High Expectations: An Observational Study of Programming and Cannabis Intoxication

Wenxin He
wenxinhe@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Manasvi Parikh
manasvi@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Westley Weimer
weimerw@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Madeline Endres
endremad@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

ABSTRACT

Anecdotal evidence of cannabis use by professional programmers abounds. Recent studies have found that some professionals regularly use cannabis while programming, even for work-related tasks. However, accounts of the impacts of cannabis on programming vary widely and are often contradictory. For example, some programmers claim that it impairs their ability to generate correct solutions, while others claim it enhances creativity and focus. There remains a need for an empirical understanding of the true impacts of cannabis on programming. This paper presents the first controlled observational study of cannabis's effects on programming ability. Based on a within-subjects design with over 70 participants, we find that, at ecologically valid dosages, cannabis significantly impairs programming performance. **Programs implemented while high contain more bugs and take longer to write ($p < 0.05$)** — a small to medium effect ($0.22 \leq d \leq 0.44$). We also did not find any evidence that high programmers generate more divergent solutions. However, programmers can accurately assess differences in their programming performance ($r = 0.59$), even when under the influence of cannabis. We hope that this research will facilitate evidence-based policies and help developers make informed decisions regarding cannabis use while programming.

CCS CONCEPTS

• **Social and professional topics** → User characteristics; *Governmental regulations*; Testing, certification and licensing; Employment issues; • **Human-centered computing** → *User studies*; • **Software and its engineering** → *Software development techniques.*

KEYWORDS

programming preferences, cannabis, controlled user study, drug policy, preregistered hypotheses

ACM Reference Format:
Wenxin He, Manasvi Parikh, Westley Weimer, and Madeline Endres. 2024. High Expectations: An Observational Study of Programming and Cannabis Intoxication. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3597503.3639145

1 INTRODUCTION

Software developers commonly use psychoactive substances while programming [15], a behavior at odds with many company drug and hiring policies [35]. *Cannabis sativa* (or "cannabis") is the world's most common illicit substance, used by more than 192 million people in 2018 [48] and representing a market of 20.5 billion USD [53]. Claims, biases, and folk wisdom about the actual effects of cannabis intoxication on programming do not agree. This lack of a firm understanding prevents individuals and companies alike from making informed policy decisions and accurately balancing risk and reward.

A 2022 study of 800 programmers (including 450 full-time developers) found that 35% had used cannabis while programming, and that 18% did so at least once per month [15]. Motivations included enjoyment, thinking of more creative programming solutions, enhancing brainstorming, and improving focus [15, Tab. 3]. A 2023 study interviewing 25 professional programmers who used psychoactive substances reported positive views on the impact of cannabis on brainstorming, neutral views on coding and testing, and negative views on debugging, design, and documentation [35, Tab. III]. Psychoactive substances including cannabis are seen as part of the historical tradition of software [32], with advocates touting focus [4] and creativity [51] benefits [50].

In broader contexts, views may be more negative. Cannabis intoxication can impair decision-making accuracy and consistency, resulting in losses being under-estimated ("treating each loss as a constant and minor negative outcome regardless of the size of the loss") [20]. It can also impair motor control and reaction times. For example, "acute cannabis intoxication is associated with a statistically significant increase in motor vehicle crash risk" [41]. These more-negative views often inform both general and software-specific regulation. Anti-cannabis hiring and retention policies are prevalent in software companies (e.g., [26, pp. 12–13] and [9, p. 12]) and almost one-third of developers reported taking a drug test for a programming job [15, Sec. 6]. However, questions have been raised about the efficacy of such policies [35, Sec. IX.B] and whether they are either needed or beneficial [27] in a modern context.

Understanding the impacts of cannabis intoxication on particular aspects of software development would help fill this knowledge gap, allowing for evidence-based corporate policies and informed decisions by developers regarding when and if to use cannabis while programming. An effective understanding would be based on an indicative sample size, ecologically-valid conditions and both quantitative and qualitative aspects of produced software. In addition, any such study must be conducted ethically, given the rapidly-changing legal landscape around cannabis use.

We conducted the first rigorous observational study of the effects of cannabis intoxication on programming. We assessed $n = 74$ participants from multiple American metropolitan areas across four states and used *pre-registered hypotheses* to mitigate researcher bias. Each participant completed two sessions on different days, one while sober and one while using cannabis, in a *randomly-assigned* order. This design permits both within-subject and between-subjects comparisons. In each session, participants completed both short targeted programming tasks as well as multiple LeetCode [2] problems using a standard development environment (Visual Studio) on their personal computers. In the cannabis condition, participants were asked to use the amount they would normally use while programming, an *ecologically-valid* context that allows us to learn actionable insights. Our contributions are:

(1) The first rigorous observational study of programming in both sober and cannabis-intoxicated conditions.
(2) We find that ecologically-valid cannabis intoxication has a **small to medium effect on program correctness**: programs written by cannabis-intoxicated programmers exhibit *more bugs*, failing 10% more test cases, on average ($p < 0.05$).
(3) Cannabis-intoxicated programmers take *more time* to write non-trivial functions than do sober programmers (11% more on time average, $p = 0.39$).
(4) Cannabis-intoxicated programmers exhibit *different typing patterns*, including deleting and rewriting code more frequently and pausing for longer without typing ($p \leq 0.003$, $d \geq 0.35$).
(5) Despite anecdotes of cannabis improving creativity, we observe no evidence that cannabis-intoxicated programmers make different algorithmic or stylistic programming choices.
(6) Cannabis-using programmers accurately recognize their programming performance, even when intoxicated, $r = 0.59$.

The low impact of cannabis compared to individual differences, and the ability of developers to recognize cannabis impairment, suggest caution when crafting anti-cannabis policies (see Section 7).

To the best of our knowledge, this is the first study of how of cannabis intoxication impacts programming. We believe aspects of our design (pre-registered hypotheses, ecologically-valid settings, broad participant pool, etc.) make it generalizable and useful for informing policies and individual developer decisions surrounding cannabis. We make our (de-identified) replication package available, including raw data, stimuli, analysis scripts, and design documents.

2 BACKGROUND AND RELATED WORK

In this section we briefly present related work on cannabis intoxication, computing-related cannabis use, and software creativity.

Impacts of Cannabis Intoxication. Cannabis is well-known for its mind-altering effects. In particular, acute cannabis use (especially for non-heavy users) can impair various cognitive processes such as memory and learning, attention control, fine motor control, and emotion processing [29]. The use of cannabis in medical contexts (e.g., to treat chronic pain) has been examined in the literature, including via observational studies [5]; we focus here on cognitive aspects such as memory, fine motor control, decision-making, and creativity [20] that are related to programming [46].

Acute cannabis use impairs memory and learning as well inhibiting motor responses and reaction times [6, 29]. These cognitive processes are used while programming (e.g., during code comprehension tasks [46] or while typing code [30]). The impact of cannabis on other programming-related cognitive processes is less understood. Recent reviews report insufficient evidence of how cannabis impairs working memory or decision making [6, 29], both essential to software [11, 38, 46]. Similarly, cannabis's impact on creativity is the subject of contradictory claims: For example, LaFrance *et al.* found that cannabis users exhibit higher creativity due to increased openness to experiences [31] while Kowal *et al.* found that cannabis impairs aspects of creativity at high dosages [28]. These conflicting claims preclude using a "bottom up" approach to infer the impact of cannabis on programming by considering its impact on relevant cognitive processes. We instead use a "top down" approach, studying the relationship between cannabis use and programming performance directly in a real-world context.

Cannabis and Programming. Endres *et al.* surveyed 800 developers. Their results provide a general context of the landscape of cannabis use in software engineering [15]. They report that over one-third of their sample had used cannabis while programming and over one-sixth did so at least once per month. They also reported on cannabis use during work-related tasks. Key motivations were not medicinal but instead focused on "perceived enhancement" to software development skills. Newman *et al.* conducted interviews of 26 developers, identifying themes related to soft skills, social stigma, and organizational policies [35]. Critically, both of these prior studies rely on *self-reported* impacts (e.g., cannabis-using programmers say that cannabis enhances or hurts their programming).

