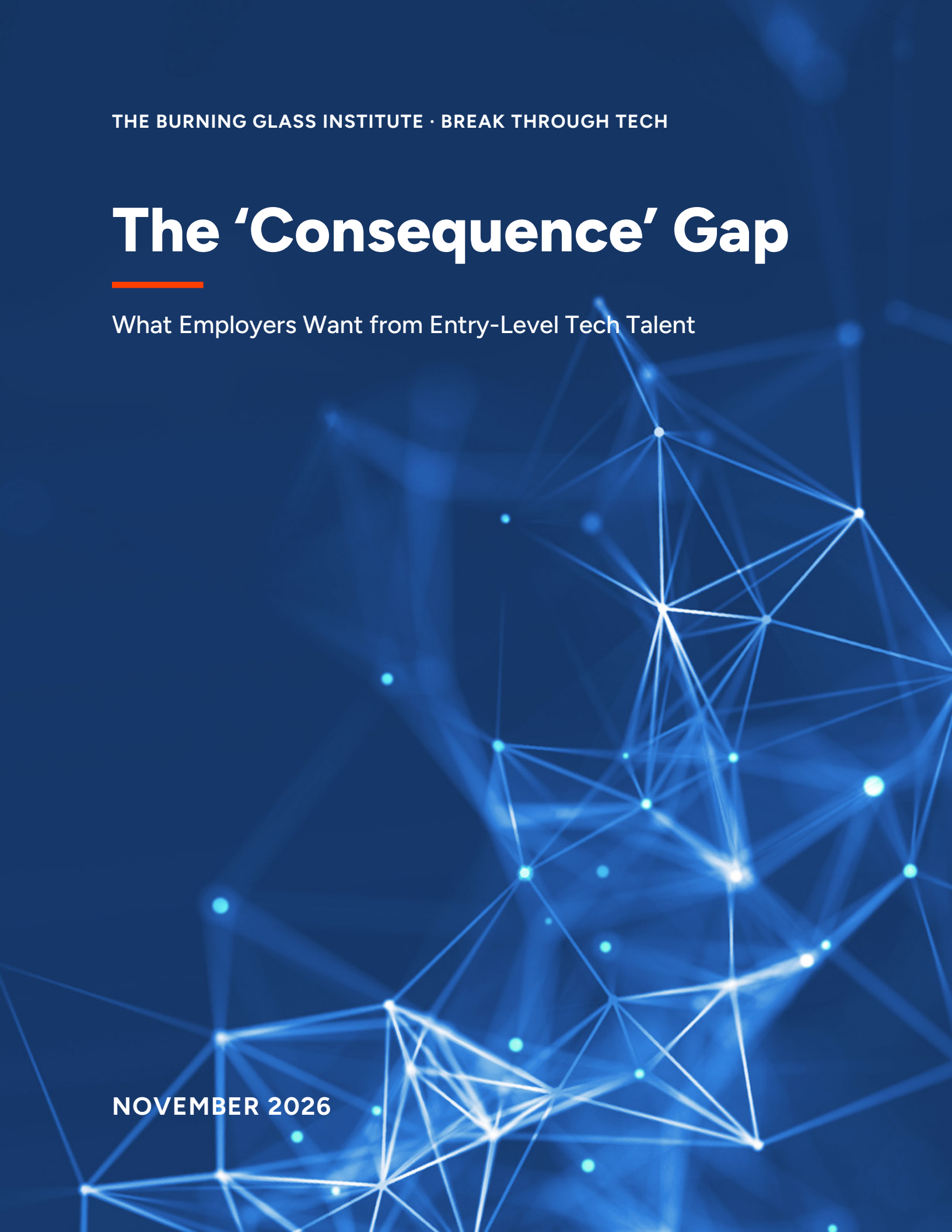
THE BURNING GLASS INSTITUTE · BREAK THROUGH TECH

The 'Consequence' Gap

What Employers Want from Entry-Level Tech Talent



NOVEMBER 2026

The ‘Consequence’ Gap

What Employers Want from Entry-Level Tech Talent

Acknowledgments

This report is the product of a collaboration between the Burning Glass Institute and Break Through Tech. Several members of the Burning Glass Institute team contributed substantially to its development, including Carlo Salerno, Cecilia Joy Perez, Suchitra Akmanchi, Caylee O’Connor, and Gillian Perkins. We are especially grateful to Judith Spitz, Founder and CEO of Break Through Tech, for her partnership, insights, and ongoing collaboration throughout this project as well as Christian McIntire and Hari Raghaven from Break Through Tech for their contributions to this work. Finally, we thank Break Through Tech for generously funding this research and making the work possible.

© Burning Glass Institute 2026

Table of Contents

Executive Summary 4

Introduction 7

1. Foundational Computer Science and Software Engineering Skills Still Matter 9

2. Demand for Code Review Has Doubled, and Its Scope Is Expanding 11

3. Employers Are Asking For More of the Work that Follows Code Generation 14

4. AI-Specific Skills Are the Fastest-Growing, But Still Account for a Small Share of the Market 16

5. AI Is Reshaping Work Across the Software Development Lifecycle 18

6. Employers Are Seeking Business, Product and Engineering 'Judgment' but Do Not Yet Share a Common Vocabulary for It 19

7. Technical Assessment Is in Transition 20

Implications and Recommendations 21

Appendix: Methodology 26

Works Cited 29

Endnotes 30

© Burning Glass Institute 2026

Executive Summary

Developments in AI have transformed the nature of technical work. AI tools can now generate code, documentation, tests, and other software artifacts across the development lifecycle — typically faster and at lower cost than before. For entry-level candidates, this shift creates uncertainty about which skills remain most important, which tasks employers will expect junior hires to perform, and how candidates can acquire and demonstrate that they are ready for the work.

This white paper examines what employers seek from new college graduates entering technology roles. It draws on two forms of evidence: analysis of more than one million U.S. job postings for entry-level positions across 32 technology occupations from January 2020 through June 2026 and structured conversations and convenings with 54 practitioners across 46 organizations conducted between May and July 2026. Job posting data show how skills demand is changing across the market. Practitioner conversations help interpret what familiar skill labels mean in practice, identify expectations that employers do not consistently state in postings, and illuminate changes in interview and assessment practices.

The findings suggest that as AI takes on more of the code generation work for many companies, employers are less likely to judge candidates on their ability to write code themselves and more on the skills and competencies that enable them to take responsibility for it. There is greater value in the ability to specify work clearly, evaluate AI outputs critically, integrate and maintain software in existing systems, and make sound technical and product tradeoffs amid ambiguity.

Five themes organize the findings that follow. The body takes them up in the same order, with the second and third themes each spanning two sections.

1. **Foundational computer science and software engineering skills remain essential, even as employers name fewer programming languages.**
   From 2020 to 2025, the share of entry-level postings naming 12 of the 16 programming languages and related coding tools tracked in this study declined. Python was the notable exception, rising from 22% to 33% of postings; Go, TypeScript, and Rust also gained. At first glance, this suggests that traditional computer science programs may be losing their claim on the curriculum. Practitioner conversations point to a different interpretation. Not one practitioner that we spoke to said that programming fundamentals had become less important. Instead, they emphasized the importance of reasoning about code, understanding data structures and algorithms, debugging, testing, and recognizing when a solution is not secure, inefficient, unnecessarily complex, or poorly suited to the problem. In many respects, coding has shifted from being the thing employers hire for to an essential competency that enables entry-level hires to reliably ship and maintain the end product. AI may reduce the value of prior exposure to a long list of languages while increasing the value of the underlying ability to write, understand, assess, and improve code in any one language.
2. **Demand is rising for work that supports code generation: reviewing it, integrating it, and keeping it running in production.**
   Employers listed code review in twice as many entry-level postings in 2025 as in 2020: 15.6%, up from 7.8%. That doubling signals growing demand for verification: employers want new hires who can review code, not only write it. However, the term "code review" doesn't necessarily mean what it used to.

Practitioner conversations suggested that it increasingly refers to the ability to evaluate AI-generated code: identifying outputs that are plausible but incorrect, vulnerable, biased, inefficient or unnecessarily complex. In short, the error patterns in AI-generated code are not necessarily the same as those in human-generated code.

Production-oriented skills also gained ground. Employers included CI/CD in 19.4% of postings and Kubernetes in 14.5% in 2025, with complementary containerization, cloud, reliability, and monitoring skills also rising. These patterns align with a growing expectation that entry-level engineers contribute not only to developing software, but also to deploying, monitoring, troubleshooting, and maintaining it after release.

3. **AI-specific skills are the fastest-growing part of the market—but still among the smallest parts of the labor market data.**

   Technical skills associated with building, integrating, or operating AI and agentic systems are clearly increasing in importance. Postings citing AI agent development rose more than twelvefold, from 0.14% in 2024 to 1.8% in 2025, and kept climbing in the first half of 2026; LangChain, retrieval-augmented generation, and prompt engineering also grew rapidly, though each appeared in fewer than 2% of postings in 2025. Practitioner conversations indicate that interviews may be moving ahead of job descriptions, with some employers testing candidates' ability to reason about AI tools, model choices, cost, context, and integration.

