KushoAI

State of Agentic API Testing 2026

Benchmarks from 1.4 Million AI-Driven Test Executions Across 2,616 Organizations

2,616	64,459	1.4M
Organizations	API test suites	Test executions

Published by KushoAI - March 2026 (link)

KushoAI

Table of Contents

About KushoAI

KushoAI is building AI-native infrastructure for software maintenance - autonomous housekeeper agents that keep production systems clean, secure, and reliable. They embed within CI/CD pipelines and continuously test, heal, and monitor code so engineering teams can focus on new product work instead of repetitive maintenance.

KushoAI is used by engineering and QA teams across enterprises in fintech, infrastructure, and consulting.

To learn more or request a demo, visit kusho.ai or write to us at hello@kusho.ai.

Methodology

The findings in this report are drawn from telemetry collected across KushoAI's platform between January and December 2025. The dataset covers 1.4 million test executions across 2,616 organizations, spanning startups, growth-stage companies, and enterprises across industries including fintech, infrastructure, and enterprise software.

All data was collected passively through normal platform usage. No surveys were conducted.

Organizations are categorized by size based on self-reported team attributes at the time of account creation: startups (under 200 engineers), growth-stage (200–500), and enterprise (500+).

Failure classification follows KushoAI's internal taxonomy, which maps HTTP response codes and assertion outcomes to categories including authentication errors, schema violations, server-side failures, and timeout events.

All data used in this report is anonymized and aggregated. No organization-level or individual-level data is identifiable in any finding.

How to cite this report

KushoAI. State of Agentic API Testing 2026: Benchmarks from 1.4 Million AI-Driven Test Executions Across 2,616 Organizations. March 2026. reports.kusho.ai/state-of-agentic-api-testing-2026

Executive Summary

APIs have become a central component of modern software systems. As organizations build increasingly API-driven architectures, testing these APIs has become an important part of ensuring system reliability.

This report analyzes behavioral data from teams using KushoAI, an AI-native API testing platform used to generate and execute automated API tests. Because the platform operates directly within the testing workflow, it provides visibility into how engineering teams generate tests, run executions, structure workflows, and respond to failures across real systems.

Over time, this has produced a dataset that reflects actual testing activity rather than reported practices.

This report analyzes anonymized data from:

- **2,616 organizations**
- **64,459 API test suites**
- **1.4 million test executions**

The analysis is based on observed platform activity, including test generation patterns, execution frequency, workflow structures, and failure signals across different teams and environments.

Unlike survey-based reports, the insights presented here are derived directly from observed usage patterns and execution telemetry. The findings highlight several patterns in how organizations generate tests, structure API workflows, detect failures, and integrate testing into their development pipelines.

Four Structural Shifts

1.

API testing is moving from endpoint validation to workflow validation.

End-to-end API testing adoption grew 63% year over year. Multi-step workflows are replacing isolated endpoint checks as the dominant regression strategy.

2.

Schema drift is eroding test reliability.

41% of APIs experience undocumented schema changes within 30 days of initial test creation.

3.

AI-assisted test generation is collapsing test creation time.

AI-assisted generation reduces time from spec upload to a runnable test suite to approximately 4 minutes. Baseline coverage that previously required hours of manual authoring is now the starting point.

4.

AI-assisted testing is increasing edge-case coverage.

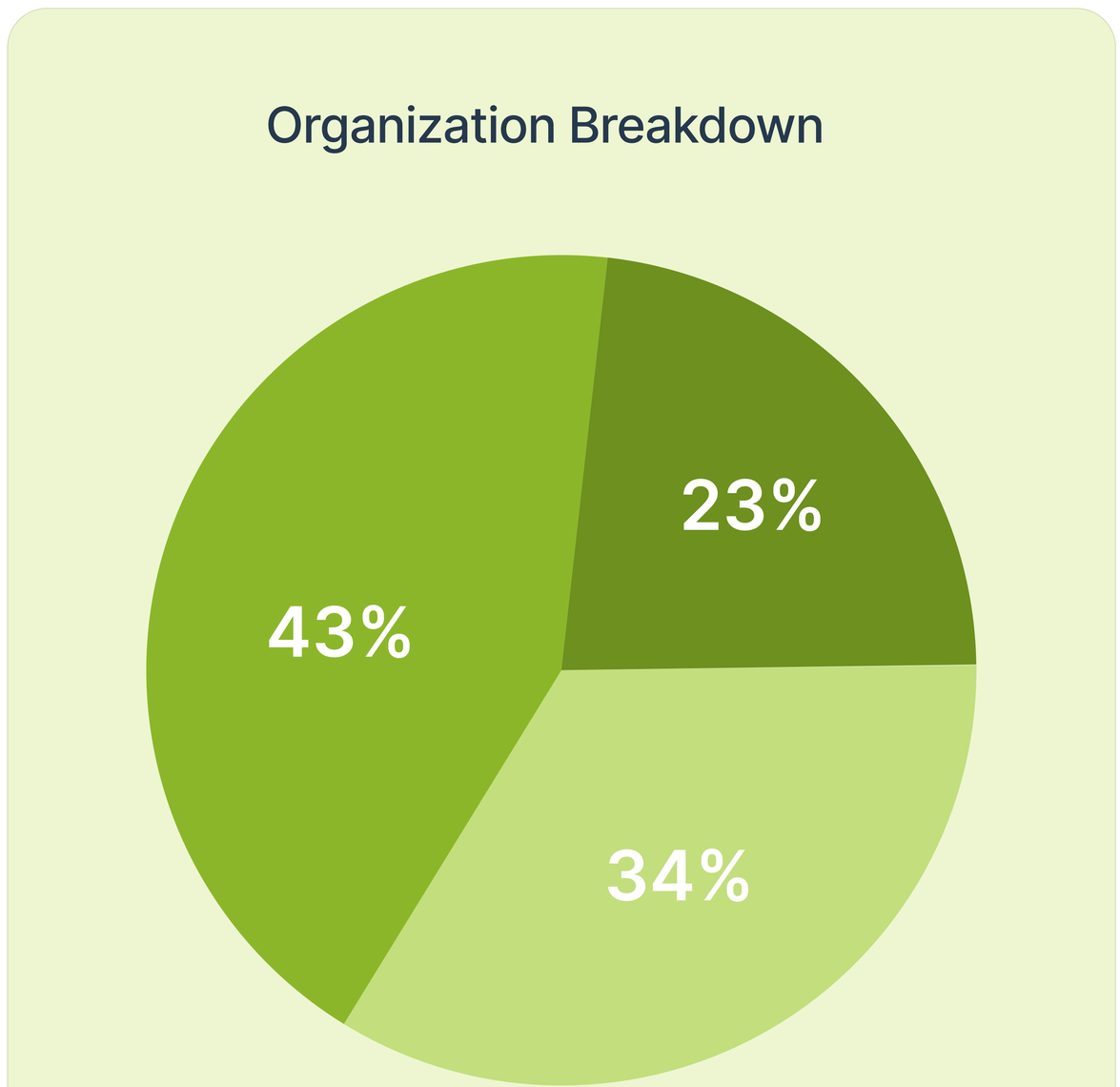
With baseline testing handled by AI, engineers can focus more on designing complex edge scenarios. At the same time, AI systematically explores parameter variations and boundary conditions that are often missed in manual testing, resulting in significantly broader edge-case coverage.

