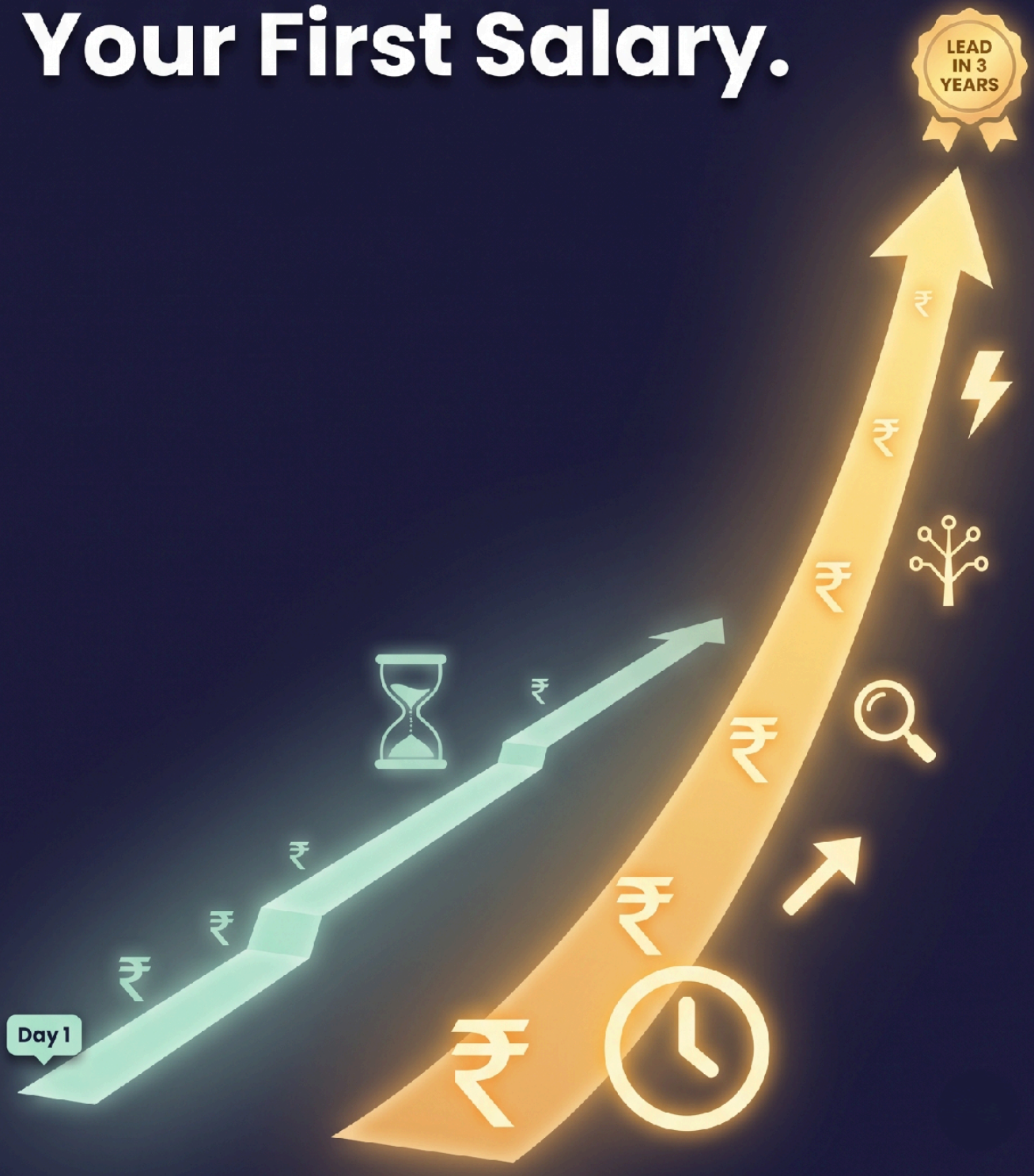
Your First 2 Years in Tech – Why Velocity Matters More Than Your First Salary.



The Two Freshers

Same company. Same joining date. Same starting salary.

Two years later, one is a team lead, trusted with critical projects. The other is still doing the same tasks, wondering why promotions pass them by.

The difference? It wasn't their college. It wasn't their CGPA. It was **Velocity**—the ability to learn, adapt, and grow faster than the environment around them.

Your first salary is temporary. Your velocity is permanent.

What Exactly Is Velocity?

In the SCAV framework, Velocity measures your **Skill and Capability Growth Potential over time**. It's not about what you know today. It's about how quickly you can learn what you'll need tomorrow.

Velocity breaks down into three critical dimensions:

1. Learning Ability (Fluid + Crystallised Intellect)

Can you grasp new concepts quickly? Can you retain and apply what you've learned? A study on code comprehension found that **three of four intelligence factors—fluid intelligence, visual perception, and cognitive speed—have significant positive effects on code comprehension performance** [1][2]. In simple terms: your ability to understand code isn't fixed. It's trainable. And it directly impacts your performance.

2. Reasoning Strength

Can you work through complex problems logically? Can you draw sound conclusions from incomplete information? The same study found that **conscientiousness—the trait of being thorough and organized—explains some of the variance in code comprehension performance** [3]. This isn't about being "smart." It's about being deliberate.

3. Critical Thinking Driven Decision Making

Can you evaluate options, weigh trade-offs, and make good decisions under uncertainty? This is what separates engineers who lead from engineers who follow.

The Learning Velocity Crisis

Here's the uncomfortable truth: **AI is accelerating the pace of change faster than most engineers can keep up**.

