Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity

Joel Becker*, **Nate Rush***, **Beth Barnes**, **David Rein**

Model Evaluation & Threat Research (METR)

Abstract

Despite widespread adoption, the impact of AI tools on software development in the wild remains understudied. We conduct a randomized controlled trial (RCT) to understand how AI tools at the February–June 2025 frontier affect the productivity of experienced open-source developers. 16 developers with moderate AI experience complete 246 tasks in mature projects on which they have an average of 5 years of prior experience. Each task is randomly assigned to allow or disallow usage of early-2025 AI tools. When AI tools are allowed, developers primarily use Cursor Pro, a popular code editor, and Claude 3.5/3.7 Sonnet. Before starting tasks, developers forecast that allowing AI will reduce completion time by 24%. After completing the study, developers estimate that allowing AI reduced completion time by 20%. Surprisingly, we find that allowing AI actually *increases* completion time by 19%—AI tooling slowed developers down. This slowdown also contradicts predictions from experts in economics (39% shorter) and ML (38% shorter). To understand this result, we collect and evaluate evidence for 21 properties of our setting that *a priori* could contribute to the observed slowdown effect—for example, the size and quality standards of projects, or prior developer experience with AI tooling. Although the influence of experimental artifacts cannot be entirely ruled out, the robustness of the slowdown effect across our analyses suggests it is unlikely to primarily be a function of our experimental design.

1 Introduction

Software development is an important part of the modern economy, and a key domain for understanding and forecasting AI capabilities [1; 2]. Frontier AI systems demonstrate impressive capabilities on a wide range of software benchmarks [3; 4; 5; 6; 7; 8; 9] and in experiments measuring AI's impact on developer productivity when completing synthetic tasks [10; 11]. However, tasks used in these *lab experiments* sacrifice realism for scale and efficiency: the tasks are typically self-contained, do not require much prior context/familiarity to understand and complete, and use algorithmic evaluation metrics which do not capture many important capabilities [12; 13; 14]. As a result, it can be difficult to draw inferences from results on these evaluations about AI's impact in practice.

To reduce the inferential gap between measurements of AI capabilities and real-world impact, one can measure the impact of AI systems in real-world settings (i.e. *field experiments*). Existing field experiments aimed at measuring AI's impact on software development measure outcomes like number of added lines of code or number of tasks completed [15; 16; 17]. However, AI systems can affect these outcomes without productivity actually increasing—for example, code can be more verbose but functionally equivalent, and tasks can be broken up into multiple smaller tasks without the total amount of work changing—making it challenging to interpret these results.

*Equal contribution. Correspondence to {nate, joel}@metr.org

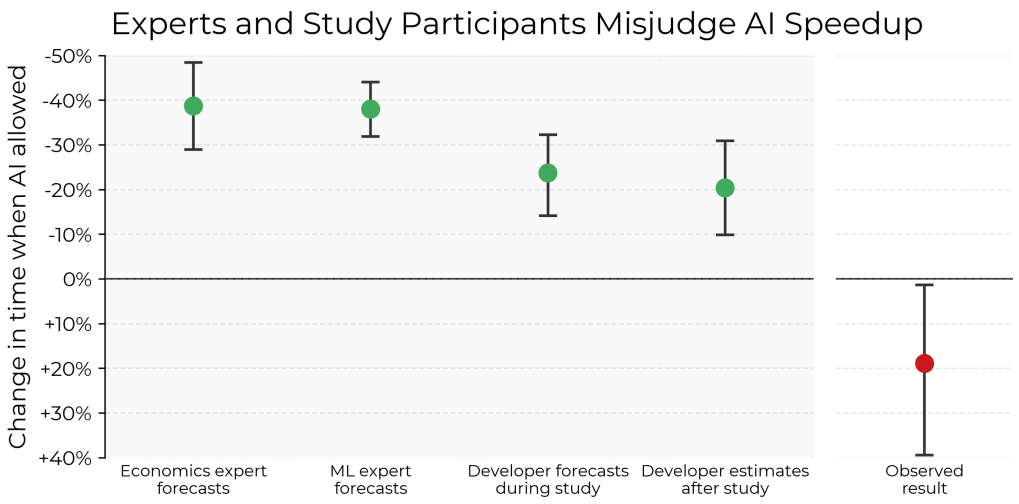

Figure 1: *Experts and study participants (experienced open-source contributors) substantially overestimate how much AI assistance will speed up developers—tasks take 19% more time when study participants can use AI tools like Cursor Pro. See Appendix D for detail on speedup percentage and confidence interval methodology.*

To directly measure the impact of AI tools on developer productivity, we conduct a randomized controlled trial by having 16 developers complete 246 tasks (2.0 hours on average) on well-known open-source repositories (23,000 stars on average) they regularly contribute to. Each task is randomly assigned to allow or disallow AI usage, and we measure how long it takes developers to complete tasks in each condition[1]. Developers, who typically have tens to hundreds of hours of prior experience using LLMs[2], use AI tools considered state-of-the-art during February–June 2025 (primarily Cursor Pro with Claude 3.5/3.7 Sonnet). We collect screen recordings as they work, providing a rich data source for analysis.

Before tasks are randomized, developers forecast that allowing AI will reduce completion time by 24%. After study participation, developers estimate that allowing AI reduced completion time by 20%. Surprisingly, we find that allowing AI actually *increases* completion time by 19%— developers are slower when using AI tooling. Figure 1 displays this observed slowdown in contrast with forecasts and post-hoc developer estimates of speedup from AI. We also collect forecasts of speedup from machine learning and economics experts in academia and industry, and find that they also substantially overestimate our observed speedup.

To understand this surprising result, we manually label 143 hours of recordings of developers' computer screens while they work (representing 29% of the total hours spent by developers), which allows us to decompose how they spend their time when working with and without AI assistance at a resolution of ~ 10 seconds. We additionally collect rich statistics from source-code management systems, interview and survey participating developers, and conduct subset analyses to better understand the nature of the slowdown result.

Using these various sources of data, we identify 21 properties of our setting and experimental design that we hypothesize *a priori* may contribute to the slowdown effect. We group these factors into four categories: a) direct productivity loss, b) experimental artifact, c) factors raising human performance, and d) factors limiting AI performance. We find evidence that 5 factors contribute to the slowdown effect, we find mixed/unclear/no evidence for 10 factors, and we find evidence against 6 factors contributing to the slowdown effect. Section 3.3 presents these factors at a high level, and Appendix C discusses each factor in detail. While we can't completely rule out the impact of exper-

[1] Crucially, the tasks are defined *before* they are randomized, limiting the impact of effects from AI assistance unrelated to productivity (e.g., more verbose but functionally equivalent code)

[2] While 93% of developers have previously used LLMs, only 44% have prior experience using the Cursor IDE.

imental artifacts, the slowdown effect appears broadly robust across a wide range of experimental design decisions.

