TURING

REPORT

Building at the Intersection of Human and Artificial Intelligence

Exploring LLM-based generative AI coding assistants for application development

Table of Contents

Executive Summary

As businesses strive to improve operational efficiency, speed up time-to-market, and navigate the challenges of talent acquisition and retention, AI coding assistants have emerged as a transformative way to maximize their engineering workforce.

This report delves into the adoption landscape of AI, the opportunities afforded to businesses that adopt it, and the results found in using these coding assistants for application development productivity. As more businesses recognize the potential of AI to transform their application development processes, those who adapt quicker have the opportunity to grow faster than their competition.

Introduction

As the software development industry grapples with challenges around improving operational efficiency, speeding up time-to-market, and finding and retaining quality talent, AI coding assistants have emerged as a potential solution.

What are AI coding assistants?

AI coding assistants, or just coding assistants, are software tools that use artificial intelligence to assist developers in writing code. They can automate repetitive tasks, provide real-time assistance, and improve code quality.

These tools can suggest code completions, detect and fix errors, and even write code snippets based on a developer's input. They can also provide insights into coding best practices and help developers understand complex code written by others.

AI coding assistants are designed to increase developer productivity by reducing the time it takes to write code and improving the overall quality of the code. They are becoming increasingly popular in the software development industry as a way to improve operational efficiency and speed up time-to-market.

Market opportunity

As organizations strive to improve operational efficiency, speed up time-to-market, and keep up with technological advancements and industry trends, the demand for AI-powered solutions is expected to rise. By addressing these pressing needs, AI can carve out a substantial niche in the application development landscape. Moreover, AI offers a promising solution to the ongoing struggle of finding and retaining quality application development talent, further underscoring its market potential.

As businesses across sectors increasingly rely on software for their operations, the demand for efficient, high-quality, and fast application development is on the rise. AI coding assistants, specifically, have the potential to solve issues around:

- **Efficiency and productivity:** AI coding assistants can significantly enhance developer productivity by automating repetitive tasks, providing real-time coding assistance, and improving code quality. This can lead to substantial cost savings and efficiency gains for businesses.
- **Talent shortages:** The tech industry is facing a talent shortage, especially for skilled developers. AI coding assistants can help bridge this gap by enabling existing developers to be more productive and making coding more accessible to people with diverse skill sets.
- **Speed to market:** In today's fast-paced digital landscape, getting to market quickly is crucial. AI coding assistants can speed up the application development process, helping businesses to launch their products faster and gain a competitive edge.
- **Innovation:** AI coding assistants can drive innovation by freeing up developers' time to focus on more complex and creative aspects of application development.

Given these benefits, the market for AI coding assistants is set to expand. As more businesses recognize the potential of AI to transform their application development processes, the industry opportunity for providers of these tools will continue to grow.

Market landscape and challenges

By 2030, the global AI market will reach $2 trillion[1]—a gigantic number, considering its relatively recent popularization and adoption. Part of that investment is likely based on the potential for this technology to support application development. Application development revenue is expected to show an annual growth rate (CAGR 2023–2028) of 7.04%, resulting in a market volume of $234.7 billion by 2028[2]. We expect companies to increase their spending as AI capabilities expand to support more application development needs over time.

However, 74% of enterprises are not using or are still evaluating AI solutions[3]. But with worldwide IT spending projected to reach $4.6 trillion in 2023 and increasing another 8.6% in 2024[4], companies who can navigate their investment into AI-powered application development will be well rewarded.

Investing in building an internal center of excellence for AI coding assistants, or partnering with external vendors who utilize AI coding assistant-based development, will provide one of the fastest opportunities to grow your software engineering capabilities. Early data suggests developers using generative AI–based tools to perform complex tasks were 25 to 30% more likely to successfully complete versus those without[5].

Challenges to adoption include concerns about data security and privacy, the need for significant upfront monetary and time investment in AI technology and infrastructure, and the requirement for training and upskilling developers to effectively use these tools.

Despite these challenges, early adopters have reported significant improvements in operational efficiency and code quality.

[1] https://www.statista.com/statistics/1365145/artificial-intelligence-market-size/
[2] https://www.statista.com/outlook/tmo/software/application-development-software/worldwide?currency=usd
[3] https://www.oreilly.com/radar/ai-adoption-in-the-enterprise-2022/
[4] https://www.gartner.com/en/newsroom/press-releases/2023-04-06-gartner-forecasts-worldwide-it-spending-to-grow-5-percent-in-2023
[5] https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai

AI coding assistant types and capabilities

Autocompletion-based coding assistants

Some of the earliest kinds of AI coding assistants generally fall under the scope of autocompletion-based assistants. Some of the most popular assistants in this category include Github Copilot, Amazon Code Whisperer, and Codeium. While fine-tuning varies between different assistants in this group, they are all defined by a few distinct capabilities.