According to the Stanford HAI 2026 AI Index, **organizational AI adoption has reached 88%** [4]. Nearly nine out of ten engineering workplaces are already integrating AI into their operations. The half-life of tooling and workflow knowledge is shrinking rapidly—the CI/CD pipeline you mastered last year may be outpaced by an AI-native alternative this year.

The Stanford HAI 2026 AI Index also found that on SWE-bench Verified—a benchmark testing AI on real-world software engineering tasks—**performance jumped from 60% to near 100% in one year** [4]. One year. The tool that was "interesting but unreliable" last quarter is now solving real problems at scale.

Your ability to learn and reinvent faster than that improvement curve isn't a career advantage—it's a survival skill. Standing still doesn't mean falling behind gradually. It means becoming obsolete in a release cycle.

The Wix CEO Avishai Abrahami, who scaled the platform to over 270 million users, emphasizes that **the first big challenge for any company is velocity—the ability to iterate quickly and release new features** [5]. And that same principle applies to individual engineers: "If you can nail this, your company is on the right track because you can test and improve faster, which is crucial for success."

The AI Trap: Why Speed Can Kill Your Velocity

Here's where it gets dangerous. AI makes you faster. But speed without understanding is a trap.

A recent controlled experiment by Anthropic compared developers who used AI assistants against those who didn't, testing identical coding tasks. The results were striking [6]:

- **Developers who relied most heavily on AI scored lower across the board**, with the biggest drop in debugging
- The group without AI **made more mistakes but understood more**—errors forced them to actually learn
- AI made tasks feel easier but did not produce a meaningful speed advantage overall
- The highest performers used a pattern called **"generation then comprehension"** —letting AI produce something, then interrogating it, questioning it, and working to understand why it functioned. They scored in the **eighties**, while pure delegation scored in the **thirties**

The difference had nothing to do with technology. It had everything to do with whether the human stayed in the driver's seat.

As LeanIX Engineering puts it: *"Over-reliance on AI leads to shallow reasoning behind implementation choices, a drop in technical ownership, and reduced learning-by-doing… If we're not careful, it may lead to the opposite of productivity: a decline in deep engineering thinking, quality, and long-term team capability"* [7].

Critical thinking isn't optional anymore. It's your differentiator.

The 3-Year Curve: What Great Developers Actually Learn

Avishai Abrahami, who has hired and mentored hundreds of engineers, breaks the developer journey into clear phases [5]:

Phase 1 (First 4 years):

You master syntax, loops, and problem-solving. You think you know how to code. But you actually only know how to solve **simple problems**. At this stage, you need consistency, attention to detail, and the ability to try things repeatedly.

Phase 2 (Next 6 years):

You learn design patterns. You think you know it all. The best attribute at this stage is **understanding that you still have a lot to learn**. If you don't know what that means, you're probably in this phase.

Phase 3:

You start thinking about systems—how everything connects, works, and grows. You're finally beginning to be qualified to build technology.

Phase 4 (Great Developer):

You can think about the **minimum level of design for the maximum impact**. You understand the big picture but can zoom into the smallest details. At this stage, **courage is the attribute most needed**—the ability to make brave, wise decisions based on experience.

The key insight? **You don't reach Phase 4 by just showing up.** You reach it by deliberately building your Velocity—learning faster, reasoning better, and thinking more critically with every passing year.

How to Build Velocity in Your First 2 Years

1. Learn Deliberately, Not Passively

Don't just use AI to generate code and move on. Follow the "generation then comprehension" pattern. Let AI produce something, then question it. Understand *why* it works. This is how skill formation accelerates [6].

2. Embrace Mistakes as Learning Opportunities

The developers without AI in the Anthropic study made more mistakes but understood more. **Struggling builds skill formation.** The debugging gap between AI users and non-users was the largest—because debugging is structured reasoning under uncertainty, and that discipline is built through repetition.

3. Build a Repeatable Learning Process

Engineers who stay relevant treat learning as an operational discipline, not a personality trait [4]. This means:

- **Small, fast learning cycles:** Spend 30 minutes with a new tool against your actual codebase, not a toy project
- **Stay connected to upstream context:** Engineers who understand the product and business model learn faster because they evaluate tools against real constraints
- **Reflect before grinding:** When stuck, ask whether you're approaching the problem with an outdated mental model

4. Practice Critical Thinking Daily

The highest-performing participants in the Anthropic study didn't avoid AI—they used it as a thinking partner, generating output and then examining it, questioning it, testing assumptions. They stayed mentally engaged and kept doing the thinking [6]. That's what builds lasting capability.

Your First Salary Is Temporary. Your Velocity Is Permanent.

The gap between the engineer who stays stuck and the one who accelerates isn't luck. It's Velocity.

Your first salary is a number. Your learning ability, reasoning strength, and critical thinking are the engine that will drive every salary you'll ever earn. Invest in the engine.

Ready to Discover Your Velocity?

Take LNBE's SCAV assessment to measure your learning ability, reasoning strength, and critical thinking. Get a personalized roadmap to build the Velocity that will define your career.

References

1. https://arxiv.org/pdf/2109.13546#4#1
2. https://ui.adsabs.harvard.edu/abs/2021arXiv210913546W/abstract
3. https://portal.fis.tum.de/en/publications/code-comprehension-confounders-a-study-of-intelligence-and-person/
4. https://utterskills.com/blog/skills-beyond-code-in-the-era-of-ai/the-adaptive-engineer-adaptability-and-learning
5. https://www.wixjapan.com/post/exclusive-interview-with-avishai-abrahami
6. https://kepner-tregoe.com/blogs/ai-is-powerful-discipline-is-essential/
7. https://engineering.leanix.net/blog/ai-won%E2%80%99t-grow-great-engineers/#main