

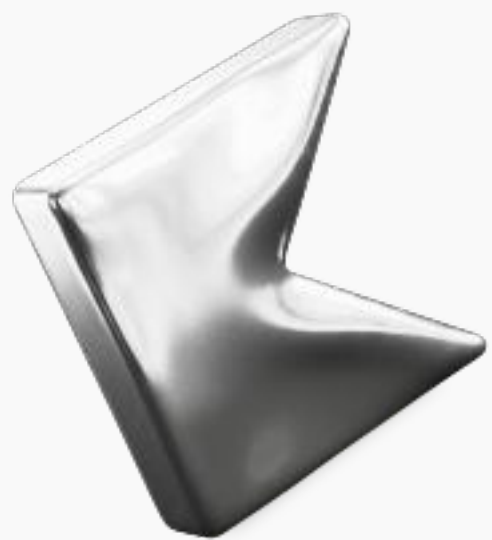
JetBrains Report

The State of Kotlin in 2026

15 YEARS OF GROWTH, TRUST, AND ECOSYSTEM MATURITY

Fifteen years after its first release, Kotlin has grown from a JVM option into a mature ecosystem spanning 8.1 million developers worldwide.

This report explains the shape of that growth: Who is adopting Kotlin, what they do with it, how they fold it into existing systems, and their plans for the future. Read on for more.



JETBRAINS



Kotlin

Table of Contents

Exec Summary

Kotlin down the years, and its role today
About the report
Our use of data

Snapshots – Kotlin: 15 Years in 6 Numbers

1. The Kotlin Adoption Curve: A 15-Year Perspective on Ecosystem Maturity

1.1 The evolution of Kotlin and key milestones
1.2 Kotlin’s expansion
1.3 Retention and ecosystem maturity
1.4 External validation from the broader market
Key Points

2. Enterprise and Organizational Adoption

2.1 Kotlin in organizations of different sizes
2.2 Adoption across industries
2.3 Interoperability between Java and Kotlin
2.4 Production usage and organizational trust
Key Points

3. Talent Pool and Workforce Readiness

3.1 Kotlin combines talent with experience
3.2 Kotlin’s access to adjacent talent pools
Key Points

4. Kotlin Multiplatform – a Strategic Platform Choice

4.1 From emerging technology to production-ready platform
4.2 Adoption is gaining momentum
4.3 How teams actually use KMP
4.4 Compose Multiplatform: sharing interface as well as logic
4.5 Organizational outcomes of KMP
Key Points

5. Backend Kotlin: Stability, Productivity, and Enterprise Readiness

5.1 Backend is now a core production use case
5.2 A gradual adoption path for JVM teams
Sidebar: What to validate in a Kotlin backend pilot
5.3 Adoption follows the JVM modernization cycle
5.4 A mature ecosystem: compatible and standardized
Sidebar: Backend Kotlin in practice – Kakao Pay
Key Points

6. Kotlin + AI: Tested in Real Workflows

6.1 Kotlin in the production layer of AI applications
Sidebar: Agents in production at Mercedes-Benz.io
6.2 AI tools are standard practice among Kotlin developers
6.3 Developers report strong results with AI on Kotlin code
Sidebar: Developers on AI + Kotlin
6.4 The Kotlin Benchmark: how well agents handle real Kotlin tasks
Key Points

The JetBrains Take

In Conclusion: A Language Fit for Purpose

Sources

Exec Summary

Kotlin down the years, and its role today

Kotlin, the programming language developed by JetBrains, is no longer young – it's mature. Fifteen years from initial release, it's reached an estimated 8.1 million developers worldwide. Since Google's official support for Kotlin on Android in 2017, the language has demonstrated sustained rather than cyclical growth, attracting developers across new domains while continuing to strengthen its presence in existing ones. Rather than peaking in a single area, Kotlin has steadily expanded across Android, backend, multiplatform, and AI-related development. Here's a summary of our findings.

- **Steady growth, confirmed by outside observers.** Independent rankings tell the same story as JetBrains' own data. Kotlin was among the top five fastest-growing languages on GitHub in 2024, is among the most admired languages on Stack Overflow, and has held a stable position in RedMonk rankings since 2024. All this data suggests it holds a durable and sustainable market position. Section 1 ahead explores it further.
- **Organizations adopt Kotlin incrementally, without rip-and-replace.** Kotlin is in play at firms of every size and across many industries, with around 80% of developers using it in production. Interoperability is the enabler: Over two-thirds of Kotlin developers used Java as their primary language before, giving Kotlin a clear route into environments without a disruptive migration. Section 2 examines how it enters and spreads through today's organizations.
- **The talent pool is maturing as well as expanding.** Two-thirds of Kotlin developers have more than two years of experience, while nearly one in five started using the language within the past year, indicating a steady influx of new developers alongside an established community. Java remains the dominant feeder pool (which lowers hiring risk), but the ecosystem is now producing Kotlin-first developers without prior Java backgrounds. This workforce is profiled in Section 3.
- **Kotlin Multiplatform (KMP) has evolved from an experimental technology into a production-ready platform,** and its adoption is accelerating in both ecosystem surveys and real-world usage. KMP's share among cross-platform developers more than doubled in a single year, and its presence among the top 10,000 App Store apps nearly tripled between early 2024 and mid-2026, growing faster than Flutter and React Native. Adoption also runs deep: Roughly 60% of KMP projects share all or almost all of their business logic across platforms. Section 4 sets out how teams use KMP.
- **Backend development is now a core use case.** Around half of Kotlin developers work on backend projects, and in JetBrains' anonymized IDE data, almost nine in ten large engineering organizations already run some Kotlin on the backend. Development cycles are 15%–20% shorter than in comparable Java work, and the ecosystem now carries enterprise weight – a strategic partnership with Spring and a formal security and support policy. Section 5 goes deeper into stability, productivity, and adoption paths.
- **Kotlin's strengths carry over to the AI era.** Nine out of ten Kotlin developers use AI coding tools, and the language works in their favor: When a model invents a method or misuses a nullable value, the code simply fails to compile, surfacing mistakes before anything runs. Developers report most AI-generated Kotlin works after minor fixes, and a production AI stack has emerged – Spring AI, Koog, and the Kotlin MCP SDK. Section 6 explores these developments.

At 15, Kotlin has become infrastructural. Its growth is broad-based, its developers have deep experience, and its ecosystem is expanding into multiplatform, backend, and now AI – signifying a language that has matured without losing momentum. Enjoy the report ahead.

About the report

The State of Kotlin 2026 report is both a retrospective – tracking how a JVM newcomer became a mature, multi-domain ecosystem and infrastructure – and a forward-looking view of the road ahead, exploring Kotlin’s role across Android, backend, multiplatform, and AI-assisted engineering.

It is written for engineering leaders, system architects, and technical decision makers who need to understand where Kotlin fits into existing systems and what its trajectory implies for hiring, tooling, and platform strategy.

Our use of data

This report draws on multiple sources, including JetBrains’ own annual State of Developer Ecosystem Survey, Kotlin Multiplatform Survey, and Kotlin Developer Survey, plus outside resources including independent industry reports, case studies, and external market data. Every claim is grounded in survey data, ecosystem facts and figures, or independent third-party sources. Where a finding is provisional or open to interpretation, we state this clearly.

Kotlin: 15 Years in Numbers

8M+

Kotlin
developers
worldwide



50%+

use Kotlin
for backend
tasks



30%+

use it on
multiplatform
projects



80%

use it in real
production
environments



81%

say AI-generated
Kotlin is usually
correct or needs
only minor fixes



87%

are satisfied
or very satisfied
with the language



Sources: JetBrains State of Developer Ecosystem Survey 2025;
JetBrains Kotlin Developer Survey 2024–2025

The Kotlin Adoption Curve: A 15-Year Perspective on Ecosystem Maturity

Kotlin is a programming language created by JetBrains – one of the company's core technologies, alongside its integrated development environments, data science and team tools, and AI offerings.

A modern language with concise syntax and built-in null safety, it runs on the JVM and interoperates fully with existing Java codebases.

That compatibility, coupled with Google's recognition of Kotlin as the preferred language for Android, has made it a trusted choice for a diverse set of applications, including mobile, backend, and multiplatform. This section looks back at Kotlin's story so far.

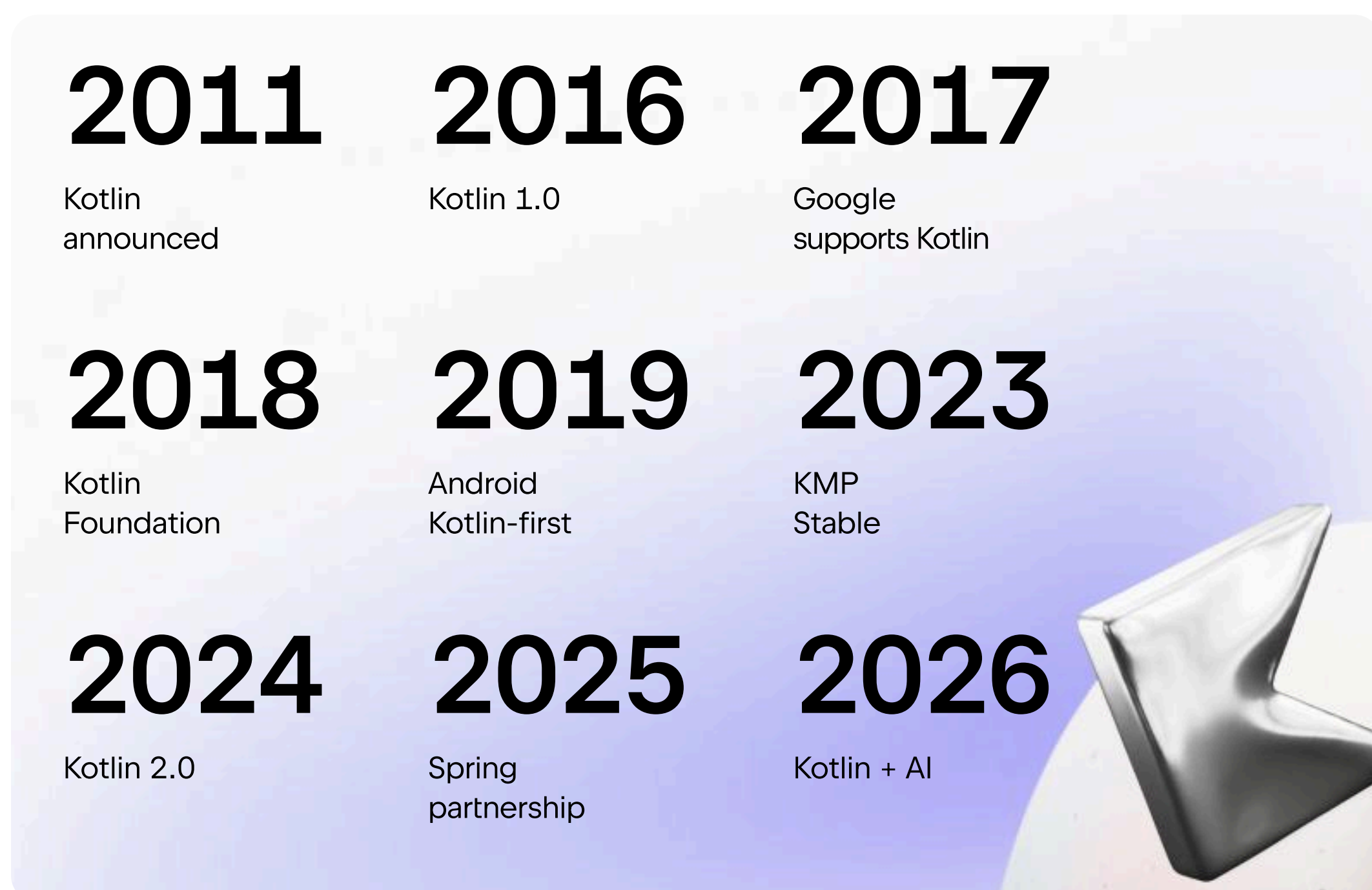
8.1M

developers worldwide
have  Kotlin experience

1.1 The evolution of Kotlin and key milestones

Kotlin's history is one of steady progress, building up experience, capabilities, and credentials over time as developers adopt and become familiar with it. Here's a brief timeline.

- In **July 2011**, at the JVM Language Summit, [JetBrains announced Kotlin](#) – a new statically typed programming language for the JVM.
- In **2016**, JetBrains released [Kotlin version 1.0](#), the first stable release of the language.
- In **2017**, Google recognized Kotlin as a [first-class language for Android](#), a move also reported in places like [TechCrunch](#). Prior to the announcement, JetBrains survey data showed just 2% of developers using it regularly; by 2018, adoption was nearing 10% and the Kotlin Foundation was launched.
- In **2019**, Google announced a [Kotlin-first developer strategy](#) at its I/O conference.
- In **2023**, [Kotlin Multiplatform became stable](#), broadening the language's set of use cases further.
- In **2024**, at KotlinConf, JetBrains announced the [release of Kotlin 2.0](#) with a stable K2 compiler.
- In **2025**, a [strategic partnership with Spring](#) boosted Kotlin's backend strengths.
- In **2026**, the agent framework [Koog reached stable release](#) and the Kotlin standard library gained a formal security and support policy.
- **Also in 2026**, JetBrains released the [Kotlin Benchmark](#), an open, public evaluation of AI coding agents on real Kotlin engineering tasks, with all datasets and test harnesses published – part of a commitment to measure the Kotlin AI ecosystem transparently – so teams can compare agents on evidence rather than vendor claims.