That said, many of the factors we find evidence for contributing to slowdown are specific to the setting we study—these results do *not* imply that current AI systems are not useful in many realistic, economically relevant settings. Furthermore, these results do not imply that future models will not speed up developers in this exact setting—this is a salient possibility given the rapid pace of progress in AI capabilities recently [2]. Finally, it remains possible that further improvements to current AI systems (e.g. better prompting/agent scaffolding, or domain-specific finetuning) could yield positive speedup in this setting.

Nonetheless, our results reveal a large disconnect between perceived and actual AI impact on developer productivity. Despite widespread adoption of AI tools and confident predictions of positive speedup from both experts and developers, we observe that AI actually slows down experienced developers in this setting.

1.1 Background

Speedup, but on synthetic tasks Literature on productivity improvements on software tasks due to AI usage broadly finds that AI tools increase productivity. Peng et al. [10] and Paradis et al. [11] find 56% and 21% speedups on coding tasks when using AI assistance, and Weber et al. [18] finds a 65% increase in the rate of task requirements satisfied with AI tools. However, these studies use artificial/synthetic tasks that make it difficult to directly draw inferences about the real-world impact of AI tools. For example, Peng et al. [10] asks developers to implement a very basic HTTP server in JavaScript to satisfy several automatic test cases that are shown to the developers—this task is a) unrepresentative of most software development work, and b) likely to be similar to a large amount of LLM training data, which may unfairly advantage AI systems relative to humans.

Speedup, but with non-fixed outcome measures Other literature uses tasks found "in the wild," either via natural experiments [16] or randomized controlled trials [15; 17], finding 14-51% increases in output productivity metrics. However, these studies use outcome measures that are not fixed in advance—i.e. lines of code written, number of code commits, and pull requests[3] (PRs) as their key outcome measures respectively. It's possible for AI assistance to affect the outcomes without actually increasing productivity, e.g. by causing developers to write more verbose but functionally equivalent code, or causing them to break up pull requests into smaller chunks of work.

Impressive AI benchmark results This general consensus around AI tooling's effect on software developer productivity is perhaps unsurprising, given the impressive apparent capabilities of frontier AIs on challenging question-answering and agentic tasks used in popular AI benchmarks [19; 20].

Heterogeneous effects by experience One important question that emerges given these impressive results is whether productivity gains are captured by individuals of all experience levels. The canonical framework of Agrawal et al. [21] treats AI as a fall in the cost of prediction, with distributional consequences depending on which complementary sub-problems the tool does not solve. Existing empirical work on the micro-level effects of generative AI tools tends to find that access to these tools benefits less experienced workers more, compressing performance distributions [22; 23; 10; 24].

These heterogeneous effects motivate our focus on highly skilled open-source developers, as there has been relatively less research in this setting.

Mixed speedup results in other domains Some literature measures the impact of frontier AI systems in settings other than software development, for example, for CBRN uplift risk assessment, finding mixed results with recent AI systems [25; 26; 27; 28]. Other research finds substantial productivity increases in non-software domains [22; 23].

Understanding AI's economic impact Finally, some literature tries to predict how AI capability advances might a) affect the rate of AI progress (e.g. if AI systems can substitute for human AI

[3] See Appendix F for a primer on open-source development terminology.

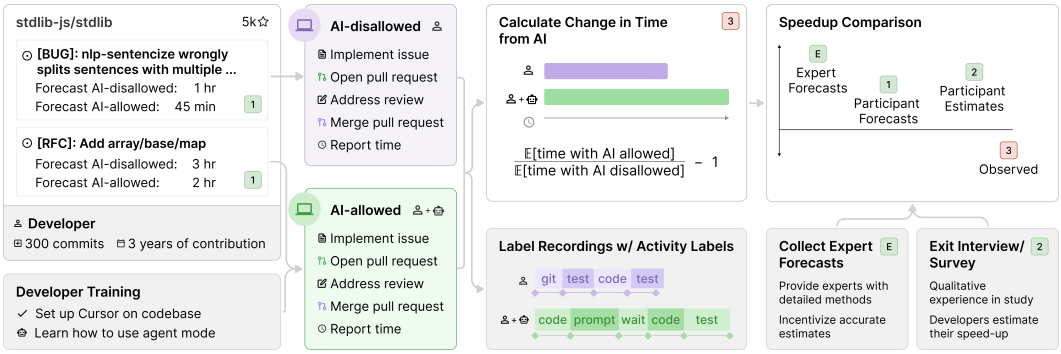

Figure 2: *Our experimental design. Tasks (referred to as issues) are defined before treatment assignment, screen recordings let us verify compliance (and provide a rich source of data for analysis), and forecasts from experts and developers help us measure the gap between expectations and observed results.*

R&D labor), or b) broadly impact the economy. Leibowich et al. [29] interview AI researchers about how full automation of AI R&D might alter the pace of advancement, several papers explore the possibility of explosive economic growth via large-scale AI labor substitution [30; 31; 32], and the economics literature includes both optimistic and skeptical perspectives on AI's productivity impact [33; 34; 35].

Our study primarily complements existing literature measuring the impact of AI on software development by:

1. Testing AI models at the February–June 2025 frontier,
2. Using unfiltered, "live" open-source repository tasks rather than synthetic or cherry-picked tasks,
3. Using a fixed outcome measure (speedup on tasks defined before randomized treatment assignment),
4. Recruiting experienced engineers with years of expertise in the target repositories, and
5. Collecting rich data on time usage, AI code suggestions, and developers' qualitative experiences.

2 Methodology

2.1 Developers and Repositories

We recruit experienced developers from large open source repositories to work on real tasks defined on these repositories. Developers come from a mix of our professional networks and from outreach to active contributors to large, popular Github repositories. The developers are experienced software engineers (typically over a decade of experience), and are regular contributors to the repositories we use—on average, they have 5 years of experience working on their repository, representing 59% of that repository's lifetime, over which time they have made 1,500 commits to the repo. As an incentive to participate, we pay developers $150/hour. Appendix G provides more detail about our recruitment and incentivization process.

The repositories themselves are large and mature. On average, they have 23,000 stars, 1,100,000 lines of code, 4,900 forks, 20,000 commits, and 710 committers, and they broadly have very high quality bars for code contributions. For example, one set of repository contribution guidelines concludes: *"Phew. While the above may be a lot to remember [..] the motivation for enforcing process is to ensure that all code contributions meet a certain quality threshold."* Section G.7 details further statistics about individual developers and repositories.