   AI's effect also reaches past any single skill to the shape of the work. While the software development lifecycle itself has not changed, what has changed is that entry-level candidates are now expected to work alongside an AI agent at every phase of that lifecycle and the amount of time spent on each phase of the lifecycle is being redistributed.

4. **Employers are seeking business, product and engineering 'judgment', but do not yet share a vocabulary for it.**

   In conversations, practitioners consistently described looking for entry-level candidates who can make and explain tradeoffs, rather than simply execute a well-defined task. They used different terms, such as engineering judgment, business judgment, product judgment, and "shot selection," but described similar capabilities: clarifying ambiguous requests, translating them into technical requirements, weighing tradeoffs among cost, reliability, security, speed, and functionality, and explaining why one approach is preferable to another.

   These are not new dimensions of strong technical work. What is changing is their importance at the entry level. While AI tools can enable less-experienced workers to generate more code or prototype more quickly, the challenge is increasingly to determine what should be built, assess whether an AI solution is sound, and make defensible choices about how it should be implemented. These expectations are difficult to observe in postings because employers do not yet consistently name them, and familiar labels may now encompass broader or different work than they once did.

5. **Technical assessment is in transition, creating uncertainty for educators and entry-level candidates.** For years, technical hiring relied on coding screens, whiteboard exercises, algorithmic problems, and platforms such as LeetCode to assess programming and problem-solving ability without assistance. Practitioner conversations indicate that some employers are now experimenting with assessments that more closely resemble AI-enabled work: open-book system-design exercises, unfamiliar-domain case studies, work on existing or legacy code, and tasks requiring candidates to identify and correct problems in imperfect AI-generated outputs.

The transition is uneven. A January 2026 Karat survey of engineering leaders found that many organizations still prohibit AI use in interviews while also reporting that AI has made technical skills harder to assess. The result is not a new consensus but a period of experimentation, with varied interview formats and limited agreement about what AI-enabled entry-level assessment should measure.

The implications for learning are clear. Undergraduate curricula and credentialing pathways should continue to devote time to mastering computer science and software engineering fundamentals. But those foundations increasingly need to be paired with opportunities to practice interpreting incomplete requirements, reviewing AI outputs, deploying systems into a production environment, working with and managing existing systems that have real constraints and generate real consequences, and making technical and product tradeoffs.

Many of these capabilities are difficult to develop through isolated, greenfield classroom projects alone. They require practice with imperfect code, existing codebases, ambiguous problems, and realistic constraints. This points toward a need for expanded applied-learning opportunities including internships and apprenticeships, but also scalable new alternatives such as structured simulations, micro-internships, practitioner-supported projects, and reusable technical environments.

Employers and funders have an important role to play by (i) clarifying the capabilities they seek, (ii) sharing assessment approaches and scoring criteria, (iii) contributing realistic technical tasks, (iv) providing sanitized codebases, and (v) helping education and workforce development organizations create and scale learning experiences that better reflect the work entry-level hires will be asked to do.

Introduction

Writing code has historically been the gateway into a career in tech. For two decades the pathway was clear to everyone involved: a college student learned the problem solving and programming skills required to produce software. In an interview for an entry-level role, a prospective employer tested whether the student could effectively leverage those skills in a software development scenario. And the first two years on the job turned that skill set and potential into something broader: the ability to solve real business problems and deliver business value with software systems. In short, the interview matched the job, and the job built the developer.

Generative AI is putting that pathway under pressure. Traditional curriculum is challenged because the competencies employers now name are ones the classroom was not designed to produce. The job itself now requires candidates to have skills that are typically learned during the first couple of years on the job. And the interview protocols that screened for a reasonably consistent set of skills and competencies are now in flux. Employers can feel the change; most cannot yet codify what they are screening for. **That is the gap this paper documents: a set of expectations that has outrun the vocabulary available to describe it, and the resulting challenge to define, teach, or test for what now matters most**.

This raises related questions for educators, employers, workforce-development organizations, and students. Which capabilities remain foundational? Which are rising in importance? Which are visible in job postings, and which are difficult for employers to name or assess consistently? And if some of the most valued capabilities are developed through exposure to existing systems, ambiguous requirements, and real operational constraints, how can more learners gain meaningful opportunities to build them?

Job postings and practitioners address these questions in different ways, and this paper is built on the difference. Postings measure how demand for a skill is moving across the whole market. They cannot tell you what a familiar skill label now means, and they cannot show you a requirement that employers have stopped writing down because it has become assumed.

This paper draws on more than one million U.S. job postings for entry-level positions across 32 technology occupations, published between January 2020 and June 2026, alongside structured conversations and convenings with 54 practitioners across 46 organizations conducted between May and July 2026. Fifteen participants held individual-contributor technical roles (e.g., software engineer, ML engineer); 39 held managerial or executive roles, ranging from engineering managers to C-suite and EVP-level leaders. Organizations ranged from Big Tech platforms and AI-native startups to financial-services firms, healthcare systems, and industrial and consumer companies with substantial internal technology organizations.

The two sources are most useful when read together. Postings show how explicitly named skill demand is changing across the market. Practitioner conversations help interpret what familiar labels mean in practice, identify emerging expectations that have not yet become standard posting language, and illuminate changes in technical interviews and assessment. At the same time, practitioner conversations cannot establish how widespread demand for any individual skill or competency is across the market.

The competencies this paper identifies share a common challenge: they are learned against systems and codebases that already exist, under business and operational constraints that closely resemble real production and professional environments. They are skills that are difficult to develop through traditional

classroom instruction or in greenfield classroom projects alone. Being prepared hinges on access as much as pedagogy. Only 44% of undergraduates who want a paid internship get one, and students of color, first-generation students, community college students and financially vulnerable students face the worst odds of all.[1] If the skills that now decide who gets hired can only be built in placements that are already rationed, then the shift this paper documents raises the price of entry most sharply for the students least able to pay it. That makes the supply of applied learning a hiring problem, not only an education one.

The sections that follow take the findings in order. Section 1 examines the fundamentals employers still expect; Sections 2 and 3, the rise of review and production work; Section 4, AI-specific skills; Section 5, how AI is redistributing effort across the development lifecycle; Section 6, the judgment employers seek; and Section 7, the emerging gap in technical assessment. A final section draws out the implications for educators, workforce-development organizations, employers, training companies, talent acquisition professionals, and funders.

Brief Methodology Note

This paper pairs time series analysis with in-depth interviews to identify the entry-level skills that get college graduates hired in entry-level tech roles today. The quantitative evidence is based on an analysis of more than one million job postings covering 32 technology sub-occupations. The qualitative evidence comes from structured conversations that Break Through Tech conducted between May and July 2026 with practitioners who lead, hire, screen, and train people for tech roles (cited throughout as "Practitioner Conversations").

This paper looks at demand across three windows: a five-year horizon covering the change between 2020 and 2025, a one-year comparison of 2024 with 2025, and a six-month view that captures the most recent skill trends, from January to June 2026. A note on time-series comparisons: job postings data are collected from public web sources, and month-to-month changes can reflect shifts in how postings are captured rather than true changes in employer demand. Readers should treat the direction and rough magnitude of trends as more reliable than any single period's value.

The methodological pairing employed in this analysis is necessary because each instrument sees what the other cannot. Postings tell us how a skill is trending: they let us quantify rising and falling skill shares, proxies for demand, across the whole market. Practitioner conversations fill two gaps that postings cannot. First, they clarify what familiar skill labels now mean in practice. Second, they surface emerging skills and competencies that have not yet appeared consistently in job-posting language. Additionally, practitioner conversations offer a window into changes in interview protocol, an early signal of where job descriptions may be heading. These conversations help us identify shifts in skill demand before those shifts are fully visible in postings.

Throughout this paper, we use job postings to measure how a skill is trending, and practitioner conversations to interpret what the skill consists of, and to identify emerging skills that are not yet reflected in labor market data. A comprehensive methodology for the study is presented in the Appendix.

