Sweat the Details: Exercise Breaks and AI Assistance in Code Summarization

Amy Wei, Priscila Santiesteban, Westley Weimer, University of Michigan, Ann Arbor, USA

Abstract—Code summarization supports program comprehension and defect detection, and developers increasingly use both human-centered and AI-based interventions when performing this task. We present a controlled human-subjects study ($N = 47$) examining the effects of **physical exercise** and **AI-generated code summaries** with varying correctness on developers' summarization and bug detection performance. Participants summarized GitHub code under different intervention conditions, with outcomes evaluated along multiple dimensions, including accuracy, completeness, conciseness, readability, defect detection, and response time. We analyzed the data using mixed-effects models to account for repeated measures across participants and code artifacts. Surprisingly, under the exercise intervention studied, we did not observe consistent benefits for summarization quality. AI assistance is generally useful for code summarization, but incorrect AI assistance substantially degrades bug detection performance. These findings provide empirical evidence on the nuanced benefits and risks of human- and tool-based interventions in code summarization and bug detection.

Index Terms—code summarization, human study, breaks, AI, software engineering

1 Introduction

Code summarization is the task of writing natural-language descriptions of source code to convey its purpose and behavior, and it has been studied as a way to help developers rapidly understand and maintain software systems [1], [2]. Developers routinely seek to understand what a piece of code does, how it does it, and why it behaves in a particular way [3]. As software systems grow in size and complexity, producing accurate and useful summaries becomes increasingly important to help developers navigate and comprehend large codebases [1], [4].

At the same time, **there has been a growing interest in interventions that improve developer performance** on software engineering tasks in general [5], [6] and on comprehension and summarization activities in particular [1], [2]. We consider two broad categories of interventions: those that make human cognition better (such as training or better environments [7]–[11]) and those that provide tool or algorithmic help, offloading some aspects of cognition to external support (such as AI tools [12]–[19]).

One exciting and less-explored intervention is **physical exercise**. Research in cognitive science has found that light physical exercise can affect attention and executive functioning in general reasoning tasks (see meta-analyses [20], [21]) and, excitingly, in engineering activities [22], [23]. Whether such effects extend to software engineering activities, such as code summarization, remains an open question.

In parallel, advances in **large language models (LLMs)** are garnering significant attention and have enabled automatic generation of prose code summaries [12]–[19]. While such tools may assist developers when correct, they remain prone to errors and hallucinations [24] and can introduce anchoring and overreliance effects [25], [26].

We conducted a controlled study in which participants summarized GitHub code under different combinations of light physical exercise and exposure to AI-generated summaries of varying correctness. Participants collectively produced 260 summaries, which we evaluated across multiple dimensions (including accuracy, completeness, conciseness, readability, usefulness for bug detection, and time required). We analyzed the results quantitatively, accounting for repeated observations within participants and across code artifacts [27]–[29]. We also analyzed the resulting summaries qualitatively, identifying patterns associated with each intervention.

Our work is guided by the following research questions:

RQ1. How does exercise influence code summarization performance and the correct identification of buggy code?
RQ2. How does AI assistance influence code summarization performance and the correct identification of buggy code?
RQ3. How does the correctness of AI-generated summaries affect human summarization and bug detection performance?

These research questions relate directly to our desire to understand the relationship between exercise, AI and code summarization outcomes. In addition, we also collect data on, and analyze, a number of important related factors, including whether or not the AI answer was correct [24] and how complex the code was using standard metrics. We make the following contributions:

- The first human study ($N = 47$) to investigate exercise and code summarization as well as exercise and static AI support.
- A formal model relating code summarization outcomes to AI and exercise interventions.
- Statistically significant, actionable relationships. Our results do not uphold the reported Engineering benefits of exercise transferring to software engineering. We do find bug detection to be much more vulnerable to incorrect AI

than is general code summarization.

2 Related Work

In this section, we discuss how our work relates to research on code summarization, AI usage, and exercise.

Exercise Breaks. Physical exercise has been used as an effective intervention in pedagogy across multiple fields and in data processing tasks. Koulanova *et al.* incorporated ten minute exercise breaks (cardio and stretching) into introductory computer science lectures and found that the breaks increased enjoyment and life satisfaction, with certain groups earning higher final course grades [22]. Balci *et al.* used exercise breaks between 30 seconds and 10 minutes and found improvements in performance for data entry and cognitive arithmetic tasks [30].

Fenesi *et al.* found that cardio exercise breaks (e.g., jumping jacks, high knees) increased test performance for both immediate and delayed assessments in introductory psychology lectures [31]. We use the cardio exercises of Fenesi *et al.* and similarly evaluate performance immediately after exercise (see Section 3.2.1).

Micro-Breaks. Nastasi *et al.* found that five minute breaks improve performance in completing simulated check-processing tasks [32]. Bennett *et al.* found that ten minute micro-breaks reduced fatigue in attention-related and simulated work tasks [33]. Liu *et al.* used three minute breaks (short-form videos, music), which increased willingness and reduced physiological stress measures for English proof-reading tasks [34]. We build on this prior work by analyzing the effect of similar-length exercise breaks on code summarization performance in software engineering (see Section 3.2.1).

Mindfulness Breaks. Other work has found varied effects of meditation and mindfulness breaks on performance [35]. Volobuev *et al.* used electroencephalography (EEG) to examine how meditation influenced neural activity during programming interview tasks, finding that it reduced neural engagement [36]. Although the mechanisms differ, this reduced engagement is broadly reminiscent of the slower response time and lower quality found in our experiments in exercise (see Section 6.1).

Montes *et al.* found that weekly 90-minute breathing exercises improved well-being and perceived productivity among IT workers [23]. Bernárdez *et al.* investigated ten-minute mindfulness sessions in requirements elicitation tasks involving conceptual modeling and observed improvements in efficiency [5]. In contrast, we chose to investigate *active* exercise breaks.

Code Summarization — Call Graphs. Prior work used eye tracking to investigate programmer attention on call graphs during code summarization. For example, McLoughlin *et al.* investigated how programmer visual attention relates to underlying function call graphs when summarizing code [37]. We follow the same qualitative analysis procedure to score participant summaries in our study (see Section 4.2). McLoughlin *et al.* found that when participants look at more methods associated with the call graph, they produce lower-quality summaries and report reduced self-confidence.

Code Summarization — Signatures. Researchers have investigated attention to method signatures during code summarization. Rodeghero *et al.* used eye tracking to investigate how much developers focused on signatures, control flow, and invocations when summarizing code [38]. Abid *et al.* similarly used eye tracking to compare novices and professionals, finding that novices relied heavily on signatures while experts returned to the method body more often [39]. In this paper we focus on behavioral metrics (see Section 4 as oppose to cognitive metrics (e.g., eye-tracking).

Code Summarization — ASTs. Researchers have also examined how developers attend to abstract syntax trees during code summarization. Karas *et al.* performed an eye tracking study to investigate programmer attention on the abstract syntax tree when reading or writing code summaries [40]. While we evaluate participant summaries according to the same scoring criteria in this study (e.g., accuracy, readability, see Section 4.2), we alternatively investigate the summarization of defect-seeded source code to approximate developer workflows on real software with bugs. In addition, we focus on larger Java methods for additional ecological validity.

Code Summarization — Reading Summaries. Other work has focused on the task of *reading* already-existing summaries. Stapleton *et al.* conducted a human study in which participants were given summaries of Java source code and answered code comprehension questions or completed coding tasks [2]. Their study found that subjects performed better when given human-written summaries than machine-generated summaries, whereas we instead investigate AI-written summaries used to assistant. We note that the machine-generated summaries in Stapleton *et al.*'s work from 2020 may have predated modern generative AI tools.

AI for Code Summarization. Prior work has used large language models to write code summaries [12]–[19]. Ahmed *et al.* found that adding semantic facts to the code when prompting LLMs resulted in better code summaries according to common metrics such as BLEU [41]. Others have used LLMs to evaluate the quality of generated code summaries [42], [43]. Wang *et al.* found that LLMs can achieve near-human evaluation on the quality of code summaries [44]. Virk *et al.* developed approaches to predict whether summaries produced by LLMs will be of comparable quality to human summaries [45].

Prior work has also used LLMs to summarize a wide range of software artifacts. For example, LLMs have been applied to code changes [46], code comments [47], bug reports [48], and code fixes based on crash traces [49]. In our study, we instead provide participants with AI-generated summaries to examine how AI can support humans during code summarization. We evaluate participant summaries using human judgments rather than NLP-based similarity metrics, which allows for a more grounded assessment of summarization performance (see Section 4.2).

AI for Fault Localization. While significant prior research has used LLMs to find bugs [50]–[52], our study uses humans *supported by* AI information to perform fault localization. For example, Liu *et al.* used LLMs on programs written by novices and found that ChatGPT-4 outperformed traditional testing methods, with prompt engineering improving the LLMs' performance [53].

