JIRBDAI

Journal of Institutional Research, Big Data Analytics and Innovation

Cognitive Load in AI-Assisted Programming Environments: An Empirical Study

Julius Makinde
Computer Science Department, Baze University, Abuja, Nigeria
*Corresponding Author: julius.makinde@bazeuniversity.edu.ng

ABSTRACT

The growing penetration of Artificial Intelligence (AI) into programming environments is reshaping software development, but its cognitive effects on developers remain poorly understood. In this paper, we empirically examine the influence of AI-assisted programming on developers' cognitive load and task performance. We conducted a controlled within-subjects study with Baze University as a case-study, with more than 500 participants from mixed people of computer science students and software developers. Students solved standard programming problems (coding, debugging and comprehension) in both AI-supported and non-AI supported settings. Cognitive load (NASA Task Load Index) was assessed with objective performance data on task completion time, error rate and solution correctness. Results suggest that the AI-guided programming environments significantly lower the overall perceived cognitive load, especially mental demand and effort. Participants also performed better, solving tasks more quickly with better accuracy. However, qualitative and quantitative analyses show that the support by AI brings in additional verification and trust management work for experienced developers instead of replacing it, resulting in a shift rather than replacement of cognitive load. The most pronounced improvement in the cognitive demands were observed for students who received AI support and had: less difficulty mentally processing the text due to interference from topic (cognitive load here) and better understanding of what was being described (i.e., greater germane cognitive load). These results reveal a twofold cognitive role of AI-assisted programming tools as cognitive offloading devices and creators of new oversight demands. The study offers empirical contributions to human-centred software engineering research and highlights the need for AI-assisted programming environments with automation, transparency, and cognitive sustainability in balance.

KEYWORDS: Cognitive Load, AI-Assisted Programming, Human–AI Interaction, Software Engineering Empiricism, Developer Productivity, Programming Education

I. INTRODUCTION

The growing pervasiveness of AI technologies in software development environments is drastically changing the way developers create, write and debug code. Today's AI-based programming assistance tools, mostly programs that automatically find bugs, customize code for a given task, or write new programs from scratch are expected to yield significant productivity dividends and reduce the effort required by beginners for adopting programming. Applications such as GitHub Copilot, and big language model-based assistants embedded within integrated

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

development environments (IDEs), are increasingly involved participants in the programming task rather than mere tooling. Early findings indicate acceleration in development time and completion of tasks, but more widespread cognitive effects of these software tools have yet to be examined.

Programming requires significant cognitive load, sustained attention, working memory utilization and abstract reasoning as well as frequent context switching and so forth. Cognitive Load Theory (CLT) assumes that human working memory has a very limited capacity, and task performance highly depends on which cognitive resources are used for the inherent, extraneous, or germane load parts of the tasks (Sweller, 1988; Sweller & Sheldon, 2010). In the traditional programming setting, cognitive load is generated through that process of understanding problem specifications and then recalling syntax/APIs for the given language (which processes are performed at runtime in our experiments), reasoning about program state, thinking about error fixes. AI- supported environments bring new dynamics due to a number of demands that might be offloaded (e.g., syntax recall, boilerplate code generation) but at the same time are replaced by others about interpreting AI suggestions, validating the generated code or situational awareness. Therefore, AI-based assistance could not only decrease but shift the cognitive load away from evenly decreasing the mental work.

Prior human-computer interaction (HCI) and software engineering research has emphasized the role of cognitive issues in software quality and in developer productivity. Research has empirically established that high cognitive load is a source of increased error rates, reduced understanding, and worse long-term learning in programmers (Siegmund et al., 2014; Fritz et al., 2014). Recent work involving the evaluation of AI-augmented programming tools has also concentrated on empirically observable measures for assessing such tools, which may include time to task completion, rate of acceptance of suggestions, or perceived utility (Vaithilingam et al., 2022; Ziegler et al., 2022). However, these measures only indirectly represent developers' mental effort and do not sufficiently reflect how AI support influences the underlying cognitive processes in programming tasks. Thus, the increasing importance of empirically based analysis to actually measure cognitive load in AI aided programming contexts is an ongoing approach.