1. Foundational Computer Science and Software Engineering Skills Still Matter

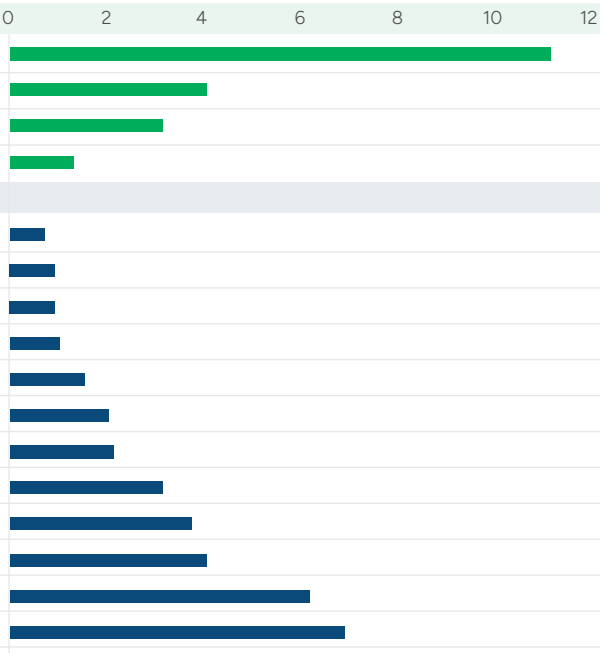
Generative AI can make it faster and easier to produce a first draft of code. But it does not remove the need for people who can understand, evaluate, debug, test, and maintain that code in the context of a larger system. As AI takes on more of the code-generation work, employers may place less value on a candidate's prior exposure to a long list of languages and greater value on the foundational knowledge that enables them to assess and take responsibility for technical outputs. Computer science and software engineering fundamentals are still central to entry-level technical work, according to practitioners, even as employers name fewer programming languages in job postings. Taken together, these signals suggest a shift toward capabilities that sit above any particular language or tool: the ability to read and reason about code, understand data structures and algorithms, and determine whether a proposed technical solution is sound.

The labor market data is unambiguous: from 2020 to 2025, demand for 12 of the 16 coding tools we measured fell (Exhibit 1) and consolidated on Python. Python, previously mentioned in one in every five postings, grew to one in every three (from 22% to 33%). Go, TypeScript and Rust also gained. The increase in these three languages is almost certainly related to the larger trends showing that entry-level jobs are increasingly focused on production support and the agentic architecture layer (more on that in Sections 3 and 4).

Exhibit 1. The authoring stack narrowed onto Python — and thinned overall

Share of entry-level tech postings naming languages or coding technology, 2020 vs. 2025 national annual averages.

Language or markup	2020	2025	5-yr change, pp
RISING (4 of 16 shown)			
Python	22.2%	**33.3%**	▲ +11.0pp
Go	4.5%	**8.4%**	▲ +4.0pp
TypeScript	3.2%	**6.3%**	▲ +3.1pp
Rust	0.2%	**1.4%**	▲ +1.3pp
FALLING (12 of 16 shown)			
Java 8	1.1%	**0.3%**	▼ -0.7pp
C#	7.5%	**6.5%**	▼ -0.9pp
Scala	2.8%	**1.9%**	▼ -0.9pp
PL/SQL	2.1%	**1.1%**	▼ -1.0pp
PHP	3.3%	**1.7%**	▼ -1.5pp
Perl	3.3%	**1.2%**	▼ -2.0pp
Ruby	4.1%	**2.0%**	▼ -2.1pp
HTML5	5.7%	**2.7%**	▼ -3.1pp
CSS	12.7%	**9.0%**	▼ -3.7pp
jQuery	5.9%	**1.9%**	▼ -4.0pp
JavaScript	23.6%	**17.5%**	▼ -6.1pp
Java	27.7%	**20.9%**	▼ -6.8pp



Source: Burning Glass Institute analysis of job postings. Universe: 32 curated technology sub-occupations, entry-level, United States.

Beyond those likely links, the data do not establish why individual languages rose or fell. But the overall pattern shows language-specific requirements became less prominent in entry-level postings over the five-year period.

Despite the drop, not one practitioner we spoke with described computer science and software engineering fundamentals as less important than before. Some practitioners believe postings undercount fundamentals precisely because they have become an 'assumed' background: the thing nobody writes down because nobody would hire without it. The practitioner conversations emphasized that coding fluency remains critical, and "nobody is vibe-coding their way into a tech engineering job." Overall, practitioners told the following consistent story: You cannot direct an AI system, or judge what it hands back, without reading code, reasoning about data structures, and recognizing when an algorithm solves the problem inefficiently or at unnecessary cost.

Practitioners suggested that depth in one primary language—often Python, given it is the most common language in training data sets—may now matter more than prior exposure to many languages. In earlier years, employers may have used language requirements as a proxy for the time and uncertainty involved in bringing a developer up to speed in an unfamiliar technical environment. AI tools can reduce some of that friction by helping developers navigate unfamiliar syntax and documentation. According to practitioners, a shorter list of language requirements is therefore not evidence that coding matters less. Rather, it may indicate that the advantage conferred by prior familiarity with a long list of languages has become less decisive, while the fundamentals underneath remain essential. Those fundamentals are what a new hire draws on to check code as well as write it, and postings show employers asking for that check far more often.

> ***"Nobody is vibe-coding their way into a tech engineering job."***

2. Demand for Code Review Has Doubled, and Its Scope Is Expanding

Between 2020 and 2025, the share of entry-level postings naming code review doubled, from 7.8% to 15.6% (Exhibit 2). A doubling in five years is a strong demand signal: employers are asking far more often for new hires who can verify code, not only write it. The rise has continued. Code review gained 1.8 percentage points in 2025 (Exhibit 3) and another 4.2 points in the first half of 2026, reaching 18.5% of postings by June (Exhibit 4). Taken at face value, the rise in mentions of code review in job postings might suggest that employers want more of something they already wanted; but that doesn't tell the full story.

The practitioners we spoke to weighed in on this, noting that review of machine-generated code is a distinct skill from review of human-generated code, with additional and sometimes different failure modes. Code review of human-generated code traditionally checked for correctness, logic errors, missed edge cases, and whether the code is testable.

Practitioners described AI code review as requiring candidates to look not only for these more clear-cut and unambiguous issues, but also for outputs that are plausible but wrong; functionally correct but expensive to run in production; unnecessarily complex, insecure, or inconsistent with an existing codebase's conventions.

Several participants noted the importance of "knowing what a great solution looks like" before collaborating with AI to avoid the risk of accepting a mediocre solution. Moreover, several participants proposed clear, usable checklists for code review that included: checking for hallucinations, code performance latency, accuracy, consistency with internal coding criteria, and checking for unneeded complexity. One participant said, "the number one determining factor why people don't pass that [coding] interview is they don't critically assess the results of the AI."

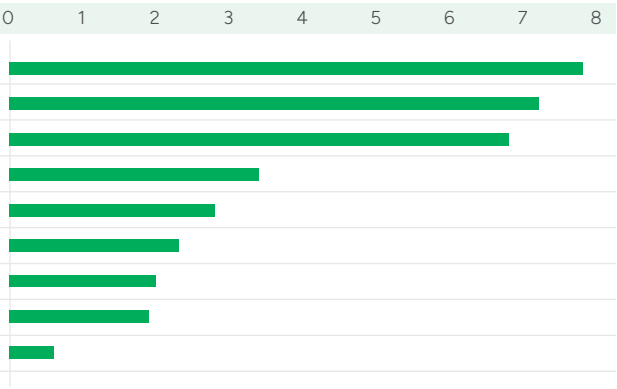
> *"The number one determining factor why people don't pass that [coding] interview is they don't critically assess the results of the AI."*

The posting trend alone cannot establish why demand for code review has increased. But practitioner accounts suggest that the skill's scope has expanded while its label has remained the same. Code review increasingly includes assessing the quality and appropriateness of AI output, not simply checking code written solely by another person. Exhibits 2 and 3 place code review alongside the operations skills taken up in Section 3, first across five years and then across the most recent year.

Exhibit 2. The code review and operations layer grew across the board, 2020–2025

Share of entry-level tech postings naming each skill in 2020 and 2025, and the change in percentage points. Each year's share is the mean of its twelve monthly shares.

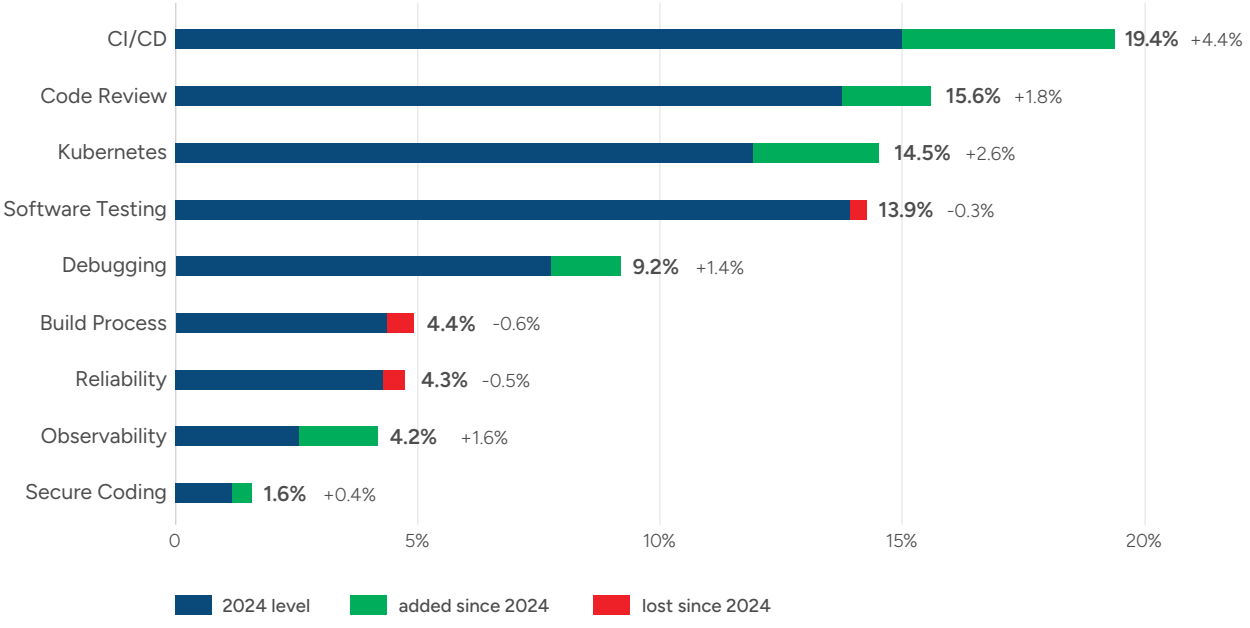
Code review and operations layer	2020	2025	5-yr change, pp
Code Review	7.8%	**15.6%**	▲ 7.8pp
CI/CD	12.2%	**19.4%**	▲ 7.2pp
Kubernetes	7.8%	**14.5%**	▲ 6.8pp
Observability	0.7%	**4.2%**	▲ 3.4pp
Reliability	1.5%	**4.3%**	▲ 2.8pp
Build Process	2.0%	**4.4%**	▲ 2.3pp
Software Testing	11.9%	**13.9%**	▲ 2.0pp
Debugging	7.3%	**9.2%**	▲ 1.9pp
Secure Coding	1.0%	**1.6%**	▲ 0.6pp