Widyasari *et al.* found that fault localization explanations generated by LLMs were rated as similarly clear and informative as human-written explanations [54]. Their study emphasized perceived quality rather than whether the explanations were factually correct. In contrast, we examine bug detection and specifically evaluate how the correctness of AI-assisted bug descriptions influences performance.

Trust in AI. Prior research has found that developer trust in AI depends on model-specific attributes including accuracy, robustness, and interpretability, as well as factors such as transparency and usability [55]. Others have found that novice developers use LLMs for a wide range of SE activities and generally perceive that AI improves productivity. Many novice developers do not blindly trust LLMs, either holding neutral views or opting not to use AI. However, they may have difficulty interpreting LLM suggestions due to lack of background or may not know when they are helpful to use [56]. In contrast, rather than directly asking participants about their trust in AI, we report indirect measures (see Section 6.1 and 6.2).

3 Experimental Methodology

We implement a controlled, IRB-approved (UMich IRB SBS HUM00276189) human study (N = 47) to investigate how exercise breaks and AI assistance affect programmers' code summarization and defect detection performance. This section outlines the task completed by participants (Section 3.1), experimental design and conditions (Section 3.2), stimulus construction (Section 3.3), and recruitment process (Section 3.4).

3.1 Code Summarization Task

Participants completed a code summarization task designed to balance ecological validity with measurable performance outcomes. Specifically, participants wrote natural language summaries of target methods drawn from large, unfamiliar code bases, reflecting common comprehension scenarios in software development [2], [3]. Because code summarization frequently co-occurs with maintenance and debugging activities [2], participants indicated whether each method contained a bug and where if so (Section 3.3.1).

Summaries were evaluated along four dimensions: accuracy, completeness, conciseness, and readability, using predefined criteria provided to participants at task onset [37], [40]. These dimensions are among the most common quality attributes used for code comment quality assessment (see [57] for a comprehensive review). Each task was designed to take ten minutes or less, with participants completing tasks in 8.4 minutes on average.

3.2 Experimental Design and Controlled Interventions

We designed a 90-minute human study in which participants completed six code summarization tasks (Section 3.1) under controlled interventions (Section 3.2.1 and 3.2.2). We manipulated two factors: exercise breaks (exercise vs. no exercise before the task) and AI assistance (assistance vs. no assistance during the task), yielding four conditions: exercise/AI, exercise/no-AI, no-exercise/AI, and no-exercise/no-AI.

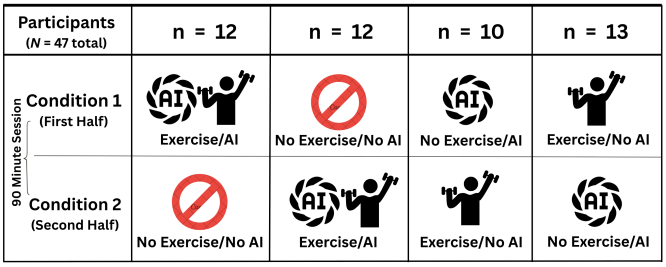

Fig. 1: Experimental design. We had four conditions: exercise/AI, exercise/no-AI, no-exercise/AI, and no-exercise/no-AI. Each participant completed two task blocks (first and second half), with a different condition in each. n is the number of participants per ordered pair of conditions.

Each participant completed two task blocks, one in the first half of the session and one in the second half, and was assigned a different condition in each block (n = $12, 12, 10, 13$ respectively, see Figure 1). For example, a participant might complete the first half under exercise/no-AI and the second half under no-exercise/AI. The assignment of condition order was counterbalanced across participants to enable both between-subject and within-subject comparisons. This design allowed us to examine both the main effects of exercise and AI assistance, as well as their possible interaction. In this section, we elaborate on the interventions and our rationale.

3.2.1 Exercise Breaks

To investigate how physical exercise influences code summarization performance, we selected an exercise intervention previously shown to affect performance in an academic but non-computing setting. To the best of our knowledge, no prior work directly investigates the impact of exercise on software comprehension (cf. [22], [36]); we thus selected a five-minute exercise activity found by Fenesi *et al.* to improve lecture comprehension test performance with a medium effect size [31]. We investigate whether such benefits extend to the software task of code summarization.

In the half of the session assigned to the exercise condition, participants completed a five-minute exercise break immediately before each code summarization task (Section 3.1). Following Fenesi *et al.*, the exercise break consisted of five calisthenic exercises (jumping jacks, heel taps, high knees, split jumps, and hamstring kickers) [31], each performed for 50 seconds with 10 seconds of rest, for a total of five minutes per task (15 minutes per session). Participants followed a guided video and were allowed to use lower-intensity modifications as needed. After each task within this condition, participants were asked to report their perceptions of the helpfulness of, and exertion required by, the exercise. Our design controls the exercise treatment within our study and investigates an activity that could be used in practice in many workplaces (cf. recommendations for walking or calisthenic breaks at work [58], [59]).

3.2.2 AI Assistance

To investigate the effect of AI assistance on code-summarization performance, we implemented a controlled form of AI support. In one treatment condition, participants were shown a static AI-generated summary of the target

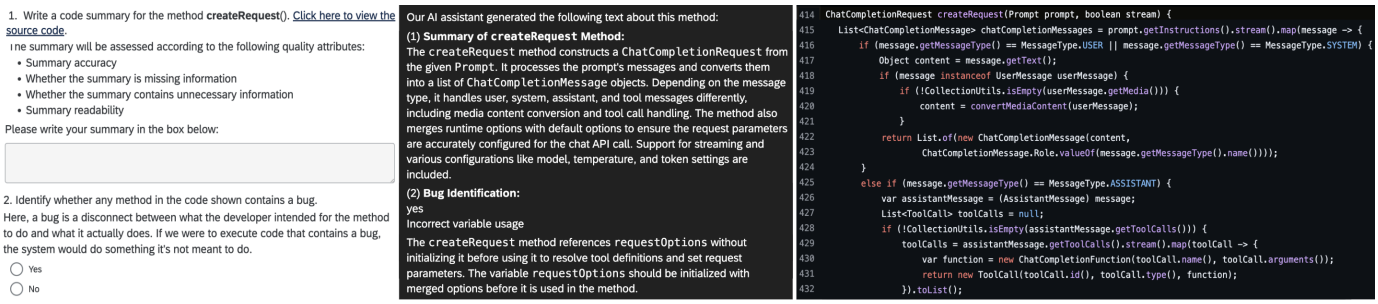

Fig. 2: Example stimuli. Participants used the remote survey interface, AI responses, and (partial) Java source code.

method they were asked to summarize (see Section 3.1). An example is shown in Figure 2. Pre-generated (rather than interactive) AI responses were used to ensure that all participants received identical input, since slight variations in "prompt engineering" can produce significant variation in AI output quality [53], [60]. The summary content was presented exactly as produced by the ChatGPT-4o model [61], and participants were permitted to consult the summaries freely during the task. After each task within this condition, participants were asked how helpful they perceived the AI summary to be.

We experimentally varied the *correctness* of the AI summaries. In this study, correctness refers specifically to whether an AI-produced summary makes a truthful claim about the presence or absence of a defect in the corresponding code stimulus (see Section 3.3.1). Participants were informed that the AI summaries might be incorrect but were not told how often or in what manner. In our experiment, AI summary correctness was balanced to support analysis of our research questions. We believe this controlled AI-summary treatment approach reflects common behavior in current LLM-based tools used by developers, which can produce both correct and incorrect explanations (e.g., hallucinations) [62]. We detail AI summary generation and control in Section 3.3.2.

3.3 Stimuli Selection and Construction

We used two types of stimuli: Java source code methods (with or without a seeded defect) and static AI-generated summaries (varying in correctness). Below, we describe how each stimulus type was motivated and constructed. All stimuli are available in the replication package (see Section 5).

3.3.1 Source Code Stimuli

In this section, we describe our approach for selecting code stimuli and seeding defects.

Open-Source Java Methods. To obtain stimuli that better reflected real-world development work while still fitting within our human study time constraints, we used code from high-quality, high-activity open-source GitHub projects. Some open-source projects provide examples of software used and maintained by practicing developers [63].

We selected the two highest-starred Java projects created within the last year (at the time of stimulus creation) on GitHub: `spring-ai-alibaba` (a framework for building AI agent applications, 171,242 LOC) and `fast-excel` (a library for reading and writing Excel spreadsheets, 555,994 LOC). Within these projects, we identified Java modules that contained at least one method that was 40 lines of code or longer. This length cutoff was chosen in alignment with prior work in code comprehension [64]. For additional ecological validity, stimuli were presented to participants using the existing GitHub web interface. However, to minimize participant bias associated with context information, we cloned the relevant modules to private repositories and removed comments containing GitHub links that could direct participants to the full repository or other code. We also removed `@author` tags, which could reveal the originating project or admit other biases (e.g., gender bias in GitHub pull requests [65] or code review [66]).