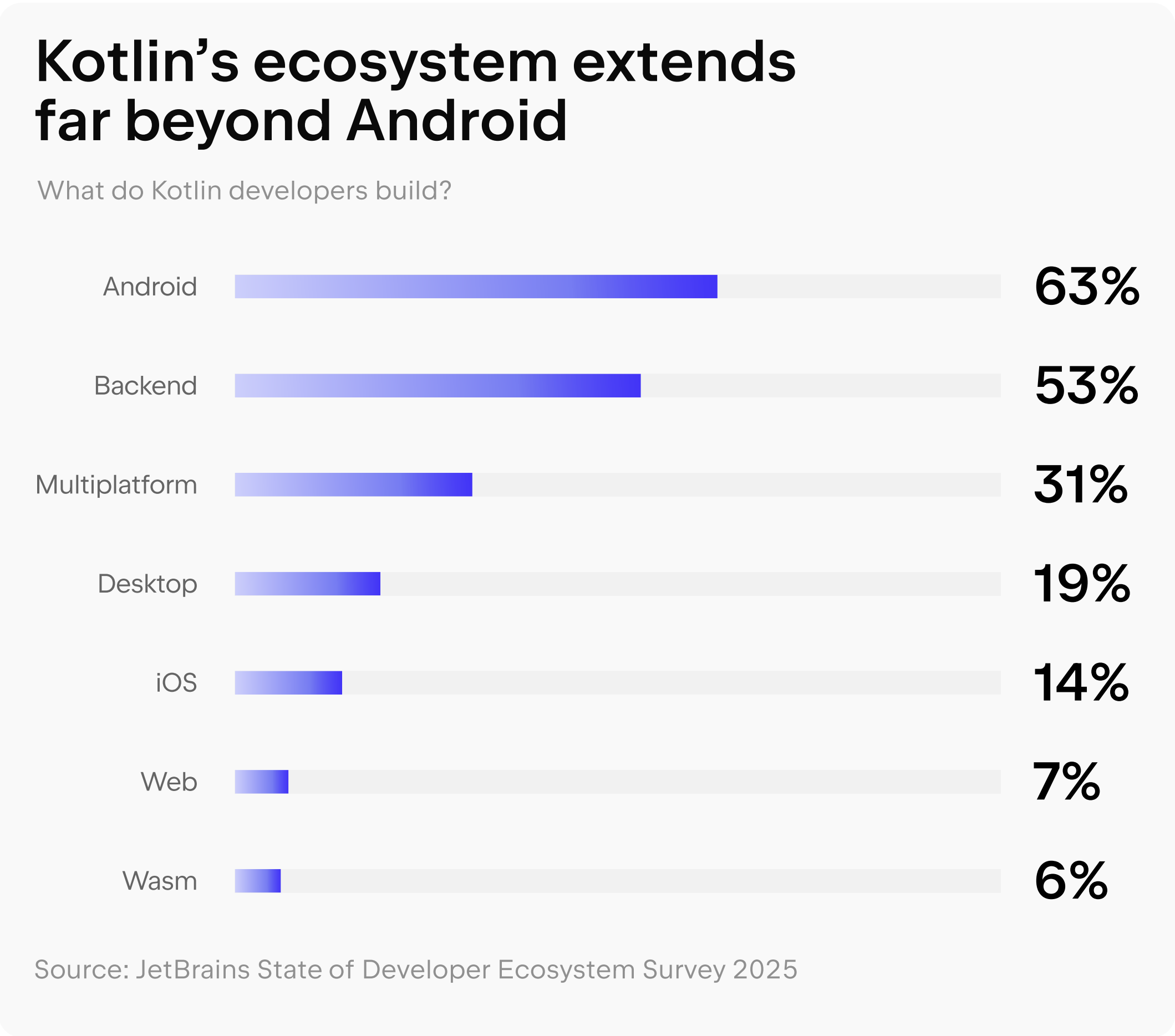


1.2 Kotlin’s expansion

Following Google’s 2017 announcement of first-class Android support, Kotlin quickly became known as “the Android language” in most developers’ minds – associated mainly with mobile apps.

In the years that followed, that association gradually became looser, driven mainly by backend adoption. And starting in the late 2010s, Kotlin’s use as a JVM backend grew faster than its mobile base – JetBrains’ 2023 Developer Ecosystem Survey report described Kotlin’s evolution from an Android-focused language into a general-purpose one.

By 2025, data showed backend development had become Kotlin’s second most popular use case, with over half of Kotlin developers using the language for backend. The language’s footprint now extends into multiplatform, desktop, and web development, but the backend is the clearest single sign that the language is no longer defined by the platform that first popularized it.



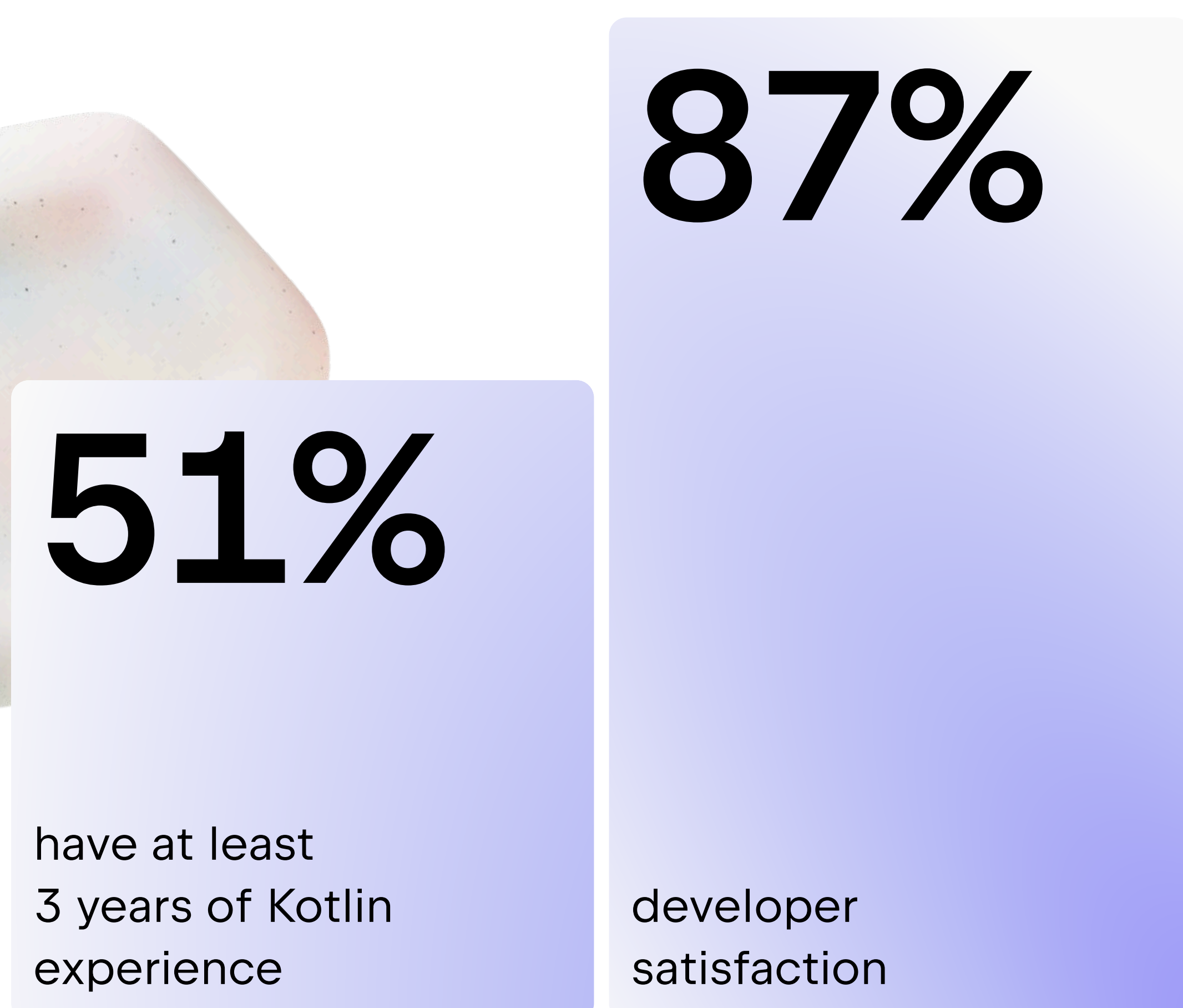
1.3 Retention and ecosystem maturity

A growing user base doesn't mean much if developers drift away after one or two projects. The data suggests Kotlin developers stay – and keep using the language year after year.

The JetBrains Developer Ecosystem Surveys show that the number of developers having at least four years of Kotlin experience nearly tripled between 2021 and 2023. By 2025, the Kotlin Developer Survey found that about half of Kotlin developers have been using the language for at least three years, and nearly one-third for five or more.

In addition, satisfaction correlates with retention. The 2024 Kotlin Developer Survey suggested developers become more satisfied with the language as they use it more, with CSAT scores higher for experienced developers than newer ones. Developers adopt Kotlin, grow their expertise, and use it more widely as they discover what it enables.

Taken together, retention and rising satisfaction describe a mature ecosystem: Developers who adopt Kotlin tend to stay, deepen their expertise, and carry the language into their next projects.



51%

have at least
3 years of Kotlin
experience

87%

developer
satisfaction

Source: JetBrains Kotlin Developer Survey 2024–2025

1.4 External validation from the broader market

While much of our data comes from in-house research, the positives show up beyond JetBrains' own datasets.

There are multiple examples of respected sources ranking Kotlin among the industry's most actively deployed and discussed languages:

- In **2024**, the [GitHub Octoverse](#) listed Kotlin among its Top 5 fastest-growing languages, showing the momentum the language has in the market.
- The **2024 Stack Overflow Developer Survey** put Kotlin among the most admired languages according to developers, with a 61% positive score demonstrating strong user satisfaction.
- **Further Stack Overflow Developer Survey** data shows Kotlin holding a steady 8%–9% share of professional developers from 2020 through 2025, ranking 15th among the most-used languages in 2025. Stack Overflow itself has noted Kotlin – alongside Go – overtaking older mainstream languages such as Ruby, which sat in the survey's top ten as recently as 2017. And a stable share of a growing developer population means the absolute number of Kotlin developers keeps rising.
- **Various RedMonk Rankings** figures put Kotlin in its top 20 for rates of usage, ranking 17 in 2024 and 14 in 2025 and 2026. This run supports Kotlin's continuing relevance to developers – not a one-hit wonder, but a stable and strong choice for ever-larger projects.

Google's 2017 endorsement transformed Kotlin from a promising JVM tool into a mainstream platform technology. With each milestone, it has grown from a language into a broader ecosystem backed by major players in software development, with applications spanning mobile frontend to backend and multiplatform. Above all, the Kotlin story demonstrates market confidence in the technology, a growing community of experienced developers, and a strong production footprint.

2.1 Kotlin in organizations of different sizes

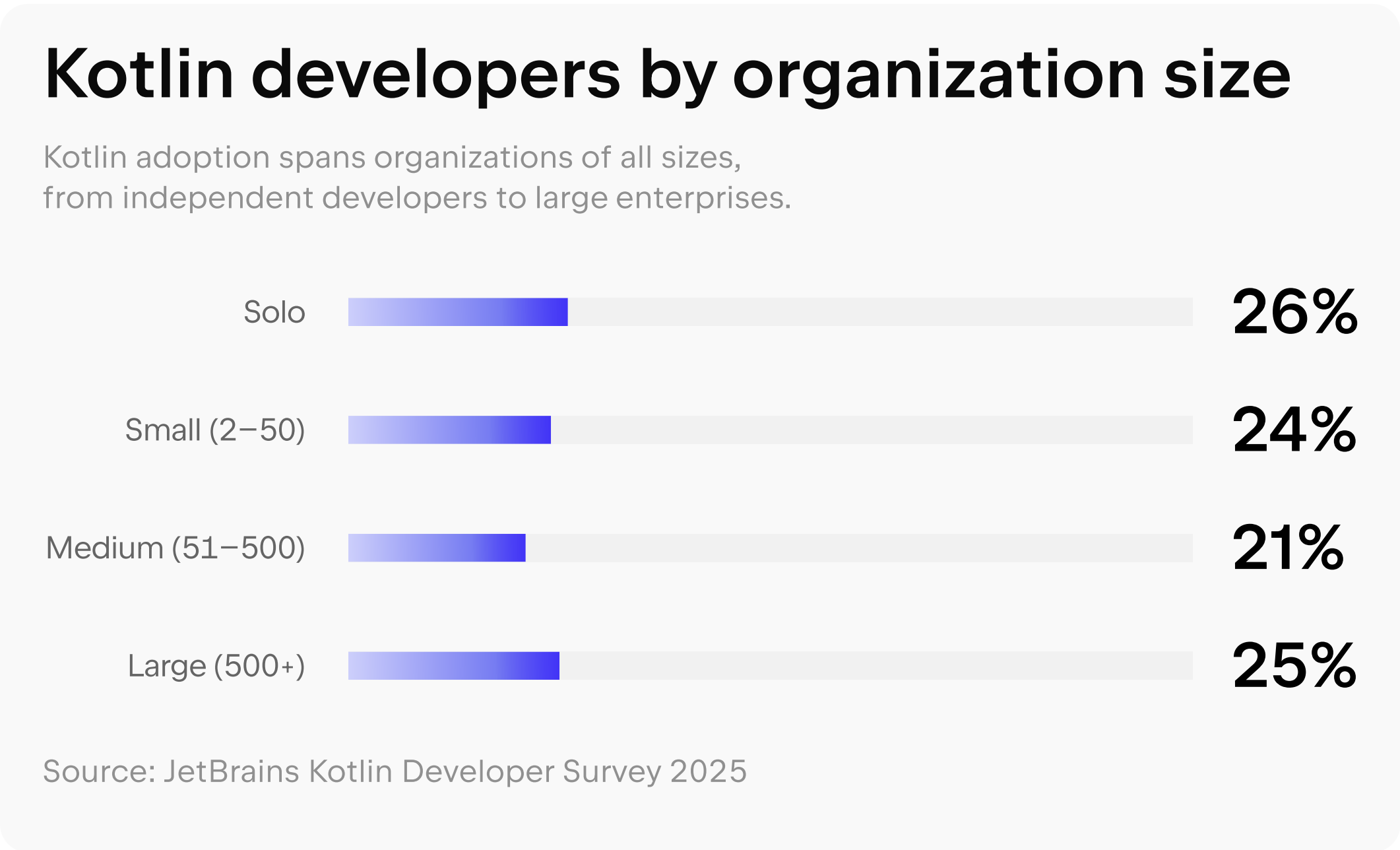
Enterprise and Organizational Adoption

Section 1 discussed Kotlin’s takeup among individual developers. In Section 2, you’ll see the picture at scale – with insights into how the language is used worldwide. Companies of all sizes, from solo developers to enterprises with over 5,000 employees, are adopting Kotlin.