Source: Burning Glass Institute analysis of job postings. Universe: 32 curated technology sub-occupations, entry-level, United States.

Exhibit 3. The code review and operations layer in a single year, 2024-2025 - where the volume is

Share of entry-level tech postings naming each of the code review and operations skills tracked in Exhibit 2, 2024 and 2025. Each year's share is the mean of its twelve monthly shares. Red bars mark share lost since 2024.



Source: Burning Glass Institute analysis of job postings. Universe: 32 curated technology sub-occupations, entry-level, United States.

Several outside studies provide some backing for the idea that increased demand for code review skills is related to the complexity of reviewing AI-generated code. Stack Overflow's 2025 survey of more than 49,000 developers found trust in AI accuracy falling to 29% from 40% a year earlier. Two-thirds of respondents voiced frustration with "AI solutions that are almost right, but not quite."[2] Faros AI's telemetry across 22,000 developers found median review time up fivefold and a 31% increase in pull requests merged without review.[3] A peer-reviewed survey of 415 developers found that 84.3% of frequent generative-AI users said the tools did not reduce their code review time despite higher volume.[4]

These studies use different samples and methods, and they should not be read as establishing a single estimate of AI's effect on code quality or review time. Taken together, however, they reinforce the practitioner observation that higher volumes of AI output can increase the importance and complexity of review.

Veracode's work suggests this is not a transitional problem. Running the same battery of coding tasks against successive model generations, they find syntax correctness now above 95% and the security pass rate stuck near 55%, essentially flat year over year.[5] In other words, models got dramatically better at writing code that runs and no better at writing code that is safe, a finding that is consistent with practitioners' emphasis on review as a separate and increasingly consequential capability.

Where the practice has moved ahead of the labor market signal: Practitioners indicated that the ability to critically assess AI output is already appearing in some technical interviews. Several described a related precondition: candidates must know what a great solution looks like before collaborating with AI—otherwise, they risk accepting a merely adequate one without noticing. To direct an AI system, or judge what it hands back, candidates must be conversant in reading code, reasoning about data structures, and recognizing when an algorithm solves the problem at a high computational cost.

Candidates must know what a great solution looks like before collaborating with AI—otherwise, they risk accepting a merely adequate one without noticing.

3. Employers Are Asking For More of the Work that Follows Code Generation

Review is one part of the work that follows code generation. Employers are also increasingly naming skills associated with deploying, operating, and maintaining software after it ships. In 2025, CI/CD (continuous integration and continuous deployment), a method of building, testing, and releasing software quickly and safely using automation, appeared in 19.4% of entry-level postings. Kubernetes (an open-source tool that helps manage computer programs running inside containers) appeared in 14.5% (Exhibit 3). Cloud, containerization, reliability, and monitoring skills also rose over the period examined. Together, these trends suggest that production-oriented work is becoming more prominent in the expectations employers state for entry-level technical hires. This pattern is consistent with a broader shift described in practitioner conversations: as AI reduces the effort required to generate an initial technical output, greater attention may be moving toward the work required to integrate, deploy, monitor, troubleshoot, and maintain that output in production.

The language data also sheds light on this trend. Go and Rust, the languages used to build much of the production infrastructure and tooling that entry-level engineers are now expected to operate, both increased in share over the five-year period. Rust rose more than eightfold, though from a small base and appearing in only 1.4% of postings in 2025 (Exhibit 1). These changes may reflect, among other factors, demand for work associated with cloud infrastructure, systems programming, and production environments.

The rising weight of production work emerged consistently across practitioner conversations, spanning Big Tech and startups alike. Practitioners described DevOps (managing infrastructure at scale, diagnosing failures in production, responding to incidents) and system architecture as rising in value for entry-level work.

The most recent six months tell an interesting story. Exhibit 4 shows the skills whose share of entry-level postings changed most between January and June 2026. The gains came in the work that sits between producing code and running it. Code review rose 4.2 points to 18.5% of postings and software testing 4.9 points to 17.9%, the two largest gains among the skills shown. CI/CD gained 1.4 percentage points as well. Skills not shown in Exhibit 4 tell the same story. Those tied to connecting and releasing software rose: systems integration by 3.5 points, software integration by 2.9 and software deployment by 2.6.

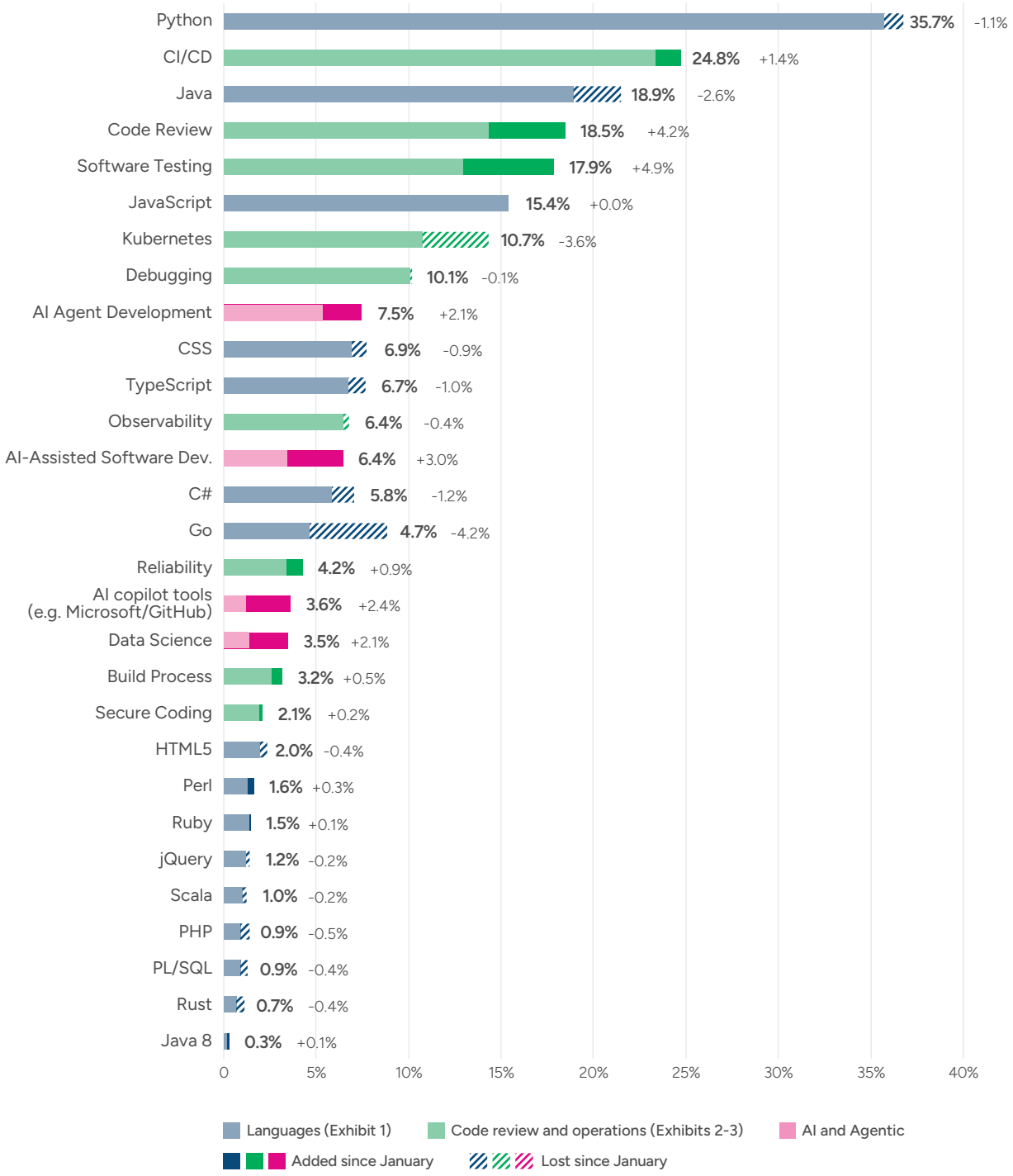
Data work rose as well. Data pipelines gained 3.3 points to 12.2% of postings, and ETL (extract, transform, load), data quality, data engineering, data science and data architecture each gained between 2.0 and 2.6 points. Named AI skills grew from lower levels: AI-assisted software development rose 3.0 points to 6.4%, AI agent development 2.1 points to 7.5%, and AI copilot tools such as Microsoft and GitHub Copilot 2.4 points to 3.6%.