We ultimately identified 17 code modules that matched our length requirements within the top two Java projects to serve as our stimulus pool (see Table 1). A partial example is in Figure 2 (full stimuli are available in the replication package). Each participant completed six of them. This larger pool increased stimulus diversity and reduced the risk that our findings would depend on the specifics of any single code example.

Defect Seeding. To more closely approximate real developer workflows, we considered two versions of each code stimulus: (1) the original version, and (2) a version containing a seeded defect. In our design, seeded defects are not a top-level experimental treatment condition (cf. exercise, AI) but instead allow our analyses to account for an additional notion of code quality.

To create buggy versions in a controlled manner, we adopted the defect-seeding procedure of Fry *et al.* [67], which has been used and validated in prior Java fault localization research. For each Java module, we randomly selected one of the 18 defect types defined by Fry *et al.*, such as using an incorrect variable or omitting a required statement. We then identified all valid locations in the method where that defect could be introduced and randomly inserted the defect at one of those locations, reusing existing code when possible. We indicate the type of defect seeded in each code stimuli on Table 1. This approach produced single-seeded-defect variations across stimuli while keeping the tasks feasible for participants to summarize within the study's time constraints.[1]

1. We note that the stimuli, which come from projects under active development, may contain unknown or latent defects. We intentionally compare quality between "existing code with a seeded defect" (lower) and "existing code" (average) — not "perfect code" (high).

Method name	LOC	File name	LOC	Seeded defect type
		`fast-excel`		
buildUserModel	42	ModelBuildEventListener	150	WrongVariable
afterWorkbookDispose	45	DimensionWorkbookWriteHandler	66	WrongMethod
createCellStyle	45	WriteWorkbookHolder	243	WrongConditional
chooseExcelExecutor	46	ExcelAnalyserImpl	205	WrongMethod
fillComment	46	AbstractExcelWriteExecutor	295	ExtraAssignment
headCellRangeList	48	ExcelWriteHeadProperty	123	UninitializedVariable
buildChain	55	AbstractWriteHolder	408	ExtraStatement
finish	63	ExcelAnalyserImpl	205	OrToAnd
doFill	96	ExcelWriteFillExecutor	532	OrToAnd
		`spring-ai-alibaba`		
after	46	DashScopeDocumentRetrievalAdvisor	157	ExtraAssignment
executeToolCall	49	ObservableToolCallingManager	233	VarShouldHaveBeenMethod
createRequest	50	DashScopeChatModel	454	UninitializedVariable
apply	61	McpNode	171	ExtraConditional
put	62	MongoSaver	207	MissingStatement
upsertPipeline	62	DashScopeApi	960	UninitializedVariable
generate	66	DiagramGenerator	147	WrongType
initBody	75	HttpNode	611	ExtraAssignment
Total: 34 unique code stimuli (17 Java files from open-source projects, each with an original and defect-seeded version)				

TABLE 1: Code summarization (and defect detection) stimuli. Each stimulus asks participants to summarize the given method ("Method name"), with the full file available as context ("File name"). In cases where a seeded defect is present the category is noted ("Seeded defect type"). We make our stimuli avaliable (see Section 5).

3.3.2 AI Summary Stimuli

We generated AI summaries for participant use as assistance during some tasks. For each code-snippet stimulus (both the original and the defect-seeded version of the Java module), we provided ChatGPT-4o with the full source code of the corresponding module and requested (1) a code-comment-style summary of the target method, and (2) a statement indicating whether it believed the module contained a bug. Both (1) and (2) served as the assistance shown to participants. The bug statement allowed us to determine the *correctness* of the AI summary with respect to the defect seeding status for that code version (i.e., whether a seeded defect was present). We also manually verified that the AI summaries never correctly identified a non-seeded (i.e., latent) defect in our stimuli.

We thus labeled each AI summary by comparing its bug statement to the ground truth seeded defect status of the corresponding code version, yielding four possible correctness outcomes: a *true positive* (the model stated that a bug was present and the code contained a seeded defect), a *true negative* (the model stated that no bug was present and the code contained no seeded defect), a *false positive* (the model stated that a bug was present but the code contained no seeded defect and our manual verification ruled out the claimed bug), and a *false negative* (the model stated that no bug was present but the code contained a seeded defect). For each stimulus, we iteratively queried the model to obtain examples of all four outcomes. We note that since we queried the model until we obtained all 4 categories that these stimuli do not represent a sample of exact AI error rates and are instead intended to help us examine the hypothetical cost of AI inaccuracies.

When necessary, we varied the prompt by adding a brief framing paragraph describing the code as written by either a novice or an expert, as seen in prior work in LLMs [53], [60]. In cases where the model repeatedly produced the same outcome, we reseeded the defect in a different location within the method and repeated the query. The exact prompt text is available in our replication package. In all cases, the output of the AI model is used verbatim (i.e., no manual editing). We believe these AI summaries provide a moderately-realistic form of developer-facing AI assistance because they were used exactly as produced by a state-of-the-art (at the time of data collection and participant performance) model, including naturally occurring inaccuracies, while the four labeled correctness outcomes enabled experimental control over the accuracy of AI assistance.

3.3.3 Stimuli Presentation

We delivered all materials through a remote online survey (see Figure 2), which shows an example of the survey interface. When first presented with the prompt to summarize code, participants were given an embedded GitHub link (as shown in the left panel of Figure 2). After clicking the link, they viewed the target method in the standard GitHub interface with syntax highlighting (right panel of Figure 2). The page automatically scrolled to the relevant method. All files were hosted in a standalone repository to prevent access to additional project context. Participants were instructed to summarize the designated method but could view the entire file.

For AI-assisted tasks, participants were shown an AI-generated response directly within the survey interface (middle panel of Figure 2), positioned below the initial prompt. We embedded the AI response on the same page to reduce tab switching. We did not change the content, but introduced minor formatting adjustments (e.g., line breaks) to improve readability without bias. We styled the AI text to appear against a dark background, similar to common

Gender	**Count**
Man	28
Woman	17
Prefer not to say	2
Race	**Count**
Asian or Pacific Islander	26
Black or African American	2
Hispanic or Latino	2
White or Caucasian	18
Multiracial or Biracial	1
Prefer not to say	2
Occupation	**Count**
Undergraduate in Computer Science	19
Undergraduate not in Computer Science	7
Graduate student in Computer Science	13
Graduate student not in Computer Science	5
Full or part-time job in Computer Science field	1
Other	2
Self-reported Coding Experience (scores in $[1$ (low), 10 (high)$]$)	**Count**
$1 \leq x < 3$	3
$3 \leq x \leq 4$	8
$5 \leq x \leq 6$	11
$7 \leq x \leq 8$	22
$9 \leq x \leq 10$	3

TABLE 2: Demographic overview of survey participants.

"dark mode" ChatGPT interfaces. At the top of the response, we described it as follows: "Our AI assistant generated the following text about this method." For tasks with exercise breaks, we embedded the guided, timed exercise video in the survey.

3.4 Recruitment and Participant Contextualization

In this section, we describe the recruitment process for our human study and our population pool.

Recruitment and Data Population. We recruited 47 participants via a combination of email, course forums, posters and in-class presentations at the University of Michigan, a large public US university. Subjects were required to be at least 18 years old, have experience in Java or C++, have taken or be enrolled in a data structures course, and be able to safely perform low-impact exercises. To participate, subjects were required to complete a pre-study to practice an example task. Consented participants could choose to complete the study either in a *synchronous supervised videoconferencing* session or in an *asynchronous self-paced* format completed independently online. In all cases, to preserve privacy, participants were instructed that video evidence of exercise could be provided occluded (e.g., the participant's body could be out of frame with just a moving arm visible to show jumping jacks, etc.). In addition, each participant had their own unique videoconferencing session and URL (e.g., one participant could never interrupt or see another, even in a virtual waiting room). Those who completed the study received $40 for the supervised format and $30 for the asynchronous format.

Certain individual responses to tasks were excluded from 6 participants for having response times more than 2 standard deviations away from the mean. Certain responses from 3 participants were excluded for not following the exercise break instructions. Some responses from 1 participant were excluded for not completing survey questions related to the task. Ten responses within the AI condition were excluded for indicating they did not read the AI. These exclusion criteria were defined before analysis. Our final analysis considered 260 task responses from 47 participants. In particular, our analysis included 129 task response data points for Condition 1 and 131 task response data points for Condition 2 (see Figure 1). Viewed another way, our analysis includes 65 task response data points for exercise/AI, 63 for exercise/no-AI, 64 for no-exercise/AI, and 68 for no-exercise/no-AI.

Population Contextualization. Table 2 summarizes our participant demographics. 28 of our participants were men, 17 were women, and 2 preferred not to report gender. 19 reported being undergraduate computer science students, 7 were undergraduate students not in computer science, 13 were graduate computer science students, and 5 were graduate students not in computer science. 1 reported having a full- or part-time job in computer science.

4 Analysis Methodology

We analyzed our results via mixed effects modeling and qualitative analysis.