The data points to a clear conclusion: API testing is becoming the first truly autonomous layer of software quality control. AI handles breadth and continuous execution; engineers focus on depth and system-specific judgment.

Who Is Testing APIs

API testing is no longer limited to large enterprises. However, testing depth varies significantly by organization size.

Segment	% of Orgs
Startup	23%
Growth	34%
Enterprise	43%



Industry Patterns

Organizations in Fintech and SaaS exhibit the highest API testing activity and the strongest integration with CI/CD workflows, reflecting a culture of rapid release cycles and strong reliability requirements. In contrast, more traditional industries (e.g. healthcare, manufacturing) show significantly lower testing volumes and slower CI/CD adoption, indicating a lag in automated API quality practices.

APIs in Fintech and supply chain systems tend to have the most assertions per endpoint, suggesting stricter validation and monitoring of API behavior. This reflects the higher operational and financial risk associated with API failures in these domains, where even small regressions can have immediate downstream impact.

The API Landscape

Protocol Distribution

REST continues to dominate the API landscape, accounting for **76% of uploaded APIs** across the dataset. Its maturity, ecosystem support, and widespread tooling make it the default choice for many external and internal APIs.

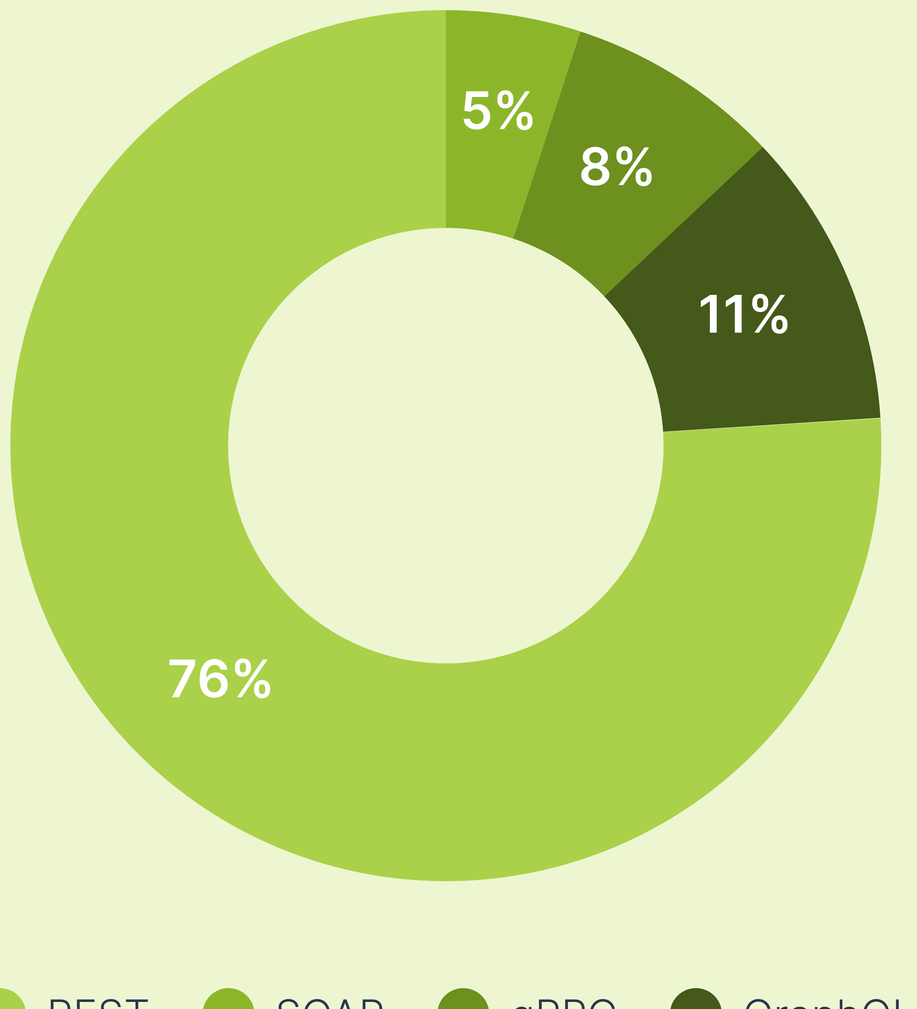
However, an important nuance emerges when looking at adoption patterns across organizations. While 76% of APIs are REST-based, **the proportion of organizations using REST is even higher.** Based on platform observations, nearly every organization in the dataset operates at least some REST APIs, even when they also use other protocols such as GraphQL or gRPC. In practice, this means **REST often acts as the foundational interoperability layer** within modern systems.

Growth trends also suggest a gradual diversification of API protocols. GraphQL and gRPC are increasingly adopted for specific architectural needs. GraphQL is commonly used in frontend-driven applications that benefit from flexible query structures, while gRPC is gaining traction for high-performance service-to-service communication in microservice environments. These newer protocols also show different reliability characteristics. GraphQL APIs exhibit **the highest average failure rate at 13.5%**, likely reflecting the complexity of resolver chains and schema dependencies. gRPC services show moderate failure rates and are typically deployed in tightly coupled internal systems. Meanwhile, SOAP APIs remain relatively stable but represent a smaller portion of the ecosystem, reflecting their continued use in legacy enterprise integrations.