Kotlin usage isn’t concentrated in a single company size; 2025 Kotlin Developer Survey’s 1,532 respondents work across the full range, from solo practitioners to the largest enterprises.

Across various surveys, independent developers are a large grouping, with some showing over a quarter of respondents identify as working for themselves – meaning that Kotlin has a strong following among individuals making their own choices. But the rest of the distribution is broad. In our own survey, small companies of 2–50 account for nearly a quarter, mid-sized organizations of 51–500 for over 20%, and larger enterprises of more than 500 employees account for another quarter, fairly evenly distributed across the 501–1,000, 1,001–5,000, and 5,000-plus bands.

This even spread suggests Kotlin turns up wherever software is written – from startups choosing a modern stack to mid-sized firms scaling their products and large enterprises maintaining substantial existing codebases. While this dataset is a snapshot and not a trendline, it shows the diversity of Kotlin adoption today.

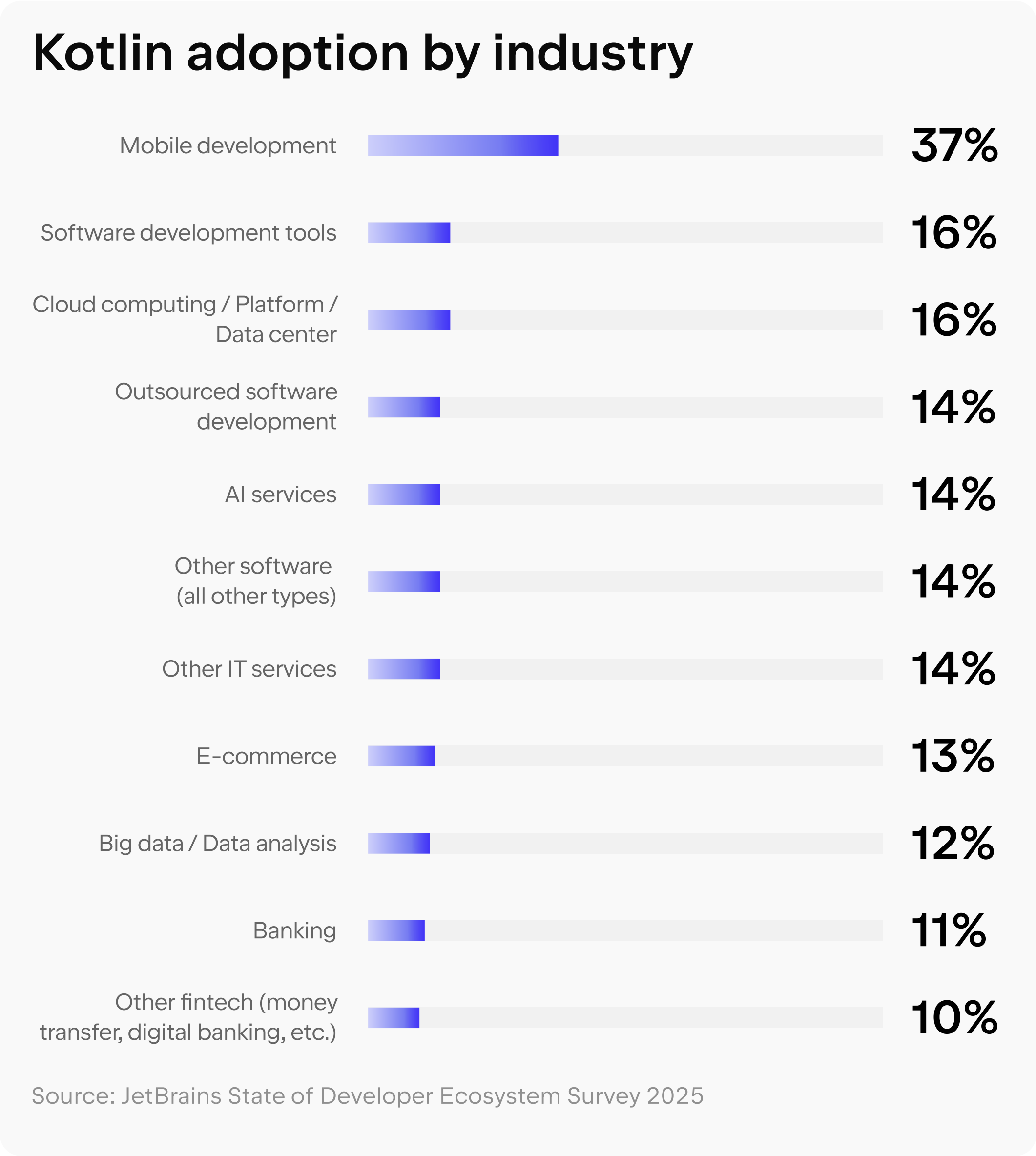


2.2 Adoption across industries

When filtered for Kotlin, 2020–2025 data from JetBrains’ annual Developer Ecosystem Survey shows a consistent industry pattern. Mobile leads the table, but there’s a broad range of sectors behind, and a long tail of representation almost everywhere software is created.

IT services is consistently the next-largest, followed by cloud computing, platform engineering, and software development tools at around 15% each. Financial technology is also a stronghold: Combining banking and other fintech, it hits the high teens and low twenties in different years. The pattern sharpens for those who use Kotlin on the backend: In the 2024 and 2025 surveys, more than one in five Kotlin backend developers (22%–23%) worked in fintech, and e-commerce climbed to fourth place at 16%–17%.

Beyond these, Kotlin maintains a steady presence across data analytics, healthcare IT, IoT and cybersecurity, and other areas. Published case studies map onto this spread – Cash App and ING in financial services, Philips in healthcare, Duolingo in education, Forbes in media, and McDonald’s in consumer-facing services.



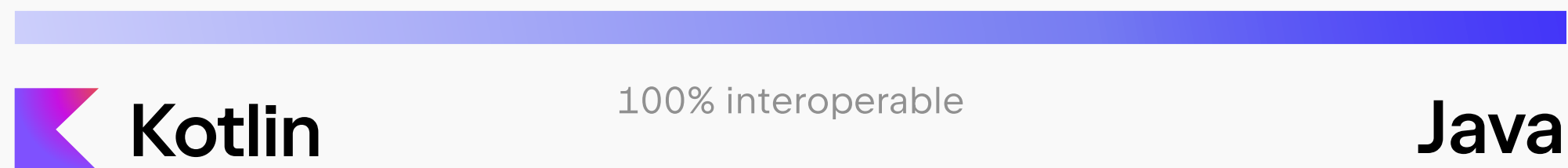
2.3 Interoperability between Java and Kotlin

Kotlin and Java are widely used together in different companies, teams, and projects, interoperating smoothly. Kotlin makes an explicit point of co-operating well with existing Java-based systems so that it can be introduced gradually into existing applications and infrastructure without wholesale rewrites.

With Java and Kotlin happily co-existing in the same codebase, organizations can preserve frameworks, libraries, tooling, and team expertise, maximizing the value of existing IT investments while benefitting from Kotlin's conciseness, null safety, enhanced developer experience, and measurable productivity gains.

A clear majority of Kotlin developers report Java being their previous choice of language, meaning they're comfortable with the ecosystem. Consequently, adopting Kotlin is widely viewed as a low-risk strategy, without disruptive rip-and-replace migrations.

Incremental Adoption Model



Organizations can adopt Kotlin incrementally because Kotlin and Java are fully interoperable and can coexist within the same codebase

2.4 Production usage and organizational trust

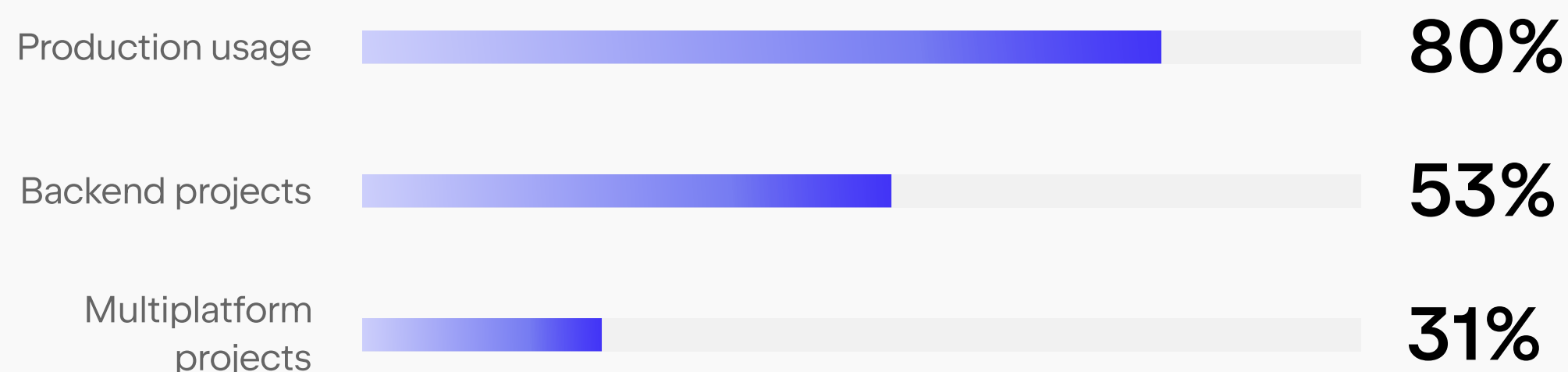
A clear sign that organizations trust Kotlin is where they run it. Production deployment is the point of high commitment: It implies code review, operational support, and a maintenance horizon measured in years.

In the 2025 Developer Ecosystem Survey, a large proportion of Kotlin developers reported using the language for real production code – not prototypes, side projects, or evaluation sandboxes, but in business systems companies depend on.

Other figures add color. As Kotlin experience has deepened, over half of developers in our survey use it for backend projects, and nearly a third for multiplatform – serious multi-discipline uses. It's evident that across domains – cloud, server, desktop, or mobile devices – Kotlin shows significant penetration into each segment.

Kotlin is increasingly used as a long-term production technology rather than a language for experimentation or isolated projects. And with long-term institutions now supporting the language – the Kotlin Foundation providing guidance and governance, and JetBrains' relationship with Spring – Kotlin is no longer an interesting tool for individual experimentation, but a genuine part of enterprise production environments.

Kotlin production footprint



Source: JetBrains State of Developer Ecosystem Survey 2025

KEY POINTS

Kotlin has evolved from an experimental technology to a strategic, low-risk multiplatform choice for companies of all sizes. Although mobile remains the largest domain, it's now common to see Kotlin in a diverse range of software-intensive industries, with organizations adopting Kotlin incrementally and expanding their usage over time.

Talent Pool and Workforce Readiness

Java professionals are Kotlin's biggest source of talent – but an increasing number of developers are Kotlin natives, and the number with multiple years of experience is rising. This section shares some data on the people employed to write Kotlin code.

The Kotlin talent pool

8.1M

51%

of surveyed Kotlin developers have at least 3 years of Kotlin experience

developer satisfaction

Sources:

¹JetBrains Kotlin Developer Survey 2025,

²Anonymized JetBrains IDE data, 2025

3.1 Kotlin combines talent with experience

In 2025, JetBrains research estimated 8.1 million developers worldwide have Kotlin experience. The pool of people who can code in Kotlin has expanded steadily in the language's second decade.

This pool has depth, too. The 2025 Kotlin Developer Survey found that about half of Kotlin developers had been using the language for at least three years, meaning a significant share have moved well beyond initial experimenting. The cohort with the most experience continues to grow, with developers actively deepening their knowledge over time.

As you'd expect, the cohort with four or more years of Kotlin experience has grown strongly since 2021. For employers, the practical effect is positive: a hiring pool populated by developers with genuine skills and expertise using Kotlin in actual business environments.