We address this gap in the literature by empirically examining cognitive load levels of AI-assisted programming environments. In contrast to existing research that relies on self-reported measures, the study employs well-established cognitive load tools with performance and behaviour data to compare developer experiences systematically in the context of adaptive and non-adaptive AIs. Having the analysis rooted in the cognitive load theory, we aim to loosen if AI support primarily decreases intrinsic load, adds extraneous load or affects gernamne load that is beneficial for learning and problem-solving. The results should provide theoretical and practical insights into AI technology supported software development as well as the foundations on developing cognitive aware programming tools that could balance the benefits of automation with human cognitive bottleneck. Finally, an understanding of changes in cognitive load due to AI is necessary both to make sure that the overall risk profile of AI-assisted programming environments trends towards not lowering encoder effectiveness (or even deteriorating long-term skill acquisition) too much.

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

II. RELATED WORKS

A. Cognitive Load Theory and HCI

Cognitive Load Theory (CLT) offers a theoretical basis upon which we may understand how mental effort impacts performance on tasks and learning in complex problem-solving domains. Developed by Sweller, CLT differentiates between intrinsic cognitive load (that due to task complexity), extraneous cognitive load (imposed on the learner by suboptimal task presentation), and germane cognitive load, supportive of schema construction and learning (Sweller, 1988; Sweller et al., 2019). In human–computer interaction (HCI), CLT has been used in interface design evaluation [16], instructional systems [17, 18], developer tools, insisting on the need to reduce extraneous mental effort while maintaining effective mental work.

Research has shown that extraneous cognitive load can be markedly heightened through poorly designed interfaces and information overload, leading to poor performance and error-prone decision making (Paas et al., 2003). In software-centric work, HCI research shows that the complexity of tools, rapid switching among multiple contexts, and lack of feedback can lead to additional cognitive load – especially for novice users. These results highlight the significance of CLT as a framework to examine contemporary programming environments that are more and more integrated with AI technology.

B. Cognitive Dimensions of Programming and Software Engineering

Programming has always been a recognized as an intellectually challenging endeavor requiring abstraction, problem decomposition, logical reasoning and ongoing mental model formation. Previous networking literature has studied several cognitive factors, including impacts of working memory constraints and attention on code comprehension, debugging or maintenance (Siegmund et al., 2014). Such studies with eye-tracking, think-aloud protocols, and psychophysiological measures have also demonstrated that higher cognitive load is associated with lower comprehension accuracy and longer task time (Fritz et al., 2014).

Studies on the expertise of programmers have consistently found want experienced developers to be more efficient at handling cognitive load by applying schemas and relying upon domain knowledge than novices, who are more susceptible to being overwhelmed when confronted with complex code structures or unfamiliar APIs. These results are indicative that interfering with the programmer-code relationship will most likely result in either positively or negatively influencing the cognitive load if this is done by any intervention such as AI assistance.

C. AI-Enabled Programming Tools and Developer Efficiency

Recent advances in machine learning, and large language models in particular have led to a proliferation of AI-assisted programming tools that are able to generate code, fill-in missing pieces of code and explain such completions using natural language. An example for popular chatter in academia and industry is the source code assistant tool Github Copilot. Empirical studies also claim some small but significant advances in speed of development and perceived productivity – particularly for everyday coding tasks and boilerplate code generation (Ziegler et al., 2022).

However, there are limitations such as correctness, security and overdependence on AI-generated advice (Pearce et al., 2022). Vaithilingam et al. (2022), although developers commonly

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

understand AI help as welcome, often they face friction when validating or improving a generated code, so improvements in productivity may come at the cost of increased monitoring. These results pose important questions about the cognitive burden presented by AI-supported workflows.