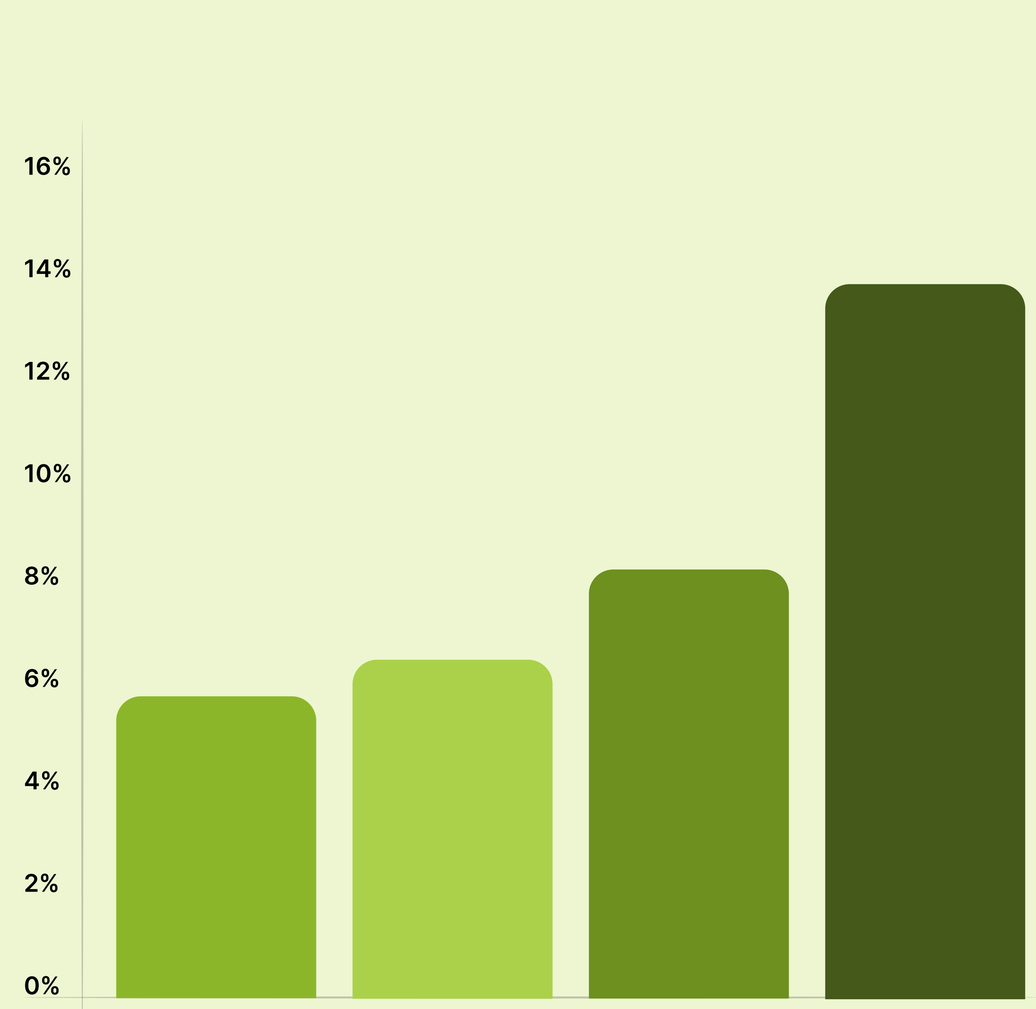
Taken together, the data suggests that **modern organizations are not choosing a single protocol**, **but operating multi-protocol API ecosystems**, with REST continuing to serve as the common foundation across systems.

Protocol	Share	Avg Failure Rate
REST	76%	6.4%
GraphQL	11%	13.5%
gRPC	8%	8.1%
SOAP	5%	5.7%

Protocol Share



Avg Failure Rate by Protocol



GraphQL's average failure rate is 2.1 times higher than REST.
72% of GraphQL regressions occur within nested response fields rather than status codes.

gRPC failures are primarily contract drift related, often triggered by protobuf evolution without synchronized client updates.

API Complexity Distribution

To understand how API complexity varies across systems, we analyzed APIs based on **the number of fields present in requests and responses**, using it as a simple proxy for structural complexity. APIs were grouped into three categories:

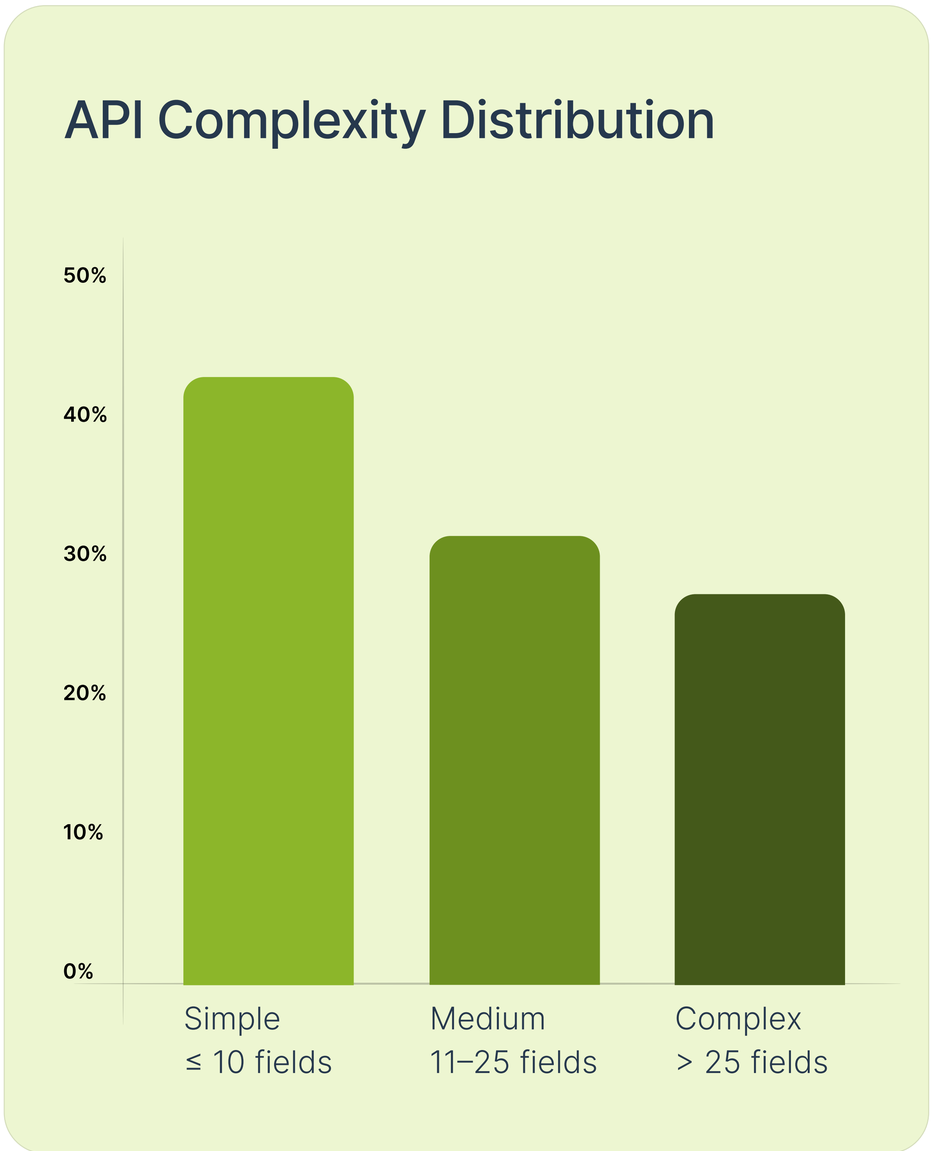
Complexity Level	Definition (approx.)	Share of APIs
Simple	≤ 10 fields	42%
Medium	11–25 fields	31%
Complex	> 25 fields	27%

As expected, **simpler APIs represent the largest share of the dataset**, typically corresponding to endpoints designed for narrow functionality such as single-resource retrieval, status checks, or small transactional operations.

However, an interesting pattern emerges when looking at the most complex APIs. **Highly complex APIs appear disproportionately in more traditional industries**, particularly domains such as Healthcare and Oil and Gas.