The specific infrastructure tools at the center of the five-year story were among the largest declines: Go fell 4.2 percentage points to 4.7% of postings and Kubernetes 3.6 points to 10.7%; Linux and Docker (not shown) fell 3.2 and 2.8 points, to 5.6% and 9.5%. This would suggest that while infrastructure and production-related skills continued to rise in importance, the specific tools/technologies were less important than the domain expertise.

The recent data suggest a split within the work that follows code generation. Reviewing, integrating and deploying code, and preparing the data that systems run on kept strengthening, while demand for specific infrastructure tools paused.

Exhibit 4. The last six months: the skills in Exhibits 1–3, plus the AI skills now moving fastest, January–June 2026

Share of entry-level tech postings naming each skill, January 2026 vs June 2026. Each bar shows the January share; solid shows the share added by June and hatching the share lost. All 25 skills from Exhibits 1-3, plus four fast growing AI skills.



Source: Burning Glass Institute analysis of job postings. Universe: 32 curated technology sub-occupations, entry-level, United States.

4. AI-Specific Skills Are the Fastest-Growing, But Still Account for a Small Share of the Market

The named AI skills that rose in early 2026 extend a climb that began earlier. Technical skills associated with building, integrating, or operating AI and agentic systems are at once the fastest-growing part of the market and among the smallest as a share of postings. Postings naming AI agent development grew more than twelvefold between 2024 and 2025, from 0.14% to 1.78% of postings (Exhibit 5). LangChain, retrieval-augmented generation, and prompt engineering grew 13- to 18-fold by the first half of 2026 (Exhibit 5). The table separates skills that already had a meaningful base in 2024 (at least 100 postings) from rarer ones. Among the rarer ones, Google Gemini, Claude and Microsoft Copilot Studio were barely mentioned in 2024 but appeared in hundreds of postings in 2025, alongside newer engineering terms such as agentic frameworks. Together, the figures suggest employers are asking for both the general methods of building with AI and familiarity with particular tools. Even so, each of these skills still appeared in fewer than 2% of entry-level postings as recently as 2025.

Exhibit 5. AI skills mushroomed from 2024 to H1 2026

Share of entry-level tech postings naming each skill. The 2024 and 2025 shares are averages of monthly shares, and the H1 2026 share averages January–June. Growth is the H1 2026 share divided by the 2024 share. Skills shown are: the ten fastest-growing core-AI skills of 2024–2025 among skills with at least 100 postings in 2024; the same ranking among skills with more than 20 postings in both years; the AI and Agentic skills in Exhibit 4. Data Science appears because it is among the AI skills that moved most in Exhibit 4; its share was flat from 2024 to 2025.

AI skill	2024	2025	H1 2026	Growth, 2024 - H1 2026
MODERATE VOLUME (at least 100 postings in 2024)			*11 skills, ranked by growth within the group*	
AI-Assisted Software Dev.	0.10%	0.81%	**4.84%**	**49.9x**
AI Agent Development	0.14%	1.78%	**5.52%**	**38.6x**
AI copilot tools (e.g. Microsoft/GitHub)	0.10%	0.71%	**2.45%**	**24.5x**
LLM Application Development	0.10%	0.45%	**2.07%**	**21.5x**
Prompt Engineering	0.26%	1.31%	**4.63%**	**17.7x**
Intelligent Automation	0.09%	0.53%	**1.32%**	**15.0x**
Retrieval Augmented Gen.	0.14%	0.63%	**2.01%**	**14.7x**
Vector Database	0.17%	0.98%	**2.39%**	**14.2x**
LangChain	0.19%	1.12%	**2.44%**	**13.0x**
OpenAI API	0.18%	1.22%	**1.75%**	**9.6x**
Data Science	1.48%	1.54%	**2.97%**	**2.0x**
LOW VOLUME (more than 20 postings in 2024)			*6 skills, ranked by growth within the group*	
Agentic Frameworks	0.03%	0.48%	**2.45%**	**81.7x**
Claude AI	0.03%	0.55%	**2.17%**	**70.4x**
Google Gemini	0.03%	0.60%	**1.09%**	**42.4x**
GenAI Workflow Orchestration	0.04%	0.33%	**1.09%**	**28.8x**
Vector Embeddings	0.05%	0.32%	**1.11%**	**23.9x**
Microsoft Copilot Studio	0.03%	0.28%	**0.57%**	**18.7x**

Growth is computed on unrounded shares; dividing the rounded columns will not reproduce it exactly.

Source: Burning Glass Institute analysis of job postings. Universe: 32 curated technology sub-occupations, entry-level, United States.

The agentic layer kept growing into 2026. From January to June 2026, postings naming AI agent development rose from 5.3% to 7.5%, and AI-assisted software development nearly doubled, from 3.4% to 6.4% (Exhibit 4). Agentic frameworks and multi-agent systems each gained about one percentage point.

API skills are technically complementary to agent development: agents commonly use APIs to call external tools and functions. API demand is large and steady. About one in four entry-level postings named at least one API skill in both January and June 2026, spread across dozens of separately named skills, from RESTful APIs to API design, integration, and testing. What is changing is the kind of API work employers name. Established labels such as RESTful API and SOAP appear less often than they did in 2020, while API skills tied to AI systems are rising quickly: the share of postings naming at least one of them, such as the OpenAI API, the Model Context Protocol, or LLM function calling, grew from 3.6% to 5.2% between January and June 2026.[6] Practitioners describe the same pairing.

As with code review, APIs are not a new skill, but what the skill demands has changed. Our practitioner conversations offered the following perspective: Consider an API integration task in 2020 against the same task in 2025. In 2020, the entry-level engineer wrote the request, handled the response, and wired it into the application. By 2025, that engineer was more often specifying what the integration needs to do, letting an agent draft the call, and then reviewing the result closely enough to catch a malformed payload or an unintended side effect before it reaches a user. Practitioners associated some of the growth in AI-related API skills with their role in AI-powered product features.

Where the practice has moved ahead of the labor market signal: Interviewers are asking candidates to reason about orchestration frameworks, agent-to-agent protocols, retrieval versus fine-tuning tradeoffs, context and memory management across turns, and model selection under cost constraints. None of that content existed in entry-level interviews two years ago. Moreover, practitioners described new baseline expectations: entry-level talent is now expected to demonstrate strong reasoning for selecting appropriate models, managing context windows, handling memory across agent turns, and reasoning about token usage and costs.

5. AI Is Reshaping Work Across the Software Development Lifecycle

AI skills change what candidates are asked to know, and AI tools also change how the work itself is organized. The software development lifecycle itself has not changed: its core phases (i.e., requirements, design, implementation, testing, deployment, and operations) remain. What has changed is how AI can support work within each phase, and how practitioners describe the distribution of human effort across them.

Practitioners we spoke to identified a variety of emerging skills that can best be summarized as a redistribution of where human judgment sits across the development lifecycle. Several described the contours of software work as increasingly "hourglass-shaped." The traditional software development lifecycle was roughly linear: requirements → design → code → test → deploy. By contrast, in the emerging AI-integrated lifecycle, context engineering and constraint-setting expand at the beginning of production, followed by a more compressed phase of development, while testing and verification expand at the end of the sequence.

This is not a claim that the conventional lifecycle has been replaced. Rather, it is that AI tools can accelerate code generation and implementation in the middle of the process, while making it more important to establish context and constraints at the outset and to test, verify, and monitor outputs at the end. Amazon Web Services has similarly coined the term "AI-Driven Development Life Cycle" to describe a process in which AI is embedded within a disciplined sequence of human-governed development.[7]

According to practitioners, someone who can write code without being able to specify it precisely upfront, review it critically in the middle, or verify it operationally at the end, is missing the skill employers need.

Across these trends, a common pattern emerges. AI can compress parts of the implementation phase of software development, while increasing the importance of work before and after code generation: defining requirements and constraints, selecting and integrating tools, reviewing outputs, testing behavior, and monitoring systems in production. The findings do not imply that all entry-level workers now own the full software-development lifecycle. What they do suggest is that employers and practitioners are placing greater value on a broader range of lifecycle capabilities earlier in a technical career.