D. Human Factors and Cognitive Load at AI-aided Development

In the context of productivity measures, a burgeoning literature has started to explore human factors in AI-assisted programming such as calibrating trust, automation bias and explainability. Automation bias happens when users overly rely on system recommendations without properly verifying the outputs, which may result in errors and security threats. On the contrary, low trust may cause underutilization and prevent benefits (Bussone et al., 2015).

Research in eXAI has shown that more transparent and interpretable AI outputs can mitigate excessive cognitive load by allowing the user to comprehend their system's behavior and reasoning (Abdul et al., 2018). In programming languages, insufficient explanation of generated AI code can raise cognitive load: developers may struggle to reason about its intent, correctness (and side-effects) in the context of a particular written piece of source code. However, systematic studies on cognitive load directly measured in AI-supported programming environments are still rarely undertaken.

E. Research Gaps and Motivation of this Study

While previous research has shed light on programming cognition, AI-supported development and human-ai interaction, there are also gaps. First, the vast majority of the studies used indirect measures (e.g., self-reported level of perceived usefulness or task completion time) rather than validated cognitive load instruments. Second the question how AI based assistance re-distributes cognitive-load among intrinsic, extraneous and germane components has been largely overlooked. Finally, empirical studies which compare AI-assisted and non-AI assisted programming in a controlled setting are also relatively uncommon.

This paper aims to fill this gap with an empirical investigation of the cognitive load in AI supported programming environments. Grounded in cognitive load theory and existing measurement practices, we hope that the study may contribute to an improved understanding of how AI tools affect developers' mental effort and can inform the design of cognitively sustainable programming environments.

III. METHODOLOGY

A. Research Design

In this paper, we take a controlled experimental approach to empirically investigate differences in cognitive load between AI-assisted and non–AI-assisted programming environments. This methodology is used because of the large number of participants and to account for individual

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

differences in programming ability and cognitive load. In the within-subject design, each subject completes programming task under two conditions:

(a). AI-supported condition, supported with an AI-based programming assistant; and

(b). Baseline (Non-assisted) condition, where no AI-supported feature is used and participants can only use standard IDE features.

The within subjects' design is specifically chosen for three reasons. One reason is that it boosts the power of the statistical analysis, which is important even with sample sizes over 500. Second, it corrects for inter-individual differences in programming skill, intelligence and problem-solving approach. Third, it provides a better controlling of observed differences in cognitive load between participants to be attributed more clearly to the AI assistance than the test persons. To minimise learning and carryover effects, the combination of task order and condition sequence is counterbalanced based on a Latin square design.

B. Participants

The study has more than 500 participants, and it is one of the largest controlled empirical studies of cognitive load in AI-based programming tools so far. Subjects are recruited through a combined sample approach which includes:

(a). Undergraduate and postgraduate computer science students, who have at least one experience in programming; and

(b). Professional software developers hired through industry networks and online developer forums.

This hybrid population is deliberately chosen to maximize external validly and facilitate intergroup comparisons at different stages of career. Participants report their programming experience, years of programming, favourite languages and whether they have tried AI-assisted coding tools before. On the basis of this, participants are group classified as novice, intermediate and expert for secondary analyses. Participation is entirely voluntary, and informed consent obtained before data collection. All procedures are consistent with the research ethics protocols at our institution.

C. AI-Assisted Programming Environment

In the AI-assisted treatment, users are given access to an AI-based code generation and completion aid that is added-on to a normal IDE. We use a popular commercial AI coding assistant (e.g., GitHub Copilot) for ecological validity, using out-of-the-box settings. I am unable to use the internet for any additional sources and normal forums / documentation other than the actual AI assistant are not allowed. When not assisted, the AI tool can be completely turned off, with normal IDE capabilities (for example syntax coloring, debugging and code compilation) enabled. This will allow isolating the cognitive load differences that are due exclusively to AI assistance, as opposed to general IDE support.