One likely explanation is **API evolution over long time horizons**. In many legacy systems, APIs are rarely redesigned from scratch. Instead, new fields and behaviors are incrementally added as products evolve and new requirements emerge. Over time, this accumulation of functionality leads to larger request and response schemas and increasingly complex API contracts.

As a result, some of the **oldest APIs in production environments tend to become the most structurally complex**, reflecting years of backward compatibility requirements and expanding system responsibilities.

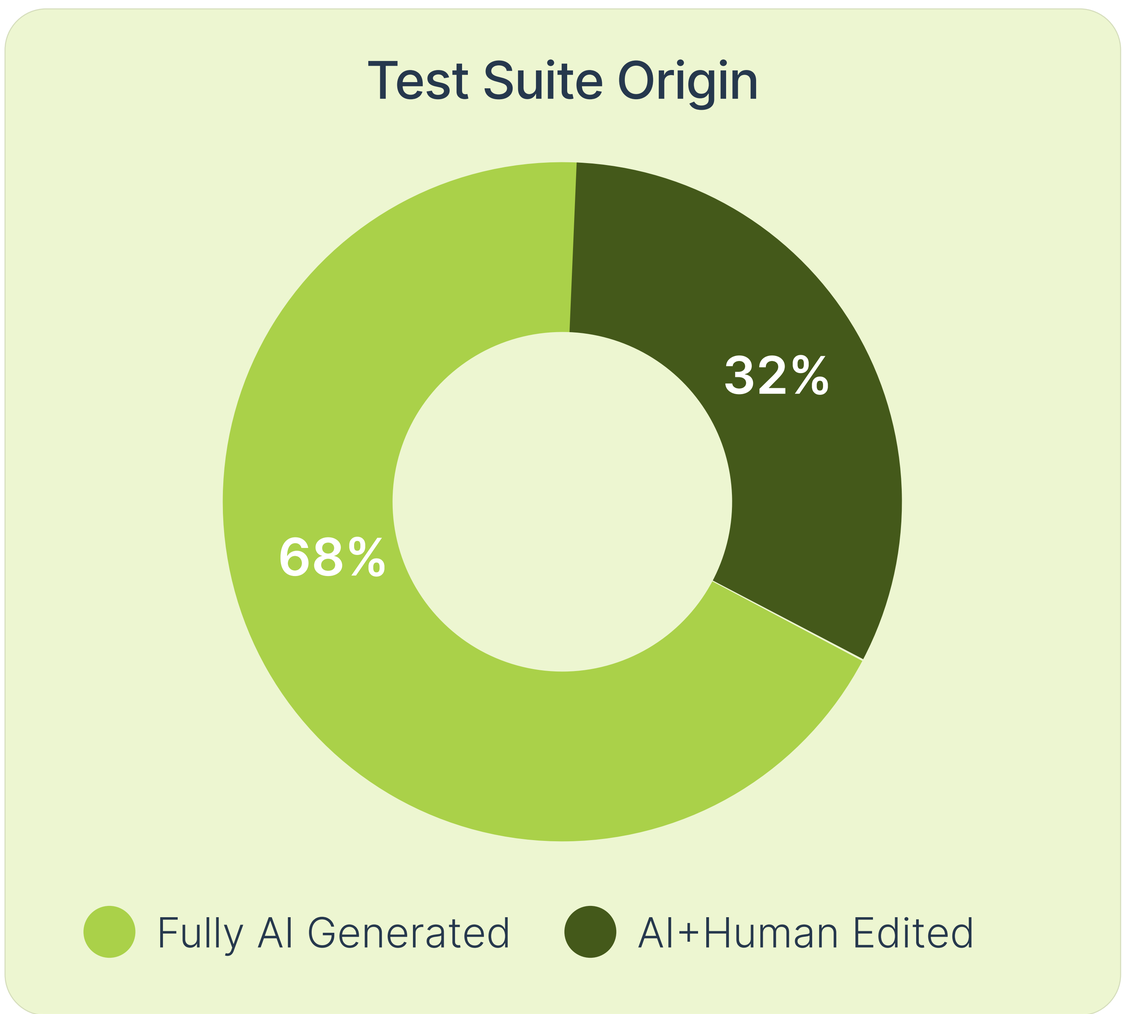


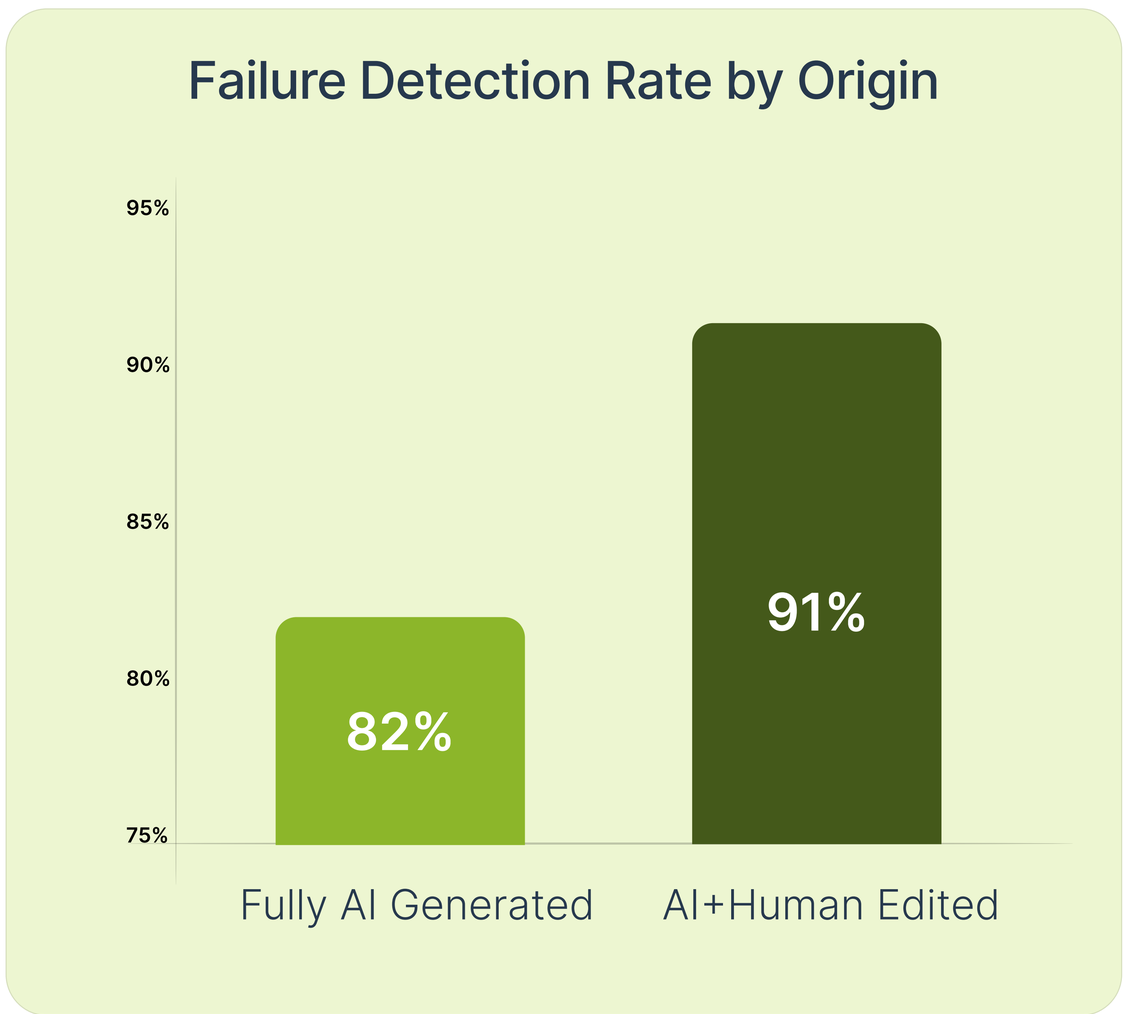
How Teams Are Testing

Test Creation Patterns

Because KushoAI is an **AI-native API testing platform**, test generation typically begins with an automated baseline. Most teams upload their API specifications or traffic definitions, after which the platform generates an initial test suite that can be executed immediately and optionally refined by engineers.

Origin	Share	Failure Detection Rate
Fully AI	68%	82%
AI + Human Edited	32%	91%





Failure detection rate measures the percentage of failing executions in which the test suite successfully surfaced a real behavioral issue in the API, such as schema violations, incorrect status codes, authentication failures, or response mismatches. The data suggests that suites refined by engineers perform slightly better at catching failures, likely due to the addition of domain-specific validations and edge scenarios.

Another pattern emerges when looking at **how engineers interact with AI-generated tests.** For **simple and medium-complexity APIs**, **teams rarely modify the generated suites**. In many cases, the automatically generated tests already provide sufficient baseline coverage, suggesting that AI can effectively handle a large portion of routine API testing.

For **more complex APIs, manual edits become more common**. However, these changes typically focus on adding specialized edge cases or domain-specific assertions, rather than rewriting the generated tests. This indicates a broader workflow shift: the time traditionally spent writing baseline tests is now being redirected toward designing higher-value scenarios that rely on engineering intuition or system-specific knowledge.