6. Employers Are Seeking Business, Product and Engineering 'Judgment' but Do Not Yet Share a Common Vocabulary for It

Generative AI can enable less experienced workers to produce prototypes, systems and products more quickly. But faster generation does not determine whether a proposed solution addresses the right problem, reflects the relevant constraints, or is worth the cost of building and operating it. Tokens, infrastructure, maintenance, and reliability all carry costs. As one practitioner put it, "You can prompt an AI system all day and it will build the wrong thing" and often inefficiently, at that.

Practitioners consistently named this as an emerging expectation for entry-level technical talent. They used overlapping terms, including business judgment, engineering judgment, product judgment, and "shot selection," but generally pointed to the same set of capabilities: understanding the business and user context; turning ambiguous asks into clear technical requirements; identifying which problems and requirements matter most; and weighing tradeoffs among cost, reliability, security, speed, and functionality.

The increased visibility of these capabilities does not mean that technical production matters less. Rather, practitioners suggested that AI makes judgment more consequential: candidates must be able to explain what should be built, why it should be built, and how technical choices will affect users, operations, and business outcomes.

Employers have not yet translated those expectations into a common vocabulary that can be consistently specified in postings or assessed in hiring. In practitioner convenings, for example, one group described product judgment as "shot selection" and understanding what a user wants. Another identified three possible dimensions: intent (does the output do what the user needs), scoping (is the request appropriately broad or specific), and consistency (will the same prompt reliably produce an acceptable result).

This lack of a shared framework creates a practical challenge. Candidates may have limited guidance about what to learn and demonstrate, while employers may struggle to assess these capabilities consistently. Defining the component skills more clearly could help both groups distinguish strong technical judgment from strong technical output alone.

> *"Many can build something that works, but only a few people can explain why it works, why they chose one approach over another, and what they'd need to change if requirements changed. That gap is something we need to notice in students."*
> ***—Engineering coach / technical lead practitioner***

> *"The key thing here is to understand the business model, understand [what the product or service is that we're selling to the customer]. What is the value proposition? What differentiates us? Then you try to reason out how my role fits into that organization … how am I helping finance achieve its goals? … Then maybe you're looking at building a contract review agent, or a billing agent, using AI."*

7. Technical Assessment Is in Transition

The missing vocabulary also complicates hiring, because technical assessment is changing at the same time. Five years ago, there was broad convergence on how to evaluate technical talent. Coding screens, whiteboard exercises, and platforms such as LeetCode were commonly used to assess both problem-solving ability and a candidate's capacity to write correct code independently.[8] Technical interviews have always varied by employer and role. Still, these formats reflected a widely shared view of what an entry-level assessment should measure: whether a candidate could solve a technical problem and produce working code without assistance.

But generative AI has changed the work that technical assessments are meant to predict. Conventional assessments were designed to measure a candidate's ability to solve problems and write code independently. As AI tools become part of routine technical workflows, employers must now decide whether and how to assess a candidate's ability to use, evaluate, manage and challenge AI output.

Practitioner conversations suggest wide-ranging experimentation with new evaluation methods, including:

- A 24-hour open-book system-design project, defended before a live panel.
- An unfamiliar-domain case study, reasoned through in real time with no preparation.
- A fix-in-place exercise on a legacy codebase.
- A take-home assignment whose front-end component is intended to surface product judgment rather than coding ability alone.
- A two-hour assessment involving a messy dataset and a retrieval-augmented generation pipeline that produces hallucinations at a high rate.
- An internal fire-drill for early-career analysts, in which financial models must be updated by 7:15 a.m. the following morning.

These examples do not constitute a validated typology, nor should they be read as representative of the full market. Their significance is narrower: despite their differences, they point toward a common set of design features. Candidates are asked to work through ambiguity, use tools available to them (including, in some cases, AI), explain their reasoning, and respond to conditions that more closely resemble real technical work than a conventional algorithmic exercise.

When the object of assessment changes, assessment practices must adapt to remain effective. The testing alternatives to conventional coding screens mentioned by practitioners show that process underway.[9] At the same time, though, a January 2026 survey of 400 engineering leaders found 62% still prohibiting AI use in interviews and 71% saying AI has made technical skills harder to assess.[10]

Taken together, the practitioner examples and survey findings point to a field deep in transition. While employers are experimenting with new assessment approaches, the evidence is directional rather than definitive: it does not establish how widespread these emerging practices are or whether any particular approach predicts job performance better than existing methods.

Employers, educators, and workforce organizations therefore have an opportunity to develop more transparent assessment approaches that map more directly to the work of entry-level technical roles. Such approaches should evaluate capabilities that extend beyond producing technical output: evaluating and improving it, and making sound decisions under conditions of ambiguity, production complexity, constraint, and AI assistance.

> *"AI did not break technical assessment. It raised the cost of screening and prepping for a job that no longer exists."*

Implications and Recommendations

The findings in this paper present both challenges and opportunities for all the players in the workforce readiness ecosystem.

First, undergraduate curricula and credentialing pathways should not treat AI fluency as a replacement for fundamental CS and software engineering skills. While employers are indeed naming fewer languages as stated requirements in job postings, no one should misread this as a sign that code authoring is retiring as a skill. Rather, fluency in one primary programming language and strong mastery of data structures and algorithms remain essential to reading code, debugging failures, testing systems, reasoning about data, and judging AI-generated output.

What appears to be changing is the composition of entry-level technical work. As AI tools make it faster to generate output, entry-level talent is now expected to evaluate whether that output is correct, secure, maintainable, useful, and appropriate for the problem at hand.

This shift places greater emphasis on the ability to use technical knowledge in contexts shaped by real constraints, competing priorities, and imperfect information, which challenges traditional approaches to technical curricular development. Applied learning approaches like capstone projects and hackathons that feature greenfield software development are important but are structurally insufficient to teach production ownership in real-world environments with real stakes attached. A 'working application' is no longer strong evidence of skills that are increasingly in demand.

The capabilities learners need to practice

Second, beyond those fundamentals, the evidence in this report points to six connected capabilities, drawn from Sections 2 through 6, that should be incorporated more deliberately into technical preparation, applied learning, and hiring assessment:

1. **Code review and critical evaluation of AI outputs.** Reviewing AI-generated code is not the same skill as reviewing human-written code. Where human code review looks for logic errors, missed edge cases, testability, security, and consistency with the surrounding codebase, review of AI-generated code requires screening for different failure modes: plausible-but-wrong output, unsupported assumptions, missing context, unnecessary complexity, hallucinated dependencies, security issues, or code that functions locally but is poorly suited to the architecture or conditions in which it will operate.

Reviewing code generated by AI requires a broader form of verification. Reviewers must examine not only whether the code is correct, but whether the problem was framed correctly; whether the tool was given adequate context; what evidence supports the output; what tests, specifications, or dependencies it relies on; and whether the proposed solution should be accepted, revised, constrained, or rejected. This competency has a precondition: candidates must know what a strong solution looks like before they can meaningfully evaluate an AI-generated one. Practitioners repeatedly identified critical evaluation of AI output as an important differentiator in technical interviews and entry-level work.

2. **Production support, integration and operational awareness.** A feature that works in isolation may fail when introduced into an existing system. Learners need greater exposure to the work surrounding code generation: integration, testing, deployment, documentation, maintenance, observability, and the reliability, security, cost, and usability implications of technical decisions. As code generation compresses, the work after code ships (e.g., deployment, monitoring, incident response, keeping a system running well under real load/traffic) has become an expectation for entry-level hires. While they may not own production systems, familiarity and experience with how software behaves after release is increasingly valuable. Learners should understand that shipping code is not the end of the engineering task: software must be integrated, observed, maintained, and improved over time.
3. **Applied AI integration.** Candidates need hands-on experience applying existing models, APIs, and frameworks to solve a defined user, organizational, or business problem (e.g., building retrieval-augmented generation systems, integrating LLM APIs into a product). The relevant capability is determining whether an AI component is appropriate for the problem; connecting it to relevant data, interfaces, and existing systems; and evaluating its performance, limitations, cost, security, and failure modes in use. For candidates working directly on AI-enabled products, particularly in larger organizations, this may also include exposure to the practices involved in operationalizing such systems: deployment, monitoring, versioning, access controls, human-review processes, and governance.
4. **AI system development lifecycle management.** As agentic tools redistribute work across the development lifecycle, candidates must be able to use AI systems across requirements, design, implementation, testing, deployment, and operations. It also includes knowing where a human checkpoint belongs in an agent-driven workflow, including when a person should approve, revise, test, or halt AI-generated requirements, architecture, code, analyses, and tests before they move further into a workflow.
5. **Engineering judgment.** Engineering judgment is the ability to select and justify an approach when multiple technically valid options involve different tradeoffs among performance, reliability, security, maintainability, cost, and delivery speed. It applies to AI-generated proposals, existing systems, and work produced without AI assistance. A candidate may be able to write code that produces the desired output while still making poor decisions about performance, complexity, security, testing, reliability, or maintainability. Learners should have repeated opportunities to compare alternatives, justify their choices, and revise their approach in response to new evidence or constraints.