4.1 Quantitative Analysis

We analyzed performance outcomes across the code summarization tasks using mixed-effects regression models. This approach is appropriate for our experimental design because it accommodates repeated observations from the same participants, accounts for variability across different tasks, and enables generalization beyond the specific participants and items included in the study [27]–[29]. Mixed-effects modeling is widely used for performance data in empirical research where multiple observations are collected from each participant [68]. We controlled the false discovery rate at $\alpha = 0.05$ using the Benjamini–Hochberg (BH) procedure and report adjusted p-values. Response time, code summary quality ratings, and bug detection were each modeled using regression families appropriate to their measurement properties. The following sections describe the modeling strategy, including fixed and random effects for each dependent variable. We claim no novelty in these statistical approaches.

4.1.1 Modeling Strategy

Our model-building process compared candidate models using information-theoretic criteria (AIC/BIC), which balances fit and model complexity [69], [70]. For all dependent variables, we specified a core set of design-driven fixed effects consisting of AI assistance, exercise condition, and trial sequence number (i.e., proxy for learning effects). Including predictors derived directly from the experimental design in an initial base model is consistent with recommended practice in model development [69], [71].

Using model information criteria (such as AIC/BIC) to identify a supported random-effects specification is consistent with recommended practice for mixed effects modeling

[70], [72], starting from a maximal random effects structure where feasible [73]. We first identified an appropriate random effects structure by comparing candidate models that held the core fixed effects constant while varying random intercepts and slopes for participants and items.

We assessed extensions to this model only after establishing the random effects structure, which is consistent with recommended practice [28], [72]. We then considered extending the fixed effects specification by adding additional predictors or interaction terms to the core model. For example, we evaluated whether including participant occupation, participant physical activity, Java project identity, if the session was done supervised or not, or the AI and exercise interaction improved model fit relative to the base specification. As part of this, we include trial sequence as a predictor to account for potential learning or adaptation effects across repeated tasks. Among the 96 candidate fixed effect formulations considered for each dependent variable, the final model was selected via the established approach of information model criteria (AIC/BIC), with lower values indicating stronger model support [69], [70]. This standard approach of using information-theoretic criteria to determine a final model is appropriate as it penalizes overfitting by favoring parsimonious models.

4.1.2 Model Specifications by Dependent Variable

We describe the our model formulations. We make our models available (see Section 5).

Response Time. Response time (RT) exhibited a right-skewed distribution, which is typical for human performance measures, so a standard square-root transformation was applied prior to analysis to improve normality of residuals [74]. The transformed RT outcome was then analyzed using a linear mixed-effects model (LMM), treating $\sqrt{RT}$ as a continuous response with approximately Gaussian residual structure. The model incorporated the fixed and random effects from model selection (Section 4.1.1).

Code Summary Qualities. Each of the code summary quality dimensions (accuracy, completeness, conciseness, readability), using a 0–5 ordinal scale via qualitative analysis (see Section 4.2), was modeled using cumulative link mixed models (CLMM) suitable for ordered categorical outcomes [71]. The ordinal specification accounts for the ordered but non-interval nature of the ratings, avoids assuming linear distance between rating categories, and yields interpretable log-odds of being at or above each rating threshold. Fixed and random effects were specified according to the modeling strategy described in Section 4.1.1.

Bug Detection. Participant bug detection performance (correct vs. incorrect; see Section 4.2) was analyzed using mixed-effects logistic regression, appropriate for binary response outcomes. The resulting models (general and AI-only; see Table 3) estimates the log-odds of successful bug identification as a function of the specified fixed effects and random variance described in Section 4.1.1.

4.2 Qualitative Annotation

We evaluated participant-written prose summaries using a qualitative annotation procedure adapted from the established four-attribute framework of McLoughlin *et al.* and Karas *et al.* [37], [40]. Two authors independently scored each summary on four dimensions: accuracy, completeness, conciseness, and readability. A shared codebook and grading rubric were agreed upon prior to annotation. Each attribute was scored on a 5-point scale, with one point deducted for each unmet rubric criterion (see Section 5).One author was a Computer Science PhD student, and the other was a fourth-year undergraduate in Computer Science.

Annotation proceeded in three stages. In the training stage, both annotators independently scored all summaries for 4 of the 17 code methods. They then met in a negotiated consensus stage (cf. [75]) to ensure consistent application of rubrics, resolve disagreements, and finalize training set scores. Finally, annotators independently scored the remaining summaries, after which we computed the intraclass correlation coefficient (ICC) to assess interrater reliability across all summaries. ICC was 0.92 across all annotations.

For the binary assessment of bug detection, we followed a semantically guided annotation procedure. One author reviewed each participant's bug description and marked it as correct or incorrect. To be considered correct, the description had to identify the appropriate line(s) where the defect occurred and explain why it constituted a defect. Correctness was evaluated generously rather than strictly: if a defect seeded on line X could also manifest on lines Y or Z, all such locations were accepted (e.g., an uninitialized variable could be reported either at declaration or at use).

5 Replication Package

We make our data, analysis scripts, derived models, and qualitative annotations available for transparency and replication of our findings. Our package is available at https://github.com/pasantiesteban/25-Code-Summarization-Exercise-AI.

6 Results

We present quantitative and qualitative results of our investigation of effect of exercise breaks and AI assistance on code summarization performance (see Section 1 for our primary research questions). P# denotes a participant number for quote attribution. All β values correspond to fixed-effect coefficients from the mixed-effects models (see Table 3) and are reported on the model link scale (e.g., log-odds for ordinal outcomes), indicating the direction and relative magnitude of effects. To aid interpretation, we also report descriptive statistics (e.g., average character length and response time), which are not part of the models but provide contextual information about observed differences.

6.1 RQ1 — Effect of Exercise on Performance

We examined whether physical exercise prior to a code summarization task influenced performance, measured by response time, summary quality, and bug detection accuracy. Although bug detection is not a direct summarization outcome, it reflects broader program comprehension closely related to summarization. Based on prior work showing exercise improves lecture-based performance [31], we hypothesized similar benefits for software engineering tasks.

TABLE 3: Fixed-effect estimates (β) of key predictors across all final mixed-effects models. Coefficients are reported on each model's link or transformed scale. Shaded cells indicate statistically significant effects ($p < .05$). The final column reports marginal (M) and conditional (C) R^2 values for each model, where marginal R^2 reflects the variance explained by the fixed effects alone, and conditional R^2 reflects the variance explained by both the fixed and random effects. The marginal R^2 values ranged from $0.02 – 0.31$ across outcomes, consistent with prior behavioral research where modest fixed-effect explanatory power is expected in complex human-performance tasks [76]. Full model specifications and estimation details are available in the replication package. * *In the AI-only bug-detection model, the AI assistance predictor reflects the effect of AI's ground truth in terms of the defect seeded status.*

Outcome	Model Link / Scale	Exercise	AI Assistance	Trial Sequence	Exercise × Trial	R^2 (M / C)
Response Time (RT)	LMM ($\sqrt{\text{RT}}$)	$\beta = 1.88$	$\beta = 0.11$	$\beta = -1.31$	*n/a*	0.27/0.86
Summary Accuracy	CLMM (logit)	$\beta = -2.50$	$\beta = 0.12$	$\beta = -0.35$	$\beta = 0.48$	0.22/0.28
Summary Completeness	CLMM (logit)	$\beta = 0.27$	$\beta = 0.50$	$\beta = -0.05$	*n/a*	0.02/0.43
Summary Readability	CLMM (logit)	$\beta = -0.98$	$\beta = -0.09$	$\beta = -0.11$	*n/a*	0.10/0.57
Summary Conciseness	CLMM (logit)	$\beta = -0.85$	$\beta = 1.02$	$\beta = 0.17$	*n/a*	0.22/0.47
Bug Detection Accuracy (general)	GLMM (logit)	$\beta = -0.73$	$\beta = 0.73$	$\beta = -0.17$	$\beta = 0.29$	0.05/0.31
Bug Detection Accuracy (AI-only)	GLMM (logit)	$\beta = 0.39$	$\beta = 2.51^*$	$\beta = 0.09$	*n/a*	0.29/0.41

Exercise Impairs Summarization Performance. Surprisingly, the net effect of our exercise intervention had a negative impact on both response time and summary quality. We find that participants in the exercise condition had significantly longer response times ($\beta = 1.88$, $p = 0.023$), indicating slower responses under the model. Descriptively, participants in the exercise condition took an average of 526 seconds per summary ($SD = 243$), compared to 486 seconds ($SD = 225$) for non-exercised participants. Exercise was also associated with significantly lower accuracy ($\beta = -2.50$, $p = 0.009$). This indicates reduced correctness in summarizing code functionality. Our qualitative analysis supports this as we found that exercised participants made significantly more *minor comprehension mistakes* than their counterpart. In particular, we find that about 14% of summaries (9/63) within the exercise/no-AI condition describe an incorrect object, compared to 6% (4/71) in the no-exercise/no-AI condition ($p < 0.05$). For example, a participant wrote that a method *"will return all combined objects"* (P1) when only one object is returned. Similarly, another participant wrote that a method will *"print out a response message based on the type of the prompt"* (P2) when it is actually based on the object `toolCall`.