Software Creativity. Researchers have identified creativity as important to multiple aspects of software engineering, from requirements elicitation [36] to Agile development [10]. Creativity in software engineering is often associated with the novelty and effectiveness of the solutions [34] and approaching problems from different angles [24]. Models and reviews have identified knowledge [25], communication [24], happiness [23], or personality [1, 24] as factors relevant to creativity. Our study looks at divergent method-level implementation choice as one piece of creativity.

3 EXPERIMENTAL SETUP AND DESIGN

To understand the impact of cannabis intoxication on programming performance, we conduct a controlled observational study with 74 participants. Eligible participants were at least 21, had used cannabis in the last year, and had smoked or vaped cannabis before. We also required Python familiarity and programming experience comparable to that of a senior undergraduate. We first explain our study design in more detail. Then, in Section 3.2, we describe the surveys, programming tasks, and metrics used in our experiment.

3.1 Study Design

We had three main design considerations: achieving sufficient statistical power for our pre-registered hypotheses (see Section 5.1), balancing ecological validity with experimental control, and maximizing participant privacy and safety.

Overall Study Design. To maintain statistical power, we use a within-subjects design. Participants completed a 20-minute questionnaire with demographics, cannabis usage history, programming history, and a four-minute training video introducing the study platform. Next, they attended two structurally-identical programming sessions: one cannabis-intoxicated and one sober. The session order was counterbalanced: participants were randomly assigned the cannabis-first or sober-first condition (35/71 cannabis first vs. 36/71 sober first). This counterbalanced and within-subject design mitigates the impact of individual differences in programming ability, cannabis tolerance, and session ordering effects in our analysis.

Study Session Structure. All programming sessions lasted 1.5 hours and included two cognitive assessments, a series of short programming problems, and three "interview-style" coding questions. We included cognitive assessments for data validation (see replication package), short programming problems for controlled observations of the impact of cannabis on programming, and "interview-style" coding questions to capture more complex coding algorithmic options. The short programming problems permit a controlled investigation of the impact of cannabis on specific aspects of programming while the "interview-style" questions enable a holistic and ecologically-valid analysis at the expense of statistical power and experimental control. Section 3.2 details all experimental tasks.

For the short programming problems, participants completed an online Qualtrics survey, a platform that permits randomization and timing collection via custom JavaScript. For the "interview-style" questions, participants wrote and executed code using a browser-based instance of VSCode (a popular programming text editor) via a Github Codespace configured to collect keystrokes, terminal/compiler interactions, and program file contents. Participants were also permitted to search (Google) for help with syntax errors. This design allowed our programming environment to have higher ecological validity, while remaining controlled enough to permit straight-forward statistical analysis. Sessions were conducted remotely (via Zoom) and participants were required to screen share.

Cannabis Session Logistics. Participants used cannabis 10–15 minutes before the start of the session, then uploaded pictures of the product and indicated the amount. Participants were instructed to consume cannabis via vaping or smoking, rather than by taking an edible to reduce variability: edibles can have very different effects than vaping or smoking [7]. We chose vaping based on its popularity in the the supplemental data from the Endres *et al.* report [15].

Participants who had used cannabis while programming before were asked to use the amount they would typically use when programming. If they had not, they were asked to use a mild to medium dosage, consistent with the amount they use when not programming. Allowing participants to choose the amount of cannabis to consume is different than the approach taken by many studies of cannabis use on behavior or cognition (cf. [12]). However, allowing participants to self-select their usual cannabis dosage improves ecological validity. Informally, in this study we are not interested in learning the amount of impairment per milligram of cannabis, but instead in the amount of impairment an average cannabis-intoxicated programmer faces while programming. We view the latter as significantly more actionable (e.g., to managers).

Ethical Considerations. The varying legality of cannabis requires special care in study design. Throughout the design and implementation of this study, we worked with our IRB (ethical review board) to ensure participant privacy and safety. Additional safety precautions were incorporated into our protocol including requiring that participants have used cannabis in the last year, and participating using a personally-owned computer (e.g., not a company-distributed laptop that may have tracking software) from a location where they would not have to travel for several hours.

Ethical research practice also influenced our selection of a remote study design, rather than an in-person lab study. Design decisions such as directly administering specific amounts of cannabis at a central location or analyzing blood samples to assess intoxication were considered and rejected given our focus on ecological validity, as well as for reasons related to logistics and privacy.

3.2 Surveys and Stimuli: Content and Metrics

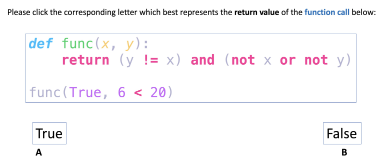
We describe our survey instruments, programming tasks, and metrics. All surveys and stimuli are in our replication package.[1]

3.2.1 Demographics. We collected demographics such as gender, age, and employment status. We also asked programming experience, cannabis usage history, and prior experiences using cannabis while programming. Questions about programming experience and prior cannabis use while programming were adapted from a published survey [15] to admit comparison with prior work. For general cannabis usage, we used the validated *Daily Sessions, Frequency, Age of Onset, and Quantity of Cannabis Use Inventory* (DFAQ-CU) [13].

3.2.2 Short Programming Problem Stimuli. Each session included a series of short programming questions. We adapted stimuli from the programming comprehension literature, specifically those that use neuroimaging [17, 30]. We chose to do this because such stimuli are explicitly designed to study targeted programming aspects in a controlled manner while still being completed quickly. We included three types of programming tasks: Boolean questions, code-tracing questions, and code-writing questions (see Figure 1 for examples). The Boolean and code-tracing stimuli were adapted from a study of novice programmer cognition [17], while the code-writing stimuli were adapted from a study of code writing and prose writing [30].

Consistent with their design and use in program comprehension studies, each stimulus was timed: participants had at most 30 seconds for each Boolean problem, 45 seconds for each code-tracing problem, and 90 seconds for each code-writing problem. In each session, each participant completed six problems from each sub-type that were randomly sampled from our corpus.

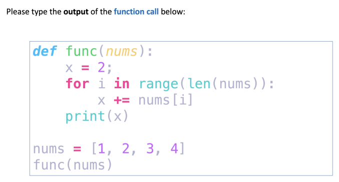
[1]The replication package can be found through the OSF pre-registration (available here: https://osf.io/g6fds). A living repository for the project can be found on GitHub at https://github.com/CelloCorgi/CannabisObservationalStudy.



Please click the corresponding letter which best represents the **return value** of the function call below:

```
def func(x, y):
    return (y != x) and (not x or not y)

func(True, 6 < 20)
```

(a) Boolean problem (answer is False)



Please type the **output** of the function call below:

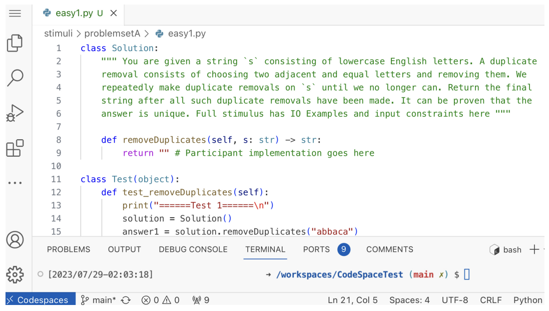
```
def func(nums):
    x = 2;
    for i in range(len(nums)):
        x += nums[i]
    print(x)

nums = [1, 2, 3, 4]
func(nums)
```

(b) Code-tracing problem (answer is 12)

Implement a function `containsDuplicate` that accepts a list of integers and returns true if it contains a duplicate element, and false otherwise.

(c) Code writing problem

Figure 1: Example short programming stimuli, adapted from the program comprehension literature.



```
class Solution:
    """ You are given a string `s` consisting of lowercase English letters. A duplicate
    removal consists of choosing two adjacent and equal letters and removing them. We
    repeatedly make duplicate removals on `s` until we no longer can. Return the final
    string after all such duplicate removals have been made. It can be proven that the
    answer is unique. Full stimulus has IO Examples and input constraints here """

    def removeDuplicates(self, s: str) -> str:
        return "" # Participant implementation goes here

class Test(object):
    def test_removeDuplicates(self):
        print("======Test 1======\n")
        solution = Solution()
        answer1 = solution.removeDuplicates("abbaca")
```

```
○ [2023/07/29-02:03:18]                  → /workspaces/CodeSpaceTest (main ✗) $
```

Figure 2: Example "interview-style" programming stimulus, presented in the study platform (shortened for space).