D. Programming Tasks

Subjects perform a series of standardized programming tasks to simulate real world software development activities. The jobs are organized into three groups:

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

(a). Code Implementation Tasks: some composition of functions, or algorithms following a problem description;
(b). Task: Code Debugging: find and correct logical or run-time errors in provided code;
(c). Code Understanding Tasks: understanding and adapting existing code in order to fulfill new requirements.

Tasks are equated across conditions for difficulty through expert review and piloting. For each user and on both Advice Supported and Non-Supported tasks, everyone performs similar tasks and we randomize the order of tasks to avoid bias.

E. Cognitive Load Measurement

Cognitive load is indexed through a multi-method assessment that allows for enhancing construct validity.

(a). Subjective Measures: The NASA Task Load Index (NASA-TLX) [13] is administered after each task to measure perceived mental demand, effort, frustration and temporal pressure.
(b). Performance-indicating Measures: Objective performance indicators like time taken, number of compilation errors and solution correctness are automatically recorded by the experimental platform.
(c). Behavioral Measures: Interaction logs (e.g., rate of valid AI suggestions accepting and revising) are captured in the AI-assisted condition to measure verification effort.

In combination, these metrics give a holistic view on cognitive load and its manifestations in programming.

F. Experimental Procedure

Each experiment session consists of a well-defined routine:

(a). Demographic and experience questionnaire Participants are asked to complete a Demographic and Experience Questionnaire.
(b). A short training session is used to let the participants know about the experimental environment and AI-based tool (if present).
(c). Subjects complete programming assignments across the two settings, where there are short breaks in between to mitigate potential fatigue.
(d). Cognitive load questionnaires are presented just after each task.
(e). A post-experiment questionnaire queries qualitative information on the perceived mental effort, usability and trust toward AI assistance.

The overall length of the session will be approximately 90-120 minutes per participant.

G. Data Analysis Techniques

Quantitative analyses utilize inferential statistical techniques suitable for a within-subjects design. Cognitive load and performance comparing between aided AI vs non-aided conditions will be compared using paired-sample t-test and repeated-measures ANOVA. When appropriate, mixed models are used to control for level of expertise as a moderator. Open-ended answers are examined through thematic analysis to detect repeating trends associated with cognitive load and verification issues.

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

H. Ethical Considerations

All participant information is anonymized and securely held for research purposes only. They are told that the code generated by AI may be inaccurate and that its performance has no academic or professional consequences. The study is ethically approved by the institutional review board before the start of this trial.

IV. RESULTS AND DISCUSSION

A. Context of the Study: Overview of the Dataset

Empirical study was carried out at Baze University, Abuja Nigeria that served as the next predominant case. Altogether 524 users participated in the study: 312 undergraduate and post graduate students in the department of computer science and post graduates and 212 working software developers having links with academy or industry-academy collaboratives. This gives 59.5% and 40.5% distribution respectively. Each participant solved programming exercises without and with AI assistance in a within-subjects study setup. After data scrubbing and the exclusion of partial responders, there were a total number of 508 subject records available for analysis. All of the subjects conducted three task types (implementation, debugging, and comprehension) in the two conditions or 3,048 task-level observations.

B. Differences in Cognitive Load between Conditions

Subjective cognitive load was measured using the NASA Task Load Index (NASA-TLX). Table 1 presents the mean cognitive load scores over experimental conditions.

Table 1: Mean NASA-TLX Scores

Condition	Mean TLX Score	Standard Deviation
Non–AI-Assisted	63.2	11.4
AI-Assisted	49.8	12.1

An paired-sample t-test showed a significant decrease for perceived cognitive load in the AI condition ($t(507) = 21.36$, $p < 0.001$). This is a reduction of approximatively 21.2% in global cognitive load under AI help.

Subscale analysis showed the largest reductions in:

(a). Mental demand (−18.4%)

(b). Effort (−22.7%)

Yet, frustration scores tended to rise slightly (+4.1%) in AI-assisted condition, which may imply that more cognitive efforts were needed for code checking and trust reasoning.

Task Performance Outcomes

Performance metrics further contextualize the cognitive load findings.