3.2 Kotlin's access to adjacent talent pools

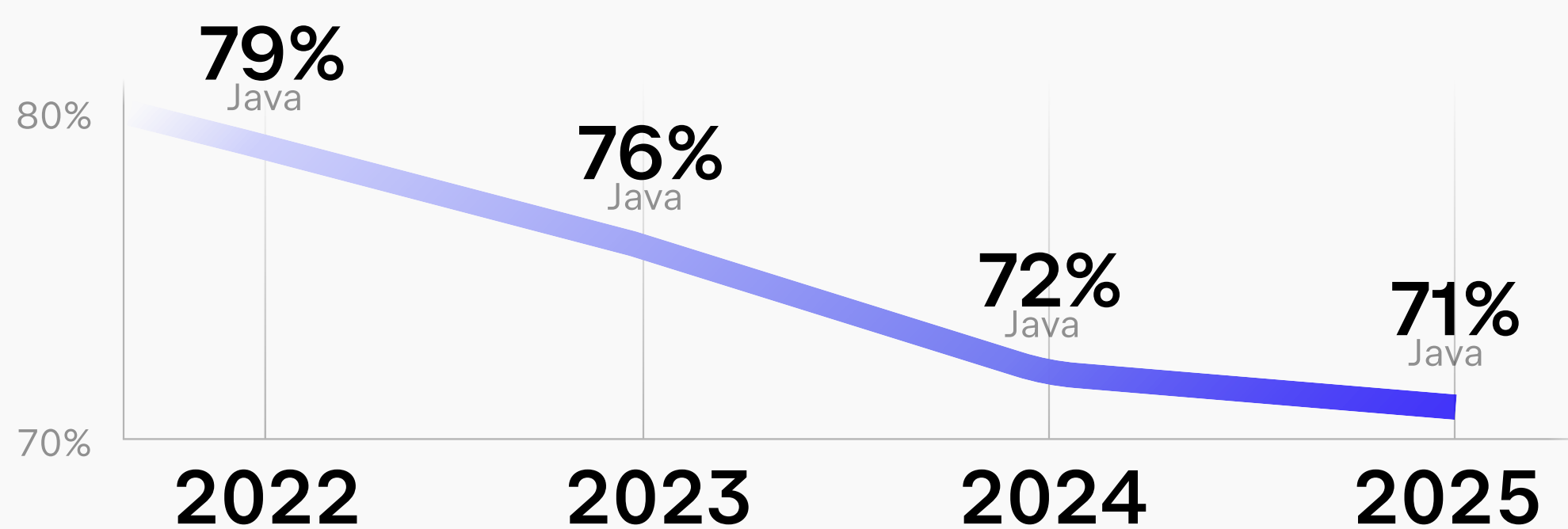
Where do Kotlin developers come from? Overwhelmingly it's from the earlier language you'd guess: Java. Across our State of Developer Ecosystem surveys, Java is consistently the language developers worked in before adopting Kotlin – by a wide margin.

Asked for their primary programming language before switching, over 70% named Java across surveys from 2022 to 2025. Earlier surveys, which asked a different question – whether respondents had used Java at all in their pre-Kotlin days – showed even higher figures of over 90%.

For organizations, this is a practical advantage. Because the language shares a talent pool with Java developers, hiring and adoption risks are lower. Companies don't need to hunt for "Kotlin developers" specifically – they can open the requirement to the vast Java-savvy population or even redeploy the Java developers they already employ. This helps explain why Kotlin adoption tends to be incremental, moving from Java to Kotlin in stages.

But there's a twist in the data. While the share arriving from Java is fairly steady (declining just a few percent from 79% to 71% between 2022 and 2025 in our data), developers are also coming to Kotlin from Python, JavaScript, TypeScript, and even C/C++. And some never switched at all: In 2025, 7% of Kotlin developers named it as the first programming language they ever learned – a hint of an ecosystem beginning to grow its own native talent.

Talent Origins: Java → Kotlin



Source: JetBrains State of Developer Ecosystem Survey 2022–2025

7%

reported  Kotlin as their first programming language (2025)

3.2 Kotlin's access to adjacent talent pools

Kotlin combines a growing pool of experienced developers with ready access to the broader Java ecosystem.

Together, these give organizations a workforce capable of supporting long-term production systems while reducing risks in hiring and onboarding. And the emergence of Kotlin-native developers suggests the ecosystem is increasingly able to sustain its own talent pipeline, rather than depending solely on Java for new entrants.

KEY POINTS

Kotlin developers often come from a Java background, but the number of “Kotlin natives” is steadily rising – with a small but significant number indicating it as their first language. When combined with the large number of Java developers who can move to Kotlin with ease, and the fact many developers are multiskilled and multiplatform, the talent pool is sizable, diverse, and flexible – excellent news for the organizations hiring them.

Kotlin Multiplatform – a Strategic Platform Choice

The previous sections mentioned Kotlin Multiplatform (KMP); Section 4 goes deeper. In a nutshell, KMP is JetBrains' cross-platform development technology for writing shared Kotlin code that runs on multiple targets – Android, iOS, desktop, web, and server.

With Kotlin Multiplatform declared production-ready in 2023 across supported platforms, and with Google's official endorsement for sharing business logic between Android and iOS, KMP is now an established part of the cross-platform ecosystem.



4.1 From emerging technology to production-ready platform

Kotlin Multiplatform builds on Kotlin's core language and ecosystem, while introducing its own capabilities for sharing code across multiple platforms. It evolved steadily from experimental tooling into a production-ready platform supporting both shared business logic and shared UI. Here are the milestones.

- **Pre-2022: Expansion of Kotlin capabilities.** Kotlin gains the ability to compile beyond the JVM, notably to iOS – vital for any genuine cross-platform code-sharing.
- **During 2022: [KMP for mobile reaches beta](#).** Sharing code between Android and iOS becomes a workable solution, although not yet with production guarantees.
- **During 2023: [Kotlin Multiplatform declared stable](#).** JetBrains announces that shared business logic is ready for production; many organizations are already using it.
- **During 2024: [Google endorses KMP](#).** At the Google I/O developer conference, the Android team announces support for sharing business logic across platforms and adopts KMP in its own apps.
- **During 2025: [Compose Multiplatform for iOS goes stable](#).** Shared UI functionality catches up with shared logic and launches into stable, enabling flexible code sharing across both business logic and UI without compromising app quality. [Compose Multiplatform for web goes Beta](#). This major milestone shows that Compose Multiplatform for web is no longer just experimental but ready for real-world use by early adopters.
- **During 2026: Ecosystem consolidation.** The [KotlinConf'26 keynote](#) reports 4,000+ community libraries on klibs.io and major Kotlin/Native performance gains – the marks of a maturing platform despite its short history.

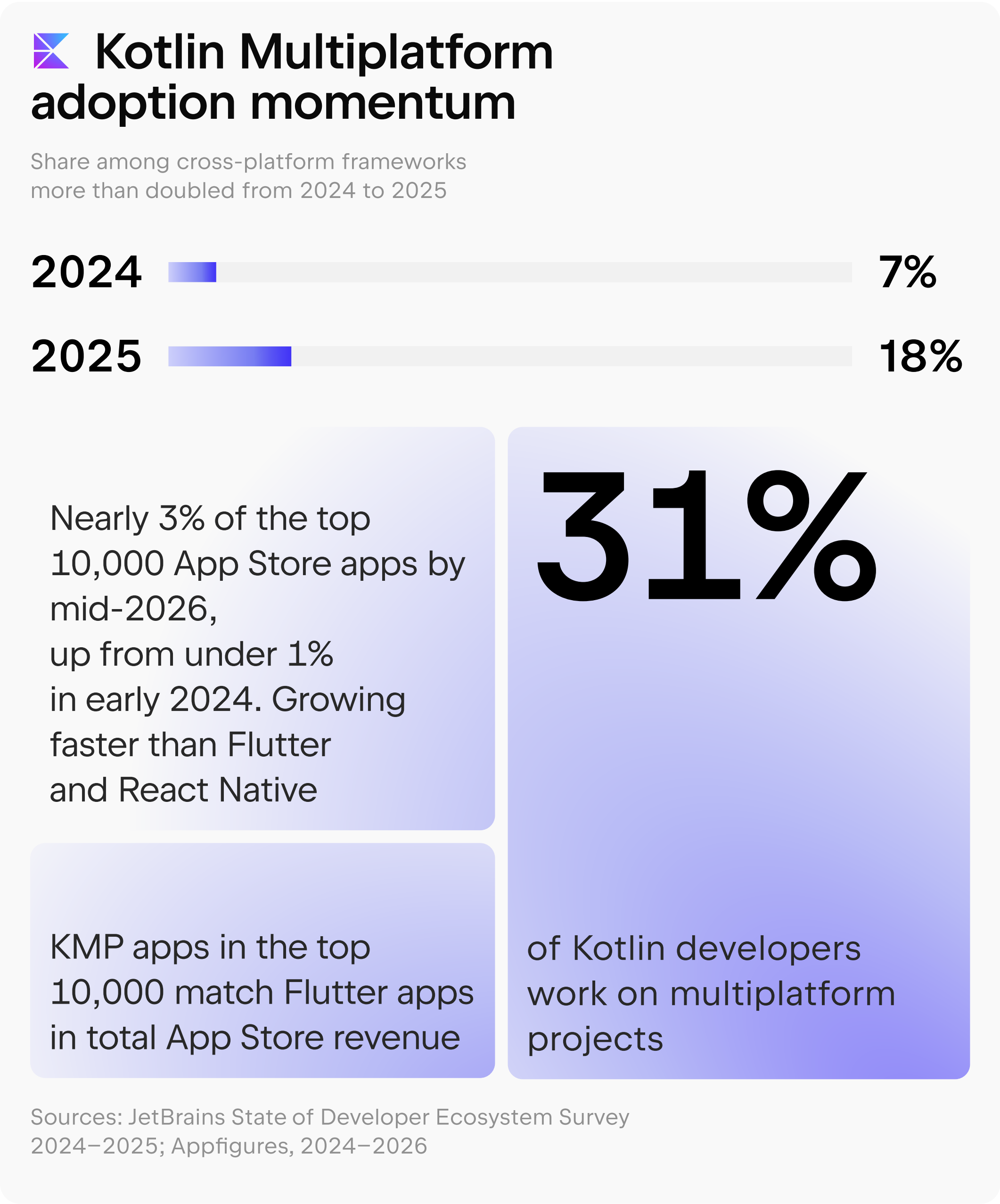
With Kotlin Multiplatform, Kotlin has evolved into a production-ready platform for cross-platform development, with both shared business logic and shared UI now stable across its primary targets. KMP gives teams the flexibility to decide how much code to share – from a single module up to 100% of their app code – while preserving native implementations wherever they make the most sense.



4.2 Adoption is gaining momentum

This move to production readiness has been matched by another trend: **acceleration in uptake**. Our Developer Ecosystem data shows that KMP's share among cross-platform developers more than doubled from 2024 to 2025, a considerable rate of change for an established framework category.

Adoption at the organizational level tells the same story. According to Appfigures data, KMP's presence among the top 10,000 App Store apps nearly tripled, from under 1% in early 2024 to almost 3% by mid-2026. Over the past year, KMP added more apps to the top 10,000 than Flutter did, despite starting from a far smaller base. The commercial weight is real, too: By mid-2026, KMP apps in the top 10,000 matched Flutter apps in total App Store revenue, and KMP-based applications serve hundreds of millions of users daily. Taken together, these are the signs of a technology going mainstream.



4.3 How teams actually use Kotlin Multiplatform

The dominant way organizations use KMP revolves around **sharing business logic** – the platform-agnostic core of an application, containing the rules and transformations that guide a workflow to a result.

And it goes deep. In the KMP Surveys for 2024 and 2025, 60% of projects reported sharing all or almost all of their business logic across platforms, and 80% reported sharing a significant portion. These results also seem steady year over year, which shows the pattern is an established practice among software developers, not a one-off spike.

Business-logic sharing in KMP projects

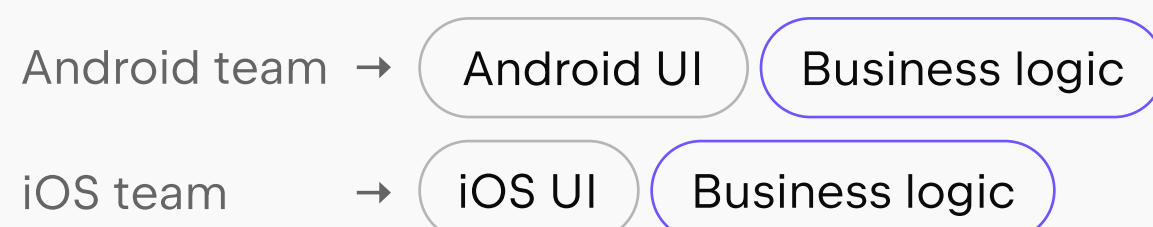
Results are statistically consistent year over year



Source: JetBrains Kotlin Multiplatform Survey 2024–2025

BEFORE

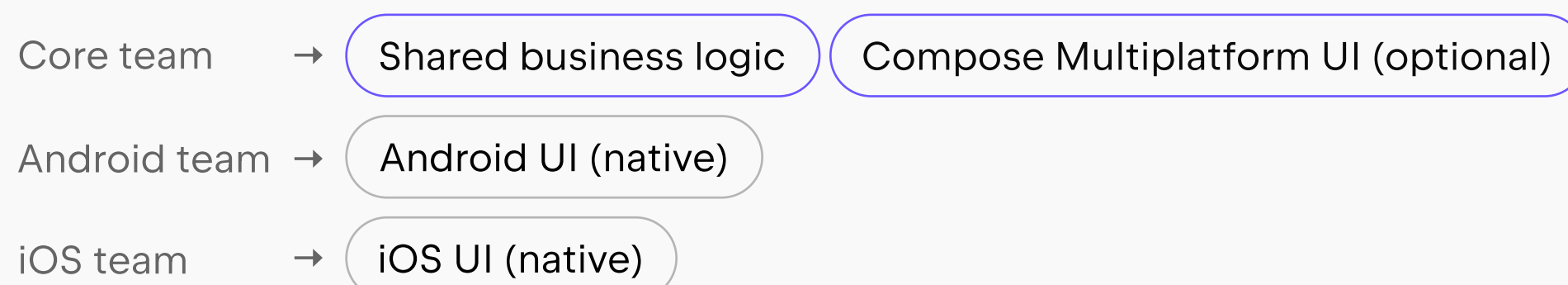
Duplication



- Parallel implementation
- Multiple fixes
- Coordination overhead

AFTER

Unified core



- Single source of truth
- Test once
- Deploy everywhere

4.4 Compose Multiplatform: sharing user interface as well as logic

While KMP is most often used to share business logic across applications and environments, teams increasingly seek to share user interface (UI) approaches, too, improving consistency of the user experience across platforms. In Android, that’s the job of Google’s **Jetpack Compose**, the declarative UI toolkit for Android – a model that JetBrains has extended beyond Android with **Compose Multiplatform**.