Metrics and Scoring. Boolean problems were graded automatically. Code-tracing and code-writing problems were assessed manually by marking responses as either correct (full score), partially correct (half score), or incorrect (zero). Manual assessment of responses was done without knowing if the response was produced while high or sober. Our full rubric is available in our replication package. Graded responses were aggregated into percent-correct values per sub-type per programming session for use in analysis.

3.2.3 "Programming Interview" Style Stimuli. Each session included three "Programming Interview" problems. These were taken directly from LeetCode, a popular platform for practicing technical interview skills. Performance on these coding challenges has been found to reflect a software engineer's fundamental computer science knowledge, ability to find efficient and scalable algorithms for unknown problems, and skill at testing or debugging a short piece of code [33]. Each problem consisted of a natural language specification, a function stub, and 2–3 basic tests. They admit implementation in a file where programmers can type, run, and edit code. We recorded the state of each program every 15 seconds, along with all keystrokes and terminal interactions. Figure 2 shows an example of the study platform and an indicative stimulus.

To choose our stimuli, we first selected 20 potential problems labeled "easy" or "medium" on LeetCode. We avoided those marked "hard" due to the time constraints imposed by our study design. We categorized these problems into three groups by their primary data structure: a 1D-array, 2D-array, or a recursive data structure (list or tree). Through a series of pilot experiments, we selected six problems (two of each type) that were non-trivial but could be completed in the available time. In each session, participants completed one each of the two 1D-array, recursive, and 2D-array problems. A participant never saw the same problem twice. Problem order was randomized across sessions and study conditions to minimize between-problem variance and learning effect impacts. We partially controlled the difficulty of the problem pairs using the LeetCode problem difficulty and solve rate. For example, the two 1D-array and two recursive problems were "easy" on LeetCode, while the two 2D-array problems were marked as "medium". Participants had 15 minutes for each easy problem (1D-array, recursive) and 20 minutes for the medium problem (2D-array).

Automated Metrics and Scoring. We assessed correctness via held-out test suites. As LeetCode does not publish its own hidden tests, we constructed our own held-out *correctness tests*. The number of correctness test cases ranged from 24 to 34 per problem. All held-out test suites achieved full branch coverage on the published LeetCode solution. We analyzed the maximum correctness score across all saved file versions for a given solution. We use the best score rather than the final score because in cases where a participant ran out of time, the last score often is much lower because the participant was mid-edit. We also made 11 additional *efficiency tests* per problem. These consisted of inputs of increasing size (including very large inputs), and were run separately from the correctness tests to ascertain the run-time efficiency of correct solutions.

Manual Annotation: We qualitatively analyzed the 1D-array problem solutions to permit a nuanced analysis of design and code style factors that may differentiate high and sober programmers. We annotate algorithmic method choices (e.g., brute force, dynamic programming, etc.), and code style features (e.g., comments, helper functions, etc.). To determine algorithmic method categories, one author manually clustered the most up-voted Python solutions on LeetCode. This initial set was validated by another author. A third author manually assigned participant solutions into these clusters. Solutions about which the annotator was unsure were shown to the others and final categorizations were confirmed via consensus.

Problem Difficulty. We validated that the two alternate problems in each pairing had similar difficulties for our population, regardless of LeetCode labels, as a potential source of bias. We used a within-subjects t-test of participant scores. For the 1D-array and recursive pairings, we find no significant differences in difficulty (p = 0.82, 0.66 respectively). For the 2D-array problems, however, there is a significant difference in difficulty (68% vs 34% on average, $p < 0.00001$). We consider this discrepancy when interpreting our results for the 2D-array problems in Sections 6.1 and 6.2.

Table 1: Demographics and experience for $n = 74$ participants.

Gender	
Man	53 (72%)
Woman	15 (20%)
Non-binary	6 (8%)
Age and Programming Experience (Average, (Min–Max))	
Age	24, (20–49)
Programming experience (years)	5,[2] (1–30)
Has 1+ years of professional programming experience	48 (65%)
Computing-related employment status (could select multiple)	
Currently Employed at a CS-related job	28 (38%)
Undergraduate Student in CS related field	37 (50%)
Graduate Student in CS-related field	12 (16%)
Unemployed or N/A	3 (4%)

4 PARTICIPANT OVERVIEW

We overview our recruitment process and then describe the demographics, programming background, and cannabis usage history of our final $n = 74$ participants to contextualize our results.

4.1 Recruitment Process

Participants were recruited via flyers posted around four American metropolitan areas (San Francisco Bay Area, Ann Arbor, Seattle, and New Haven). Each area is in a US state where recreational cannabis is legal (California, Michigan, Washington, and Connecticut). Within these areas, posters were primarily placed near the offices of technology companies and on university campuses. Prospective participants were directed to an online pre-screening form for eligibility requirements (Section 3.1). This verification included eight programming questions to ensure sufficient Python performance. Of the 640 who completed the pre-screening, 247 obtained the perfect score required to participate. Participants were contacted in batches on a first-come-first-serve basis, with preference to those with more professional programming experience. In total, we sent 205/247 invitation emails before closing recruitment.

Of the 205 invited, 85/205 finished the initial survey and scheduled programming sessions, a response rate of 42%. 74 attended at least one session and 71 attended both, a study retention rate of 84%. This overall rate aligns with previous software engineering studies with multiple sessions [16, 19]. Upon completion of the study, participants were compensated with an 80 USD gift-card to the company of their choice. All data collection occurred in 2023.

4.2 Population Contextualization

To better contextualize our results, we now describe the demographics, programming experiences, and cannabis use histories of our population. A summary of these numbers is available in Table 1.

Participants ranged in age from 20 to 49 (average 24), with 72% men, 15% women, and 8% non-binary. Our population had a mix of students and full time professional developers. About half (37/74) were undergraduates in a computing field. The remainder were either graduate students (16%, 12/74) or professional programmers (38%, 28/74). A few reported both student status and current programming employment. This split between students and professionals is consistent with the locations we put up recruitment posters.

Participants reported a range of programming experiences. While all had at least one year of programming experience (a requirement to participate), eight participants had over 10 years of experience, and the median was 5.[2] As for professional programming experience, participants ranged between none and over 20 years. The majority of participants in our sample had professional programming experience (94%), with the median participant reporting between 1 and 5 years: 65% had at least one year of professional experience. Of those who had at least one year, the most common job titles reported were "software engineer" or "software developer".

Cannabis Usage History. All eligible participants had used cannabis in the last year. Specific usage levels varied substantially, ranging from once a year to more than once every day. The median participant reported using twice a week. In addition, the majority of our participants had experience programming while high (66%, 49/74). The frequency of this use ranged from under once a year to five or more days a week. The wide array of cannabis usage histories, both with programming and without, allows us to systematically investigate if the magnitude of prior cannabis use mediates the impacts of cannabis intoxication on programming performance.

5 RESEARCH QUESTIONS AND ANALYSIS

We structure our analysis in two parts: a primary hypothesis-driven analysis and a secondary exploratory analysis. The primary hypotheses and analysis plan was preregistered to mitigate biases and increase confidence in our results. However, as investigating the impact of cannabis on programming is relatively novel, we perform additional exploratory analysis to glean insights for further study. In the rest of this section, we present our research questions for both analysis parts, and outline our statistical methods.

5.1 Primary Analysis: Pre-registered Questions

Following more recent best practices in both software engineering (e.g., the "Registered Reports" track of Mining Software Repositories [45]) and psychology and the social sciences (e.g., [47]), we *pre-registered* our hypotheses before conducting our analysis.

In pre-registration, the "research rationale, hypotheses, design and analytic strategy" are submitted before beginning the study [22]. As a result, biases associated with researchers choosing which results to present after the fact may be mitigated: "pre-registration can prevent or suppress HARKing, p-hacking, and cherry picking since hypotheses and analytical methods have already been declared before experiments are performed" [54]. Similarly, pre-registration may mean that "researchers will not be motivated to engage in practices that increase the likelihood of making a type I error" [22].