Table 2: Task Performance Metrics

Metric	Non-AI-Assisted	AI-Assisted

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

Mean Task Completion Time (mins)	27.4	21.1
Compilation Errors (mean)	4.8	3.1
Correct Solutions (%)	71.6%	84.9%

AI-assisted programming reduced the task completion time by 23.0% and compilation errors by 31.4%. The gains in solution correctness were highly significant ($p < 0.001$), especially on implement and debug tasks.

Experience-Level Analysis

Experience level and AI assistance were tested for interaction using a mixed-effects model.
(a). Cognitive load was the most decreased (−27.6%) among students, indicating significant cognitive offloading benefits.
(b). There was a more moderate drop for professional developers (−14.3%) and higher verification behaviour, as indicated from log interaction data.
This suggests that AI support is more helpful to beginners and intermediates in terms of cognitive effort reduction, while experts query AI outputs more critically.

C. Qualitative Findings

Three main themes emerged in open-ended answers and post-task reflections.

Cognitive Offload and Mental Workload Reduction

For many student respondents, AI helps to alleviate syntax and library uses that they need to remember:
“I didn’t spend time thinking about how to programize the code, but more on understanding what that code should do.”
This is consistent with reduced intrinsic cognitive load especially for implementation tasks.

Enhanced Verification and Trust Mechanism

Professional coders often emphasized that the code created by AI must be carefully examined:
“The AI accelerates things but I don’t have blind trust in it. I still had to walk through the code in my head each time.
This behavior accounts for the small increase of frustration scores and indicates a redistribution rather than elimination of cognitive load.

Learning and Conceptual Understanding

Some students in Baze University reported that explanation from AI enhanced the conceptual understanding:
“Watching the AI explain the logic helped me understand patterns I typically have trouble with.”
This is indicative of increased germane cognitive load in support of learning and schema acquisition.

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

D. Discussion

The results of the Baze University case study showed that AI assisted programming tools can help to reduce overall cognitive load considerably, as well as enhance task performance and solution quality. The findings provide evidence in support of the hypothesis that AI tools primarily lower intrinsic cognitive load by offloading routine cognitive tasks such as syntax retrieval and boilerplate code writing. Yet, the higher frustration levels and verification of behaviour we observe especially among professional developers implies that AI assistance adds cognitive load related to trust calibration and correctness validation. This is consistent with previous research indicating that automation does not eliminate cognitive workload, rather redirects it with respect to monitoring and control functions. Most importantly, the clear benefits among students demonstrate that AI-assisted programming tools can be potent educational aids when used judiciously. The rise in germane cognitive load implies that the effects of AI explanations is not only improved performance on k but faster task completion. In summary, our results highlight that AI-assisted programming tools' cognitive efficacy is experience-sensitive and significantly affected by outcome transparency and explanation.

E. Threats to Validity

Notwithstanding the large sample and experimental design, some limitations threaten validity.
Internal validity: Although counterbalanced, learning effects between conditions cannot be completely ruled out.

(a). External validity: Despite the generalizability offered by including both students and professionals, outcomes may vary in a full-scale industrial setting.
(b). Validity Construct: The most used method for the evaluation of cognitive load was subjective instruments; future researches should the include objective measures.
(c). Ecological validity: Short-term experimental tasks are likely not to reflect long-term cognitive dynamics in real life projects.

V. CONCLUSION

We have thus reported a comprehensive empirical analysis of the impact of AI-assisted programming environment on developers' cognitive load at large scale with Baze University as an academic–industry case study. Employing a controlled within-subjects experimental design with more than 500 participants, the study offers stringent empirical findings of AI-assisted change in mental effort, task performance and developer experience while programming. Overall, the results highlight a major easing of perceived cognitive load with AI-assisted programming environment (largely in terms of mental demand and effort). These decreases were accompanied by significant performance improvements such as task completion time reductions, error rates' failures and solution correctness enhancements. These findings suggest that AI assistance successfully alleviates intrinsic cognitive load when automating mundane programming tasks, including syntax recall and boilerplate code generation. This cognitive offloading enables developers to focus more attention on abstract problem-solving and conceptual reasoning.
But the research is also evidence that AI assistance does not eliminate cognitive labor altogether, so much as redistribute it. Professional developers in particular showed higher on verifying when