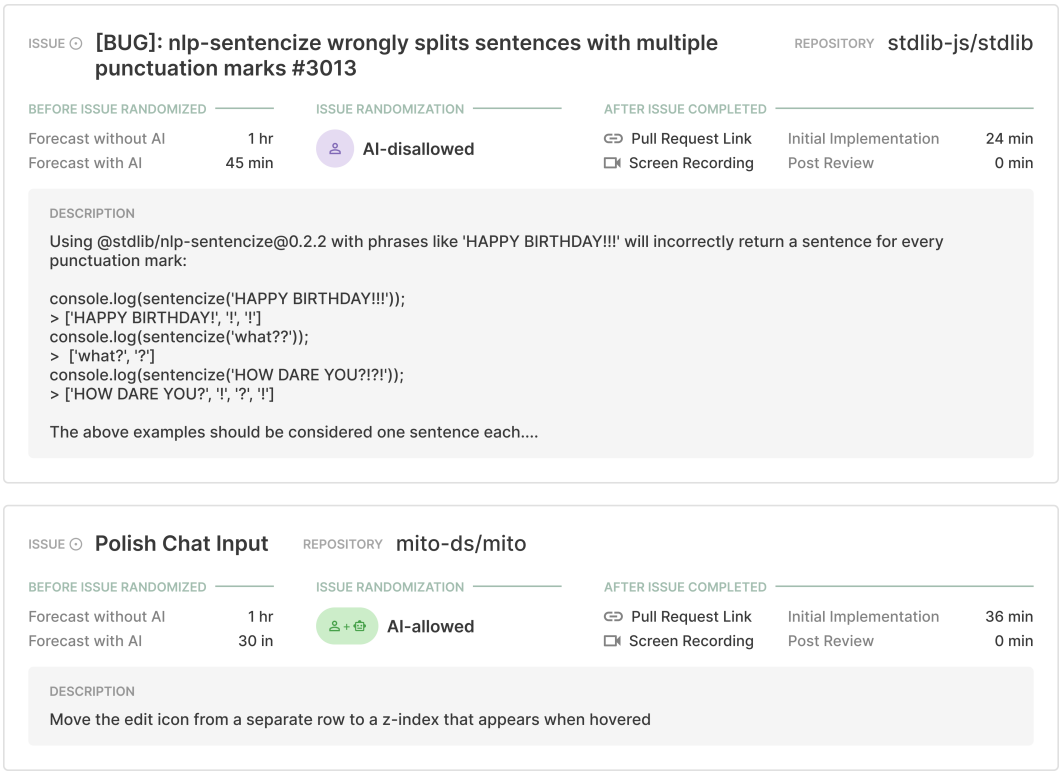

Figure 3: *Real issues completed during the study from the* `stdlibjs` *and* `mito` *repositories*

2.2 Experimental Design

Each developer provides a list of real issues in their repository to work on as part of this study. Issues are typically bug reports, feature requests, or work items used to coordinate development. They range from brief problem descriptions to detailed analyses and represent work ranging from minutes to hours. Two example issues are shown in Figure 3. Many issues are defined before the study period begins, but some are created during the study period.[4]

After collecting this issue list, developers forecast how long each issue would take if they were to complete it both with and without AI assistance. We use these forecasts as a proxy for issue difficulty, and to measure per-issue speedup anticipated by the developer. These issues are then randomized to one or the other condition via a simulated fair coin flip.[5] If AI is allowed, developers can use any AI tools or models they choose, *including no AI tooling if they expect it to not be helpful*. If AI is not allowed, no generative AI tooling can be used.[6]

Developers then work on their assigned issues in their preferred order—they are allowed to flexibly complete their work as they normally would, and sometimes work on multiple issues at a time. After completing an issue to their satisfaction, they submit a pull request (PR) to their repository, which is typically reviewed by another developer. They make any changes suggested by the PR reviewer,

[4] About half of issues included in the study were not formally defined on the repository, and instead were tracked separately for our experiment. Importantly, all issues represent real work that developers wanted to contribute to their repositories, and all work completed by developers is submitted and reviewed through each repository's standard source-code management system (e.g. GitHub/GitLab). Developers are asked to contribute issues taking a maximum of two hours, or to break up issues taking longer into multiple issues.

[5] 25 issues early in the study were randomized differently. We show that results are not sensitive to inclusion/exclusion of these issues in Section C.3.4, and we include these issues for statistical power.

[6] AI-based tab autocomplete is disallowed in the AI-disallowed condition if it uses LLMs (e.g. GitHub Copilot) but allowed otherwise. Search engines, which sometimes use AI under the hood, remain allowed in the AI-disallowed condition.

and merge their completed PR into the repository[7]. As the repositories included in the study have very high quality and review standards, merged PRs rarely contain mistakes or flaws. Finally, they self-report how long they spend working on each issue before and after PR review.

See Section G.2 for the full written instructions given to developers before they start working.

2.2.1 AI Tools and Training

Two popular means of using modern large language model (LLM) based AI tools are via web-based user interfaces (e.g. chatgpt.com) and the integrated development environment (IDE) Cursor Pro (which we provide a subscription for). Cursor is a fork of the widely used VSCode IDE with near-identical features, that additionally includes extra AI features like a language model chat interface, and an AI agent tool that can search and edit files, run arbitrary bash commands, prompt/ask the user for more details when relevant, and iterate/debug programs without constant input from users. Developers have a range of experience using AI tools: 93% have prior experience with tools like ChatGPT, but only 44% have experience using Cursor.

We provide developers with Cursor Pro subscriptions and conduct live basic training, validating that developers are able to prompt Cursor effectively to edit files in their own codebase, accept changes, and revert to previous checkpoints. However, we don't require that they use Cursor specifically. Developers working on issues for which AI is allowed can use any AI tools of their choosing, or no AI tools if they prefer. See Section F.2 for further information on these two methods of accessing AI assistance, and Appendix G for more detail about our training and onboarding process.

2.2.2 Data Collection

Contributors completed issues largely as they would outside of our experiment, with a few exceptions: they typically record their screen as they work (providing us with a source of rich data on their AI usage and working patterns), when using AI they often use the Cursor IDE, which sometimes differs from their normal development environment (e.g. neovim, PyCharm), and they record details of their working patterns after each issue is completed. Broadly, developers reported that they were not significantly inconvenienced or affected by these differences compared to their normal workflows (see section Section 3.3 for discussion of how experimental design may affect results). Developers self-report the length of time they spend working on each issue before and after PR review.