For this study, we pre-registered four research questions and associated hypotheses with the Open Science Framework (OSF), along with our data collection strategy and statistical analysis methods:[3]

[2] Estimate: Years of experiences was reported in ranges (e.g., 1-2, 3-5, 6-10, etc.)
[3]OSF pre-registration is available here: https://osf.io/g6fds. We note that in the pre-registration, *RQ3* mentions *creativity* instead of *program method divergence*. As suggested by our reviewers, we change the name in this paper to better match our methods.

- **RQ1—Program Correctness:** How does cannabis intoxication while programming impact program correctness?
  - *Hypothesis:* Programs will be less correct when written by cannabis-intoxicated programmers.
- **RQ2—Program Speed:** How does cannabis intoxication while programming impact programming speed?
  - *Hypothesis:* Cannabis-intoxicated programmers will take longer to write programs.
- **RQ3—Program Method Divergence:** Does cannabis use influence programmer algorithmic method choice?
  - *Hypothesis 1:* Correct programs by high participants will run slower than those by sober participants (i.e. are less efficient or have higher algorithmic complexity).[4]
  - *Hypothesis 2:* Solutions to free-form programming problems by cannabis-intoxicated programmers will exhibit greater method choice divergence and diversity.
- **RQ4—Cannabis Use History:** Does cannabis usage history mediate intoxication's on programming outcomes?
  - *Hypothesis:* The impact of cannabis use while programming will be lessened for heavy vs. moderate users.

5.2 Exploratory Analysis

We also consider two exploratory research questions:

- **E-RQ1—Code Style:** Does cannabis intoxication impact stylistic code properties (e.g., code comments, etc.)?
- **E-RQ2—Self Perception:** Are programmers able to accurately assess how cannabis impacts programming performance?

5.3 Statistical Methods

Our analysis was primarily conducted in a Python Jupyter Notebook using Pandas [52]. Some analyses, especially those informed by data visualization, were done using Excel. For statistical tests, we primarily used the SciPy [49] and Statsmodels [44] libraries.

Significance. We consider results significant if $p < 0.05$. When testing for a significant difference between sober and cannabis-intoxicated programming using continuous variables (e.g., percent correctness scores or response time such as in RQ1 and RQ2), we use a paired samples t-test unless otherwise noted. Assumptions of normality are confirmed through inspection of histograms. While we primarily use paired tests (as is appropriate with our within-subjects design), in some cases (e.g., missing data, etc.) we use a non-paired test and note the specific test used in the text.

For categorical values (e.g., program method choice in RQ3), we use a χ^2-test. For the difference between two binary variables (such as if a solution has comments or not in E-RQ1), we use the n-1 χ^2-test (i.e., the proportions z-test) [8]. We treat the responses to Likert questions (e.g., self-perception of cannabis impact in E-RQ2) as continuous variables.[5] The Student's t-test is thus appropriate.

Multiple Comparisons. We investigate multiple research questions and conduct multiple statistical tests per research question. To avoid fishing or p-hacking, we pre-registered our primary hypotheses and analysis plan and report results for each. Within each research question, we correct for multiple comparisons for all tests used to accept or reject the null hypothesis. We use Benjamini-Hochberg Correction, with a false discovery threshold of $q = 0.05$: unless stated otherwise, all significant results pass this threshold.

[4]In our pre-registered hypotheses, this was listed under RQ2. We present it under RQ3 for thematic and narrative clarity.

[5]Although they are ordered categorical variables and normality cannot be assumed, with large samples, parametric tests are sufficiently robust for analysis [37].

(a) Code produced by participant when sober

```
def is_sorted(integers):
  for i in range(len(integers)-1):
    if integers[i] > integers[i+1]:
      return False
  return True
```

(b) Code from same participant when intoxicated

```
def is_sorted(input_list):
  return helper(None, input_list)

def helper(min_val, input_list):
  if len(input_list) == 0: return True
  if min_val > input_list[0]: return False
  return helper(input_list[0],
       input_list[1:])
```

Figure 3: Indicative example comparing code produced while high vs. sober by the same participant for the same problem. The intoxicated code is more complicated and contains a bug.

Effect Size. We use Cohen's d (with pooled standard deviation) to assess the size of differences tested by t-tests. We consider $d > 0.2$ a small effect, and values above 0.5 a medium effect. For correlations, we use Pearson's r, with $0.1 < r \leq 0.3$ a weak correlation, $0.3 < r \leq 0.5$ a moderate correlation and $0.5 < r$ a strong correlation.

6 RESULTS

We present the results of our pre-registered (RQ1-RQ4 Section 5.1) and exploratory (E-RQ1, E-RQ2, Section 5.2) questions. Section 3.2 details experimental tasks, including the metrics used. A discussion of the statistical our statistical methods is in Section 5.3.

6.1 RQ1 — Impacts on Program Correctness

We first investigate how programming while intoxicated impacts program correctness. We do this by using paired t-tests to compare sober vs. cannabis session percent correctness for each short program comprehension task type and "interview style" coding question. This results in six total significance tests.

At a high level, we find strong evidence supporting our hypothesis that cannabis-intoxicated written programs are less correct. For all correctness score comparisons, participants' cannabis correctness scores were *lower*, on average, than sober scores. For four out of the six comparisons, this difference was statistically significant ($0.0005 < p < 0.05$) with small to small-medium effect ($0.28 \leq d \leq 0.44$). Table 2 summarizes our top-level results.

Short Programming Problems. Participants completed several Boolean logic problems, code-tracing problems, and code-writing problems during each session. For the Boolean task, we

Table 2: The impact of cannabis intoxication on programming correctness (RQ1) and speed (RQ2). All significance tests are paired t-tests, while effect size calculations use Cohen's d. Cells that are bold and highlighted in green are those differences that are significant with Benjamini-Hochberg correction (p< 0.05, q=0.05). Cells in italics and highlighted in yellow indicate a trend that did not reach significance. Notice that all differences, even those that did not reach significant, indicate decreased performance while high: cannabis-intoxicated programmers take more time to write more incorrect programs.

	Sober	High	Diff	p	BH-p	d	Sober	High	Diff	p	BH-p	d
RQ1: Short Programming Problems, Correctness Scores							*RQ2: Average Stimulus Time (in seconds)*					
Boolean	81.5%	81.0%	-0.5%	0.846	0.846	0.03	14.2	14.7	+0.5	0.310	0.465	0.10
Code-tracing	62.3%	52.1%	-10.2%	<0.001	**0.003**	0.42	31.7	32.1	+0.4	0.656	0.656	0.06
Code-writing	56.9%	46.4%	-10.6%	<0.001	**0.003**	0.44	67.4	70.4	+3.0	*0.065*	*0.130*	0.23
RQ1: "Programming Interview" Problems, Correctness Scores							*RQ2: Average Overall Time (in min)*					
Problem 1: Strings and 1D arrays	65.9%	56.4%	-9.5%	0.033	**0.049**	0.28	9.7	11.1	+1.4	0.012	**0.039**	0.32
Problem 2: Recursive Lists and Trees	48.5%	34.5%	-14.0%	0.012	**0.024**	0.35	11.8	13.0	+1.2	0.013	**0.039**	0.33
Problem 3: 2D-arrays	53.9%	48.4%	-5.5%	0.383	0.460	0.15	16.2	16.8	+0.6	0.500	0.656	0.11

do not find evidence of cannabis intoxication impairing performance (p = 0.85). However, for both the code-tracing task and the code-writing task, we find that cannabis intoxication has a significant negative impact on performance (p = 0.0009, p = 0.0005 respectively). For both of these tasks, the negative impact was of a small-medium effect (d = 0.42, d = 0.44). To put this in perspective, correctness scores were, on average, 10% lower when intoxicated (52% vs. 62% for code-tracing and 46% vs. 57% for code-writing). Cannabis impairment at ecologically-valid levels can have a negative impact on two fundamental software development tasks: code reading via tracing and code writing. Additionally, the presence of an effect for the two more cognitively-demanding tasks, but not for the simpler Boolean task, may indicate that cannabis intoxication impacts scale with task complexity; if so, this trend could relate to known effects of cannabis on working memory [29].

