient generation of scalable systems.

Hardware and software development depends on various auxiliary tools, such as simulation, model checking and timing analysis tools (for hardware), debuggers (for software), as well as performance and energy analysis tools, verification, static analysis and code coverage tools. Human intuition and creativity will increasingly be replaced with AI-based tools, but traditional optimization algorithms are extremely powerful and should continue to have a place at the lowest level. These tools often have idiosyncratic interfaces, and they are hindered by the multiple levels of abstraction between the machine and the high-level code, that may need to be traversed to understand what has gone wrong. The key is to operate at the right level of abstraction to solve the issue, as high as possible, while being able to drop to the lowest levels where needed. This presents a significant opportunity for AI-driven tools to drive developer tool use through natural language interaction, automate tools integration within a larger AI controlled workflow, and translate cryptic error messages into higher-level code suggestions.

Support a European ecosystem that includes basic research in AI

Europe's universities, research centres and companies must be at the forefront of basic research in AI, pursuing important research topics such as the following:

- Correctness, safety and security. As discussed above, this is the greatest barrier to the adoption of AI. Formal methods can be used to prove correctness and security properties (see for example `[GoogleAlphaIMO]`), but they are cumbersome for large systems and should be the subject of basic research.
- Programming languages and abstractions.As discussed above, it is not clear how programming languages should evolve as they are increasingly targeted by AI-based tools. It is unclear whether the choice of abstractions should mirror those designed for human developers or be created specifically to exploit the strengths of generative AI methods, whatever that entails. A key issue will be the lack of training data for any new programming language or language features.
- Open-ended problems. For hardware design, AI-based tools can be given an open-ended problem, such as. "design a CPU that executes these programs, as fast as possible, given this transistor/power budget". This problem includes design space exploration but is much broader in scope, as it is not constrained by parameters defined ahead of time by people.
- Optimization of neural networks.In addition, the increasing and tremendous complexity of neural networks, present in all machine-learning applications, will require more and more reliance on automated AI-based tools to help design efficient solutions and master their huge complexity. These tools will need to exploit multi-

criteria optimization methods and to generate optimized code for a given hardware, in order to take into account the numerous embedded constraints that it must guarantee. These constraints can cover the induced power, the memory size, the prediction accuracy or for instance the type of operations used to remain compatible with the final hardware. The supported hardware must be compatible with the latest innovations and computing trends, including for instance heterogeneous system-on-chips (SoCs) with dedicated neural networks accelerators. The output of these AI-based tools, based on neural architecture search (NAS) methods, should be able to design optimized and frugal AI applications, for all AI applications using LLMs, transformers or Mamba algorithms. These tools will also have to integrate trustable and explainable methods to bring to the user the knowledge used by the tools to obtain the final results, in order to integrate critical embedded systems.

Develop European agents, tools and infrastructure

In today's geopolitical climate, European sovereignty over its AI models is crucial, especially as AI-based technologies increasingly influences national security, economic competition and social governance. AI-based tools for hardware and software development stand out from general AI due to the foundational role they can play in building and shaping future technology, as well as their role in innovation and competitive advantage.

AI development tools will serve as the backbone of the digital economy, facilitating the creation of chips, communication networks, cloud infrastructures, middleware, and applications that support all other AI-based applications, from autonomous vehicles to smart cities. If Europe lags behind in this area, it will become dependent on foreign suppliers whose interests may not align with European priorities. In a worst-case scenario, this dependency could lead to hardware and software being compromised or containing hidden backdoors, creating significant national security risks.

The race to build AI development tools is, in essence, a competition for leadership in the global tech economy. European countries must have access to the most advanced tools and be able to influence their development, in order to compete with global giants from the US and China, and help Europe to remain a leading force in key industries such as automotive manufacturing, telecommunications and fintech.

At the same time, Europe is recognized for its strong commitment to ethics, legal, socioeconomic and cultural aspects of the use of AI-based technologies and its unique regulatory frameworks. Some global companies have already opted to withhold support for their most advanced AI rather than adjusting to European regulations. If this trend continues and worsens, especially in times of geopolitical tension, it could stifle economic competitiveness. In the worst case, there will be significant pressure to undermine European ethics.

Focus on education, training and jobs

As of 2025, AI tools can fully automate the creation of simple code, consisting of a few hundred lines, and they are powerful assistants to human developers in full-scale development projects. However, as described so far, these tools are cannot yet replace proficient and experienced developers. As these tools advance, important questions arise about the future of the workforce in the hardware and software industries, which currently employ millions of people globally.

Over the next few years, AI tools are likely to continue to assist developers, particularly in routine and repetitive tasks, freeing developers to focus on higher-level design and problem-solving. In this period, many routine tasks will be automated, leading to a shift in work for

developers. Entry-level positions may be affected, but mid-level and senior developers will still be in high demand to oversee complex projects, integrate AI-generated code, and ensure quality and creativity in the final product. This will place greater and distinct demands on education, which may be alleviated by individualized AI-based training helping to make programming more fun and learnable by people at a younger age.