2.3 Effect Estimation

We aim to measure the percentage change in issue time to completion due to AI, defined as:

$$S = \frac{\mathbb{E}[\text{completion time with AI allowed}]}{\mathbb{E}[\text{completion time with AI disallowed}]} - 1$$

A value of -50% would indicate issues assigned to AI-allowed taking half the time of issues assigned to AI-disallowed; 0% would indicate that issues take the same time to complete with and without AI-allowed; 100% would indicate that issues assigned to AI-allowed take twice as long as issues assigned to AI-disallowed. For convenience and clarity, we typically report the absolute value of S, and colloquially refer to negative values of as *speedup*, and positive values as *slowdown*[8]

We use the total implementation time (including both pre- and post-pull request review time) developers self-report spending on each issue to estimate the percentage change in issue completion time due to AI. A small number of issues do not end up getting reviewed/merged for various reasons—we impute the post-review implementation time of these issues with the mean post-review time per treatment condition. Section C.3.4 shows that our results are robust to the specific imputation method used.

We estimate S via a log-linear regression for total implementation time, incorporating estimates of issue difficulty to reduce noise in our estimate. We also use adapted versions of this regression when

[7]Not all PRs end up getting merged by the end of the study period—see Section C.3.4 for discussion of how this may affect results.

[8]This is a mild abuse of language, because speedup and slowdown refer to *rates*, while the quantity we actually estimate and report is the percentage change in time from AI.

examining how performance varies in different subsets of the data (to efficiently account for any post-randomization issue difficulty imbalances that may be more pronounced within subsets). See Appendix D for details of the regression approach used.

2.4 Supplementary Data Collection

Developer Forecasts and Post hoc Estimates Before randomizing each issue to either AI-allowed or AI-disallowed, we ask developers to forecast how long they expect each issue to take if they are allowed to use AI assistance, and if they are disallowed from using AI. This provides a measure of issue difficulty independent of random treatment assignment, and enables us to assess developer calibration and expectations regarding the usefulness of AI assistance for their work.

After the study's conclusion, we ask developers to estimate post hoc how much AI tooling affected the time it took them to complete issues. This helps us understand the relationship between individuals' beliefs about AI helpfulness and our measured effects.

The detailed instructions given to developers for per-issue forecasts and exit surveys are available in Section G.2.

Expert Forecasts We solicit forecasts from experts in machine learning and economics in order to compare our results against expert prior expectations regarding AI capabilities [36]. Machine learning experts are a mix of industry professionals from frontier AI labs, and academics from universities with leading graduate computer science programs. Economics experts are primarily academics and PhD students from universities with leading economics graduate programs. See Section E.5 for summary statistics of expert forecasts by expert group.

Experts are given detailed information about our study design, repositories participating in our experiment, and the proportion of developers who have prior experience with AI tooling (Section G.9). They then report their prediction of our point estimate for $\frac{\mathbb{E}[\text{time with AI disallowed}]}{\mathbb{E}[\text{time with AI allowed}]}$ [9].

To incentivize experts to make accurate forecasts, we offer to pay $\max(0, \$50 \times (1 - (\text{true answer} - \text{guessed answer})^2))$ for their point estimates.[10][11]

Cursor Analytics Data We collect per-developer analytics data from Cursor's analytics dashboard, for the 13 developers we provide Cursor Pro subscriptions to. For each developer, we collect their most-used model and the number of suggested and accepted lines of AI-generated code.

Exit Interviews and Surveys After the study's conclusion, we conduct exit interviews and surveys with all developers to assess where they found AI helpful, what strategies they used to effectively elicit work from AI, whether they felt they improved at using AI tooling over the course of the study, and to estimate how much they were sped up by AI during the study. Full details of the exit interviews are available in Section G.5.1.

Qualitative Evidence Throughout the study we collect qualitative evidence from developers, to form a more comprehensive understanding of their experiences using AI tools. Developers are instructed to take detailed notes regarding their experience and usage of AI tools, and we use inductive coding—where we iteratively read through the data to identify recurring patterns, create and refine categories as they emerge, and reorganize excerpts until stable themes develop—to cluster excerpts from these notes. Qualitative results in Section 3.3 (particularly quotes from developers) were collected primarily with this methodology. However, to investigate initial hypotheses, we often ask developers probing/targeted questions, so we cannot rule out bias from this type of qualitative evidence.

[9] Note that the estimate we use for Figure 1 is transformed to represent $\frac{\mathbb{E}[T \mid \text{AI=1}]}{\mathbb{E}[T \mid \text{AI=0}]} - 1$. It is not necessarily true that forecasters' belief about the point estimate of $\frac{\mathbb{E}[T \mid \text{AI=1}]}{\mathbb{E}[T \mid \text{AI=0}]}$ is equal to the reciprocal of their belief about the point estimate of $\frac{\mathbb{E}[T \mid \text{AI=0}]}{\mathbb{E}[T \mid \text{AI=1}]}$.

[10] Approximately one-third of forecasters are offered a maximum of \$100 rather than \$50.

[11] Taking a maximum with 0 makes our scoring rule improper [37]—their reward is not necessarily maximized at their true belief. We use this scoring rule for simplicity and clarity.