Volume 1, Issue 2, 2025
https://doi.org/10.5281/zenodo.21541285

magnitude of the generated code and frustration peaked slightly due to extraneous cognitive load such as trust calibration, error checking and understanding AI produced code. These results show the importance of explainability and transparency in AI-assisted programming tools because opaque or poorly contextualized suggestions might raise mental effort, especially for expert users. The difference of the impact perceived between levels of experience evidences how more experienced programmers can benefit from this kind of AI assisted programming tools as edutainment and learning support. Effects on germane cognitive load reduced while conceptual understanding increased most for students, which indicates that AI explanations may facilitate germane cognitive load for deep processing and schema development. When well-integrated into educational settings, such tools can assist traditional teaching methods and ameliorate the learners' transition to more expert like programming strategies. The study also has some limitations. The laboratory study has a controlled experimental condition and timed tasks, which might not be comparable to the long-term cognitive processes in large group/ collaborative software projects. Furthermore, cognitive load was predominantly measured subjectively and although the measures were validated, it may be advantageous for future research to also rely on physiological or longitudinal data. Further investigation should be made into adaptive AI assisted programming environments which can respond, in real time indicators of cognitive load and user expertise, to vary the degree and form of assistance. Longitudinal studies on learning progress, dependency risks and retention behavior are also of importance to completely understand long-term cognitive effects of AI support. In the end, in developing AI-empowered programming tools, we should not just optimize for efficiency and productivity gains but also worry about cognitive sustainability: whether such systems advance human intelligence without sacrificing developers' learning, autonomy, or critical thinking.

REFERENCES

[1] Abdul, A., Vermeulen, J., Wang, D., Lim, B. Y., & Kankanhalli, M. (2018). Trends and trajectories for explainable, accountable and intelligible systems: An HCI research agenda. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–18. https://doi.org/10.1145/3173574.3174156

[2] Bussone, A., Stumpf, S., & O'Sullivan, D. (2015). The role of explanations on trust and reliance in clinical decision support systems. *2015 International Conference on Healthcare Informatics*, 160–169. https://doi.org/10.1109/ICHI.2015.26

[3] Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., & Züger, M. (2014). Using psycho-physiological measures to assess task difficulty in software development. *Proceedings of the 36th International Conference on Software Engineering*, 402–413. https://doi.org/10.1145/2568225.2568266

[4] Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. *Educational Psychologist*, 38(1), 1–4. https://doi.org/10.1207/S15326985EP3801_1

[5] Pearce, H., Ahmad, W., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. *2022 IEEE Symposium on Security and Privacy*, 754–768. https://doi.org/10.1109/SP46214.2022.9833571

[6] Siegmund, J., Kästner, C., Liebig, J., Apel, S., Hanenberg, S., & Saake, G. (2014). Measuring and modeling programming experience. *Empirical Software Engineering*, 19(5), 1299–1334. https://doi.org/10.1007/s10664-013-9276-4

[7] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4

[8] Sweller, J., Ayres, P., & Kalyuga, S. (2019). *Cognitive load theory* (2nd ed.). Springer. https://doi.org/10.1007/978-3-030-34327-9

[9] Vaithilingam, P., Zhang, T., Glassman, E. L., & Guo, P. J. (2022). Expectations, outcomes, and challenges of modern code completion with GitHub Copilot. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–18. https://doi.org/10.1145/3491102.3517440

[10] Ziegler, J., Kalliamvakou, E., Bird, C., & DeLine, R. (2022). Productivity assessment of neural code completion. *Proceedings of the ACM/IEEE International Conference on Software Engineering*, 1–12. https://doi.org/10.1145/3510003.3510208