Exercise was also associated with reduced conciseness ($\beta = -0.85$, $p = 0.023$), indicating lower conciseness ratings under the model. Descriptively, exercised participants produced longer summaries on average (394 characters, $SD = 294$) than non-exercised participants (326 characters, $SD = 167$). We see qualitative evidence in the summary structures. Exercised participants tended to summarize source code in a *line-by-line* manner, describing the details of each statement rather than abstracting to retrain important functionality. For example, one participant walked through each line of the method: *"First the function checks if either the writeWorkBookHolder is null or if the actual workbook which is a member variable of the holder is null. If so it returns. Next..."* (P3). We note that the participant wrote this at the beginning of their summary, when the effects of exercise were less likely to have worn off. We find that about 17% (11/63) of summaries within the exercise/no-AI condition are written line-by-line, compared to about 5% (4/71) within the no-exercise/no-AI condition ($p < 0.02$). We do not find evidence that exercise influences readability or completeness.

Exercise Effects Diminish Over Repeated Trials. We observe that the net negative effects of exercise are not uniform over time. A significant interaction between exercise and trial sequence on summary accuracy ($\beta = 0.48$, $p = 0.023$) indicates the effect weakens across trials, suggesting diminishing initial impairment.

Exercise Does Not Help Seeded Bug Detection. We assess participants' ability to identify seeded defects in source code. Across all trials (including both AI and non-AI cases), exercise showed a negative ($\beta = -0.73$) but non-significant effect on bug detection ($p = 0.40$). Thus, we find no evidence that exercise affects bug detection.

Possible Explanation. This result was both surprising and theoretically informative. We hypothesize that recent exercise induces a heightened state of physiological arousal that can interfere with the calm, sustained abstraction and executive control associated with high-quality code summarization. This interpretation is consistent with prior work showing that the cognitive effects of acute high-intensity aerobic exercise are mixed, with studies reporting improvements, impairments, or no change depending on experimental design, timing, and task demands [77], [78]. Importantly, prior work in the exercise-physiology literature shows that following light or moderate exercise, heart rate and related autonomic measures gradually return to resting levels over time [77]. Our finding that the negative effects of the exercise intervention on summary accuracy diminish over repeated trials is consistent with this prior work. Prior results demonstrating benefits of exercise for lecture comprehension and attention [31] may therefore reflect selective enhancement of relatively passive processes, such as vigilance and sustained attention, rather than complex generative tasks such as code summarization. However, this explanation is inferential: while it is consistent with prior work, our study did not directly measure participants' exercise intensity or physiological state.

Participant self-reports help contextualize these findings. For the prompt "I felt more prepared to work after the exercise break," a plurality of responses disagreed (48%; 61/128), compared to 31% that agreed (40/128) and 21% that were neutral (27/128). Similarly, for the prompt "The exercise break helped me concentrate on this task," 47% of

responses disagreed (60/128), compared to 35% that agreed (45/128) and 18% that were neutral (23/128). This pattern suggests that the observed performance differences are unlikely to be driven by reduced motivation or disengagement, but instead reflect changes in how participants approached the task. Despite these perceptions, we do not find evidence that participants in the exercise/no-AI condition were less confident that they successfully completed the task than participants in the no-exercise/no-AI condition ($p = 0.51$).

Overall, under the exercise protocol used in our study, participants assigned to the exercise condition did *not* exhibit improved code summarization performance and instead tended to produce more verbose summaries without improved correctness.

RQ1 — How does exercise influence performance?

We find that brief exercise impairs code summarization performance. Participants in the exercise condition exhibit significantly longer response times ($\beta = 1.88$, $p = 0.023$) and reduced summary accuracy ($\beta = -2.50$, $p = 0.009$) and conciseness ($\beta = -0.85$, $p = 0.023$). Qualitatively, exercised participants more often introduce minor comprehension mistakes and adopt a line-by-line summarization style. These negative effects on summarization accuracy diminish over repeated trials ($\beta = 0.48$, $p = 0.023$). In contrast, we find no evidence that exercise improves bug detection performance ($p = 0.40$).

6.2 RQ2 — Effect of AI Assistance on Performance

We assess the effect of providing static AI code summaries and bug descriptions on performance. We use the same evaluation criteria: (1) response time and (2) summary quality. We discuss bug detection accuracy in RQ3. We first present our main result (AI's benefits) and then an explanation.

AI Affects Summarization Style. Our primary finding is that AI assistance positively changes how participants summarize code, even when the assistance contains incorrect or hallucinatory bug-related information. AI assistance is associated with greater conciseness ($\beta = 1.02$, $p = 0.014$). Descriptively, participants in the AI condition produced summaries with an average length of 326 characters ($SD = 193$), compared to 393 characters ($SD = 276$) without AI. Notably, participant-written summaries were substantially shorter than the AI assistance they were provided with (mean = 852 characters, $SD = 312$), indicating that participants did not copy the AI output wholesale but instead engaged in additional human abstraction. On average, participant summaries were less than half the length of the AI-provided assistance.

AI assistance showed a positive association with completeness ($\beta = 0.27$), but this effect did not remain significant after BH correction ($p = 0.077$). In addition, we do not find evidence that AI improved summarization correctness, readability, or response time. We discuss bug detection outcomes in RQ3.

Possible Explanation. Qualitative analysis suggests that AI assistance improves summarization style by providing a linguistic and structural scaffold that participants selectively reuse when writing their own summaries. Rather than copying AI-generated summaries verbatim, participants appear to compress and rephrase the provided content, retaining key terminology and structural elements while producing substantially shorter summaries.

One manifestation of this pattern is the *reuse of vocabulary* introduced by the AI. For example, in one task, several participants used the term "ingestion," even though this concept is not explicitly described in the source code and appears only as part of a variable name (`ingestionId`). However, the term "ingestion" appears in the AI-generated summary for this task, and all participants who used this term in their summaries were in the AI condition. This suggests that AI assistance can introduce or surface domain-relevant terminology to participants (as in Liu *et al.* [79]).

We also observe *structural similarities* between participant-written summaries and the corresponding AI-generated assistance. In some cases, participant sentences closely match the phrasing of the AI output. For example, one participant wrote: *"It sends a PUT request to upsert the pipeline and a POST request to initiate document ingestion into the managed pipeline"*, followed by *"If there's problem with the startPipelineResponse, the function will throw DashScopeException error"* (P5). These sentences closely mirror the AI-generated description, which states: "It sends a PUT request to upsert the pipeline and a POST request to initiate document [...] Throws DashScopeException if there are errors in pipeline creation or document ingestion." Overall, 50% (32/64) of summaries in the AI condition share notable phrases or structures with the AI assistance.

Participant self-reports further contextualize these findings. Participants report greater confidence in completing the task in the no-exercise/AI condition compared to the no-exercise/no-AI condition ($p = 0.03$). They also generally perceive the AI assistant positively: 53% of responses agreed with the statement "I am confident in the AI assistant," compared to 26% who disagreed and 21% who were neutral; similarly, 58% disagreed that "The AI assistant is deceptive." Despite these positive perceptions, we do not find evidence that AI assistance improves summarization correctness, readability, or completeness. Prior work suggests that information presented in a fluent, well-structured, and authoritative style is often perceived as more credible and persuasive, independent of its correctness [80]. This may help explain why participants adopt AI-provided language and structure, even when it does not improve summarization quality. In RQ3, we probe the impact of AI correctness (or hallucinations).

RQ2 — How does AI assistance influence performance?

AI assistance improves certain dimensions of code summarization but *not* overall correctness. AI-assisted participants produce significantly more concise ($\beta = 1.02$, $p = 0.014$). AI provides a linguistic and structural scaffold that participants selectively reuse, compressing and rephrasing AI content rather than copying it verbatim. In contrast, we find no evidence that AI assistance improves summary accuracy, completeness, readability, or time.

6.3 RQ3 — Effect of AI Correctness on Performance

We assess the effect of correct and incorrect AI-provided bug detection guidance (Section 3.2.2). While prior work

has examined reliance on automated tools and AI assistance (e.g., [81]), we focus on three goals: to (1) *quantify* the magnitude of this effect under controlled conditions, (2) *characterize* how strongly users follow incorrect guidance in seeded bug detection tasks, and (3) *compare* these effects across code summarization and seeded bug detection. In all cases, the guidance was presented verbatim from a generative AI system and varied in correctness regarding defect presence. We present the main results, then an explanation. Because the correctness of the AI guidance was experimentally manipulated (see Section 3.2.2), these results estimate the impact of receiving incorrect AI advice rather than the real-world frequency or overall risk of encountering incorrect AI outputs.