As of 2025, at the height of the hype curve for AI, it is important to maintain a historical perspective. In 1954, IBM's Fortran specification claimed that "[s] ince FORTRAN should virtually eliminate coding and debugging, it should be possible to solve problems for less than half the cost that would be required without such a system" [FORmula]. Similar claims were made in the 1980s, for fourth-generation languages, such as SQL, ABAP and COBOL 85. While these technologies did reduce development cost and time (by much more than half), the belief that they would eliminate the need for software developers was wildly optimistic. In practice, the necessary skills moved from assembly language coding to the wide class of skills needed for large scale software development today.

Nevertheless, while history is a guide, it is not guaranteed to repeat. In the long term, AI tools may evolve to the point where they can build increasingly complex systems autonomously. Will AIs be able to replace a team of human developers, with a human taking on the role of a chief architect or chief technology officer interacting with AI? What happens when something goes wrong? At this point we do not know.

Conclusion

In conclusion, the integration of AI tools into hardware and software development offers transformative potential, and it has the potential to inject a major discontinuity into the development process. By utilizing natural language interfaces and leveraging AI's capabilities, development processes can become more efficient, reducing time and costs, while also democratizing access to advanced technologies. However, the risks associated with AI-generated outputs, such as safety, correctness, security, and confidentiality, must not be overlooked. Europe must prioritize basic research in AI, develop its own AI tools and models, and ensure that AI's role in development remains aligned with ethical, regulatory, and security standards. Furthermore, as AI tools evolve, the future workforce will need to adapt, with AI serving as a powerful assistant to human developers rather than a complete replacement. The success of Europe in this rapidly advancing field will depend on fostering a robust AI ecosystem, ensuring technological sovereignty, and investing in education and training for the next generation of developers.

References

ALSAQER: Shadan Alsaqer, Sarah Alajmi, Imtiaz Ahmad, Mohammad Alfailakawi, "The potential of LLMs in hardware design", Journal of Engineering Research, 2024. ISSN 2307-1877. https://doi.org/10.1016/j.jer.2024.08.001

ChatEDA: Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, B. Yu, "ChatEDA: A large language model powered autonomous agent for EDA", In: 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD), IEEE, 2023, 1-6.

ChipGPT: K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, X. Li, "ChipGPT: How far are we from natural language hardware design", arXiv preprint arXiv: 2305.14019 (2023).

ChipNeMo: https://research.nvidia.com/publication/2023-10_chipnemo-domain-adapted-llms-chip-design

CodeLlama: https://ai.meta.com/blog/code-llama-large-language-model-coding/

CodeLlamaLicence: https://github.com/facebookresearch/llama/blob/main/LICENSE

Codestral: https://mistral.ai/news/codestral/

FORmula: Preliminary Report. Specifications for The IBM Mathematical FORmula TRANslating System. 1954. https://www.softwarepreservation.org/projects/FORTRAN/BackusEtAl-Preliminary%20Report-1954.pdf

FTAug2024: https://www.ft.com/content/4868bd38-613c-4fa9-ba9d-1ed8fa8a40c8?utm_source=chatgpt.com

GitHubCopilot: https://github.com/features/copilot

GoogleAlphaIMO: Google's AlphaProof and AlphaGeometry use the Lean theorem prover to check their solution to problems from the International Mathematics Olympiad (IMO) https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/

GoogleCodeAssist: https://cloud.google.com/products/gemini/code-assist

GoogleJules: https://developers.googleblog.com/en/the-next-chapter-of-the-gemini-era-for-developers/

LLM4HPC: Pedro Valera–Lara. LLM4HPC: Towards an AI-autonomous HPC world. https://www.hpcuserforum.com/wp-content/uploads/2024/10/Pedro-Valero-Lara-ORNL_LLM4HPC-Towards-an-AI-autonomous-HPC-World_HPC-UF-BSC-Oct-2024.pdf

Metzger: Metzger, A. (2024). "AI-Assisted Software Engineering (AISE)". HiPEAC Vision 2024, Rationale. https://doi.org/10.5281/zenodo.10874754[KordonZaourar

MicrosoftCopilot: https://copilot.microsoft.com/

OpenAICodex: https://openai.com/index/openai-codex/

Qiu24: S. Qiu, B. Tan, H. Pearce, "Explaining EDA synthesis errors with LLM", arXiv preprint arXiv: 2404.07235 (2024).

SpecLLM: M. Li, W. Fang, Q. Zhang, Z. Xie, "SpecLLM: Exploring generation and review of VLSI design specification with Large Language Model", arXiv preprint https://arxiv.org/abs/2401.13266 (2024).

StackOverflow2024: https://survey.stackoverflow.co/2024/ai#sentiment-and-usage-ai-sel-prof