To understand the factors driving this decreased performance, we perform an informal qualitative analysis of the types of errors that cannabis-intoxicated programmers are likely to make on the code-tracing and code-writing tasks. We observe that high programmers often complicate their solutions and add extraneous conditionals while still missing edge cases. Figure 3 shows an indicative example: code produced by the same participant for the same problem (on different days) while high and sober. The code produced while intoxicated features a more complicated structure (recursion via a helper function) as well as more opportunities for simple mistakes (e.g., three indexing operations vs. two, two conditional branches vs. one, etc.). The intoxicated solution contains one such error, comparing `None` with a number, which raises a `TypeError`.

"Interview style" Problems. While the short problems highlight the impact of cannabis on specific programming aspects, for a more holistic understanding we consider the three longer "interview style" problems. Participants never completed the same interview problem twice: problems were paired and counterbalanced across sessions by algorithm type and difficulty to permit within-subjects analysis. The first problem always involved 1D-arrays, the second recursive trees or lists, and the third 2D-arrays.

We find that cannabis significantly impaired correctness for the 1D-array problems and recursive problems. For the 1D-array problems, participants passed 10% fewer correctness tests in the cannabis-intoxicated session than in the sober session (56% vs 65%, p = 0.033, d = 0.28). For the recursive problems, participants passed 14% fewer correctness tests in the cannabis-intoxicated session than in the sober session (34% vs. 48%, p = 0.012, d = 0.35).

For the 2D-array problems, while participants passed 6% fewer correctness tests in the cannabis session, this trend did not rise to the level of significance (48% vs 54%, p = 0.383, d = 0.15). We note, however, that unlike the 1D-array and recursive problems, the two 2D-array problems that we chose were not of equivalent difficulty for our population (Section 3.2.3). This confound may explain the lack of an observed significant difference on this problem pair.

In our pre-registered analysis plan, we stated that "The null hypothesis will be rejected if high programmers have lower scores on the majority of interview-style[6] problems when high." As we found a significant difference in 2/3 interview-style problems, we reject the null hypothesis and conclude that cannabis intoxication has a significant negative effect on program correctness. Specifically, this result shows that the impairment we observed on the controlled short programming tasks persists when implementing more complicated functions. We discuss implications in Section 7.

We find support for our hypothesis that **cannabis use decreases program correctness** with a small-medium effect (0.0005 < p < 0.05, 0.28 ≤ d ≤ 0.44, 10–14% fewer passed tests). Cannabis impairs *writing* and *tracing* through programs.

6.2 RQ2 — Impacts on Programming Speed

We next investigate the impact of cannabis on programming speed. We hypothesized that cannabis-intoxicated people will program more slowly. For the short programming problems, we use a paired t-test to compare the average stimulus completion time per task type (Boolean, code-tracing, code-writing) per condition (sober or intoxicated). For the "interview-style" questions, we compare each problem type's total completion time. Our results are in Table 2.

Short Programming Problems. We find no significant evidence that cannabis intoxication impacts programming time for simpler programming tasks. For the code-writing task (the most

[6]The original references LeetCode explicitly. We use interview-style in this paper.

complicated of the three), we observe a trend toward significance with task completion taking longer when intoxicated with small effect ($p = 0.065, d = 0.23$). However, this difference neither reaches significance nor passes our multiple comparison threshold.

"Interview Style" Problems. In contrast, we do find a significant programming speed difference for 2/3 of the more complex "interview style" problems. For the 1D-array problems, high participants spent an average of 84 more seconds than sober participants (11.1 minutes vs. 9.7 minutes, $p = 0.01, d = 0.32$). For the recursive problems, high participants spent an average of 83 more seconds than sober participants (13.0 minutes vs. 11.8 minutes, $p = 0.01$, $d = 0.33$). We do not observe a significant difference between completion times for the 2D-array problems. However, as with correctness (Section 6.1), this lack of result may be attributable to uneven difficulty matching for this problem pair (Section 3.2.3).

Why Are Programmers Slower? We investigate the factors driving the difference for the "interview style" questions: are intoxicated programmers slower because they physically type slower, because they make more typing corrections, or because they have less "active typing time" (time spent not actively programming, but rather thinking or searching online for help)?

We find that all three factors contribute! We compared participants' overall typing speed in both sessions using a paired t-test. Sober participants typed 6 more characters per minute than cannabis-intoxicated participants on average (84 chars/min vs. 90 chars/min, $p = 0.0004, d = 0.32$). We excluded the time when participants were not actively typing, defined as any period between two keystrokes longer than 8s, from the total time used to calculate typing speed. For corrections, cannabis-intoxicated participants delete more of their keystrokes than sober participants (20.9% vs. 18.5%, $p = 0.00003, d = 0.35$). This result, along with the slower typing speed, aligns with work on general cannabis impairment, which finds a negative impact on fine motor control [29]. Finally, we find that cannabis-intoxicated participants spend more of their total time not actively typing code (64.9% vs. 60.6%, $p = 0.003, d = 0.36$). We visualize these typing-related differences between the sober and cannabis sessions for a single indicative participant in Figure 4.

For "interview-style" tasks, **cannabis use impairs programming speed** ($p < 0.04$, $d = 0.3$, 10–14% slower). This decrease in speed is associated with typing slower, deleting more characters, and more time spent not typing.

6.3 RQ3 — Method Choice and Divergence

We now investigate if high and sober programmers *choose to solve the same programming problem in different ways.* We consider both the *efficiency* of solutions and also the *algorithmic method* implemented. We have two hypotheses: first, correct programs by high participants will be less efficient than those of sober participants. Second, we hypothesize high participants will show more divergence choices in algorithmic or methodological approaches (one potential aspect of "creativity"), compared to sober participants.

We focus on the "interview-style" problems, as those tasks are complex enough for a meaningful algorithmic analysis. For RQ3 we analyze all six problems separately (i.e., two 1D-array, two recursion, and two 2D-array problems). This is done because the

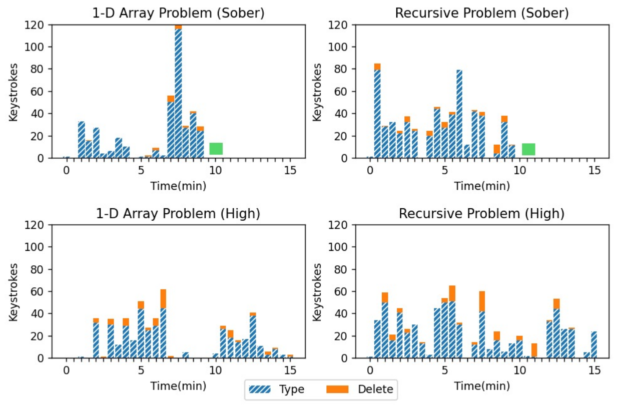


Figure 4: Histograms with typed characters (dark blue with lines) and deletions (orange) over time for the same participant while sober and high. The high condition features longer pauses and more deletions. The participant also had a higher maximum typing speed sober (120 keystrokes in 30 seconds vs. only 65 while high). This participant finished both problems early sober (noted by the green box) but ran out of time when high.

running times of difficulty-paired problems may vary significantly. For example, for the 1D-array problem type, one instance features a 1D-array as a Python list while the other uses a Python string. While conceptually similar, Python treats these very differently from an efficiency standpoint (e.g., list instances are mutable while strings are not). The statistical comparisons in this research question thus use non-paired tests unless otherwise noted.

Program Efficiency. We measure the efficiency of correct solutions on very large program inputs (Section 3.2.3, average of three trials). All program running times, were generated using the same multi-core Linux server and were run sequentially. Despite generous timeouts for the efficiency tests, some particularly-inefficient correct solutions failed to terminate for our biggest inputs. For these problems, we assign them our maximum timeout of 60 seconds.

We compare the efficiency test runtimes for correct solutions written while high to those written while sober for each of the six "interview-style problems". For 5/6 problems, the differences are not significant. For one of the two 1D-array problems, the difference is significant before multiple comparison correction ($p = 0.03$, $d = 1.03$). For this problem, the solutions by sober participants are actually *less efficient* than those by high participants (5.0 seconds vs. 21.7 seconds). While intriguing, this result does not survive correction for multiple comparisons (corrected $p = 0.18$). Additionally, few high and sober participants correctly implemented the problem (9 and 12 respectively), leading to low statistical power. We cannot reject the null hypothesis regarding efficiency.