AI Hallucinations Matter in Seeded Bug Detection. AI correctness strongly predicted human performance on our seeded bug detection task ($\beta = 2.51$, $p < 0.0001$). When AI was correct about the presence or absence of a seeded bug, participants were 81% likely to report the correct answer. Conversely, when the AI was incorrect, participants were only 26% likely to be correct.

Possible Explanation. These results suggest that participants relied heavily on AI-provided information for the seeded bug detection task, amplifying both its benefits when correct and its costs when misleading. This indicates that participants did not sufficiently double-check AI assistance and were particularly vulnerable to AI bias in this task. This pattern aligns with similar reliance on AI for code summary construction (Section 6.2).

Qualitatively, this appears in participant explanations for the seeded bug detection task. When the AI assistance was correct, participants often detected the defect. For example, one participant wrote: *"The AI is correct that there is a use of pipelineId on line 776 when that variable is never initialized"* (P6). However, when the AI assistance was incorrect, participants often included it unchallenged. For example, in one task, the AI incorrectly claimed that there was a bug related to the "returnDirect boolean initialization and updating logic... This can cause incorrect behavior if assumptions about returnDirect are made later in the code." A participant then wrote, in agreement with the AI assistance: *"returnDirect is never updated to anything other than null. This could cause errors further down the line if future code assumes returnDirect was initialized to a non-null value"* (P7). In the no-exercise/AI condition, 72% (46/64) of participant bug descriptions aligned with the AI responses. When the AI was correct, agreement was high: 81% (13/16) for true positives (correctly identified defects) and 80% (16/20) for true negatives (correctly identified absence of defects). When the AI was incorrect, agreement remained substantial: 73% (11/15) for false negatives (missed defects) and 46% (6/13) for false positives (false alarms).

This is particularly relevant because it suggests that AI accuracy matters more in some summarization and comprehension tasks than in others. Incorrect AI still helped summarization style (RQ2), but it impaired seeded bug detection in our experiment (RQ3). An organization that prioritizes summarization (or handles bug detection separately) may thus benefit from newer or speculative AI support, even if its correctness shows a higher variance.

RQ3 — How does AI correctness influence performance?

The correctness of AI assistance strongly predicts human bug detection performance ($\beta = 2.51$, $p < 0.0001$). When AI guidance is correct, participants identify defects accurately in 81% of cases; when it is incorrect, accuracy drops to 27%. Participants frequently accept AI-provided bug information without verification, amplifying both the benefits of correct AI and the costs of hallucinations.

7 Threats to Validity

We discuss how we mitigate threats to validity.

Recruitment. Subjects who completed the study with synchronous videoconferencing supervision produced more accurate and complete summaries than those who completed it asynchronously. This provides nuance to prior findings that in-person *interruptions* increased self-reported stress [82]. In that study, researchers interrupted participants during coding tasks, whereas in our study the researcher only supervised without interacting. This indicates that participant supervision (in-person or online) may be effective for obtaining higher quality data in human studies.

Data Population. Our participants are predominantly students (see "Occupation" in Table 2), which limits generalizability to professional developers. Prior work suggests that experienced developers are better at critically evaluating AI outputs and are less affected by hallucinations [83]. Thus, the over-reliance observed here may represent an upper bound. However, 47% (22/47) of participants reported prior paid programming experience (e.g., internships). This suggests our results remain relevant to novice or early-career developers (see Section 8).

Tasks and Stimuli. Our bug detection setup may not reflect real-world tasks, where defects are not limited to single seeded bugs and developers can use debugging tools or test cases (cf. [84]). We focus on the effect of AI support and use an intentionally simplified task setting. As a result, our findings may not fully generalize to more complex or tool-supported development settings, and should be validated in industrial contexts.

Treatments: AI. Participants received static, pre-generated AI summaries rather than interacting with an AI system in real time. In real-world settings, developers may use AI iteratively, request clarifications, or refine prompts dynamically. In particular, presenting a single, fixed summary may encourage anchoring on an initial explanation [26], whereas interactive systems allow users to question, refine, or correct outputs, potentially leading to different reliance behaviors. This difference may reduce ecological validity relative to actual AI-assisted programming workflows. We mitigated this by using responses generated from a state-of-the-art LLM at the time of the experiment. Participants received representative and realistic AI-produced summaries, even if the interaction modality was not dynamic.

Treatments: Exercise. We discuss two threats associated with our exercise treatment. First, slower response times after exercise may partly reflect short-term physical fatigue rather than changes in task performance [85]. Because we did not measure fatigue directly (e.g., heart rate or perceived exertion), exercise intensity is not explicitly modeled, and

our results reflect the net effect of introducing a brief exercise break before the task. Second, the diminishing effect of exercise over repeated trials may reflect physiological recovery, variation in exercise intensity across participants, or other time-dependent factors. Our study was not designed to distinguish among these possibilities.

8 Discussion

We discuss and explore the implications of our findings.

Possible implications on when to exercise. The net negative effects of exercise on both response time and accuracy suggest that the intervention, as implemented, introduced performance costs. One interpretation is that participants operated beyond an optimal level of "arousal" (a psychology term for physiological and psychological activation linked to alertness), as cognitive performance is often characterized by an inverted U-shaped relationship with arousal or stress, where moderate levels can benefit performance but higher levels can impair it [86]. Consistent with this view, prior work shows that the cognitive effects of acute exercise vary with intensity, duration, and the timing of assessment relative to the exercise bout [77]. In our setting, tasks were performed soon after exercise, which may have placed participants in a higher-arousal or fatigued state, contributing to the observed slowdown and reduced accuracy. Although prior work does not specify an optimal recovery interval, post-exercise timing may be an important design dimension for future work.

An alternative interpretation of our findings relates to task type. Our exercise protocol is adapted from prior classroom settings in which brief exercise breaks during lectures improved attention and learning outcomes, both immediately and at delayed assessments [31]. However, those settings involve passive learning tasks with interleaved activity, whereas our study evaluates immediate performance on active tasks. This suggests that the effectiveness of exercise interventions may depend on task demands, potentially benefiting sustained activities (e.g., formal code inspection) more than tightly scoped tasks requiring immediate, precise responses (e.g., fine-grained bug detection). Based on results, we do not recommend exercise right before a short school activity (such as a timed quiz), but we leave open the possibility of using exercise for longer-duration academic tasks.

Possible implications for early-career developers. Our results suggest that AI assistance can reduce performance when its output is not carefully examined (see Section 6.2 and 6.3). Although our sample includes participants with varying levels of experience (i.e., 47% reported prior paid programming experience), it remains skewed toward students preparing to enter the workforce. Therefore, we hypothesize that these observations may point to potential challenges in academic and early professional settings. In particular, if similar patterns hold among early-career developers, over-reliance on AI assistance without critical evaluation may impact tasks that require synthesizing intent and design decisions, such as documentation creation, which plays a central role in software maintenance [1], [2]. While not our primary focus, we also did not observe differences in AI assistance usage between easier and more difficult problems, suggesting relatively uniform reliance. As AI tools become more integrated into learning environments [87], future studies should examine whether training that emphasizes interpreting and validating AI-generated content improves outcomes for students and early-career developers, and whether these findings generalize to experienced professionals.

9 Conclusion

We investigate how brief physical exercise and static AI assistance influence code summarization and bug detection tasks. To the best of our knowledge, we present the first study of student developers ($N = 47$) to investigate exercise and code summarization as well as exercise and static AI support. Although both interventions are often assumed to be beneficial, our results show that their effects are task dependent and can impose unintended costs, consistent with prior work on human-based interventions [77], [78].

Contrary to expectations drawn from studies reporting benefits of exercise for attention, learning or arithmetic [30], [31], performing coding tasks immediately after exercise does *not* improve summarization performance and instead leads to slower and lower-quality summaries, with significant declines in accuracy and conciseness ($p < 0.03$).

AI assistance shapes how developers summarize code. AI-supported summaries are more concise ($p < 0.05$), but AI assistance does *not* improve summarization accuracy. For bug detection, however, AI correctness is decisive. Correct AI guidance substantially *improves* bug detection, while incorrect guidance sharply *degrades* performance ($p < 0.0001$): uncritical reliance on AI assistance is risky [25], [26].

Overall, **these findings indicate that neither exercise nor AI assistance should be assumed to universally enhance developer performance.** For practitioners, our findings suggest that AI assistance is most valuable when its output is actively verified, particularly for tasks involving bug detection where incorrect guidance can be costly. For software engineering educators, these results motivate emphasizing critical evaluation of AI-generated content alongside traditional programming skills for students. In addition, our results, based predominately student participants, suggest caution for students using AI assistance to understand new code bases. Future work should examine how developers calibrate trust in AI systems and how interfaces can better support verification in AI assisted programming tasks.

References

[1] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2015.

[2] S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang, "A human study of comprehension and code summarization," in *International Conference on Program Comprehension*, 2020, p. 2–13.

[3] J. Sillito, G. C. Murphy, and K. De Volder, "Asking and answering questions during a programming change task," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434–451, 2008.