Within the KMP community, Compose Multiplatform adoption is near-ubiquitous. In the 2025 KMP Survey, around 8 out of 10 KMP users reported using it for UI tasks – meaning that for many teams, sharing a common user interface, not just the logic beneath, has become part of the multiplatform value proposition.

Satisfaction has also kept pace as Compose Multiplatform adoption continues to grow. A large majority of users said the toolkit’s feature set is sufficient for their needs. (In UI frameworks, this is a high bar to clear, since missing components or platform gaps often block progress.) We see consistent signals across different datasets that the ecosystem is becoming more capable and its users more confident in building full production applications. This suggests the toolkit is closing the gap between “usable” and “production-complete”.

Compose Multiplatform has become a core part of the KMP ecosystem, with rising satisfaction and a fast-growing share of teams able to ship complete applications with it.



Compose Multiplatform Maturity

81%

use Compose Multiplatform

83%

93%

of Compose Multiplatform users say it meets iOS look and feel expectations

of Compose Multiplatform users report that the feature set is sufficient for their needs

Source: JetBrains Kotlin Multiplatform Survey 2024-2025

4.5 Organizational outcomes of KMP

What do organizations actually get from KMP? The reported outcomes cluster around a single mechanism – writing code once for broader rollout across platforms, enabling downstream effects on speed, quality, and team/task coordination. Here are some outcomes supported by data from JetBrains’ Kotlin for Business and other sources.

- By sharing business logic, **teams can write an estimated 40%–80% of their mobile codebase once** rather than once per platform.
- Teams also report **15%–30% faster release cycles** after introducing shared logic.
- Full **return on investment** is typically reached within three to six months.
- Around **72% of users report improved development times**, with a similar share stating improved project quality.
- Approximately **55% report better collaboration** between their Android and iOS teams.

Published case studies from Duolingo, Booking.com, Sony, BiliBili, Philips, Bolt, Cash App, McDonald’s, and others describe the same benefits qualitatively: less duplicated work, faster feature delivery, more consistent behavior across platforms, and smoother coordination between mobile teams. The common thread is: **KMP improves delivery efficiency without a wholesale rewrite**. Shared business logic, gradual adoption paths, and retained native UX let teams capture the savings of code-sharing – while keeping the flexibilities specific to each platform.

Business outcomes organizations associate with KMP adoption

40–80%

of mobile stack
built once

15–30%

faster release
cycles

3–6

months
to ROI

Primary source: Kotlin for Business – kotlinlang.org/lp/kotlin-for-business

4.5 Organizational outcomes of KMP

KEY POINTS

The data suggests Kotlin Multiplatform is now a strategic platform choice. KMP's adoption momentum, deep integration into production workflows, and trust in its capabilities show organizations now treat it as a reliable and practical technology for sharing code across platforms, with no leap of faith needed. The insight is the rise in organizational confidence as KMP and its tools go forward.



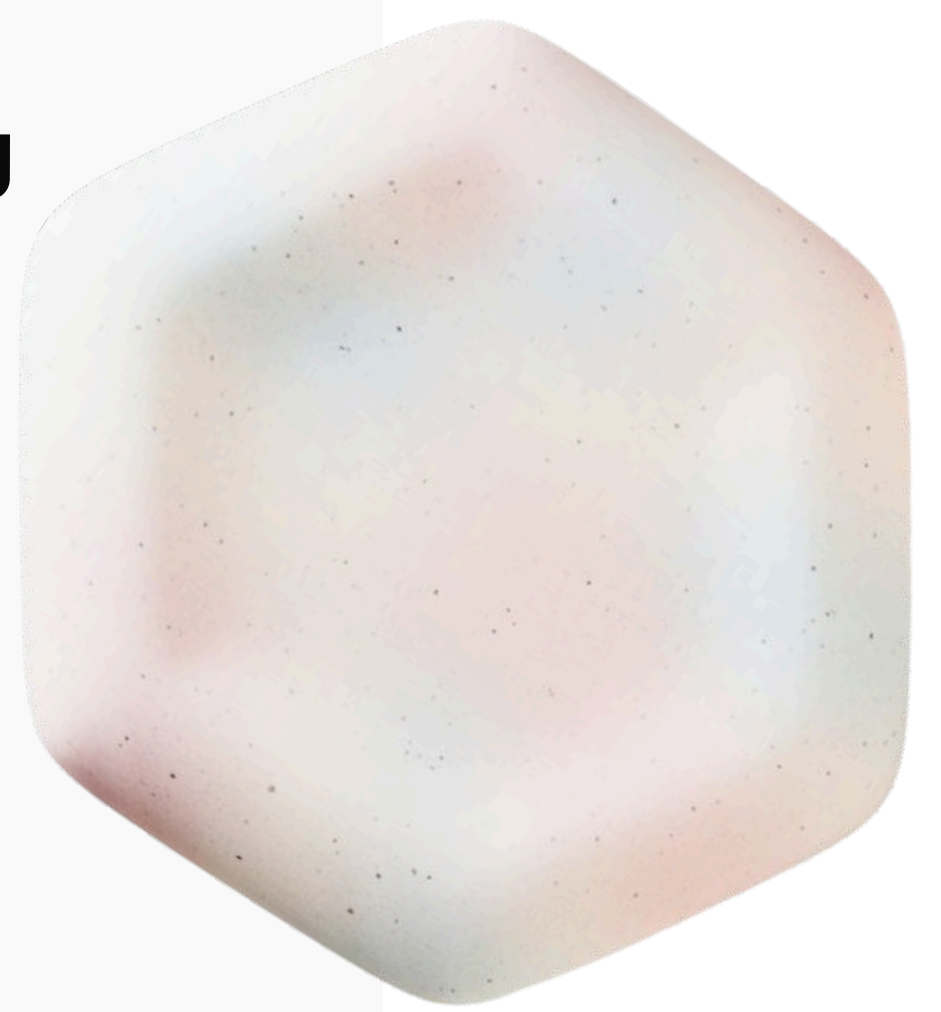
Backend Kotlin: Stability, Productivity, and Enterprise Readiness

Section 4 showed Kotlin reaching across platforms; In this section, you'll see it reaching across the stack.

Backend development has become one of Kotlin's largest domains, with over half of Kotlin developers working on backend projects.

That growth did not come from replacing Java systems. Kotlin compiles to the same bytecode, runs in the same JVM process, and calls existing Java code directly – so it spreads service by service through systems organizations already run and trust.

This section draws on two kinds of evidence: what developers report in surveys, and what anonymized JetBrains IDE data shows about commercial organizations doing backend development.



5.1 Backend is now a core production use case

The data leaves little doubt that backend development is a primary domain for Kotlin today. In our 2025 data gathering, over half of Kotlin developers reported working on JVM backend projects. More telling than the headcount is what they do with it: Nearly 80% of these developers run Kotlin in live production environments, not in prototypes or internal tooling.

The backend work itself spans several critical layers of modern infrastructure. Among developers voting Kotlin their primary language:

- **Over 40%** work on server, infrastructure, or cloud systems.
- **Around 40%** build and provide APIs of their own.

Surveys count developers; telemetry lets us also count organizations. In JetBrains telemetry, more than a quarter of commercial organizations doing backend development now use Kotlin on the backend – up from about one in five three years ago. And the larger the organization, the more likely Kotlin is already there: among companies with over a hundred backend developers, almost nine in ten have Kotlin somewhere in their backend today.

For a decision maker, the two views compound: Kotlin on the server is production-proven today – and in a large organization, adopting it rarely means going first. Someone nearby is likely already running it.

Kotlin backend adoption – developers

53%

of Kotlin developers
work on JVM
backend projects

78%

of them use Kotlin
in production
environments

43%

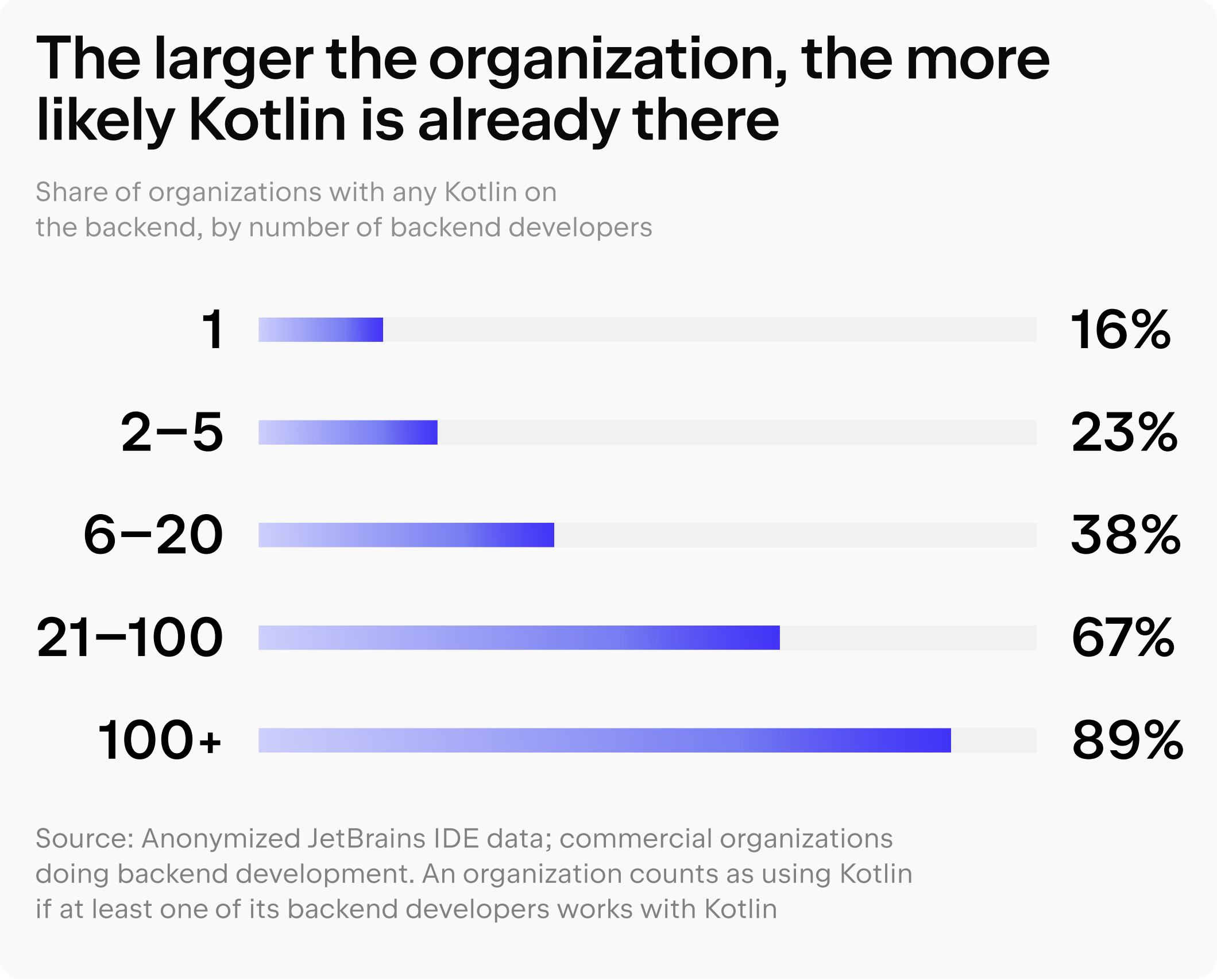
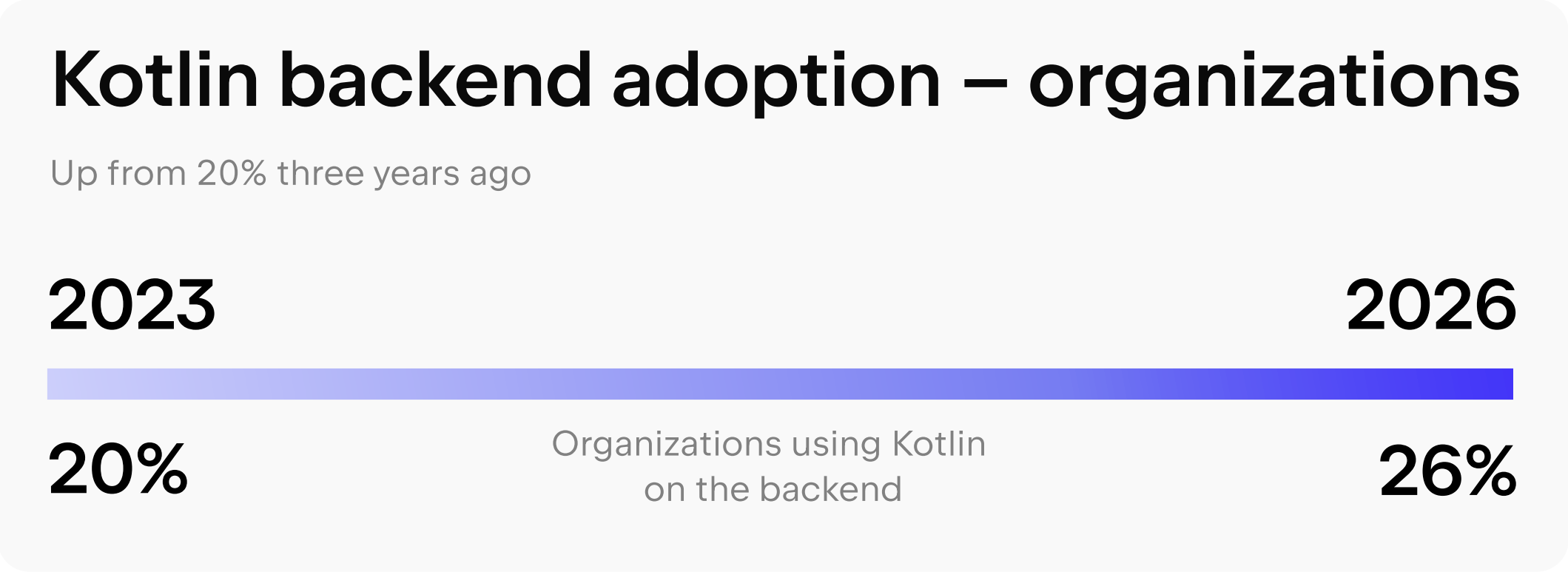
work on server,
infrastructure,
or cloud systems