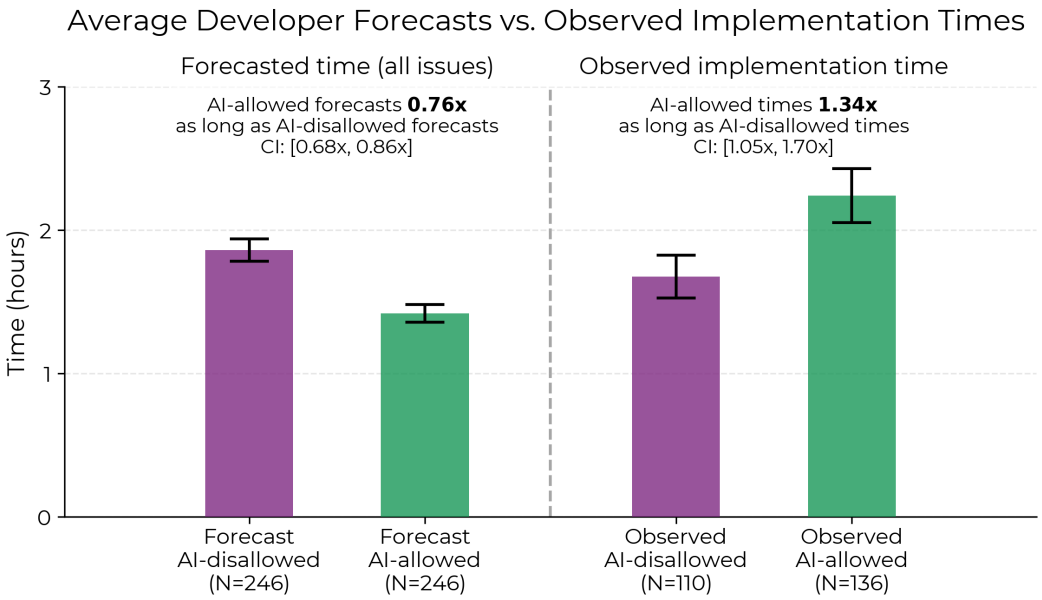

Figure 4: *Left: Raw average forecasted implementation times. Right: Raw average observed implementation times. The ratio of observed implementation times gives a more extreme slowdown estimate than regression-based estimates (Section D.1) because AI-allowed issues are forecasted (importantly, before treatment assignment) by developers to take slightly longer, which the regression corrects for. Both: Section D.5 describes confidence intervals around ratios of average times.*

Fine-Grained Screen Recording Activity Labels To compare how developers spend their time with and without AI assistance, we manually label a subset of 128 screen recordings with fine-grained activity labels, totaling 143 hours of video. In results based on these labeled screen recordings, we filter to remove issues where we find cheating, issues where the screen recording is broken for $>10\%$ of recording time, and issues with a $>20\%$ discrepancy between self-reported time and the recording time. This results in 74 recordings representing 84 hours of video for further analysis.

We label whether developers are: actively writing code, testing and debugging their code, reading or searching for information, using Git or managing their environment, prompting an AI system, waiting on an AI system to generate output, reviewing AI outputs, or idling/doing other miscellaneous work. Each high-level label is further broken down into one of 27 fine-grained categories. Labels have a resolution of ~ 10 seconds. Section G.8 describes the instructions and process used for labeling screen recordings.

3 Results

Developers complete 136 issues with AI-allowed and 110 issues with AI-disallowed. Section G.7 shows the number of issues completed across repositories and developers, respectively. We find that when developers use AI tools, they implement issues in 19% *more* time on average (Figure 1), and nearly all quantiles of observed implementation time see AI-allowed issues taking longer (Figure 5). That is, developers are slower when using AI is allowed. Colloquially, we refer to this result that issues with AI-allowed take longer than issues with AI-disallowed as *slowdown*.

3.1 Forecasts

Developer Forecasts and Post hoc Estimates Before developers complete each issue, they forecast how long they expect them to take with and without AI assistance. On average, they forecast speedup of 24%. Interestingly, after the experiment they post-hoc estimate that they were sped-up by 20% when using AI is allowed—after they used AI assistance, they estimate similar speedup as

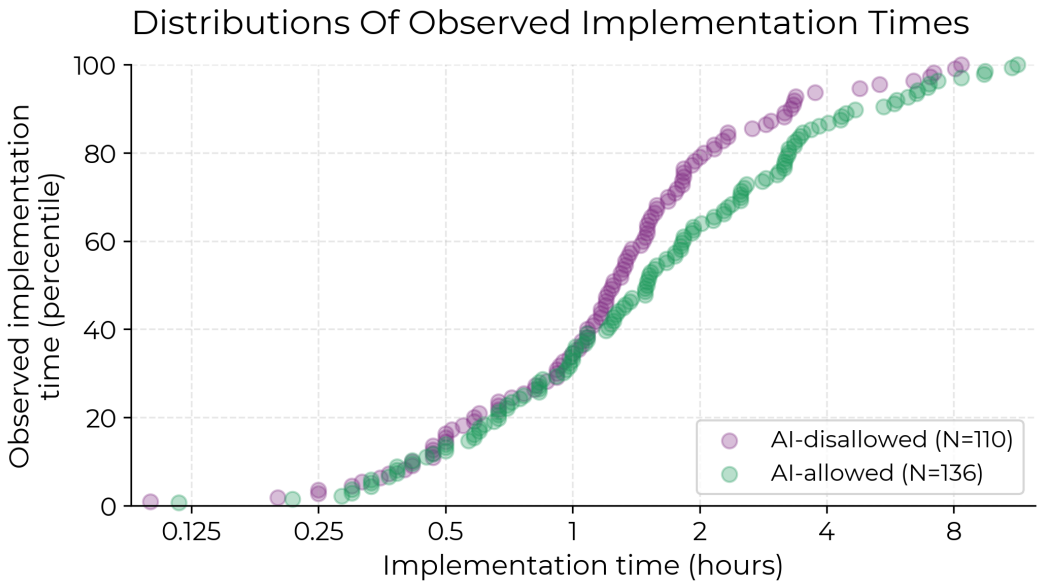

Figure 5: *Empirical cumulative distribution functions of observed implementation times. Percentile ordering is calculated separately for each treatment group.*

before, despite the fact that they are in fact slowed down by 19% (Figure 1). Figure 4 displays the raw average forecasted and observed implementation times[12].

Despite developers forecasting speedup from AI (while they are slowed down), developer forecasts *are* informative about completion time—the Pearson correlation between the time developers forecast AI-allowed issues taking and the actual time they take is 0.64, and the correlation between the time developers forecast AI-disallowed issues taking and the actual time they take is 0.59. This suggests that developers are broadly well-calibrated on the relative amount of time that issues will take, but their expectations regarding the usefulness of AI assistance are reversed.

Expert Forecasts Speedup forecasts from 34 economics experts and 54 machine learning experts overestimate speedup even more drastically than developers, predicting AI will lead to decreases in implementation time of 39% and 38%, respectively (Figure 1). We show distributions of expert forecasts in Section E.5.

3.2 Activity Labels

On a subset of 74 issues for which we have valid screen recordings, we manually label the activities developers engage in while they work. Figure 6 shows the percentage of time developers spend for each type of issue (AI-allowed or AI-disallowed). When allowed to use AI, developers spend a smaller proportion of their time actively coding and reading/searching for information. Instead, they spend time reviewing AI outputs, prompting AI systems, and waiting for AI generations. Interestingly, they also spend a somewhat higher proportion of their time idle, where their screen recording doesn't show any activity. Section E.4 shows the number of minutes spent on average in each category (instead of percentage time spent), as well as the distributions of labels broken down into more fine-grained activities.

[12]The raw percentage difference in implementation times between AI-allowed and AI-disallowed issues is 34%, which is larger in absolute value than the 19% slowdown estimated using the regression specified in Section D.1. This is true because AI-allowed issues ended up being slightly more difficult than AI-disallowed issues after randomization, as measured by the forecasted AI-disallowed times (not statistically significant; see Table 4). Our regression accounts for this, while this simple ratio estimator does not. See Figure 13 for results implied by alternative estimators.