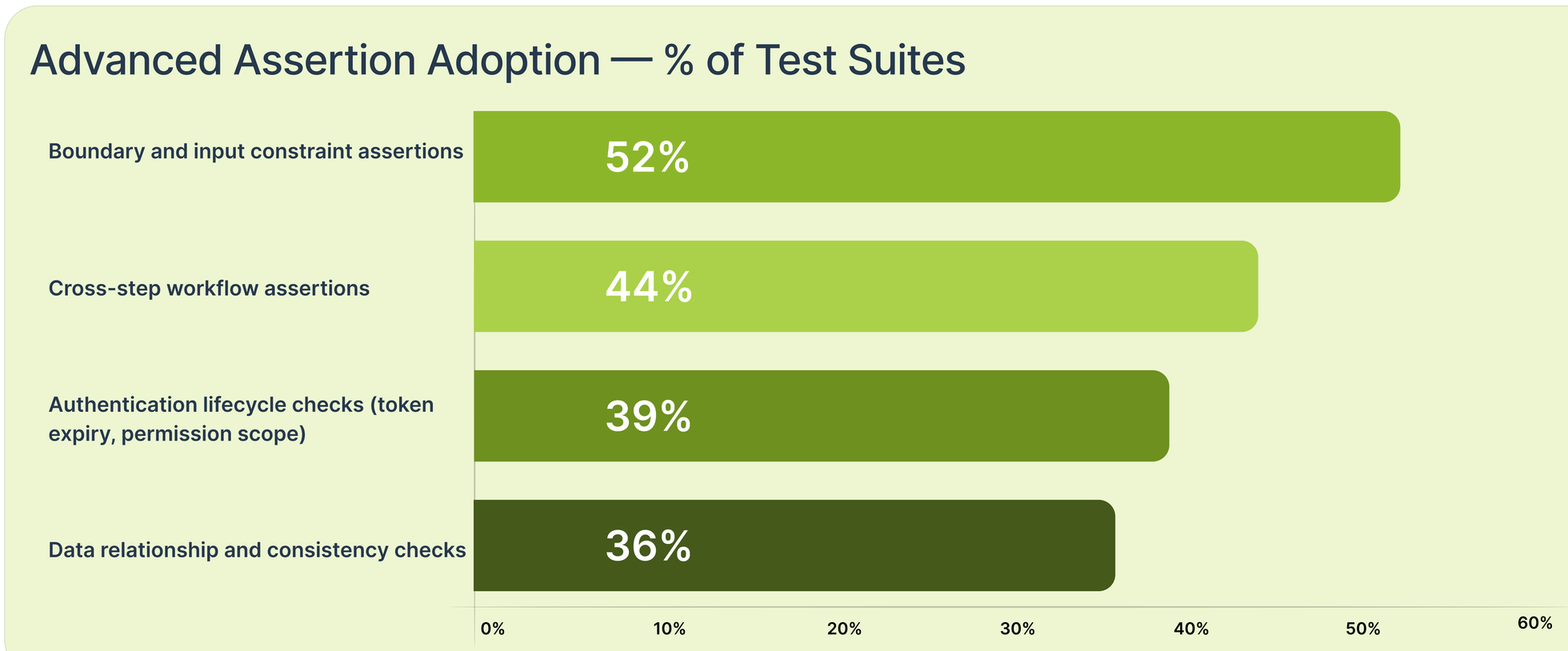
Overall, the data suggests that teams are increasingly comfortable relying on AI-generated tests for foundational coverage, using their saved bandwidth to strengthen testing around complex behaviors and edge conditions.

Assertion Maturity

KushoAI provides **baseline validation assertions out of the box**, including status code checks and schema validation. As a result, every generated test suite already includes fundamental correctness checks by default. This shifts the role of engineers from writing basic validations to **adding deeper behavioral assertions.**

Across the dataset, we observe that teams frequently extend these baseline suites with more advanced validations:

Assertion Type	Share of Test Suites
Boundary and input constraint assertions	52%
Authentication lifecycle checks (token expiry, permission scope)	39%
Cross-step workflow assertions	44%
Data relationship and consistency checks	36%



These percentages are **notably higher than what is typically seen in fully manual test suite**s, where much of the effort is spent writing basic success-path validations. Because KushoAI already generates the foundational checks, teams appear more willing to invest their time in **higher-value assertions** that validate system behavior under less obvious conditions.

This trend aligns with established testing practices. Techniques such as **Boundary Value Analysis** focus on validating behavior at the limits of input ranges because defects frequently appear at those boundaries rather than within normal operating ranges.

In practice, AI-generated baseline assertions appear **to raise the floor of testing maturity.** Instead of spending time writing repetitive checks like status codes or schema validation, engineers can focus their effort on assertions that validate system workflows, security behavior, and domain-specific edge conditions.