40%

provide APIs
and services

Source: JetBrains State of Developer Ecosystem Survey 2025

5.1 Backend is now a core production use case



5.2 A gradual adoption path for JVM teams

In any organization of meaningful size, Kotlin almost always arrives into an existing Java codebase rather than being there from the start. The Java systems continue to run untouched, and the two languages coexist long-term, as most Kotlin backend developers still write Java, as well.

The most common unit of adoption is a whole service. A team builds its next service in Kotlin, and that service is essentially all-Kotlin from day one, while the services around it stay Java. Roughly three-quarters of backend codebases that contain any Kotlin are almost entirely Kotlin. The pattern is the same in every country we observed, from the most Kotlin-heavy markets to those where adoption is just beginning.

The classic entry paths experts describe are the small-scale version of the same motion – Kotlin enters next to working Java, in a bounded piece, without touching the rest:

- **Tests first.** The first Kotlin in many codebases is JUnit test code: low-risk, easy to review, and immediately less verbose.
- **One module at a time.** A new module is written in Kotlin and runs in the existing codebase, next to working Java code, with the same Jackson serialization and JPA persistence.
- **A new Spring Boot service.** The most common first Kotlin project is a service on the framework the team already knows.

There is also a measurable payoff that keeps the expansion going once the first service ships. JetBrains’ 2026 research analyzed IDE data from over 300,000 software developers and found Kotlin development cycles to be 15%–20% shorter than comparable Java cycles for everyday tasks. The reason isn’t just faster coding. According to JetBrains’ own observations, as projects grow more complex, Java’s cycle times tend to climb, while Kotlin’s rise more gradually.

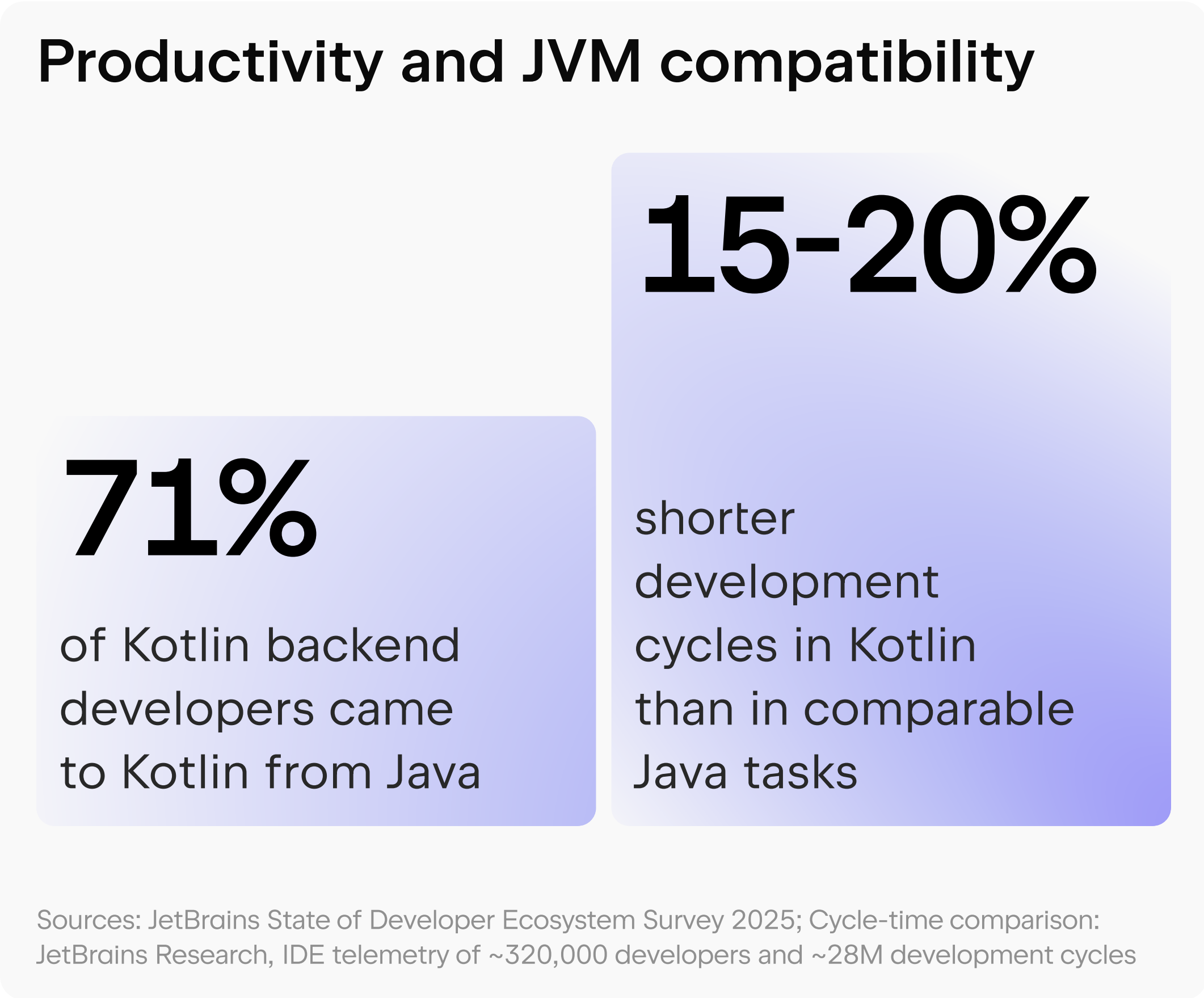
For decision makers, the combination is practical: Kotlin can be introduced without replacing the JVM stack, and its effect can be tested in a controlled pilot before broader adoption. Hiring reflects the same reality – Kotlin backend roles are typically advertised as "Java/Kotlin", so the talent pool stays the JVM talent pool.

BACKEND KOTLIN IN PRACTICE – KAKAO PAY

“As someone responsible for the server side of our system, my priority is building services that are stable, readable, and scalable. Since I started using Kotlin two years ago at Kakao Pay, I've been able to meet these goals step by step while developing a variety of applications.”

Kuyho Chung,
backend developer at Kakao Pay

5.2 A gradual adoption path for JVM teams



How Kotlin arrived, by company size

Number of backend developers	Born with Kotlin	Added into existing Java
1	50%	50%
2–5	36%	64%
6–20	24%	76%
21–100	16%	84%

Source: Anonymized JetBrains IDE data

What to validate in a Kotlin backend pilot

Kotlin can enter a JVM estate gradually, but interoperability does not remove every adoption cost. A bounded pilot – one new service or module – lets a team test Kotlin against its own architecture, build pipeline, and operating model before expanding

What stays the same

- The JVM and JDK
- Existing Java services and code
- Spring and most Java libraries
- Deployment pipelines and release process
- Runtime observability and monitoring
- Infrastructure and platform tooling

What changes

- The Kotlin compiler and standard library in the build
- Build configuration and compile times
- Coding conventions and review practices
- API contracts at the Java–Kotlin boundary (nullability, defaults, exceptions)
- Persistence and serialization setup (JPA plugins, Jackson module)
- Annotation processing and code generation (kapt/KSP)
- Static-analysis and security-scanning coverage
- Team onboarding and mixed-language ownership

What to measure

- Development cycle time
- Build and CI time
- Pull request review time
- Defects and rollbacks
- Onboarding effort for Java developers
- Interoperability issues at the boundary
- Runtime and memory performance
- Developer satisfaction

Decision point: Expand Kotlin only where the pilot shows a clear benefit without unacceptable build, integration, or operational costs.

Learn more in [The Ultimate Guide to Successfully Adopting Kotlin in a Java-Dominated Environment](#)

5.3 Adoption follows the JVM modernization cycle

Java teams add Kotlin at modernization moments. Projects that went through the Spring Boot 2→3 migration were eight times more likely to add Kotlin than those that didn't upgrade, and about seven in ten new Kotlin backend adopters already had Spring in the project when Kotlin arrived. Teams revisit their language choice when they are already reworking the codebase, not on an ordinary Tuesday.

The teams most likely to take that step are those that proactively maintain their tech stacks. For instance, teams running recent JDKs add Kotlin at a significantly higher rate than those stuck on JDK 8. At first glance, that might seem backward – if a team has already modernized its Java, hasn't modern Java met its needs? The data says the opposite. Upgrading the JDK is not a finish line; it marks a team that treats modernization as a habit. The same habit that took a team to the latest JDK takes it to Kotlin at the next opportunity. Kotlin adoption is not a break from the Java roadmap – it is what actively modernizing JVM teams tend to do next.

That makes the present moment notable: The Spring Boot 3 line reached open-source end-of-life in June 2026, and the ecosystem is moving to Spring Boot 4. For teams planning that migration, the record suggests this is exactly when the language question opens – and a new or rewritten service is the easiest place to answer it with Kotlin.

Kotlin adoption happens at modernization moments

8× higher

Kotlin adoption among projects that migrated Spring Boot 2→3, compared with teams that stayed put

0.9%

of projects that stayed on Spring Boot 2

7.2%

of projects that migrated Spring Boot 2→3 added Kotlin

Sources: Anonymized JetBrains IDE data

5.3 Adoption follows the JVM modernization cycle

70%

of new  Kotlin backend adopters already had Spring in the project

The newer the Java, the more likely the switch to Kotlin

Kotlin adoption rate among Java Spring projects, by JDK version (JDK 8 = baseline):

JDK ≤ 8	1×
JDK 17	~9×
JDK 21+	~14×

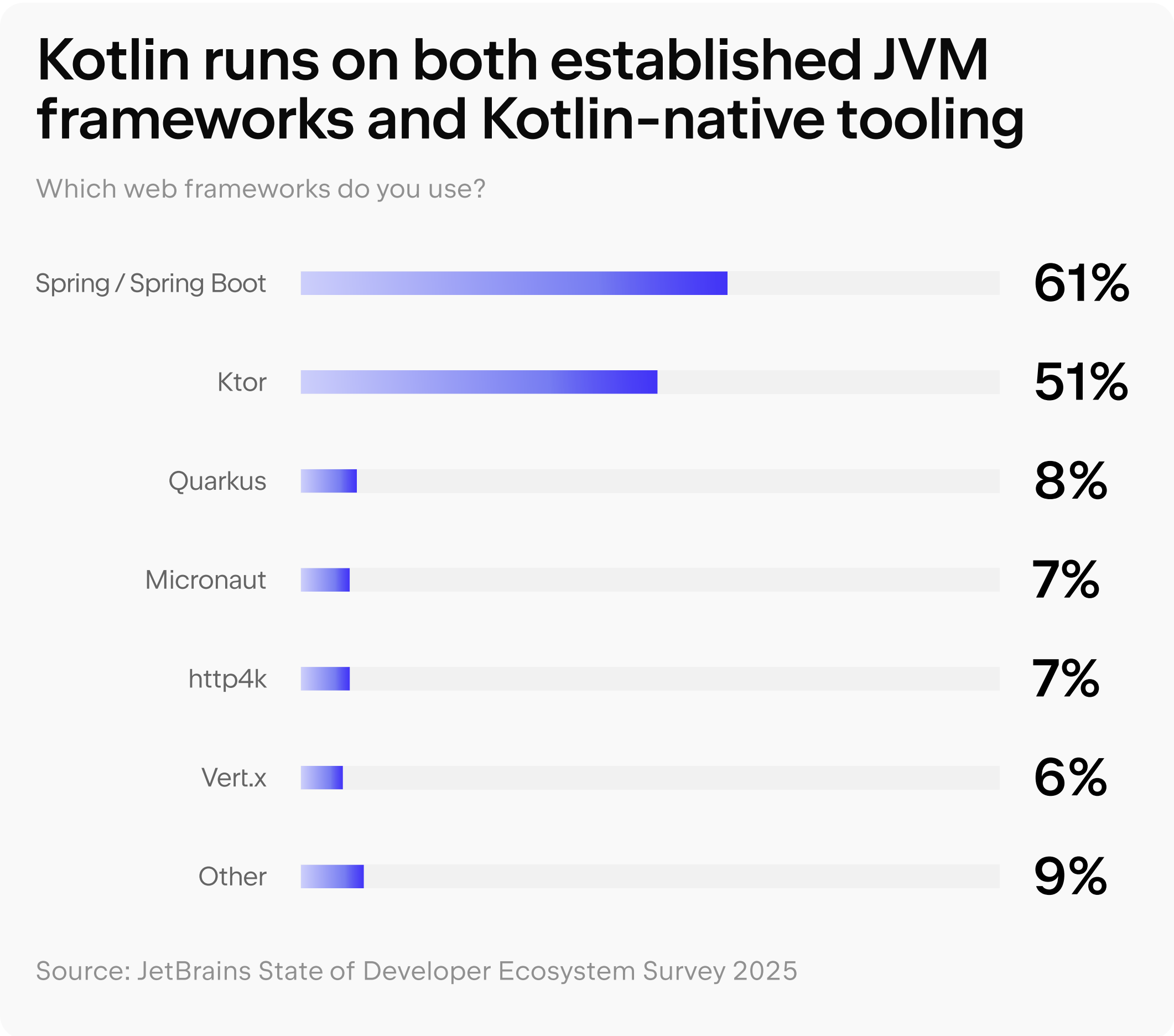
Sources: Anonymized JetBrains IDE data.
Share of Java-only Spring projects that added Kotlin within a four-month window (Jan–May 2026): 0.07%, 0.61%, and 1.01% respectively