[4] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–37, 2018.

[5] B. Bernárdez, A. Durán, J. A. Parejo, and A. Ruiz-Cortés, "A controlled experiment to evaluate the effects of mindfulness in software engineering," in *Empirical Software Engineering and Measurement*, 2014.

[6] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Foundations of Software Engineering*, 2014, pp. 19–29.

[7] O. A. L. Lemos, F. F. Silveira, F. C. Ferrari, and A. Garcia, "The impact of software testing education on code reliability: An empirical assessment," *Journal of Systems and Software*, vol. 137, pp. 497–511, 2018.

[8] B. Johnson, T. Zimmermann, and C. Bird, "The effect of work environments on productivity and satisfaction of software engineers," *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 736–757, 2019.

[9] A. Razzaq, J. Buckley, Q. Lai, T. Yu, and G. Botterweck, "A systematic literature review on the influence of enhanced developer experience on developers' productivity: Factors, practices, and recommendations," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–46, 2024.

[10] E. Murphy-Hill, C. Jaspan, C. Sadowski, D. Shepherd, M. Phillips, C. Winter, A. Knight, E. Smith, and M. Jorde, "What predicts software developers' productivity?" *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 582–594, 2019.

[11] M. Zhou and A. Mockus, "Does the initial environment impact the future of developers?" in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 271–280.

[12] T. Ahmed and P. Devanbu, "Few-shot training llms for project-specific code-summarization," in *Automated Software Engineering*, ser. ASE '22, 2023.

[13] Y. Wang, C. Xie, W. Zhang, and X. Qiu, "FRCS-LLM: A framework for refining code summarization in large language models via pre-trained models," in *Computers, Software, and Applications Conference*, 2025, pp. 1244–1253.

[14] R. v. Kulkarni, P. Kothekar, H. Kadival, R. Khinchi, and S. Gunjal, "A comprehensive survey on code summarization," in *International Conference on Progressive Innovations in Intelligent Systems and Data Science*, 2024, pp. 437–444.

[15] L. Cao, H. He, H. Huang, J. Wang, and Y. Cai, "Rethinking-based code summarization with chain of comments," in *International Conference on Computational Linguistics*, O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, and S. Schockaert, Eds., Jan. 2025, pp. 3043–3056.

[16] J. Zhu, Y. Miao, T. Xu, J. Zhu, and X. Sun, "On the effectiveness of large language models in statement-level code summarization," in *Software Quality, Reliability and Security*, 2024, pp. 216–227.

[17] M. Fang, X. Yuan, Y. Li, H. Li, C. Fang, and J. Du, "Enhanced prompting framework for code summarization with large language models," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, Jun. 2025.

[18] Y. Zhang, C. Huang, Z. Karas, T. D. Nguyen, K. Leach, and Y. Huang, *Enhancing Code LLM Training with Programmer Attention*. Association for Computing Machinery, 2025, p. 616–620.

[19] J. Li, Y. Zhang, Z. Karas, C. McMillan, K. Leach, and Y. Huang, "Do machines and humans focus on similar code? exploring explainability of large language models in code summarization," in *International Conference on Program Comprehension*, ser. ICPC '24, 2024, p. 47–51.

[20] D. Moreau and E. Chou, "The acute effect of high-intensity exercise on executive function: a meta-analysis," *Perspectives on Psychological Science*, vol. 14, no. 5, pp. 734–764, 2019.

[21] G. Li, G. Teng, W. Zhang, T. Song, Y. Li, Z. Wang, and A. Chen, "Comparative effects of different physical exercises on cognitive function and intervention adherence in older adults with cognitive impairment: A systematic review and network meta-analysis," *Clinical Psychology Review*, p. 102604, 2025.

[22] A. Koulanova, A. Maharaj, B. Harrington, and J. Dere, "Fit-breaks: incorporating physical activity breaks in introductory cs lectures," in *Innovation and Technology in Computer Science Education*. Association for Computing Machinery, 2018, p. 260–265.

[23] C. M. Montes, F. Sjögren, A. Klevfors, and B. Penzenstadler, "Qualifying and quantifying the benefits of mindfulness practices for it workers," in *International Conference on ICT for Sustainability*, 2024, pp. 272–281.

[24] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM computing surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[25] C. Rastogi, Y. Zhang, D. Wei, K. R. Varshney, A. Dhurandhar, and R. Tomsett, "Deciding fast and slow: The role of cognitive biases in ai-assisted decision-making," *Human-Computer Interaction*, vol. 6, pp. 1–22, 2022.

[26] M. Nourani, C. Roy, J. E. Block, D. R. Honeycutt, T. Rahman, E. Ragan, and V. Gogate, "Anchoring bias affects mental model formation and user reliance in explainable ai systems," in *Proceedings of the 26th International Conference on Intelligent User Interfaces*, 2021, pp. 340–350.

[27] R. H. Baayen, D. J. Davidson, and D. M. Bates, "Mixed-effects modeling with crossed random effects for subjects and items," *Journal of memory and language*, vol. 59, no. 4, pp. 390–412, 2008.

[28] D. Bates, M. Mächler, B. Bolker, and S. Walker, "Fitting linear mixed-effects models using lme4," *arXiv preprint arXiv:1406.5823*, 2014.

[29] A. Gelman and J. Hill, *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2007, ch. 12-13.

[30] R. Balci and F. Aghazadeh, "Effects of exercise breaks on performance, muscular load, and perceived discomfort in data entry and cognitive tasks," *Computers & Industrial Engineering*, vol. 46, no. 3, pp. 399–411, 2004.

[31] B. Fenesi, K. Lucibello, J. A. Kim, and J. J. Heisz, "Sweat so you don't forget: Exercise breaks during a university lecture increase on-task attention and learning," *Journal of Applied Research in Memory and Cognition*, vol. 7, no. 2, pp. 261–269, 2018.

[32] J. A. Nastasi, I. B. Tassistro, and N. E. Gravina, "Breaks and productivity: An exploratory analysis," *Journal of Applied Behavior Analysis*, vol. 56, no. 3, pp. 539–548, 2023.

[33] A. A. Bennett, A. S. Gabriel, and C. Calderwood, "Examining the interplay of micro-break durations and activities for employee recovery: A mixed-methods investigation," *J. Occup. Health Psychol.*, vol. 25, no. 2, pp. 126–142, Apr. 2020.

[34] Y. Liu, Q. Gao, and L. Ma, "Taking micro-breaks at work: Effects of watching funny short-form videos on subjective experience, physiological stress, and task performance," in *Cross-Cultural Design. Applications in Arts, Learning, Well-being, and Social Development*, P.-L. P. Rau, Ed., 2021, pp. 183–200.

[35] M. v. Vugt, *Mindfulness as a Potential Tool for Productivity*. Apress, 2019, pp. 293–302.

[36] V. Volobuev, P. Turischeva, A. Gainutdinov, R. Sahibgareev, M. Mazzara, and M. Farina, "Effects of mindfulness meditation on software developers' performance," in *2021 International Conference "Nonlinearity, Information and Robotics" (NIR)*, 2021, pp. 1–6.

[37] S. McLoughlin, Z. Karas, R. Wallace, A. Bansal, C. McMillan, and Y. Huang, "Programmers' visual attention on function call graphs during code summarization," in *Automated Software Engineering*, 2025.

[38] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An eye-tracking study of java programmers and application to source code summarization," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1038–1054, 2015.

[39] N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, and J. I. Maletic, "Developer reading behavior while summarizing Java methods: Size and context matters," in *International Conference on Software Engineering*, 2019, pp. 384–395.

[40] Z. Karas, A. Bansal, Y. Zhang, T. Li, C. McMillan, and Y. Huang, "A tale of two comprehensions? analyzing student programmer attention during code summarization," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 7, Aug. 2024.

[41] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr, "Automatic semantic augmentation of language model prompts (for code summarization)," in *International Conference on Software Engineering*, 2024.

[42] Y. Wu, Y. Wan, Z. Chu, W. Zhao, Y. Liu, H. Zhang, X. Shi, H. Jin, and P. S. Yu, "Can large language models serve as evaluators for code summarization?" *IEEE Transactions on Software Engineering*, pp. 1–12, 2025.

[43] G. Crupi, R. Tufano, A. Velasco, A. Mastropaolo, D. Poshyvanyk, and G. Bavota, "On the effectiveness of llm-as-a-judge for code generation and summarization," *IEEE Transactions on Software Engineering*, vol. 51, no. 8, pp. 2329–2345, 2025.

[44] R. Wang, J. Guo, C. Gao, G. Fan, C. Y. Chong, and X. Xia, "Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, Jun. 2025.

[45] Y. Virk, P. Devanbu, and T. Ahmed, "Calibration of large language

models on code summarization," *Proc. ACM Softw. Eng.*, vol. 2, no. FSE, Jun. 2025.