Line-level code autocompletion

This capability predicts and automatically completes lines of code as the developer is typing. It can reduce the amount of typing required, minimize syntax errors, and help developers remember and correctly implement various programming constructs and API calls.

Function-level code autocompletion

This capability predicts and automatically completes entire functions as the developer is typing. It can suggest function names, parameters, return types, and even the body of the function.

Block-level code generation

This capability generates entire blocks of code based on the developer's intent. It can create larger structures of code, such as loops, conditionals, and even entire classes.

Product	Line-level code autocompletion	Function-level code autocompletion	Block-level code generation	Fine-tuning capable?
Amazon CodeWhisperer	▲ Strong	▲ Strong	● Neutral	No
CodeComplete	▲ Strong	● Neutral	● Neutral	Yes
Codeium	▲ Strong	▲ Strong	▲ Neutral	Yes
FauxPilot	▲ Strong	● Neutral	● Neutral	No
Github Copilot	▲ Strong	▲ Strong	▲ Strong	No
Google Duet AI	▲ Strong	▲ Strong	▲ Strong	Yes
Replit Ghostwriter	▲ Strong	▲ Strong	▲ Strong	No
SourceGraph Cody	▲ Strong	▲ Strong	▲ Strong	Yes
Tabby	▲ Strong	● Neutral	● Neutral	Yes
Tabnine	▲ Strong	▲ Strong	▲ Strong	Yes (enterprise only)

Figure 1 - Autocompletion-based coding assistant capabilities

It's worth noting that the quality of the generated code can vary. These models are typically trained on common code repositories, which may not always adhere to the specific coding standards of certain industries or businesses. However, some tools offer the ability to fine-tune their models based on specific codebases, thus improving their potential for generating high-quality code.

The developer still needs to design each code module, so the design quality from the developer plays a critical role in the quality of the output from these products.

Chat-based coding assistants

As large language models (LLMs) continue to evolve, we're seeing the emergence of chat-based coding assistants—a concept likely inspired by ChatGPT. The process begins when a developer describes a coding task as a prompt and inputs it into the LLM, which in turn generates the initial code. The developer and LLM then work together, iteratively refining the produced code.

Product	Details	Recommendation
ChatGPT	Noted for its intuitiveness, this product has been embraced by developer communities worldwide. However, businesses worried about data security may opt to deploy their own on-prem, GPT-powered solution.	▲ Strong
Codeium Chat	Available as a VSCode plugin, Codeium Chat presents an efficient alternative to raw ChatGPT prompt-based coding. However, there are considerable constraints regarding the underlying LLM's coding ability. The basic version is free to use.	● Neutral
Cody AI Chat	This product has the added benefit of being able to provide sources for every answer and can be trained on your company's knowledge base. It has tight integration with Sourcegraph's products.	▲ Strong
Copilot Chat	Copilot is currently considered the industry standard. It's available as a VSCode plugin with an enterprise plan, with its standard version integrating directly into Neovim, JetBrains IDEs, Visual Studio, and Visual Studio Code.	▲ Strong
Cursor	Centered around Chat UX, Cursor signifies an upgrade from GPT prompt-based coding. While it relies purely on GPT-3 and GPT-4, other platforms such as Ghostwriter, Tabnine, and Codeium have developed their own proprietary models.	▲ Strong
Duet AI Chat	Although not released at the time of this publication, Google will also be releasing a chat-based coding assistant based on Palm-2. It has great integration with the GCP workstation, so ideal for a remote developer environment.	▲ Strong
StarCoder	This is HuggingFace's take on a GPT prompt-based solution and has shown quality akin to GPT-3 outputs.	● Neutral
Tabnine Chat	Tabnine Chat runs inside the IDE and is contextualized to the user's code. It boasts the ability to connect to customer repositories while maintaining security and compliance requirements.	▲ Strong

Figure 2 - Some of the current chat-based coding assistants in market

Chat-based coding assistants are shaping up as an interesting development in the field of LLMs, offering new ways to streamline and enhance the coding process.

Agent-based coding assistants

While previous coding assistant methodologies primarily focus on local automation, which involves partial automation of a developer's work, they often fail to capture intricate aspects of coding projects such as file dependencies and version control. These methods could theoretically incorporate such dependencies into tools like ChatGPT, but this approach has proven to be ineffective so far.