6. **Business and product judgment.** Technical work also requires understanding what problem is worth solving and what constraints shape a viable solution. Practitioners described a growing need for candidates who can clarify ambiguous requests, ask productive questions, identify who is affected by a decision, and distinguish between the technically appealing solution and the resource-efficient one. This does not mean that entry-level hires must arrive as product managers. It means they should be able to connect technical choices to user needs, organizational goals, and practical limits.

These competencies share a common thread: they are defined by the consequences of technical work. Reviewing AI-generated code matters because a plausible-but-wrong answer can reach a user or introduce risk into a system. Integration and operational awareness matter because software behaves differently once it encounters real dependencies, usage patterns, and constraints. Business and engineering judgment matter because the fastest path to a working prototype is not always the most secure, maintainable, cost-effective, or useful path.

The point is not that technical work has only recently acquired consequences. Rather, as AI makes plausible technical output easier to produce, the ability to recognize, anticipate, and manage the consequences of that output becomes a more visible source of value. The distance between producing that output and answering for its consequences is the gap this paper's title names.

Rethinking the bridge between education and employment

The six capabilities described above, and the fundamentals beneath them, are not all learned in the same way. Foundational technical fluency can be developed through structured coursework, practice, and guided feedback. The six capabilities, by contrast, tend to develop through a combination of project work, collaboration, self-directed learning, and workplace experience.

This finding matters because many traditional technical learning experiences are organized around bounded, greenfield tasks that are drifting farther from the job. Learners receive a defined problem, build a solution from scratch, and are assessed primarily on whether the resulting artifact works. These experiences can build important technical foundations, including programming fluency, problem decomposition, and persistence. But on their own, they may provide limited practice with incomplete requirements, legacy code, conflicting stakeholder needs, imperfect documentation, changing constraints, or the need to justify a decision among several technically viable options.

AI makes this gap more visible. When a tool can quickly generate plausible code or a functioning prototype, the educational task cannot end with determining whether a learner can produce an artifact. Learners also need practice judging whether that output is trustworthy and whether it fits the system and the problem it is meant to serve. These capabilities are unlikely to develop through standalone "soft skills" modules or through unstructured access to AI tools. They require repeated opportunities for critique, comparison, explanation, feedback, and revision.

Programs should therefore supplement—not abandon—foundational coursework, capstones, hackathons, and greenfield projects with experiences that recreate the conditions in which those capabilities are used on the job. Where the previous section describes what learners need to practice, the experiences below describe where that practice can happen:

- maintaining, debugging, or extending an existing codebase;
- working from incomplete, changing, or conflicting requirements and identifying the questions that must be resolved;
- working with realistic constraints, such as performance limits, accessibility requirements, security vulnerabilities, cloud-computing budgets, reliability targets, or maintainability expectations;
- participating in code-review, design-review, deployment, monitoring, or incident-style exercises; and
- receiving practitioner or stakeholder feedback on the reasoning behind a decision, not only on whether the final product works.

These experiences can take many forms: course-based simulations, practitioner-reviewed projects, client or community technology partnerships, open-source contributions, employer-sponsored challenges, micro-internships, and structured work-based learning. The essential feature is that learners encounter the ambiguity, constraints, feedback, and revision cycles that characterize contemporary technical work.

Traditional internships, apprenticeships, and cooperative education remain valuable[11] because they can expose learners to established systems and enterprise architectures, cross-functional teams, and the ongoing maintenance of software. But access to applied learning experiences is uneven, and the learning opportunities they provide vary widely. Only 44% of undergraduates who want a paid internship obtain one, according to the Business-Higher Education Forum; students of color, first-generation students, community-college students, and financially vulnerable students face some of the greatest barriers to access.[12] Closing this applied learning gap will therefore require both improving conventional placements and expanding alternative contexts for experiential learning: simulation environments, micro-internships, service-learning placements, as well as additional new, innovative approaches that are designed to deliver the applied skills described in this paper at a scale traditional entry-level placement models were never built to reach.

What this asks of employers and funders

Employers have a central role in making these expectations more explicit. Job postings often provide incomplete signals about how entry-level work is changing, while practitioner interviews suggest that employers themselves do not always share a stable vocabulary for the capabilities they seek. Terms such as "engineering judgment," "product judgment," and "AI fluency" can describe overlapping expectations without specifying how candidates should demonstrate them. Many of the learning experiences described above depend on inputs that most education providers and workforce development intermediaries cannot easily create on their own: realistic codebases, current technical environments, meaningful constraints, practitioner feedback, and a clearer view of how employers assess applied judgment. Employers and funders can help catalyze the necessary collaboration and resource sharing. A starter set of ideas might include:

- Providing sanitized, runnable codebases with commit history, test suites, CI configurations, and open defect backlogs. Existing systems give learners opportunities to practice maintenance, code review, debugging, integration, and technical decision-making that are difficult to recreate through greenfield projects alone.
- Developing and sharing a curated bank of technically grounded, judgment-under-ambiguity tasks. These might ask learners to evaluate competing solutions, clarify incomplete requirements, diagnose a failure, review AI-generated code, or explain the tradeoffs involved in a technical decision.
- Contributing engineering subject matter expertise to help learners compare strong and weak solutions to the same problem, understand why a plausible output may be insufficient, and receive feedback on the reasoning behind their choices.
- Supporting a shared framework for business, product, and engineering judgment. Funders and industry groups can bring employers and educators together to agree on the component skills behind terms that practitioners currently use interchangeably, so that these capabilities can be consistently specified in postings, taught in programs, and assessed in hiring.
- Working toward clearer, more transparent models for AI-permitted entry-level technical assessment. Employers who have already begun adapting their processes can help the field by publishing the kinds of work they assess, the role of AI in assessment, and the scoring dimensions they use.
- Participating in, and helping to evaluate, pilots of new work-based learning models that can expand access beyond the scale of conventional internships and apprenticeships. The goal should be not only more experiences, but experiences intentionally designed to develop the six capabilities this paper identifies.

A more transparent partnership among employers, educators, workforce intermediaries, and funders can make the new expectations more teachable, more assessable, and more equitably accessible to entry-level talent.

Appendix: Methodology

This paper focuses on entry-level technology roles for college graduates. The universe is composed of 32 technology roles selected with Break Through Tech. Our quantitative insights come from job postings data, and qualitative insights from conversations with practitioners. We describe both data sources below.

These 32 roles are:

Agile and Scrum Managers	Application Security Engineers	Back-End Developers	Build and Release Managers
Business Intelligence Analysts (General)	Business Intelligence and Data Warehouse Developers	Cloud Data Engineers	Cloud Security Engineers
Computer Systems Analysts (General)	Data Engineers	Data Scientists (General)	DevOps and Release Engineers
DevOps and Reliability Systems Engineers	Digital Product Designers	Enterprise Platform Architects	Front-End Web Developers
Full-Stack Developers	Full-Stack Web Developers	Information Security Engineers (General)	Integration and Middleware Developers
Machine Learning and AI Developers	Machine Learning Engineers and Scientists	Mobile Developers	Product Analytics Scientists
Security Operations and Incident Response Engineers	Software Developers (General)	Software Project Managers	Software Quality Assurance Analysts and Testers (General)
Test Automation Engineers	User Experience Designers	User Interface Designers	Web Developers (General)

Quantitative data

We use job postings data to help us identify how the skills needed in entry-level tech roles are changing. Job postings come from the Burning Glass Institute's September 2026 postings release. We use postings from January 2020 through June 2026, covering United States entry-level postings across those 32 sub-occupations. We define entry-level postings as those for full-time roles which require a bachelor's degree and no more than two years of experience. Once we collect these postings, we remove listings from aggregators, staffing firms, AI-labeling firms, and listings with no employer name specified. We are left with an analysis sample of 1,028,622 entry-level tech postings that name at least one skill, from January 2020 to June 2026.

The skills associated with each role come from the postings themselves. Job postings include the skills that employers ask for in the text of the posting. We extract the text that contains those raw skills, parse, and standardize them to the Burning Glass Institute skills taxonomy. We are then able to observe how the pool of skills across all 32 entry-level tech roles is changing over time. All figures reported as "share of postings" are the proportion of postings in the relevant universe and window that name the skill at least once. Annual shares are the mean of the twelve monthly shares, and H1 2026 shares are the mean of January through June, so each month counts equally. The January–June 2026 comparison uses single-month shares.