5.4 A mature ecosystem: compatible and standardized

In addition to meshing with all the major JVM frameworks (Spring, Quarkus, Vert.x, Micronaut, http4k, and Javalin), Kotlin is compatible with the wider landscape of Java-based libraries and tooling; a team adopting Kotlin inherits decades of working code. The usage data reflects this: **About six in ten Kotlin backend developers work with Spring or Spring Boot**, making it the most common framework choice in the Kotlin backend world.

Spring already provides Kotlin extensions, coroutine support, and configuration DSLs. [The 2025 strategic partnership](#) expands this work into areas such as framework-wide null-safety metadata, official Kotlin learning materials, reflection performance, and further DSL development.

The second most-used choice is Ktor (51%), JetBrains' own lightweight framework, built from the ground up with Kotlin and coroutines. Teams reach for it where they want something lean and flexible – microservices, smaller services, and multiplatform HTTP clients – and the two frameworks frequently coexist in the same organization, which the multi-select numbers reflect.



One long-standing concern for regulated JVM teams was resolved in 2026. Organizations in regulated industries told us directly that the lack of formal support commitments blocked adoption – whatever the language's technical merits, compliance and security departments require guarantees. [The Kotlin standard library security and support policy](#), announced in May 2026, provides exactly that: defined support periods and security patching for stdlib versions. Compiler and build-tool support remain outside the policy's scope.

5.4 A mature ecosystem: compatible and standardized

KEY POINTS

Kotlin is a production-grade backend language: Over a quarter of backend organizations have it in their backend, including almost 90% of the largest. It enters as whole new services beside a running Java estate, not as a rewrite – and adoption clusters at modernization moments, with the Spring Boot 4 migration underway right now. The ecosystem is compatible with the major JVM frameworks and covered by formal support guarantees.

Kotlin + AI: Tested in Real Workflows

"Kotlin and AI" refers to two separate topics, and this section covers both. The first is building AI-powered systems in Kotlin: LLM-backed endpoints, retrieval pipelines, and agents are becoming ordinary backend workloads – a natural extension of the backend story in Section 5. The second is using AI coding tools – assistants and agents – to write Kotlin itself. Here is the production stack for the first, and for the second – how many developers do it, what they report about the results, and what the public benchmark measures.



6.1 Kotlin in the production layer of AI applications

Kotlin now has a growing JVM-oriented stack for adding AI capabilities to production applications: Spring AI and Ktor at the application level, Koog 1.0 (now stable) for agentic flows, and the Kotlin MCP SDK for connecting agents to tools and resources. Coroutines and flows handle the streaming and concurrency that LLM integrations demand, and typed data classes keep model inputs and outputs explicit.

This stack is not JetBrains talking to itself. When Anthropic – the company behind Claude – shipped [an official Java SDK](#), it wrote that SDK predominantly in Kotlin, providing an external example of Kotlin being used to deliver a Java-facing JVM library.

AI-Powered Backend Services

Three levels of adding AI to a production JVM backend

LEVEL 1

Call a model

A Kotlin service calls an LLM API and streams the response

Official model SDKs

Ktor HTTP client

Typed request/response

LEVEL 2

Build AI features into the application

LLM-backed endpoints, retrieval (RAG) pipelines, and embeddings inside the frameworks teams already run

Spring AI

Ktor

LEVEL 3

Orchestrate agents

Multi-step agentic workflows with tools, memory, and tracing; agents reach tools and data through MCP

Koog 1.0

Kotlin MCP SDK

AGENTS IN PRODUCTION AT MERCEDES-BENZ.IO

“Thanks to Kotlin’s strong typing and concise syntax, reasoning steps remain explicit and testable, while Koog also provides execution and tracing.”

Bruno Ferreira,
Mercedes-Benz.io

6.2 AI tools are standard practice among Kotlin developers

Use of AI tools has become standard practice among Kotlin developers. Among Kotlin users who took part in the 2025 JetBrains Developer Ecosystem Survey, AI tools had already become part of regular development practice: Over 90% regularly use at least one AI coding tool, and four in five already use or are likely to try AI coding agents that carry out multistep tasks rather than generate single snippets.

Company adoption is broad but measured. Nearly every organization is doing something with AI: A third have reached partial adoption, most of the rest are piloting or still exploring, and a smaller group has taken AI into extensive use. The pattern is high but gradual engagement – scaling what works rather than leaping into the unknown.

Kotlin holds a useful advantage here. Its static typing and null safety work as guardrails on AI-generated code. When a model invents a method or misuses a nullable value, the code fails to compile – and modern agents use that signal, building the project and reading compiler errors to fix their own output before presenting a diff. The compiler cannot catch flawed logic that passes the type checker; developers still review what they ship. Some type and nullability errors are surfaced mechanically, giving an agent an opportunity to correct them before presenting a final change, which is what makes AI output in a statically typed language cheaper to review.

For a decision-maker, the takeaway is narrower but real. Kotlin doesn't remove the need to review AI output – nothing does – but it catches a class of failures mechanically and makes each review cheaper. The next section shows how developers handle the rest.

AI Adoption among Kotlin developers

81%

already use or are likely to try AI coding agents

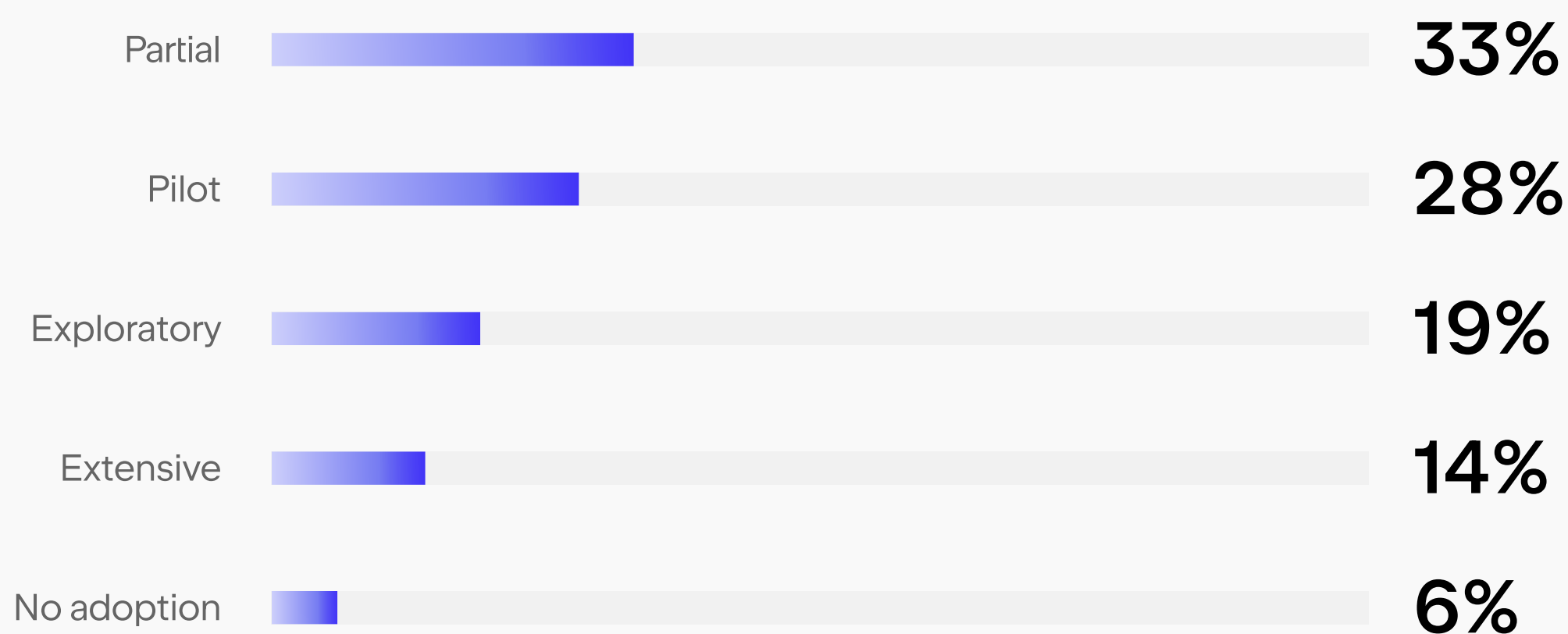
93%

use at least one AI coding tool

Source: JetBrains Developer Ecosystem Survey 2025

6.2 AI tools are standard practice among Kotlin developers

AI adoption in companies where Kotlin developers work



Source: JetBrains Developer Ecosystem Survey 2025

DEVELOPERS ON AI + KOTLIN

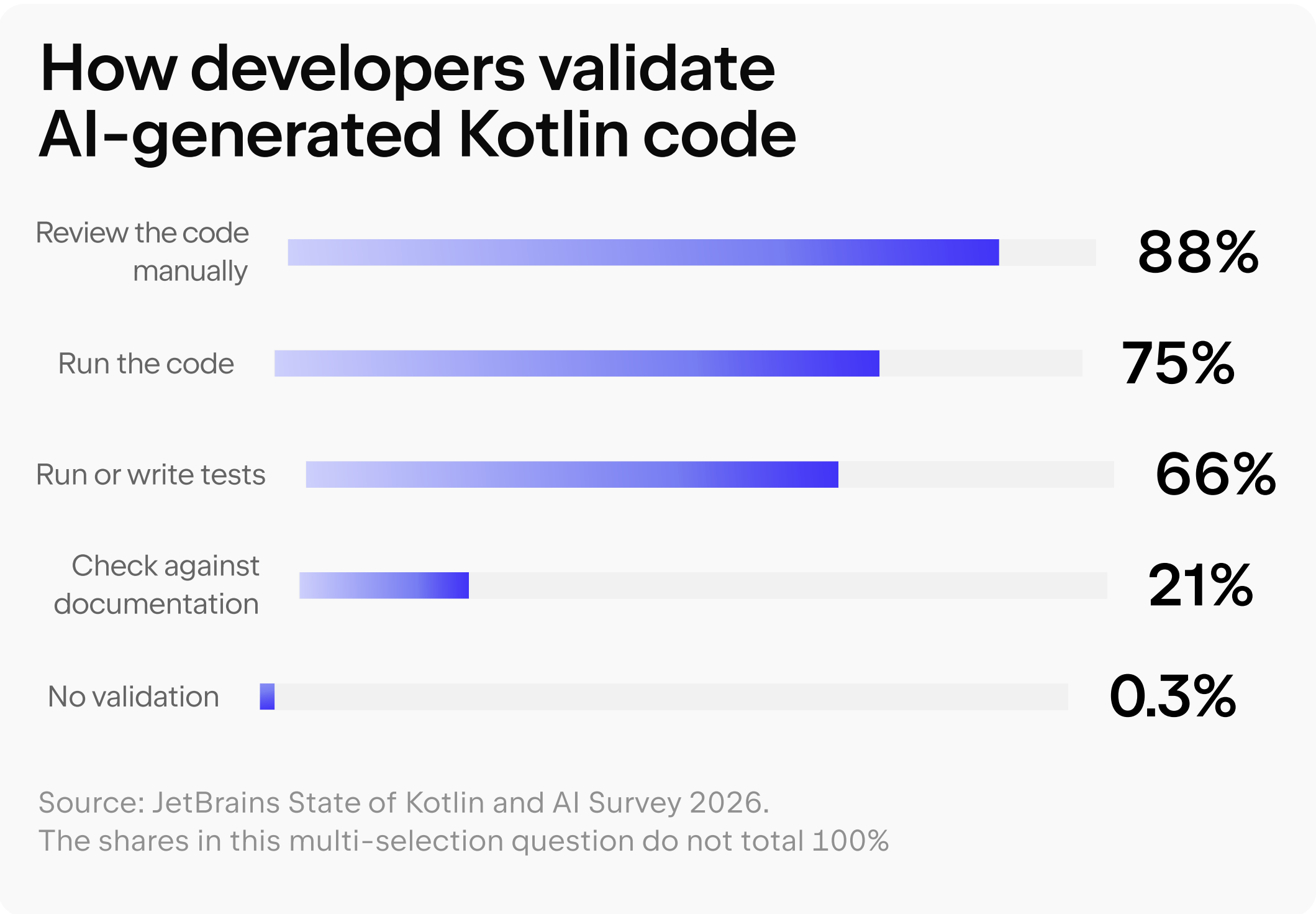
“Kotlin’s strict compiler and clear build-time errors give it an edge. The AI can easily understand and fix compilation errors before running the code. I’m comparing it to runtime languages like Python or JavaScript, where tracking down bugs is more complex because they only surface during execution.”