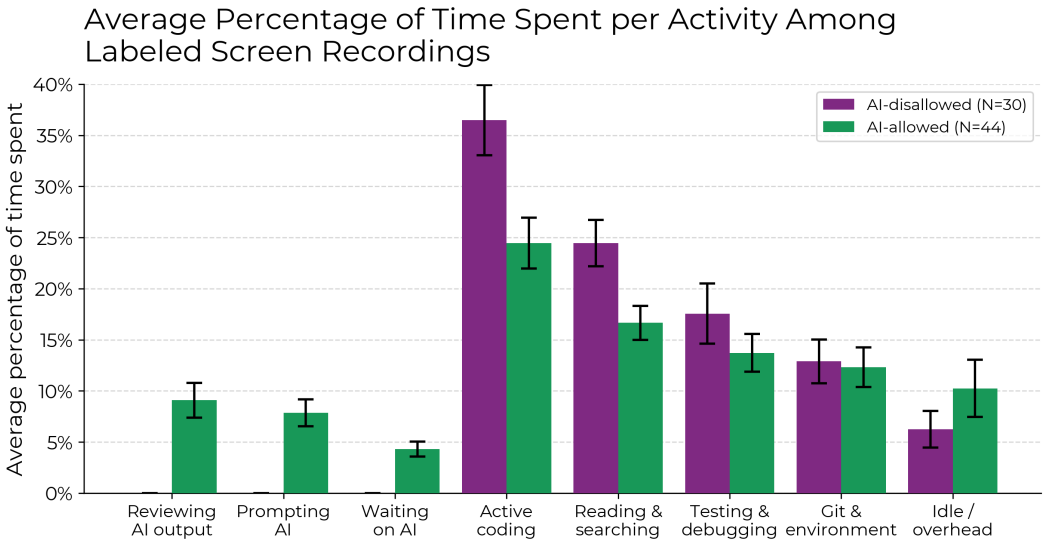

Figure 6: *On the subset of labeled screen recordings, when AI is allowed, developers spend less time actively coding and searching for/reading information, and instead spend time prompting AI, waiting on and reviewing AI outputs, and idle. Figure 19 shows the absolute (average) minutes spent in each category, and Figure 21 presents these results broken down into 27 fine-grained categories.*

3.3 Factor Analysis

Given the surprising nature of this result, we investigate 21 potential contributing factors that may contribute to developers spending more time on tasks when AI usage is allowed. We group these factors into four categories:

- **Direct productivity loss** (⏳): mechanisms by which the use of AI tools actively slows down development.
- **Experimental artifact** (🧪): confounders from our experimental setup or procedures that may introduce biases, or limit the external validity.
- **Raises developer performance** (👤): attributes of the issues, repositories, or setting that improve developer ability relative to AI.
- **Limits AI performance** (🤖): attributes of the issues, repositories, or AI/environment tooling that diminish AI's effectiveness relative to developers.

Using entry and exit surveys, screen recordings, developer interviews, and subset analyses we find qualitative and quantitative evidence that 5 of the 21 factors contribute to slowdown, we find mixed/unclear/no evidence that 10 of the factors contribute to slowdown, and we find evidence against 6 of the factors contributing. However, we strongly caution against over-indexing on the basis of any individual pieces of evidence, as we are not powered for statistically significant multiple comparisons when subsetting our data. This analysis is intended to provide speculative, suggestive evidence about the mechanisms behind slowdown. Appendix C discusses the evidence for/against each factor in Table 1.

4 Discussion

We provide evidence that recent AI systems slow down experienced open-source developers with moderate AI experience completing real issues on large, popular repositories they are highly familiar with. This observed slowdown serves as some evidence that AI capabilities in the wild may be lower than results on commonly used benchmarks may suggest.

Furthermore, we show that both experts and developers drastically overestimate the usefulness of AI on developer productivity, *even after they have spent many hours using the tools*. This underscores

Factors likely to contribute to slowdown

Factor	Type	Relevant Observations
Over-optimism about AI usefulness (C.1.1)	⏳	• Developers forecast AI will decrease implementation time by 24% • Developers post hoc estimate AI decreased implementation time by 20%
High developer familiarity with repositories (C.1.2)	👤	• Developers slowed down more on issues they are more familiar with • Developers report that their experience makes it difficult for AI to help them • Developers average 5 years experience and 1,500 commits on repositories
Large and complex repositories (C.1.3)	🤖	• Developers report AI performs worse in large and complex environments • Repositories average 10 years old with >1,100,000 lines of code
Low AI reliability (C.1.4)	🤖	• Developers accept <44% of AI generations • Majority report making major changes to clean up AI code • 9% of time spent reviewing/cleaning AI outputs
Implicit repository context (C.1.5)	🤖👤	• Developers report AI doesn't utilize important tacit knowledge or context

Factors with unclear effect on slowdown

Factor	Type	Relevant Observations
Experimentally driven overuse of AI (C.2.1)	🧪	• Developers sometimes report overuse due to experiment • Similar slowdown from developers reporting overuse vs. normal use
Unrepresentative task distribution (C.2.2)	🧪	• Developers report issues are standard but on the shorter side • Excludes non-programming tasks developers complete in normal work
AI increasing issue scope (C.2.3)	🧪	• Developers who report scope creep see *less* slowdown • Mixed developer reports on AI's impact on scope • 47% more lines of code per forecasted hour in AI-allowed issues
Bias from issue completion order (C.2.4)	🧪	• Developers decide order post randomization
Sampling bias in developer recruitment (C.2.5)	🧪	• Developers who rely heavily on AI may be less likely to participate
Trading speed for ease (C.2.6)	⏳	• Some developers report using AI is less effortful • High developer retention on Cursor
Low quality initial pull requests (C.2.7)	⏳	• Minor difference in mean post-review times between conditions • Qualitatively similar PR quality between conditions
Below-average use of AI tools (C.2.8)	🤖	• Similar slowdown for developers with prior Cursor experience • No clear learning effect across first 30-50 hours of Cursor usage • Developers appear qualitatively in distribution for Cursor Pro users
AI generation latency (C.2.9)	🤖	• Mixed developer reports that waiting on AI generations was important • Developers spend 4% of time waiting on AI generations
Suboptimal elicitation (C.2.10)	🤖	• Developers use Cursor agents/chat in majority of AI-allowed issues • Developers sample few tokens from models • But existing literature finding positive speedup also uses few tokens • Unused elicitation strategies could improve AI reliability