[46] J. Shen, X. Sun, B. Li, H. Yang, and J. Hu, "On automatic summarization of what and why information in source code changes," in *Computer Software and Applications Conference*, vol. 1, 2016, pp. 103–112.

[47] P. Sukkasem, C. Soomlek, and C. Dechsupa, "Llm-based code comment summarization: Efficacy evaluation and challenges," in *Knowledge and Smart Technology*, 2025, pp. 335–340.

[48] Y. Zhang, Y. Li, M. Fang, X. Yuan, and J. Du, "BRMDS: an LLM-based multi-dimensional summary generation approach for bug reports," *Automated Software Engineering*, vol. 33, no. 1, p. 10, Sep. 2025.

[49] H. Wang, X. Xia, D. Lo, J. Grundy, and X. Wang, "Automatic solution summarization for crash bugs," in *International Conference on Software Engineering*, 2021, pp. 1286–1297.

[50] S. Jiang, J. Zhang, W. Chen, B. Wang, J. Zhou, and J. Zhang, "Evaluating fault localization and program repair capabilities of existing closed-source general-purpose llms," in *Workshop on Large Language Models for Code*, 2024, p. 75–78.

[51] A. Z. H. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large language models for test-free fault localization," in *International Conference on Software Engineering*, ser. ICSE '24, 2024.

[52] C. Xu, Z. Liu, X. Ren, G. Zhang, M. Liang, and D. Lo, "Flexfl: Flexible and effective fault localization with open-source large language models," *IEEE Transactions on Software Engineering*, vol. 51, no. 5, pp. 1455–1471, 2025.

[53] Y. Liu, H. Liu, Z. Yang, Z. Li, and Y. Liu, "Empirical evaluation of large language models for novice program fault localization," in *Software Quality, Reliability and Security*, 2024, pp. 180–191.

[54] R. Widyasari, J. W. Ang, T. G. Nguyen, N. Sharma, and D. Lo, "Demystifying faulty code: Step-by-step reasoning for explainable fault localization," in *Software Analysis, Evolution and Reengineering*, 2024, pp. 568–579.

[55] D. Khati, Y. Liu, D. N. Palacio, Y. Zhang, and D. Poshyvanyk, "Mapping the trust terrain: Llms in software engineering - insights and perspectives," *ACM Trans. Softw. Eng. Methodol.*, Oct. 2025.

[56] S. Ferino, R. Hoda, J. Grundy, and C. Treude, "Novice developers' perspectives on adopting llms for software development: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, Mar. 2026.

[57] P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, and O. Nierstrasz, "A decade of code comment quality assessment: A systematic literature review," *Journal of Systems and Software*, vol. 195, p. 111515, 2023.

[58] L. DiPietro, S. S. Al-Ansari, S. J. Biddle, K. Borodulin, F. C. Bull, M. P. Buman, G. Cardon, C. Carty, J.-P. Chaput, S. Chastin *et al.*, "Advancing the global physical activity agenda: recommendations for future research by the 2020 who physical activity and sedentary behavior guidelines development group," *International Journal of Behavioral Nutrition and Physical Activity*, vol. 17, no. 1, p. 143, 2020.

[59] R. S. Alqhtani, H. Ahmed, A. Alshahrani, A. R. Khan, and A. Khan, "Effects of whole-body stretching exercise during lunch break for reducing musculoskeletal pain and physical exertion among healthcare professionals," *Medicina*, vol. 59, no. 5, 2023.

[60] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Comput. Surv.*, vol. 55, no. 9, Jan. 2023.

[61] OpenAI. (2024) Hello gpt-4o. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[62] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, Mar. 2023.

[63] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. M. German, "Open source-style collaborative development practices in commercial projects using github," in *2015 IEEE/ACM 37th IEEE international conference on software engineering*, vol. 1. IEEE, 2015, pp. 574–585.

[64] M. Wyrich, J. Bogner, and S. Wagner, "40 years of designing code comprehension experiments: A systematic mapping study," *ACM Comput. Surv.*, vol. 56, no. 4, Nov. 2023.

[65] J. Terrell, A. Kofink, J. Middleton, C. Rainear, E. Murphy-Hill, C. Parnin, and J. Stallings, "Gender differences and bias in open source: Pull request acceptance of women versus men," *PeerJ Computer Science*, vol. 3, p. e111, 2017.

[66] Y. Huang, K. Leach, Z. Sharafi, N. McKay, T. Santander, and W. Weimer, "Biases and differences in code review using medical imaging and eye-tracking: genders, humans, and machines," in *Foundations of Software Engineering*, 2020, p. 456–468.

[67] Z. P. Fry and W. Weimer, "A human study of fault localization accuracy," in *2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–10.

[68] J. C. Pinheiro and D. M. Bates, *Mixed-effects models in S and S-PLUS*. Springer, 2000.

[69] K. P. Burnham and D. R. Anderson, *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2002, ch. 2-3.

[70] E.-J. Wagenmakers and S. Farrell, "Aic model selection using akaike weights," *Psychonomic bulletin & review*, vol. 11, no. 1, pp. 192–196, 2004.

[71] J. E. Helmreich, "Regression modeling strategies with applications to linear models, logistic and ordinal regression and survival analysis," *Journal of Statistical Software*, vol. 70, pp. 1–3, 2016.

[72] H. Matuschek, R. Kliegl, S. Vasishth, H. Baayen, and D. Bates, "Balancing type i error and power in linear mixed models," *Journal of memory and language*, vol. 94, pp. 305–315, 2017.

[73] D. J. Barr, R. Levy, C. Scheepers, and H. J. Tily, "Random effects structure for confirmatory hypothesis testing: Keep it maximal," *Journal of memory and language*, vol. 68, no. 3, pp. 255–278, 2013.

[74] R. Ratcliff, "Methods for dealing with reaction time outliers." *Psychological bulletin*, vol. 114, no. 3, p. 510, 1993.

[75] C. E. Hill, B. J. Thompson, and E. N. Williams, "A guide to conducting consensual qualitative research," *The Counseling Psychologist*, vol. 25, no. 4, pp. 517–572, 1997.

[76] S. Nakagawa and H. Schielzeth, "A general and simple method for obtaining r2 from generalized linear mixed-effects models," *Methods in ecology and evolution*, vol. 4, no. 2, pp. 133–142, 2013.

[77] M. Sudo, J. T. Costello, T. McMorris, and S. Ando, "The effects of acute high-intensity aerobic exercise on cognitive performance: A structured narrative review," *Frontiers in behavioral neuroscience*, vol. 16, p. 957677, 2022.

[78] T. Ishihara, E. S. Drollette, S. Ludyga, C. H. Hillman, and K. Kamijo, "The effects of acute aerobic exercise on executive function: A systematic review and meta-analysis of individual participant data," *Neuroscience & Biobehavioral Reviews*, vol. 128, pp. 258–269, 2021.

[79] Z. Liu, X. Xia, C. Treude, D. Lo, and S. Li, "Automatic generation of pull request descriptions," in *Automated Software Engineering*, 2020, p. 176–188.

[80] B. J. Fogg, C. Soohoo, D. R. Danielson, L. Marable, J. Stanford, and E. R. Tauber, "How do users evaluate the credibility of web sites? a study with over 2,500 participants," in *Designing for User Experiences*, 2003, pp. 1–15.

[81] M. O. Ahmad, O. Al-Baik, M. A. Al-haija, A. Husein, and M. Morales-Trujillo, "Artificial intelligence learning paradox: How over reliance on it undermines mastery in agile software project management," *Procedia Computer Science*, vol. 278, pp. 1877–1884, 2026.

[82] Y. Ma, Y. Huang, and K. Leach, "Breaking the flow: A study of interruptions during software engineering activities," in *International Conference on Software Engineering*, ser. ICSE '24, 2024.

[83] J. Chen, X. Lu, Y. Du, M. Rejtig, R. Bagley, M. Horn, and U. Wilensky, "Learning agent-based modeling with llm companions: Experiences of novices and experts using chatgpt & netlogo chat," in *Human Factors in Computing Systems*. Association for Computing Machinery, 2024.

[84] R. Amankwah, J. Chen, H. Song, and P. K. Kudjo, "Bug detection in java code: An extensive evaluation of static analysis tools using juliet test suites," *Software: Practice and Experience*, vol. 53, no. 5, pp. 1125–1143, 2023.

[85] R. D. Moore, M. W. Romine, P. J. O'connor, and P. D. Tomporowski, "The influence of exercise-induced fatigue on cognitive function," *Journal of Sports Sciences*, vol. 30, no. 9, pp. 841–850, 2012.

[86] S. Nieuwenhuis, "Arousal and performance: revisiting the famous inverted-u-shaped curve," *Trends in cognitive sciences*, vol. 28, no. 5, pp. 394–396, 2024.

[87] T. Abbas, S. A. Rathore, A. Turki, S. Khan, O. Alghushairy, and A. Daud, "Enhancing software engineering with AI: Innovations, challenges, and future directions," *IET Software*, vol. 5691460, 2025.