The Rise of End-to-End API Testing

As API-driven architectures have matured, the limitations of single-endpoint testing have become increasingly apparent. Validating individual APIs in isolation tells you whether an endpoint responds correctly but it does not tell you whether the system works. KushoAI was built on the premise that the more meaningful question is whether complete backend workflows behave correctly end-to-end, across services, state transitions, and integration boundaries. The data from engineering teams using the platform confirms this direction.

In discussions with enterprise and growth-stage teams, an estimated **80% expressed the need for multi-step API testing.** The motivation was consistent across organizations: while UI automation is useful, it often requires significant setup, maintenance, and infrastructure.

In contrast, API-level testing can validate the same underlying system behavior with far less operational overhead. For many teams, multi-step API testing emerged as a **practical middle ground between single-endpoint tests and full UI automation suites.**

Following the release of **end-to-end API workflow testing** roughly a year ago, adoption accelerated quickly.

- **58% of organizations,** now use multi-step API workflows
- **84% of enterprise teams**, have adopted the feature
- **Over 11,200 API workflows,** have been automated on the platform
- Teams execute an average of **~50 workflow runs per week**

58%

of all organizations use multi-step API workflows

84%

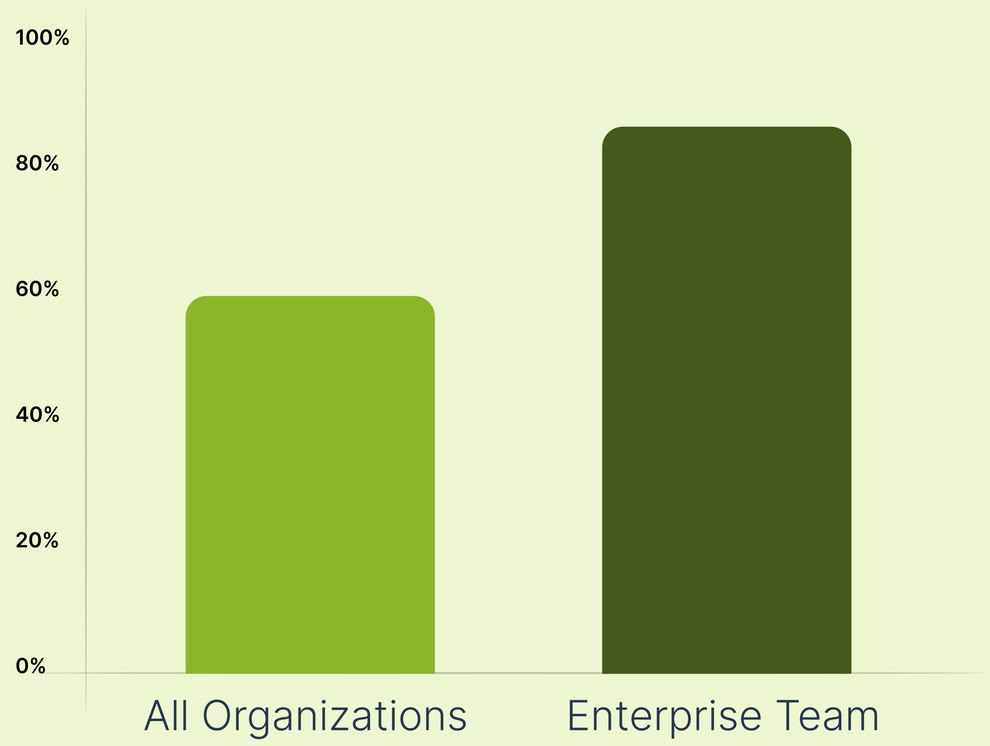
of enterprise teams have adopted E2E workflow testing

The majority of these workflows originate from **growth-stage and enterprise companies,** where backend systems often span multiple services and integrations.

These workflows are increasingly being integrated directly into CI pipelines and release processes.

Instead of relying solely on UI automation for regression testing, many teams now use **API workflows as deployment gates,** validating that critical backend flows continue to function correctly before changes reach production.

E2E Workflow Adoption by Segment



This shift reflects a broader evolution in testing strategy. As modern systems become more API-driven, **end-to-end API workflows are emerging as a scalable alternative to UI-heavy automation,** offering strong failure detection while remaining easier to maintain and faster to execute.

Execution & CI/CD Behavior

Execution behavior across organizations shows a clear divide between different stages of engineering maturity.

Growth-stage and enterprise teams are far more likely to integrate API testing directly into their **CI/CD pipelines**, using automated test runs as part of their deployment and regression validation process.

In contrast, **startups and smaller teams** more commonly execute tests manually, even when those tests are AI-generated. For these teams, automated test suites are often used as on-demand validation tools rather than continuously running safeguards in the delivery pipeline.

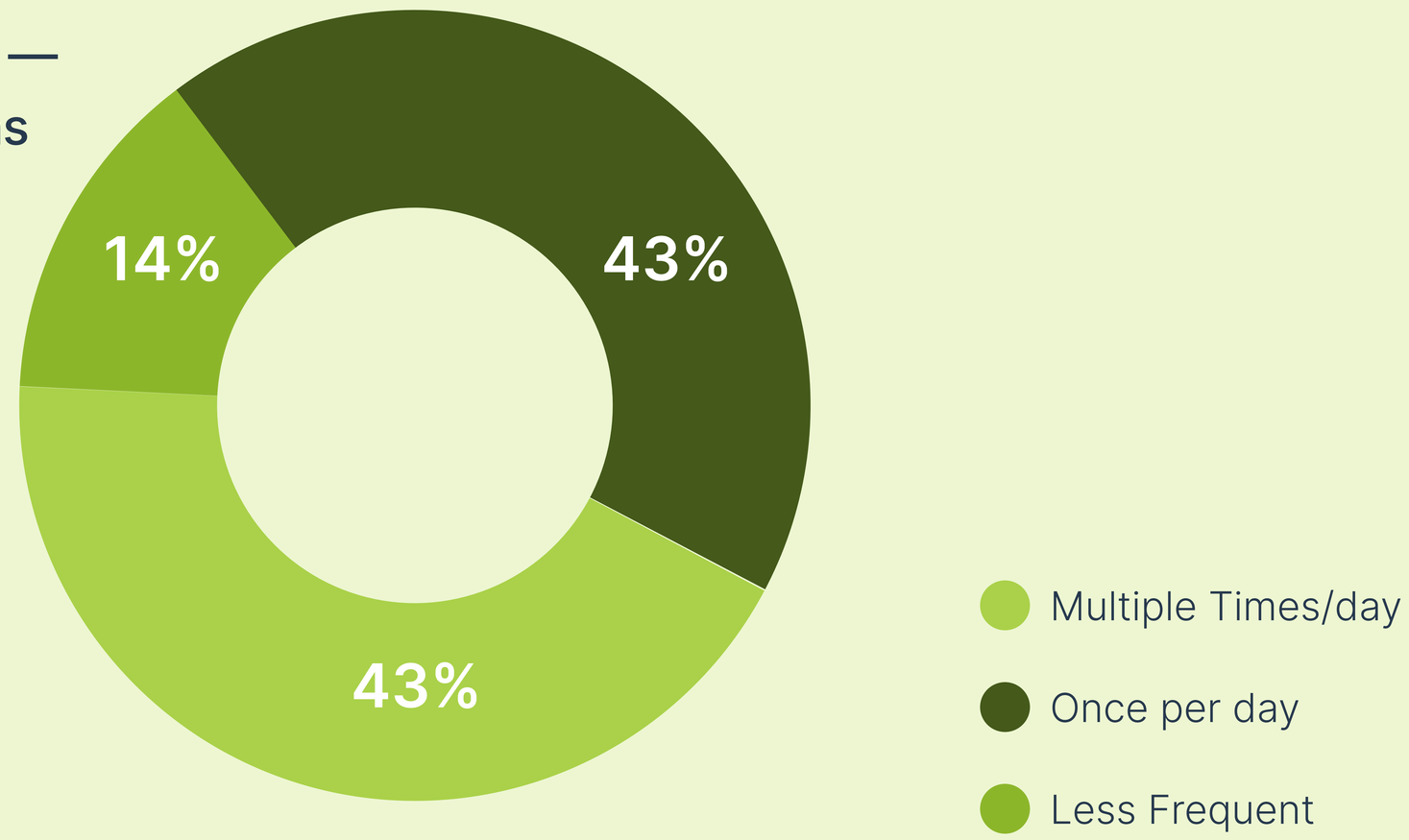
Another noticeable trend is the **increasing automation of multi-step workflows**. As organizations adopt end-to-end API testing, many teams are beginning to automate complete backend flows rather than testing individual APIs in isolation.