Factors unlikely to contribute to slowdown

Factor	Type	Relevant Observations
Unfamiliar development environment (C.3.1)	🧪	• Most developers use comparable IDEs between treatment conditions • These developers still see slowdown of 24% • No clear learning effects across first 30-50 hours of Cursor usage
Cheating or under-use of AI (C.3.2)	🧪	• AI used in all but 16.4% of allowed cases with labeled screen recordings • Only 3 cheating instances in 54 screen recordings
Issue dropout (C.3.3)	🧪	• Developers with no accidental dropout see similar slowdown • Issues dropped intentionally are qualitatively unbiased
Non-robust outcome measure (C.3.4)	🧪	• Alternative outcome measures yield similar slowdown
Non-robust estimator (C.3.5)	🧪	• Alternative estimators yield similar slowdown
Non-frontier model usage (C.3.6)	🤖	• Developers mostly use (at the time) frontier models

Table 1: *Summary of factors that may* a priori *explain or contribute to slowdown, grouped by the state of evidence for or against their impact on the slowdown effect.* 👤 *are factors that raise human performance,* 🤖 *are factors that limit AI performance,* 🧪 *are experimental artifacts that may bias/-confound results, and* ⏳ *are factors that directly contribute to productivity losses.*

the importance of conducting field experiments with robust outcome measures, compared to relying solely on expert forecasts or developer surveys.

4.1 Key Caveats

Setting-specific factors We caution readers against overgeneralizing on the basis of our results. The slowdown we observe does *not* imply that current AI tools do not often improve developer's productivity—we find evidence that the high developer familiarity with repositories and the size and maturity of the repositories both contribute to the observed slowdown, and these factors do not apply in many software development settings. For example, our results are consistent with small greenfield projects or development in unfamiliar codebases seeing substantial speedup from AI assistance.

AI-specific factors We expect that AI systems that have higher fundamental reliability, lower latency, and/or are better elicited (e.g. via more inference compute/tokens, more skilled prompting/scaffolding, or explicit fine-tuning on repositories) could speed up developers in our setting (i.e. experienced open-source developers on large repositories).

Agents can make meaningful progress on issues We have preliminary evidence (forthcoming) that fully autonomous AI agents using Claude 3.7 Sonnet can often correctly implement the core functionality of issues on several repositories that are included in our study, although they fail to fully satisfy all requirements (typically leaving out important documentation, failing linting/styling rules, and leaving out key unit or integration tests). This represents immense progress relative to the state of AI just 1-2 years ago, and if progress continues apace (which is *a priori* at least plausible, although not guaranteed), we may soon see significant speedup in this setting.

5 Acknowledgments

We thank the open-source developers who participated in this study. Your hard work, diligent record keeping, and excellent software made it a pleasure to work with you. Thanks to Aaron Diamond-Reivich, Alan Akbik, Domenic Denicola, Dens Sumesh, Jaden Fiotto-Kaufman, João Gante, Liam DeVoe, Matthew Pickering, Muhammad Haris, Philipp Burckhardt, Quentin Anthony, Ruben Bloom, Sam Derbyshire, and other participating developers.

We thank the following reviewers for feedback on the experimental design and paper drafts: Adrien Ecoffet, Alexander Barry, Ali Merali, Ajeya Cotra, Andres Campero, Andrey Fradkin, Basil Halperin, Cozmin Ududec, Eli Lifland, Ernest Davis, Gregory Sun, Hjalmar Wijk, James Requeima, Jide Alaga, Josh Jacobson, Lawrence Chan, Megan Kinniment, Michael Sklar, Neev Parikh, Rif A. Saurous, Rob Miles, Ryan Greenblatt, Seraphina Nix, Sydney Von Arx, Thomas Kwa, and Tom Cunningham.

We thank the following individuals for help with data collection: Adam Hanson, Amy Ngo, Chris Canal, Jebastin Nadar, Luis Slyfield, and Martin Milbradt.

We thank Sami Jawar and Thomas Broadley for technical support throughout the project.

We thank the following for their operational support through the project: Bhaskar Chaturvedi, Emma Abele, Kit Harris, Kris Chari, Kyle Scott, Rebecca Baron, and Rae She.

The authors thank Stephanie He for graphic design contributions.

The authors especially thank Aron Lajko, Chris Painter, Jasmine Dhaliwal, and Steve Newman for close review, feedback, and support throughout the project.

References

[1] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarlasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. Artificial intelligence index report 2025, 2025. URL https://arxiv.org/abs/2504.07139.

[2] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long tasks, 2025. URL https://arxiv.org/abs/2503.14499.

[3] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts, 2025. URL https://arxiv.org/abs/2411.15114.

[4] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. URL https://arxiv.org/abs/2410.07095.

[5] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai's ability to replicate ai research, 2025. URL https://arxiv.org/abs/2504.01848.

[6] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, Zekun Wang, Jian Yang, Zeyu Cui, Yang Fan, Yichang Zhang, Binyuan Hui, and Junyang Lin. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings, 2025. URL https://arxiv.org/abs/2501.01257.

[7] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering?, 2025. URL https://arxiv.org/abs/2502.12115.

[8] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[9] David Rein, Joel Becker, Amy Deng, Seraphina Nix, Chris Canal, Daniel O'Connel, Pip Arnott, Ryan Bloom, Thomas Broadley, Katharyn Garcia, Brian Goodrich, Max Hasin, Sami Jawhar, Megan Kinniment, Thomas Kwa, Aron Lajko, Nate Rush, Lucas Jun Koba Sato, Sydney Von Arx, Ben West, Lawrence Chan, and Elizabeth Barnes. Hcast: Human-calibrated autonomy software tasks, 2025. URL https://arxiv.org/abs/2503.17354.

[10] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of ai on developer productivity: Evidence from github copilot, 2023. URL https://arxiv.org/abs/2302.06590.

[11] Elise Paradis, Kate Grey, Quinn Madison, Daye Nam, Andrew Macvean, Vahid Meimand, Nan Zhang, Ben Ferrari-Church, and Satish Chandra. How much does ai impact development speed? an enterprise-based randomized controlled trial, 2024. URL https://arxiv.org/abs/2410.12944.

[12] Inioluwa Deborah Raji, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. AI and the everything in the whole wide world benchmark. *CoRR*, abs/2111.15366, 2021. URL https://arxiv.org/abs/2111.15366.

[13] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2024. URL https://arxiv.org/abs/2405.14782.

[14] Ernest Davis. Benchmarks for automated commonsense reasoning: A survey, 2023. URL https://arxiv.org/abs/2302.04752.