Moreover, it's important to consider the broader context of coding. This process encompasses more than just writing code—it involves a series of steps both before and after the coding stage.

We believe this will be the defining space for application development at an enterprise level, incorporating the best chat-based and autocompletion-based capabilities into a more robust development flow that combines human and AI inputs and outputs at every step.

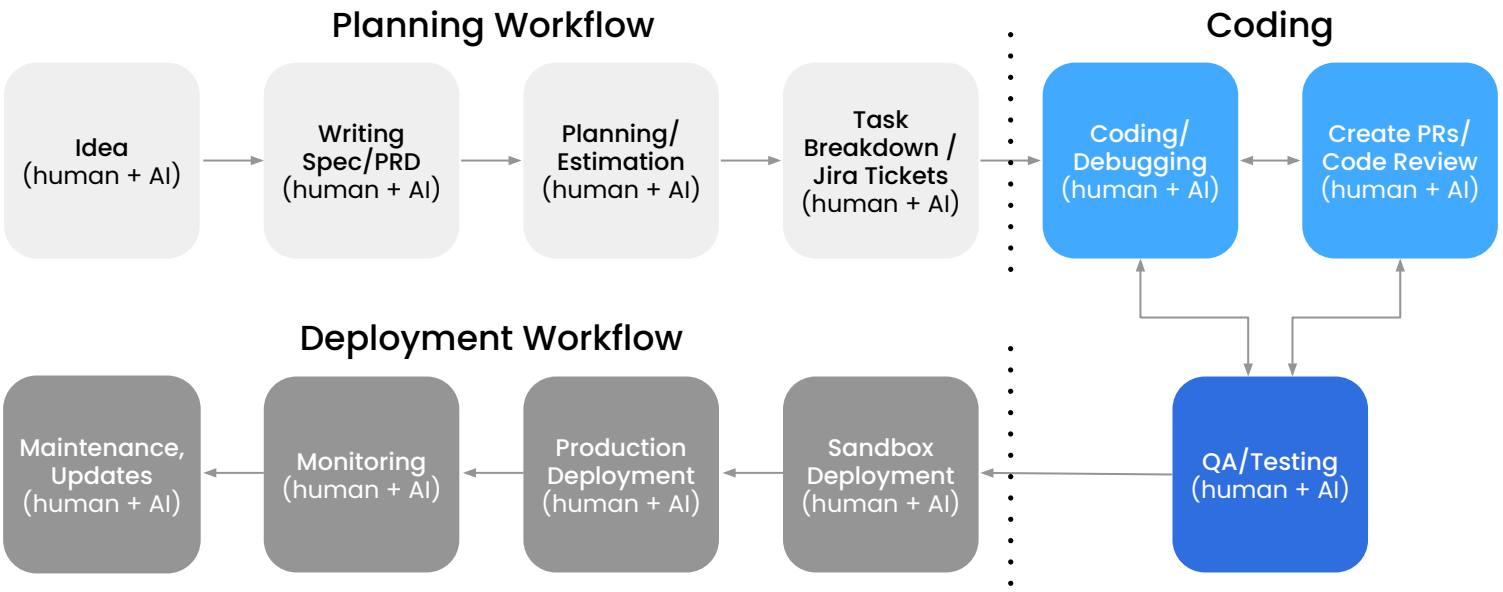

Figure 3 - Turing's GPT-powered software development flow

Agent-based assistants aim to streamline coding by creating a technical product requirements document (PRD) that an LLM can interpret and implement for an entire project. The LLM token limit currently restricts these projects, making it difficult to extend this approach to more complex tasks. However, with the anticipated increase in GPT-4's token limit from 100K to 1M, this constraint is expected to diminish.

Product	Details
AutoGPT	Still in its infancy, AutoGPT shows promise but isn't yet optimized for coding tasks. However, experimental demos have shown its potential.
GPT Engineer	GPT Engineer is made to be easy to adapt, extend, and make your agent learn how you want your code to look. It generates an entire codebase based on a prompt.
Smol Developer	Smol Developer is considered a prototype "junior developer" agent, scaffolding an entire codebase from a given product specification. Unlike other tools, it doesn't promise artificial general intelligence but offers a versatile alternative and allows developers to create custom applications.
Sweep	Sweep is an assistant that turns Github issues into Github pull requests, suggests fixes for the issue, and writes/pushes code to Github via a pull request while addressing comments made on the pull request.