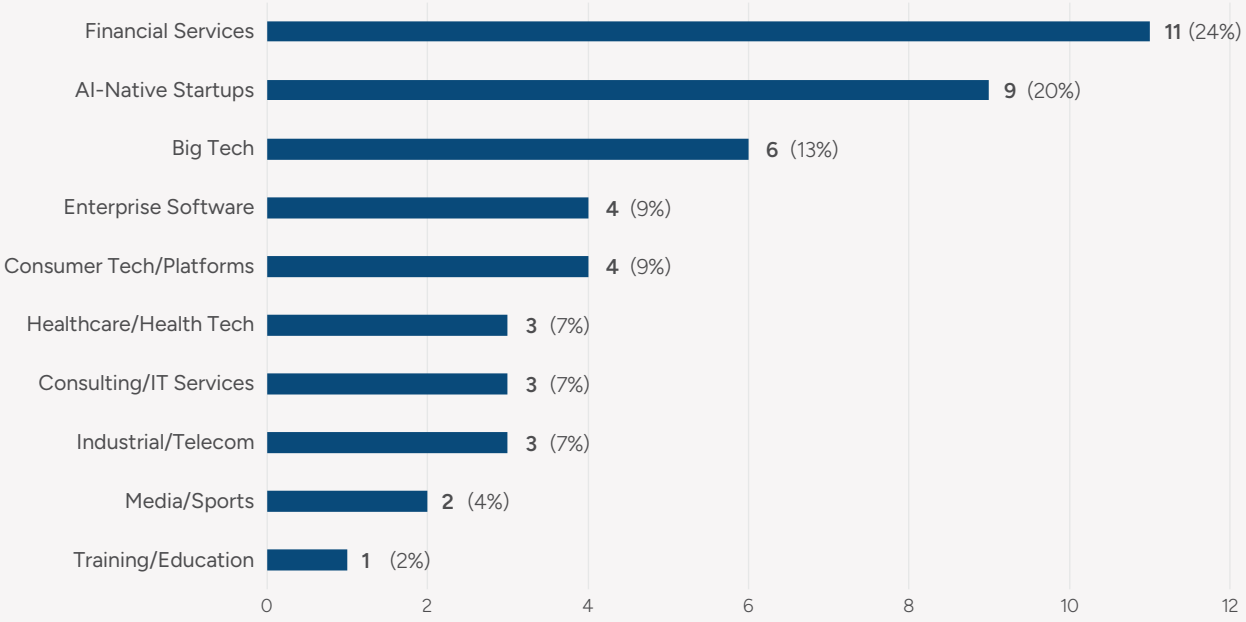
There are three time windows in this paper, and they answer different questions. The 2020-versus-2025 comparison is the workhorse. Skill growth in this window is the difference between the average monthly share of postings naming a skill in 2025 and the same average in 2020, and a skill must appear in more than 20 postings in both years to be ranked. The 2024-versus-2025 comparison carries the one-year claims, including the growth of AI skill vocabulary; it compares each skill's average monthly share in 2024 with its average monthly share in 2025. Just as in the 2020-2025 comparison, each skill must appear in more than 20 postings in both 2024 and 2025 to be ranked. January-versus-June 2026 carries the current-momentum claims. Skill growth in this window is the difference between the shares in those two single months, and a skill must appear in more than 3 postings in each month to be ranked. Exhibit 5 reports growth as a skill's H1 2026 share divided by its 2024 share, and groups skills by 2024 volume: high volume (at least 100 postings in 2024) and low volume (more than 20 postings in both 2024 and 2025). Where a skill moves one direction in the short window and the other direction across five years, we report both.

Qualitative data

Break Through Tech conducted and coded structured conversations with 54 individuals across 46 organizations between May and July 2026, along with two group convenings held in the same timeframe. Fifteen participants held individual-contributor technical roles (e.g., software engineer, ML engineer); 39 held managerial or executive roles, ranging from engineering managers to C-suite and EVP-level leaders. Participants were selected for their oversight of, or work within, technology functions, regardless of whether their employer's core business is technology — organizations ranged from Big Tech platforms and AI-native startups to financial-services firms, healthcare systems, and industrial and consumer companies with substantial internal technology organizations (Exhibit 6).

Exhibit 6. Financial services and AI-native startups account for nearly half of the organizations in our sample.

Number of employers by industry sector (n = 46 employers)



Source: Break Through Tech interview records.

Works Cited

Outside sources referenced in the text, in order of first appearance. Items marked † are vendor or practitioner research rather than peer-reviewed work, and we label them as such wherever they carry weight in the argument.

Break Through Tech (2026, August). *Analysis of Convenings*. Practitioner conversations conducted May–July 2026 and coded by Break Through Tech.

Business-Higher Education Forum. *Expanding Internships: Meeting Employer Demand for Work-Based Learning*. † https://www.bhef.com/sites/default/files/BHEF_Expanding_Internships.pdf

Stack Overflow (2025). *2025 Developer Survey*. † https://survey.stackoverflow.co/2025

Faros AI (2026). *The AI Engineering Report 2026: The Acceleration Whiplash*. † https://www.faros.ai/blog/ai-acceleration-whiplash-takeaways

Afroz, S., Feng, Z., Menezes, T., Kimura, K., Trinkenreich, B., Steinmacher, I., & Sarma, A. (2026). The Fast and Spurious: Developer Productivity with GenAI. *Companion Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering*. https://arxiv.org/abs/2510.24265

Veracode (2025, July; updated. 2026, March 24). *GenAI Code Security Report and Spring 2026 GenAI Code Security Update*. † https://www.veracode.com/wp-content/uploads/2025_GenAI_Code_Security_Report_Final.pdf

Raja SP et al. (2025, July 31). *AI-Driven Development Life Cycle: Reimagining Software Engineering*. AWS DevOps Blog; and AWS (2025, November), Open-Sourcing Adaptive Workflows for AI-DLC. †

Behroozi, M., Shirolkar, S., Barik, T., & Parnin, C. (2020). Does Stress Impact Technical Interview Performance? *Proceedings of ESEC/FSE 2020*, 481–492. https://dl.acm.org/doi/10.1145/3368089.3409712

Dungan, R. (2025, August 1). Meta Trialling AI-Assisted Interview Process for New Coding Hires. *HR Grapevine USA*.

Shibu, S. (2026, May 8). Google Is Testing a New Rule That Could Transform Job Interviews. *Entrepreneur*.

Karat (2026, January). *AI Workforce Transformation Report*. Survey of 400 engineering leaders. † https://karat.com/resource/ai-workforce-transformation-report/

Craig, R. (2023). *Apprentice Nation: How the "Earn and Learn" Alternative to Higher Education Will Create a Stronger and Fairer America*. BenBella Books.

Endnotes

[1] Business-Higher Education Forum. Expanding Internships: Meeting Employer Demand for Work-Based Learning. Available at: https://www.bhef.com/sites/default/files/BHEF_Expanding_Internships.pdf

[2] Stack Overflow (2025). 2025 Developer Survey. Available at: https://survey.stackoverflow.co/2025

[3] Faros AI (2026). The AI Engineering Report 2026: The Acceleration Whiplash. Available at: https://www.faros.ai/blog/ai-acceleration-whiplash-takeaways

[4] Afroz, S., Feng, Z., Menezes, T., Kimura, K., Trinkenreich, B., Steinmacher, I., & Sarma, A. (2026). The Fast and Spurious: Developer Productivity with GenAI. Companion Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering. Available at: https://arxiv.org/abs/2510.24265

[5] Veracode (2025, July; updated 2026, March 24). GenAI Code Security Report and Spring 2026 GenAI Code Security Update. https://www.veracode.com/wp-content/uploads/2025_GenAI_Code_Security_Report_Final.pdf

[6] API skills are the roughly 80 skills in the Burning Glass Institute skills taxonomy whose names refer to application programming interfaces or to the protocols, formats, and tools used to build and test them, for example RESTful API, SOAP, GraphQL, JSON, API design, API testing, and Postman. They include the API skills tied to AI systems: the OpenAI API, AI integration, the LLM Model Context Protocol (MCP), LLM function calling, AI/LLM API gateways, and agent memory and tool use. Broader architecture skills such as microservices and web services are not counted. Unlike the single-skill figures elsewhere in this paper, these shares count postings that name at least one skill in the group, so a posting that names several API skills counts once. Each share is the number of such postings in the month divided by all entry-level postings in that month that name at least one skill.

[7] Raja SP (2025). AI-Driven Development Life Cycle: Reimagining Software Engineering. (AWS DevOps & Developer Productivity Blog). Available at: https://aws.amazon.com/blogs/devops/ai-driven-development-life-cycle/

[8] On the validity of that format, see Behroozi, M., Shirolkar, S., Barik, T., & Parnin, C. (2020). Does Stress Impact Technical Interview Performance? Proceedings of ESEC/FSE 2020, 481-492. https://dl.acm.org/doi/10.1145/3368089.3409712

[9] Publicly reported examples include Dungan, R. (2025, August 1). Meta Trialling AI-Assisted Interview Process for New Coding Hires. HR Grapevine USA; and Shibu, S. (2026, May 8). Google Is Testing a New Rule That Could Transform Job Interviews. Entrepreneur.

[10] Karat (2026, January). AI Workforce Transformation Report. Survey of 400 engineering leaders. Available at: https://karat.com/resource/ai-workforce-transformation-report/

[11] On apprenticeship as an alternative pathway, see Craig, R. (2023). Apprentice Nation: How the "Earn and Learn" Alternative to Higher Education Will Create a Stronger and Fairer America. BenBella Books.

[12] See note 1.