In practice, these API workflows are increasingly used as a **lightweight alternative to UI test automation**, validating critical product flows without the complexity and maintenance overhead associated with UI testing frameworks.

Among organizations that have integrated KushoAI into their CI/CD pipelines, roughly **86% run at least one automated test execution per day**, with nearly half of these teams executing tests multiple times daily.

This pattern suggests that API tests are increasingly being treated as **continuous regression gates**, providing fast feedback before code changes move further into the deployment pipeline.

Daily Execution Frequency — CI-Integrated Organizations



Overall, the data indicates a broader shift: as backend systems become more API-driven, **automated API workflows are becoming a central layer in modern CI/CD validation strategies**, often complementing or partially replacing traditional UI-heavy testing setups.

Failure Intelligence

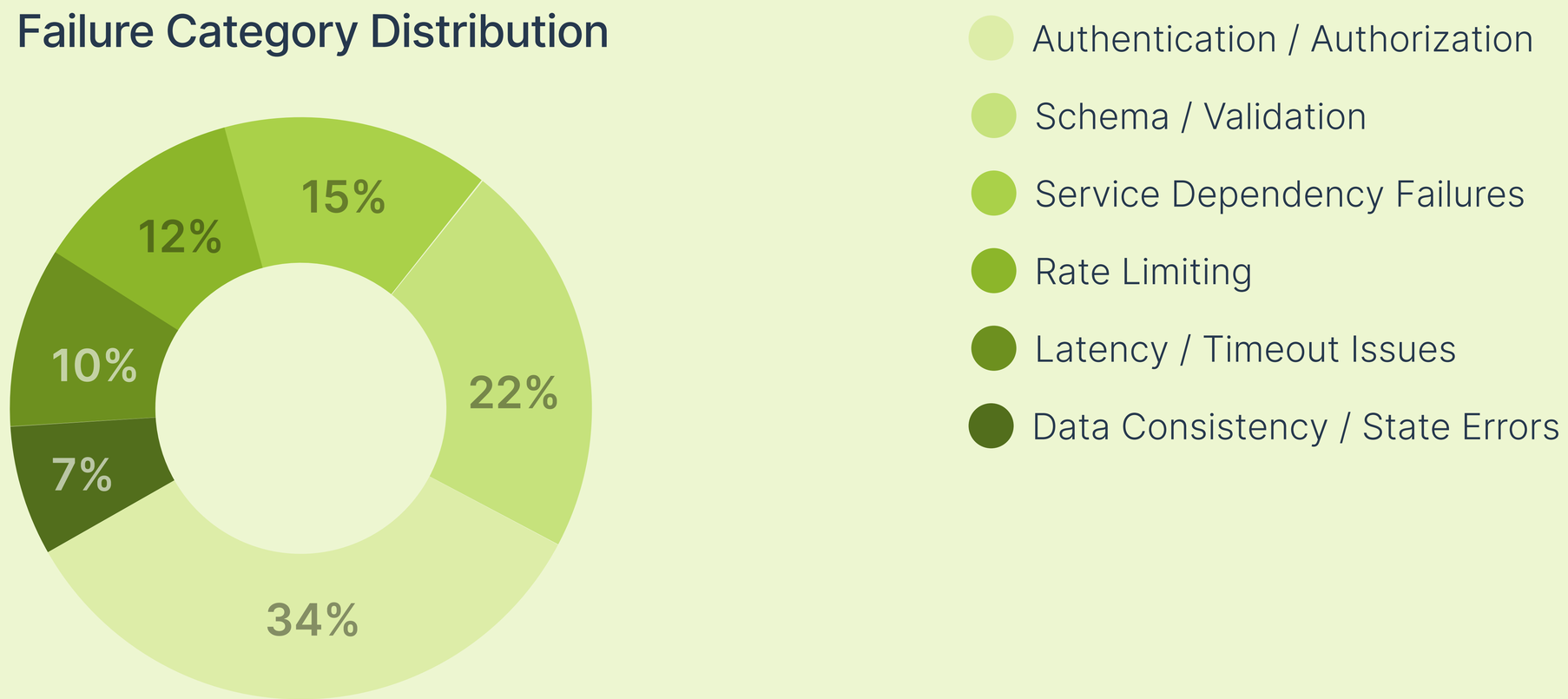
Distribution of Failures

The most common assumption about API failures is that they are server-side problems: crashes, timeouts, and 5xx errors that point to something broken in the backend. The data in this dataset tells a different story. Authentication and authorization issues account for 34% of all observed failures. Schema and validation errors account for another 22%. Combined, client-side and contract-related failures represent the majority of API regressions, while 5xx server errors account for less than 10%. Most API failures are not the system going down. They are the system behaving in ways that were not anticipated, validated, or caught before reaching production.

The following data reflects failures observed during API test executions across organizations using KushoAI. Each failure corresponds to an execution where the observed API behavior deviated from the expected assertions defined in the test suite.

Because KushoAI interacts with APIs as a black-box testing system, the exact internal cause of a failure is not always directly visible. Instead, failure categories are approximated using observable signals, including response payloads, error messages, authentication responses, validation errors, and behavioral patterns across a large sample of executions. Based on these signals, failures can be broadly grouped into the following categories.