Solution Divergence. We manually annotate the solutions to both 1D-array problems to obtain a more nuanced understanding of method choice differences between high and sober participants. We note that, unlike for our efficiency analysis, we manually annotate the solutions for all participants, even those who did not arrive at the correct solution. This is so because there is anecdotal evidence that cannabis use might improve programming *creativity* [15, 35]. If this is the case, even if solutions produced by cannabis-intoxicated programmers contain more bugs, other benefits may offset this cost; informally, a developer might generate a solution while using cannabis and then come back the next day to fix any errors.

To the best of our knowledge, while creativity is an important aspect of the software development process, a robust metric for it remains an open problem [24]. One approach used by prior work is to measure *divergence* in computational patterns [3]. Divergence tests have long been the basis of common creativity measurements [42]. We adapt this use of solution divergence to method choice.

The possible method choices were specific to the first or second problem instance, and included categories such as brute force using a loop, brute force using recursion, or a stack data structure. For both problem instances, a couple of submissions did not fall into any category and were labeled other, or were categorized as completely incorrect. Section 3.2.3 overviews our method annotation process in more detail. We applied a χ^2 test with the groups as sober and intoxicated and the categories as the different methods for each problem. For the first problem instance, $\chi^2 = 1.68, p = 0.89$. For the second problem instance, $\chi^2 = 8.44, p = 0.077$. We find no evidence that method choice differed significantly between high and sober participants. As a result, it does not make sense to investigate if methods chosen by high participants were more diverse because there was no significant difference between the two distributions. Overall, we find no evidence to support our hypothesis that high programmers generate more diverse or more creative programming solutions. We discuss implications in Section 7.

We found **no statistically-significant evidence that cannabis intoxication impacts solution efficiency or implementation divergence** ($p \geq 0.08$). We do not reject the null hypothesis that programmers using cannabis exhibit the same divergence of method choice as sober counterparts.

6.4 RQ 4 — Influence of Cannabis Use History

For our last pre-registered hypothesis, we investigate if cannabis use history mediates the negative impact of cannabis intoxication on program correctness. We hypothesize the impact of cannabis use while programming will be lessened for those that are heavy cannabis users vs. those that are moderate users.

We divide participants into two groups for analysis: heavy users and light users, classified by if their aggregated z−transformed scores on the DFAQ-CU use frequency questions are positive or negative [14].[7] 38 participants are classified as light users while 33 are classified as heavy users (roughly, those participants who use cannabis more than two times a week). We then calculate the per-participant difference between high and sober sessions for all correctness scores for which we observed significant general impairment: code-tracing problems, code-writing problems, the 1D-array problem type, and the recursive-data structure problem type. We use an independent t-test to compare the performance differences of light and heavy users on each test to see if one group experiences significantly more cannabis-related impairment than the other.

We find no significant differences in cannabis-related impairment between heavy and light users ($0.35 \leq p \leq 0.88$). To confirm this null result, we additionally compute pairwise correlations between inter-session performance correctness differences and two cannabis-related features: self-reported current intoxication level and lifetime cannabis usage amount. We find no significant correlations with these comparisons ($-0.26 < r < 0.13$ for all correlations).

We find **no significant evidence that cannabis impacts heavy users less than others** ($p \geq 0.35$). We do not reject the null hypothesis, and instead conclude that cannabis impairs all programmers equally, regardless of cannabis use history.

6.5 E-RQ1 — Impacts on Code Style

We explore the impact of cannabis use on code style. While annotating participant method choices for the two 1D-array problems, we also annotated responses for various stylistics features. In particular, we marked if a participant added comments, print statements, helper functions, or additional test cases to their implementation. We also counted the number and maximum nesting depth of loops and conditionals to get a measure of branch and loop complexity of the code. We compare proportions for stylistic features between high and sober participants using the n-1 χ^2-test. This analysis determines if the correctness impairment of cannabis extends to stylistic properties which, while non-functional, facilitate software readability and maintainability [18, 21].

We find no significant style differences between programs written while high vs. sober ($0.20 \leq p \leq 0.85$). While exploratory, this may mitigate concerns about cannabis substantially compromising code clarity and ease of understanding, which are critical for successful collaboration and future program modifications.

We find **no significant evidence that cannabis impacts programming style** (e.g., comments, helpers, etc.) ($p \geq 0.20$).

6.6 E-RQ2 — Self Perception of Impact

For our second exploratory analysis, we investigate if participants are able to correctly perceive their relative programming performance, even when intoxicated. We explore the answer to this question because prior studies have reported that cannabis-using developers self-assess task contexts and potential impairment when making usage decisions [35]. To answer this question, we first investigate how participants perceived their programming performance while high vs. sober. At the end of their second sessions, we asked all participants about their performance on the "interview style" questions in that session compared to the previous one. 63% thought they performed worse during their cannabis session, 20% could not tell either way, and only 17% thought they performed better while high. Figure 5 breaks down those reports.

While participants did, on average, have decreased performance while high, this was not universal. We thus investigate *if participant perceptions of their performance while high is accurate*: we correlate perceived performance difference (a 7-point Likert scale)

[7]In the pre-registered hypotheses, we said we would use three groups: light cannabis users (at most 3–4 times per month), moderate cannabis users (1–2 times per week), and heavy cannabis users (2+ times per week). We use an aggregated score instead after a closer inspection of the DFAQ-CU assessment's scoring instructions [13].

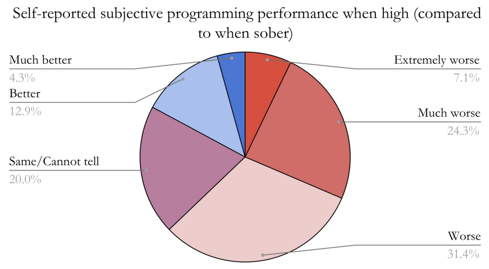


Figure 5: Self-reported subjective programming performance in cannabis-using sessions compared to sober sessions. Most participants report perceiving decreased performance when high (63%) compared to only 17% who perceived improvement.

with actual performance difference (average difference between percent correctness scores for all three "interview style" problems). We find a strong, positive correlation between self-reported relative performance while high and actual performance: $r = 0.59$. Of the 49/75 participants who showed decreased performance on the interview problems while high, only four of them (8%) incorrectly perceived increased performance. Together, these results indicate that programmers are generally able to accurately judge their relative programming performance, even when under the influence of cannabis. This has implications for policy (see Section 7).

Most programmers **can accurately judge relative programming performance while high** ($r = 0.59$).

7 DISCUSSION

We observe a significant impairment associated with ecologically-valid cannabis use while programming (10% fewer correct tests, 10% slower programming). At the same time, we do not observe a significant method divergence benefit. Previously, anecdotal evidence (Section 2) was either conflicted or suggested that purported creativity benefits were worth the impairment. *Our results paint a more nuanced picture, especially for situations without a robust mechanism to catch bugs or with deadline pressure.* We thus consider what motivates developers to use cannabis. After completing both sessions, we asked participants what was different when programming while high. The majority emphasized that it was harder to focus and easier to get distracted, which is contrary to prior survey results (i.e., improved focus [15, Tab. 3]). However, some participants did indicate more enjoyment, fewer worries, and decent insight into alternative perspectives. We note we only considered some software solution divergence aspects (e.g., we do not assess architecture or design creativity, etc.). "Interview-style" questions may be too structured to admit certain creative freedoms. This is relevant as programmers self-report self-regulating cannabis use by software task, using more when tasks are open-ended [15]. Participants also reported stress from timing and researcher observation.

Although the variance we observe in outcomes for cannabis intoxication is consistent and significant, we note that it is *much less than the productivity variance already found in new hires.* A classic study reported 16–25× differences in coding times and 26–28× differences in debugging times for programmers [43, Tab. III], with no correlations to class grades or other hiring distinctions. This general pattern has continued, with a recent Microsoft study reporting that the time to first code check-in (in weeks), a job-relevant productivity metric, was 57% lower in some geographic locations than others [39, Sec. 4.1]. A 10% difference is not large compared to such already-existing variance. In addition, some programmers in our sample received full correctness scores even while high, or performed better when high. Most were able to accurately recognize their own cannabis-related impairment or the lack of it. Blanket employment policy may thus *not* be well-motivated.