Figure 4 - Current state of chat-based coding assistants

The evolution of agent-based coding assistants necessitates a shift in PRD development, requiring developers and project managers to craft documents that an agent can convert into actual code. This approach leverages not only the coding capabilities of an LLM but also its reasoning abilities for devising the project structure. Thus, akin to having personal junior developers, these assistants serve to enhance project management efficiency and output quality.

Testing methodology & evaluation metrics

Our internal research and development team examined the capabilities of several coding assistants and chose two for experimentation due to the depth of features available, such as fine-tuning the model and the ability to create a secure development environment that protects our proprietary code and information. We used our own internal, chat-based GPT solution enhanced by these coding assistants.

Then, we chose internal Turing developers who were actively engaged in application development work using the same programming languages and working on similar projects. This consisted of both developers who had used a coding assistant before and developers who hadn't.

The experiment included full-stack developers, data scientists, data engineers, and ML engineers.

AI coding assistant selection

While running experiment 1, we experienced adoption issues with our experimental group. A major issue with tooling adoption, due to the experiment group's remote employment, is they work within a fully cloud-based workstation. However, most of the existing coding assistants are designed for local integrated development environments (IDEs).

As a result, we additionally sourced a cloud-native coding assistant, which aligned better to the experimental group's development behaviors for experiment 2.

Methodology

Pre-intervention measurements

1. Gather engineer performance ratings and code quality metrics from team leads or managers for each participant.
2. Measure the number of lines of code (LOCs) and the number of successful pull requests (PRs) merged by each participant within a specified time frame.
3. Record pre-intervention measurements for both the experimental and control groups.

Intervention

1. Provide training and resources to the experimental group to help them effectively use the assistant in their daily work.
2. Encourage the experimental group to use the assistant for their tasks, including automating unit tests and writing Cypress tests.

Post-intervention measurements

1. Gather engineer performance ratings and code quality metrics from team leads or managers for each participant.
2. Measure the number of LOCs and the number of successfully reviewed and merged PRs generated by each participant within a specified time frame after the intervention.
3. Record post-intervention measurements for both the experimental and control groups.

Data analysis

1. Compare the pre- and post-intervention measurements of engineer performance ratings, code quality metrics, LOCs, and PRs for the experimental group to assess the efficiency improvement.
2. Compare the post-intervention measurements of the experimental group with the control group to evaluate the effectiveness of the intervention.

Time to complete tasks

1. Investigate the feasibility of measuring the time to complete tasks using Jira tickets.
2. If feasible, include time to complete tasks as an additional evaluation metric.

Evaluation metrics

Primary metrics

1. Change in engineer performance ratings
2. Change in code quality metrics
3. Change in the number of pull requests (PRs)
4. Change in the number of lines of code (LOCs)

Secondary metrics

1. Time to complete tasks (if feasible)
2. Qualitative feedback from participants regarding the usability and effectiveness of the internal GPT solution and the assistant
3. Internal GPT usage data

Experiment setup

The experiment included full-stack developers, data scientists, and data and ML engineers. We measured PR merge rates 5 weeks before and after assistant access for experiment 1, and 4 weeks before and after access for experiment 2. The developers selected for this experiment were divided into three groups:

Group A	No previous experience with a coding assistant. Given access to the assistant during the experiment.
Group B	No previous experience with a coding assistant. Not given access to the assistant during the experiment.
Group C	Previous experience with a coding assistant. Given access to the assistant during the experiment.

Merging group A and group C data in results

Since we observed PR increases in both groups A and C, we believe these two groups represent the different stages of efficiency improvement from the assistant. Thus, it's fairer to merge and compare them against those without the assistant.

Findings and analysis

Due to statistical outliers, like exceptionally large increases in results that were far out of band compared to others, we filtered some developers out of the results of this experiment.

Group	Participant count	PRs before	PRs after	% change
A	11	128	177	38.3%
B	25	620	547	-11.8%
C	22	475	519	9.3%
A + C	34	603	696	15.4%
B (After assistant access)	37	565	541	-4.2%

Figure 5 - Results from AI coding assistant experiment 1

Group	Participant count	PRs before	PRs after	% change
A	16	143	214	49.7%
B	49	450	497	10.4%
C	24	378	482	27.5%
A + C	40	521	696	33.6%
B (After assistant access)	49	971	1193	22.9%

Figure 6 - Results from AI coding assistant experiment 2

We saw an increase in successfully reviewed PRs, regardless of assistant

This is based on the results from both group A and group C. Group A shows a 38.3% to 49.7% increase in PRs, likely due to the developers using a coding assistant for the first time and having heavy initial productivity gains. Group C shows a 9.3% to 27.5% increase in PRs. These developers have used a coding assistant previously, but we still see improvement despite their familiarity.