Adeeb,
Android developer

6.3 Developers report strong results with AI on Kotlin code

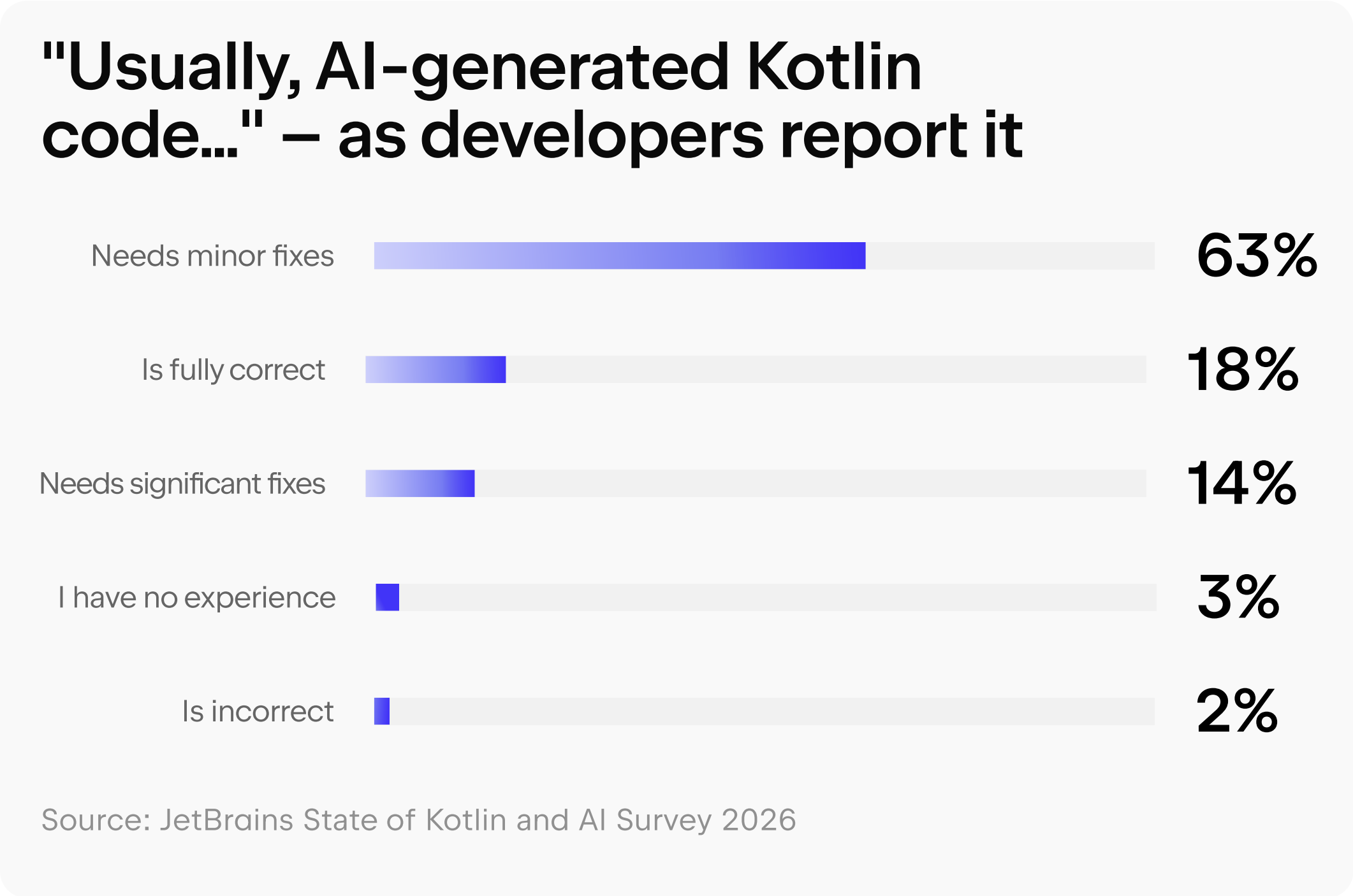
During and around KotlinConf 2026 (late May 2026), we surveyed 623 active Kotlin users from an engaged, early-adopter audience about **how they use AI and how AI performs on Kotlin code**.

The dominant working pattern is to scope a task, let AI generate or edit the code, and then review, test, and iterate. Practically everyone validates the result – most review the code manually, and a majority also run it or put it through tests; just 0.3% report shipping AI-written code without any validation. AI speeds up the middle step of the loop, while the developer owns the other two.



6.3 Developers report strong results with AI on Kotlin code

Asked how AI-generated Kotlin code usually turns out, respondents were positive but precise: In about four cases out of five, the result is working code after minor fixes. Only about one in seven report significant rework. The gaps respondents describe are rarely syntax; most of the time, they are project context, idioms, architecture, and current APIs. That is where Kotlin's compiler and type system earn their keep (see 6.1), as a whole class of AI mistakes fails to compile before it can fail in production.



6.4 The Kotlin Benchmark: how well agents handle real Kotlin tasks

What developers report is one kind of evidence; a public measurement is another. The Kotlin Benchmark, released by JetBrains in 2026, is an open evaluation of AI coding agents on real Kotlin software engineering tasks. It consists of 105 issues sourced from active open-source repositories, where an agent has to read a real issue description, navigate the project, and produce a working patch. The benchmark applies the same standard developers apply to AI output (see 6.3), i.e. a task counts as resolved only when the generated solution passes test verification in a containerized environment. All datasets and test harnesses are public, so anyone can compare agents on evidence rather than vendor claims.

Leading agents already complete a large share of the tasks. In the first public iteration, the top setup – Claude Code with Opus 4.7 xhigh – resolved 90 of the 105 tasks, with JetBrains Junie (Opus 4.7 max) and Codex (GPT 5.5 xhigh) close behind. The scores are a signal, not a guarantee: Real-world results depend on your architecture, internal APIs, coding standards, and validation process.

The Kotlin Benchmark:

Leading AI coding agents already resolve the large majority of real Kotlin engineering tasks

See the live leaderboard →

Setup (Agent + LLM)	
1	Claude Code + Opus 4.7 xhigh
2	Junie (Opus 4.7 max)
3	Codex + GPT 5.5 xHigh
4	Claude Code + Opus 4.6 max
5	Codex + GPT 5.3 Codex xhigh

This is a first iteration, and the benchmark is run as a continuous measurement pipeline rather than a one-off study. The roadmap extends the task mix toward more Kotlin domains – including Android and Kotlin Multiplatform – and adds evaluation dimensions beyond correctness, including cost, performance, maintainability, and code quality, across more agents and open-weight models. The leaderboard updates as new evaluations land. Follow the current results at kotlinlang.org/benchmark.

6.4 The Kotlin Benchmark: how well agents handle real Kotlin tasks

KEY POINTS

Kotlin developers have moved AI from experiment to daily practice – 93% use AI coding tools, and companies are scaling adoption deliberately, pilot by pilot.

The results hold up: Over 80% of surveyed KotlinConf attendees find AI-generated Kotlin code correct or near-correct, and validation is universal practice, with the compiler and type system lowering its cost. Current benchmark and survey results show that leading coding agents can complete a high share of evaluated Kotlin engineering tasks. Results remain task-, repository-, and setup-dependent.

The JetBrains Take

JetBrains, of course, is the creator of Kotlin. We built it, we maintain it, and of course we love it. That's why we always sense-check our own data with broader developer surveys and independent sources like GitHub, Stack Overflow, and RedMonk. We accept we're not neutral observers.

But according to multiple datasets, it seems the market agrees with us. The language has grown into a comprehensive ecosystem of applications both backend and front-end, running services at enterprise level with confidence. And the newest test – whether a language holds up in AI-assisted development – is one Kotlin is passing in public, on an open benchmark anyone can inspect.

Our philosophy is using the right tool for your needs. JetBrains AI tools – Junie, Air, JetBrains Central – work with Anthropic's Claude, OpenAI's GPT, Google's Gemini, or xAI's Grok, so you're never locked into a model provider you didn't pick. You can also point them at a local model on your own infrastructure, for full control.

So whether you're adopting Kotlin for the first time, scaling it across teams, or wiring AI into a JVM backend, we'd like you to try JetBrains tools in your own workflows – and judge them by the same standard we've used throughout this report: what the evidence shows.

That's Kotlin at 15. We look forward to improving it for many more years.



In Conclusion: A Language Fit for Purpose

Many languages enjoy a few years of rapid adoption before plateauing or fading into a niche. Kotlin's defining characteristic is that its growth has been sustained rather than cyclical. It continues adding developers while retaining existing ones; it keeps moving into new domains rather than peaking in one.

As we've seen from our sources, Kotlin has become a general-purpose platform. More than half of its developers now work on backend, nearly a third on multiplatform, and the language carries real production loads across finance, healthcare, and retail.

Its talent pool has matured, too. Two-thirds of Kotlin developers have more than two years' experience, most have previous experience with Java, and a growing minority are Kotlin-native. It's a diverse and flexible workforce – precisely what today's organizations need.

Organizations also need a language that holds up in the age of AI. Kotlin does, for a structural reason: When machines write more of the code, the premium moves to verifying it – and Kotlin's strict compiler, strong type system, and production-grade tooling catch mistakes regardless of whether a human or a model wrote them. The qualities that made it a safe adoption bet are exactly the ones the AI era rewards.

Fifteen years in, Kotlin is an established part of the software landscape: broadly adopted, deeply trusted, and still extending its reach. If you're not yet using it, start at kotlinlang.org/lp/kotlin-for-business.



Sources

JetBrains surveys and research

- State of Developer Ecosystem 2025 – devecosystem-2025.jetbrains.com/
- State of Developer Ecosystem 2024 – jetbrains.com/lp/devecosystem-2024/
- Kotlin Developer Survey 2025
- Kotlin Multiplatform Survey 2024–25
- State of Kotlin and AI Survey 2026
- JetBrains developer productivity research (IDE data, 300,000+ developers) – blog.jetbrains.com/kotlin/2025/12/the-ultimate-guide-to-successfully-adopting-kotlin-in-a-java-dominated-environment/

Independent and third-party sources

- GitHub Octoverse 2024 – github.blog/news-insights/octoverse/octoverse-2024/
- Stack Overflow Developer Survey 2024 – survey.stackoverflow.co/2024/technology#2-programming-scripting-and-markup-languages
- RedMonk Programming Language Rankings – <https://redmonk.com/sograde/2026/04/14/language-rankings-1-26/>
- TechCrunch: Google makes Kotlin a first-class language (2017) – techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/
- Mercedes-Benz.io: The guardrails your LLM needs – mercedes-benz.io/blog/2025-11-14-the-guardrails-your-llm-needs-reliable-agent-based-systems
- The Kotlin Benchmark leaderboard – <https://kotlinlang.org/benchmark/>

JetBrains and Google announcements and materials

- Google announces support for Kotlin (2017) – android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html
- Android Kotlin-first – developer.android.com/kotlin/first
- Kotlin Multiplatform Stable (2023) – blog.jetbrains.com/kotlin/2023/11/kotlin-multiplatform-stable/
- Compose Multiplatform 1.8.0: Compose Multiplatform for iOS is stable (2025) – blog.jetbrains.com/kotlin/2025/05/compose-multiplatform-1-8-0-released-compose-multiplatform-for-ios-is-stable-and-production-ready/
- Strategic partnership with Spring (2025) – blog.jetbrains.com/kotlin/2025/05/strategic-partnership-with-spring/
- KotlinConf'26 keynote highlights – blog.jetbrains.com/kotlin/2026/05/kotlinconf26-keynote-highlights/
- Kotlin for Business – kotlinlang.org/lp/kotlin-for-business/

Case studies (cited in Section 2.2 and 4.5)

- Cash App – kotlinlang.org/case-studies/cash-app/
- ING – youtu.be/mexcjkGZlm8
- Philips – youtu.be/Qu3jZX8RyFk
- Duolingo – youtu.be/RJtiFt5pbfs
- Booking.com – <https://medium.com/booking-com-development/kotlin-multiplatform-in-production-two-real-world-use-cases-from-booking-com-46ffe13a773d>
- Sony – <https://youtu.be/VVf6txPZk3Y>
- Bilibili – <https://mp.weixin.qq.com/s/yRwkbQxFsRBNZW3Z-S1A8Q>
- Forbes: Forbes mobile app shifts to Kotlin Multiplatform – forbes.com/sites/forbes-engineering/2023/11/13/forbes-mobile-app-shifts-to-kotlin-multiplatform/
- McDonald's – youtu.be/uCkYZ-PvCmw
- Bolt – <https://youtu.be/Qu3jZX8RyFk?si=IRLKDXeZgTYfrJbR>