While our study design features safeguards for the privacy and safety of our individual participants (Section 3.1), following Robson and McCartan [40, Ch. 10], we note that job-seekers may be considered a vulnerable group because of their economic situations. Our research may have future implications for job-seekers (e.g., if it informs hiring policies), a risk the researchers, and the IRB, weighed against the benefits of conducting the research. While we believe that the results are supportive (i.e., restrictive job policies may not be merited), we acknowledge that the situation is nuanced.

Anecdotally, we note that several participants reached out during the study to reschedule because they had an upcoming drug screening for a new job. This aligns with the qualitative results of Newman *et al.* who found that developers view drug policies in software as ineffective and easy to circumvent [35]. Additionally, the mere existence of an anti-drug policy can serve as a deterrent for hiring and retention [35]. This, combined with the low observed magnitude of cannabis impairment, may indicate that strict drug policies might not be optimal uses of resources.

8 THREATS TO VALIDITY

Although our observational results give confidence in our characterization of cannabis intoxication effects on programming, our results may not generalize. We highlight a number of considerations.

First, our participants are not a random sample of the population. Our selection may be biased to those interested in cannabis-related studies or with a positive perception of cannabis. We partially mitigate this by assessing the cannabis usage history of our participants (see Section 3.2). In addition, we are interested in understanding how programmers who routinely use cannabis are affected by it in development settings: in that context, a participant who has not used cannabis before is less indicative of the daily impact on a company. Similarly, the legal status of cannabis in some locations may deter participation in our study. We partially mitigated this by recruiting in four US states where this sort of cannabis use is legal.

Second, our larger programming tasks were taken the LeetCode repository of skills-based interview questions. These questions may not be indicative of industrial practice [2]. This is partially mitigated by the fact that they are indicative of programming tasks people carry out and study for in the hiring process.

Third, our notions of code quality and divergence may not generalize. We assess code correctness via tests and assess divergence and style by expert annotation. There are other useful notions of utility (e.g., formally proving correctness or using other static analyses, measuring maintenance efforts, etc.) that we do not capture. We partially mitigate this concern, noting that automated regression testing remains a dominant activity in SE and that manual assessment is relevant for both code reviewing and hiring decisions. There are other indicators revealing creativity in software engineering problem solving [24], and other factors linked to programming creativity (e.g., knowledge [25], personality [1, 24]), but we only measure divergence in products.

Fourth, we are unable to control the amount of cannabis affecting participants. Our IRB protocol did not permit *dispensing* cannabis, directing participants to take a particular amount, or collecting blood samples — instead, our *observational* study involves participants using cannabis anyway. We partially mitigate this via photographs of cannabis products used (and include these self-reported amount of marijuana and THC in our replication materials), and by restricting attention to one delivery method (smoking/vaping, but not edibles). Experienced and novice cannabis users may make different dosage decisions and have different tolerances (e.g., [5]), something our approach does not capture.

9 CONCLUSION

In a controlled observational study with 74 participants, we find that at ecologically-valid dosages, **cannabis intoxication has a significant small-medium impairment on both program correctness and programming speed** ($p < 0.5, 0.22 \leq d \leq 0.44$). We did not find evidence of cannabis increasing solution divergence. We also did not find that past cannabis usage history significantly mediates programming impairment. However, even when under the influence of cannabis, programmers correctly perceive differences in their programming performance ($r = 0.59$). We hope our results contribute to the development of evidence-based policies and assist software developers in making informed decisions.

ACKNOWLEDGMENTS

We gratefully acknowledge the partial support of the US National Science Foundation (2211749) as well as the support of Dan Clauw and Kevin Boehnke through the Michigan Psychedelic Center.

In addition, we thank Danielle Hu for her help in running a portion of the experimental sessions, as well as multiple lab members for acting as pilot participants. We also thank Li Morrow and other members of Michigan's IRB for their help in designing an experimental protocol that prioritized participant safety and privacy.

REFERENCES

[1] Aamir Amin, Shuib Basri, Mobashar Rehman, Luiz Fernando Capretz, Rehan Akbar, Abdul Rehman Gilal, and Muhammad Farooq Shabbir. 2020. The impact of personality traits and knowledge collection behavior on programmer creativity. *Information and Software Technology* 128 (2020), 106405.

[2] Mahnaz Behroozi, Chris Parnin, and Titus Barik. 2019. Hiring is Broken: What Do Developers Say About Technical Interviews?. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2019, Memphis, Tennessee, USA, October 14-18, 2019*, Justin Smith, Christopher Bogart, Judith Good, and Scott D. Fleming (Eds.). IEEE Computer Society, 1–9. https://doi.org/10.1109/VLHCC.2019.8818836

[3] Vicki E. Bennett, KyuHan Koh, and Alexander Repenning. 2013. Computing Creativity: Divergence in Computational Thinking. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. 359–364. https://doi.org/10.1145/2445196.2445302

[4] Marc Berman. 2020. How CBD Oil Can Help Programmers Focus. https://programminginsider.com/how-cbd-oil-can-help-programmers-focus/. Accessed: 2021-03-07.

[5] Kevin F. Boehnke, J. Ryan Scott, Evangelos Litinas, Suzanne Sisley, David A. Williams, and Daniel J. Clauw. 2019. Pills to Pot: Observational Analyses of Cannabis Substitution Among Medical Cannabis Users With Chronic Pain. *The Journal of Pain* 20, 7 (2019), 830–841. https://doi.org/10.1016/j.jpain.2019.01.010

[6] Samantha J. Broyd, Hendrika H. van Hell, Camilla Beale, Murat Yücel, and Nadia Solowij. 2016. Acute and Chronic Effects of Cannabinoids on Human Cognition—A Systematic Review. *Biological Psychiatry* 79, 7 (2016), 557–567. https://doi.org/10.1016/j.biopsych.2015.12.002 Cannabinoids and Psychotic Disorders.

[7] Thomas S. Burt, Timothy L. Brown, Gary Milavetz, and Daniel V. McGehee. 2021. Mechanisms of cannabis impairment: Implications for modeling driving performance. *Forensic Science International* 328 (2021), 110902. https://doi.org/10.1016/j.forsciint.2021.110902

[8] Ian Campbell. 2007. Chi-squared and Fisher–Irwin tests of two-by-two tables with small sample recommendations. *Statistics in medicine* 26, 19 (2007), 3661–3675. https://doi.org/10.1002/sim.2832

[9] Cisco. 2019. 2019 Code of Business Conduct. https://www.cisco.com/c/dam/en_us/about/cobc/2019/english-2019.pdf. Accessed: 2021-08-09.

[10] Broderick Crawford and Claudio León de la Barra. 2007. Enhancing Creativity in Agile Software Teams. In *Agile Processes in Software Engineering and Extreme Programming*, Giulio Concas, Ernesto Damiani, Marco Scotto, and Giancarlo Succi (Eds.). 161–162.

[11] Will Crichton, Maneesh Agrawala, and Pat Hanrahan. 2021. The Role of Working Memory in Program Tracing. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 56, 13 pages. https://doi.org/10.1145/3411764.3445257

[12] C. Cuttler, E.M. LaFrance, and A. Stueber. 2021. Acute effects of high-potency cannabis flower and cannabis concentrates on everyday life memory and decision making. *Sci. Rep.* 11, 13784 (2021).

[13] Carrie Cuttler and Alexander Spradlin. 2017. Measuring cannabis consumption: psychometric properties of the daily sessions, frequency, age of onset, and quantity of cannabis use inventory (DFAQ-CU). *PLoS One* 12, 5 (2017), e0178194.

[14] Carrie Cuttler and Alexander Spradlin. 2017. Measuring cannabis consumption: psychometric properties of the daily sessions, frequency, age of onset, and quantity of cannabis use inventory (DFAQ-CU). *PLoS One* 12, 5 (2017), e0178194.

[15] Madeline Endres, Kevin Boehnke, and Westley Weimer. 2022. Hashing It out: A Survey of Programmers' Cannabis Usage, Perception, and Motivation. In *International Conference on Software Engineering*. 1107–1119.

[16] Madeline Endres, Madison Fansher, Priti Shah, and Westley Weimer. 2021. To read or to rotate? comparing the effects of technical reading training and spatial skills training on novice programming ability. In *Foundations of Software Engineering*. 754–766.