Productivity improvement continues over time, regardless of assistant

Even with developers who had been using assistants since they were first commercially available, about 6 months ago, they were still experiencing efficiency gains. There's potential for further productivity improvement as developers are given more training on ways to use these assistants.

Larger increase in group A with experiment 2

Due to the aforementioned filtering of out-of-band results, the sample size is relatively low and may contribute to the larger variance in final results. However, the increase in PRs is high as these developers tend to be higher intent ones who would spend more time testing new technology, so we're being conservative when measuring the productivity gain. This is further impacted by experiment 2 having easier and higher adoption rates due to the cloud-based nature of the assistant used.

Changes with group B

During experiment 1, group B showed a -11.8%, which measures their productivity without a coding assistant. Afterwards, we provided them with a coding assistant and minimal training. The -4.2% result for their group is based on the initial results of this access and shows that even without much training they're able to show modest productivity gains.

For experiment 2, some of group B's 10.4% change in productivity is attributable to common development project fluctuations. Again, we provided this group with coding assistant access after the experiment, with minimal training, and saw a 22.9% change in productivity.

We saw a maximum average 33% productivity improvement

With groups A and C, we saw productivity improvements via successful PRs from 15-33%. Our research indicates that AI coding assistants can significantly enhance developer productivity by automating repetitive tasks, providing real-time assistance, and improving code quality.

With this early evidence, we believe further developer training on using these assistants optimally will lead to even higher productivity gains.

Further improvements on the horizon

Today, most coding assistants are based on GPT-3.5 or an open source equivalent. GPT-4 is more powerful, but has higher latency and associated costs, so it's less suitable for coding autocomplete use cases today. However, we expect GPT-4 equivalents to become faster and cheaper in the coming year—pushing productivity enhancements even farther.

Additionally, retrieval-augmented generation (RAG) shows promise in supporting greater coding assistant capabilities. RAG is an AI framework that enhances LLMs by retrieving information from an external knowledge base. This knowledge base allows LLMs to have access to the most accurate, up-to-date information on one or more subjects. This means coding assistants could train on and interpret extremely large code bases and not just the current and related files it may train on today.

Conclusion

Engineering excellence and AI-powered speed are within reach for most businesses willing to invest in AI coding assistants. However, building the necessary infrastructure in house to support it may prove challenging.

Considerations for adoption

Look for a versatile coding assistant

You want an assistant that can autocomplete lines of code and entire files, understand the local codebase via indexing, and debug potential issues by reading error messages or stack information.

Invest in more powerful LLM models

More powerful LLM models, including GPT-4 and Duet AI, are expected to provide more positive ROI.

Diversify to avoid productivity bottlenecks

Software developers typically spend ~30% of their time coding with the rest on other development activities. This includes planning, debugging, integration, deployment, QA, and maintenance. As we continue to see enhancements for coding, better ROI will come from similar AI tooling for productivity improvements across these other tasks.

Consult third-party expertise

We recommend working with vendors who have experience building and working with these kinds of AI solutions to build a Center of Excellence (CoE) within your organization to make the strategy, adoption, and implementation easier.

Alternatively, leveraging vendors who can provide engineering solutions using this technology may be the most economical in the short term, allowing your business to enjoy the operational efficiency and faster go-to-market without the financial and time investments needed to operate at scale. This also helps overcome the challenge of ongoing training and support needed to maximize the benefits of these tools.

Despite these challenges, the overall impact of AI coding assistants on the software development landscape is positive, offering increased flexibility, scalability, and reach.

About Turing

Based in Palo Alto, California, Turing is the world's first AI-powered tech services company. It has reimagined tech services from the ground up with AI by offering AI-vetted and matched talent, AI-accelerated development, and access to AI transformation experts who have built many of the most iconic Silicon Valley companies.

Founded in 2018, the company has experienced tremendous growth with over two million global developers on its Talent Cloud and 900+ clients. Turing has received numerous awards, including *Forbes*'s 2022 "One of America's Best Startup Employers," being ranked #1 in *The Information*'s 2021 Annual List of most promising B2B Companies and *Fast Company*'s "Annual List of the World's Most Innovative Companies." The company's most recent private fundraising round was oversubscribed and valued the company at $1.1 billion. Subsequent oversubscribed SAFEs were completed on a $4 billion valuation cap. The company's leadership team comprises both AI technologists from leading organizations including Meta, Google, Microsoft, Apple, Amazon, Twitter, Stanford, Caltech, MIT as well as tech consulting veterans from Accenture, Cognizant, Capgemini, McKinsey, Bain, and more.