[15] Leonardo Gambacorta, Han Qiu, Shuo Shan, and Daniel M Rees. *Generative AI and labour productivity: a field experiment on coding*, volume 1208. Bank for International Settlements, Monetary and Economic Department, 2024.

[16] Doron Yeverechyahu, Raveesh Mayya, and Gal Oestreicher-Singer. The impact of large language models on open-source innovation: Evidence from github copilot, 2025. URL https://ssrn.com/abstract=4684662.

[17] Zheyuan Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. The effects of generative ai on high-skilled work: Evidence from three field experiments with software developers, June 2025. URL https://ssrn.com/abstract=4945566.

[18] Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. Significant productivity gains through programming with large language models. *Proc. ACM Hum.-Comput. Interact.*, 8(EICS), June 2024. doi: 10.1145/3661145. URL https://doi.org/10.1145/3661145.

[19] OpenAI. OpenAI o3 and o4-mini System Card. https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf, 2025. [Accessed 23-06-2025].

[20] Anthropic. Anthropic Claude 4 System Card. https://www-cdn.anthropic.com/6be99a52cb68eb70eb9572b4cafad13df32ed995.pdf, 2025. [Accessed 23-06-2025].

[21] Ajay Agrawal, Joshua S. Gans, and Avi Goldfarb. Artificial intelligence: The ambiguous labor market impact of automating prediction. *Journal of Economic Perspectives*, 33(2):31–50, May 2019. doi: 10.1257/jep.33.2.31. URL https://www.aeaweb.org/articles?id=10.1257/jep.33.2.31.

[22] Erik Brynjolfsson, Danielle Li, and Lindsey Raymond. Generative ai at work*. *The Quarterly Journal of Economics*, 140(2):889–942, 02 2025. ISSN 0033-5533. doi: 10.1093/qje/qjae044. URL https://doi.org/10.1093/qje/qjae044.

[23] Shakked Noy and Whitney Zhang. Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, 381(6654):187–192, 2023. doi: 10.1126/science.adh2586. URL https://www.science.org/doi/abs/10.1126/science.adh2586.

[24] Jonathan H. Choi and Daniel Schwarcz. Ai assistance in legal analysis: An empirical study. *Journal of Legal Education*, 73(2), 2025. URL https://jle.aals.org/home/vol73/iss2/5/.

[25] Anthropic. Responsible scaling policy evaluations report – claude 3 opus. Technical report, Anthropic, 2024. URL https://cdn.sanity.io/files/4zrzovbb/website/210523b8e11b09c704c5e185fd362fe9e648d457.pdf. Accessed June 2025.

[26] C. Mouton, Caleb Lucas, and Ella Guest. The operational risks of ai in large-scale biological attacks. Research report, RAND Corporation, Santa Monica, 2024. URL https://www.rand.org/content/dam/rand/pubs/research_reports/RRA2900/RRA2977-2/RAND_RRA2977-2.pdf.

[27] Aaron Grattafiori et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[28] Tejal Patwardhan, Kevin Liu, Todor Markov, Neil Chowdhury, Dillon Leet, Natalie Cone, Caitlin Maltbie, Joost Huizinga, Carroll Wainwright, Shawn (Froggi) Jackson, Steven Adler, Rocco Casagrande, and Aleksander Madry. Building an early warning system for LLM-aided biological threat creation. https://openai.com/index/building-an-early-warning-system-for-llm-aided-biological-threat-creation/, January 2024. Accessed: 2025-06-25.

[29] Jared Leibowich, Nikola Jurkovic, and Tom Davidson. Could advanced ai accelerate the pace of ai progress? interviews with ai researchers, 2024. URL https://ssrn.com/abstract=5115692.

[30] Ege Erdil and Tamay Besiroglu. Explosive growth from ai automation: A review of the arguments, 2024. URL https://arxiv.org/abs/2309.11690.

[31] Ege Erdil, Andrei Potlogea, Tamay Besiroglu, Edu Roldan, Anson Ho, Jaime Sevilla, Matthew Barnett, Matej Vrzla, and Robert Sandler. Gate: An integrated assessment model for ai automation, 2025. URL https://arxiv.org/abs/2503.04941.

[32] Tom Davidson. What a compute-centric framework says about takeoff speeds. https://www.openphilanthropy.org/research/what-a-compute-centric-framework-says-about-takeoff-speeds/, June 2023. Open Philanthropy. Accessed: 2025-06-25.

[33] Daron Acemoglu. The simple macroeconomics of ai. *Economic Policy*, 40(121):13–58, 08 2024. ISSN 0266-4658. doi: 10.1093/epolic/eiae042. URL https://doi.org/10.1093/epolic/eiae042.

[34] Ajay Agrawal, Joshua Gans, and Avi Goldfarb. Economic policy for artificial intelligence. *Innovation Policy and the Economy*, 19:139–159, 2019. doi: 10.1086/699935. URL https://doi.org/10.1086/699935.

[35] Jason Furman and Robert Seamans. *AI and the Economy*, pages 161–191. University of Chicago Press, May 2018. doi: 10.1086/699936. URL http://www.nber.org/chapters/c14099.

[36] Stefano DellaVigna, Nicholas Otis, and Eva Vivalt. Forecasting the results of experiments: Piloting an elicitation strategy. *AEA Papers and Proceedings*, 110:75–79, May 2020. doi: 10.1257/pandp.20201080. URL https://www.aeaweb.org/articles?id=10.1257/pandp.20201080.

[37] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. doi: 10.1198/016214506000001437. URL https://doi.org/10.1198/016214506000001437.

[38] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[39] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

[40] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

[41] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

[42] Alberto Abadie, Susan Athey, Guido W Imbens, and Jeffrey M Wooldridge. When should you adjust standard errors for clustering?*. *The Quarterly Journal of Economics*, 138(1):1–35, 10 2022. ISSN 0033-5533. doi: 10.1093/qje/qjac038. URL https://doi.org/10.1093/qje/qjac038.

[43] A. Colin Cameron and Douglas L. Miller. A practitioner's guide to cluster-robust inference. *Journal of Human Resources*, 50(2):317–372, 2015. ISSN 0022-166X. doi: 10.3368/jhr.50.2.317. URL https://jhr.uwpress.org/content/50/2/317.

[44] R.M. Bell and Daniel Mccaffrey. Bias reduction in standard errors for linear regression with multi-stage samples. *Survey Methodology*, 28:169–181, 01 2002.

[45] Stack Overflow. 2024 developer survey: Technology - integrated development environment, 2024. URL https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment. Accessed: June 26, 2025.