[17] Madeline Endres, Zachary Karas, Xiaosu Hu, Ioulia Kovelman, and Westley Weimer. 2021. Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study. In *International Conference on Software Engineering*. 600–612.

[18] Sarah Fakhoury, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. 2018. The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load. In *International Conference on Program Comprehension*.

[19] Denae Ford, Margaret-Anne D. Storey, Thomas Zimmermann, Christian Bird, Sonia Jaffe, Chandra Shekhar Maddila, Jenna L. Butler, Brian Houck, and Nachiappan Nagappan. 2022. A Tale of Two Cities: Software Developers Working from Home during the COVID-19 Pandemic. *ACM Trans. Softw. Eng. Methodol.* 31, 2 (2022), 27:1–27:37. https://doi.org/10.1145/3487567

[20] Daniel J. Fridberg, Sarah Queller, Woo-Young Ahn, Woojae Kim, Anthony J. Bishara, Jerome R. Busemeyer, Linda Porrino, and Julie C. Stout. 2010. Cognitive mechanisms underlying risky decision-making in chronic cannabis users. *Journal of Mathematical Psychology* 54, 1 (2010), 28–38. https://doi.org/10.1016/j.jmp.2009.10.002 Contributions of Mathematical Psychology to Clinical Science and Assessment.

[21] Zachary P. Fry, Bryan Landau, and Westley Weimer. 2012. A human study of patch maintainability. In *ISSTA*. ACM, 177–187.

[22] Joseph E. Gonzales and Corbin A. Cunninghham. 2015. The promise of pre registration in psychological research. *American Psychological Association* (2015).

[23] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. 2014. Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ* 2 (March 2014), e289.

[24] Wouter Groeneveld, Laurens Luyten, Joost Vennekens, and Kris Aerts. 2021. Exploring the Role of Creativity in Software Engineering. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Society, ICSE (SEIS) 2021, May 25-28, 2021*. IEEE, Madrid, Spain, 1–9. https://doi.org/10.1109/ICSE-SEIS52602.2021.00009

[25] Reshma Hegde and Gursimran Walia. 2014. How to enhance the creativity of software developers: A systematic literature review. *International Conference on Software Engineering and Knowledge Engineering* (2014), 229–234.

[26] IBM. 2018. Business Conduct Guidelines. https://www.ibm.com/investor/att/pdf/BCG_accessible_2019.pdf. Accessed: 2021-08-09.

[27] Leo Kelion. 2014. FBI 'could hire hackers on cannabis' to fight cybercrime. https://www.bbc.com/news/technology-27499595. Accessed: 2021-03-07.

[28] Mikael A Kowal, Arno Hazekamp, Lorenza S Colzato, Henk van Steenbergen, Nic JA van der Wee, Jeffrey Durieux, Meriem Manai, and Bernhard Hommel.

2015. Cannabis and creativity: highly potent cannabis impairs divergent thinking in regular cannabis users. *Psychopharmacology* 232, 6 (2015), 1123–1134.

[29] Emese Kroon, Lauren Kuhns, and Janna Cousijn. 2021. The short-term and long-term effects of cannabis on cognition: recent advances in the field. *Current Opinion in Psychology* 38 (2021), 49–55. https://doi.org/10.1016/j.copsyc.2020.07.005 Cannabis.

[30] Ryan Krueger, Yu Huang, Xinyu Liu, Tyler Santander, Westley Weimer, and Kevin Leach. 2020. Neurological Divide: An fMRI Study of Prose and Code Writing. In *International Conference on Software Engineering*.

[31] Emily M. LaFrance and Carrie Cuttler. 2017. Inspired by Mary Jane? Mechanisms underlying enhanced creativity in cannabis users. *Consciousness and Cognition* 56 (2017), 68–76. https://doi.org/10.1016/j.concog.2017.10.009

[32] John Markoff. 2005. *What the dormouse said: How the sixties counterculture shaped the personal computer industry.* Penguin Group, New York, NY, USA.

[33] Gayle Laakmann McDowell. 2015. *Cracking the coding interview—189 programming questions and solutions.* CareerCup.

[34] Rahul Mohanani, Prabhat Ram, Ahmed Lasisi, Paul Ralph, and Burak Turhan. 2017. Perceptions of Creativity in Software Engineering Research and Practice. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 210–217. https://doi.org/10.1109/SEAA.2017.21

[35] Kaia Newman, Madeline Endres, Westley Weimer, and Brittany Johnson. 2023. From Organizations to Individuals: Psychoactive Substance Use By Professional Programmers. In *International Conference on Software Engineering*. 665–677.

[36] Lemai Nguyen and Graeme Shanks. 2009. A framework for understanding creativity in requirements engineering. *Information and Software Technology* 51, 3 (2009), 655–662.

[37] Geoffrey Norman. 2010. Likert scales, levels of measurement and the "laws" of statistics. *Advances in health sciences education : theory and practice* 15, 5 (02 2010), 625–32. https://doi.org/10.1007/s10459-010-9222-y

[38] Norman Peitek, Sven Apel, Chris Parnin, André Brechmann, and Janet Siegmund. 2021. Program Comprehension and Code Complexity Metrics: An fMRI Study. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 524–536. https://doi.org/10.1109/ICSE43902.2021.00056

[39] Ayushi Rastogi, Suresh Thummalapenta, Thomas Zimmermann, Nachiappan Nagappan, and Jacek Czerwonka. 2017. Ramp-up Journey of New Hires: Do Strategic Practices of Software Companies Influence Productivity?. In *Proceedings of the 10th Innovations in Software Engineering Conference*. 107–111.

[40] Colin Robson and Kieran McCartan. 2016. *Real world research: a resource for users of social research methods in applied settings.* Wiley.

[41] Ole Rogeberg and Rune Elvik. 2016. The effects of cannabis intoxication on motor vehicle collision revisited and revised. *Addiction* 111, 8 (2016), 1348–1359. https://doi.org/10.1111/add.13347 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/add.13347

[42] M. A. Runco and S.M. Okuda. 1988. Problem discovery, divergent thinking, and the creative process. *Journal of Youth and Adolescence* 17, 3 (06 1988), 211–220. https://doi.org/10.1007/BF01538162

[43] H. Sackman, W. J. Erikson, and E. E. Grant. 1968. Exploratory Experimental Studies Comparing Online and Offline Programming Performance. *Commun. ACM* 11, 1 (jan 1968), 3–11.

[44] Skipper Seabold and Josef Perktold. 2010. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference.* SciPy, Austin, TX, US, 92–96.

[45] Emad Shihab, Patanamon Thongtanunam, and Bogdan Vasilescu. 2023. Mining Software Repositories. *IEEE* (2023).

[46] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. 2014. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering*. 378–389.

[47] Joseph P. Simmons, Leif D. Nelson, and Uri Simonsohn. 2020. Pre-registration: Why and How. *J. Society for Consumer Psychology* (Dec. 2020). https://doi.org/10.1002/jcpy.1208

[48] United Nations Press Team. 2020. *UNODC World Drug Report 2020: Global drug use rising; while COVID-19 has far reaching impact on global drug markets.* United Nations. https://www.unodc.org/unodc/press/releases/2020/June/media-advisory---global-launch-of-the-2020-world-drug-report.html

[49] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[50] Charlotte Walsh. 2011. Drugs, the Internet and change. *Journal of psychoactive drugs* 43, 1 (2011), 55–63.

[51] Mary Walton. 2019. Programming and Cannabis — 5 Things to Know. https://simpleprogrammer.com/programming-and-cannabis/. Accessed: 2021-03-07.

[52] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). SciPy, Austin, TX, US, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

[53] Laura Wood. 2021. *Global Cannabis Market (2020 to 2026) - Emergence of Cannabis Legalization in Asia-Pacific Presents Opportunities - ResearchAndMarkets.com.* Business Wire. https://www.businesswire.com/news/home/20210216005966/en/Global-Cannabis-Market-2020-to-2026---Emergence-of-Cannabis-Legalization-in-Asia-Pacific-Presents-Opportunities---ResearchAndMarkets.com/

[54] Yuki Yamada. 2018. How to Crack Pre-registration: Toward Transparent and Open Science. *Frontiers in Psychology* 9, 1831 (2018). https://doi.org/10.3389/fpsyg.2018.01831