Failure Category	Share of Failures
Authentication / Authorization	34%
Schema / Validation	22%
Service Dependency Failures	15%
Rate Limiting	12%
Latency / Timeout Issues	10%
Data Consistency / State Errors	7%



Authentication and authorization issues represent the largest category of failures, accounting for roughly one-third of all observed regressions. These failures commonly arise from expired tokens, misconfigured permission scopes, incorrect authentication headers, or changes in access control rules.

Schema and validation failures form the second largest category. These typically occur when APIs return responses that no longer match the expected contract, such as missing fields, type mismatches, or unexpected payload structures. Such failures often surface when backend changes are deployed without corresponding updates to API contracts.

Most failures originate from client-side interactions, validation rules, or system dependencies, rather than purely internal server errors. This suggests that many production-impacting issues arise from integration behavior and contract mismatches, rather than catastrophic backend failures alone.

Schema Drift

One recurring pattern across the dataset is **schema drift**, where the behavior or structure of an API changes but the corresponding API schema and test definitions are not updated.

In practice, this typically occurs when developers modify an API response or request structure, for example by adding a new field or changing a data type, without updating the API specification or associated tests. As a result, tests that previously passed begin to fail even though the underlying system change may have been intentional.

Using execution telemetry, KushoAI approximates schema drift by detecting patterns where **previously passing APIs begin to fail,** followed shortly by observable changes in response structures or schema definitions. Based on these signals, we observed the following drift frequency:

- **41 percent of APIs experience schema drift within 30 days**
- **63 percent experience schema drift within 90 days**

41%

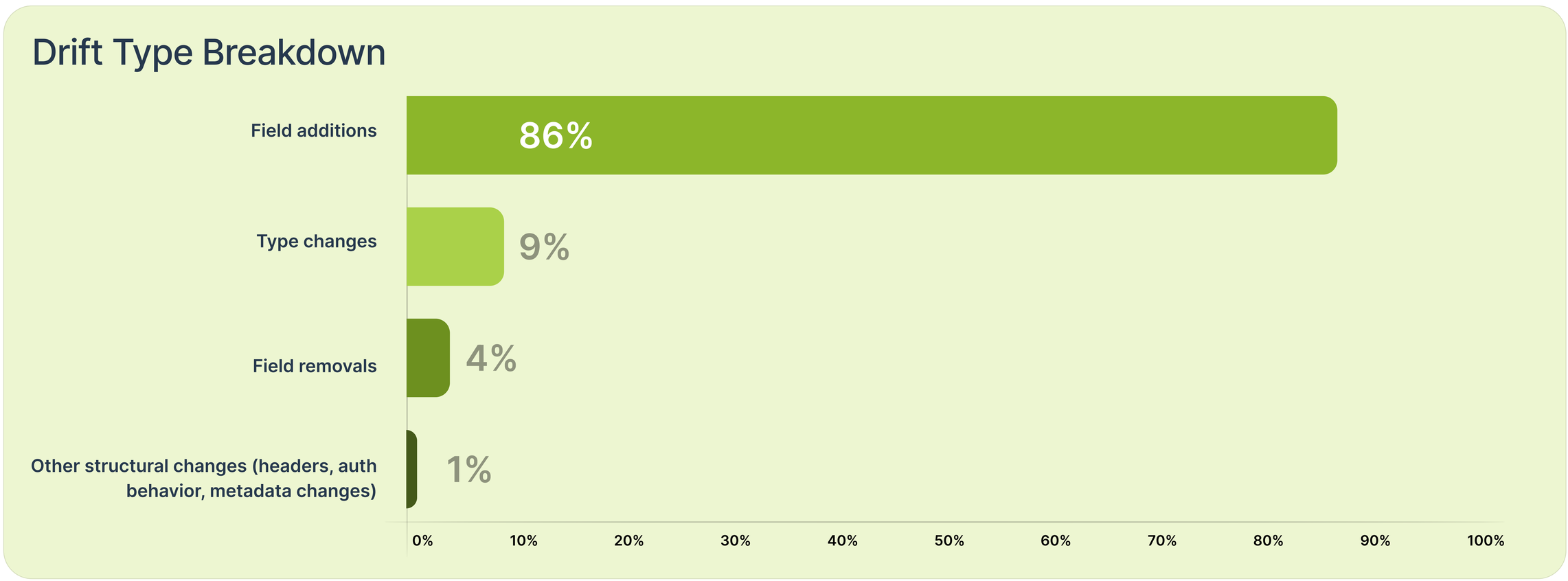
experience drift within 30 days

63%

experience drift within 90 days

Among detected drift events, the most common structural changes include:

Drift Type	Share
Field additions	86%
Type changes	9%
Field removals	4%
Other structural changes (headers, auth behavior, metadata changes)	1%



Field additions represent the majority of drift events, followed by type changes within existing fields. Field removals occur less frequently, while other structural changes such as authentication behavior changes, new headers, or metadata updates account for the smallest share.

An important behavioral pattern also emerged. In most cases, **schema drift is detected reactively**, after test failures begin appearing in CI pipelines or execution logs. In other words, the schema change happens first, and test suites are updated only after failures surface.

This reactive pattern highlights a gap in many API testing workflows. Without automated drift monitoring, schema changes propagate silently until they begin to break existing tests or integrations. Platforms that can automatically detect schema changes and adjust validation rules proactively have the potential to significantly reduce this form of test instability.

AI's Impact on API Testing

The shift from manual to AI-driven API testing is not incremental. Across the organizations in this dataset, the data points to a fundamental change in how testing effort is structured, and what engineers actually spend their time on.

1. **Test creation time has compressed significantly.**

Traditionally, writing a comprehensive API test suite meant manually authoring request permutations, assertions, and authentication logic, a process that routinely took hours or days depending on API complexity. **Across organizations using KushoAI, the average time from spec upload to a fully runnable test suite is now approximately 4 minutes.** Baseline coverage that once required a dedicated sprint task is now a starting point.

2. Coverage breadth has increased.

AI-generated suites go beyond what engineers would have written and systematically explore parameter spaces, boundary conditions, and authentication edge cases that manual authors routinely skip. This shows up in the assertion data: 52% of suites include boundary condition tests, 44% include cross-step workflow assertions, and 39% explicitly validate authentication lifecycle behavior, all significantly higher than what is typically seen in manually authored suites.

3. The role of QA engineers is shifting.

Rather than writing foundational tests, engineers using AI-assisted workflows are spending their time on scenarios that require genuine system knowledge: domain-specific edge cases, multi-step state validation, and complex failure mode handling. AI-generated suites that are human-edited achieve a 91% failure detection rate, compared to 82% for fully automated suites. Instead of writing foundational tests, engineers are adding **domain-specific edge cases, multi-step validation logic, and complex workflow assertions that AI alone cannot infer.**

The primary advantage of AI in testing is therefore **not speed alone, but coverage breadth and better allocation of engineering effort.** By automating the repetitive portions of test creation, AI enables QA teams to concentrate on the complex scenarios that are most likely to surface production issues.

What this creates in practice is testing that runs continuously with minimal human intervention. For API testing specifically, this level of autonomy is now within reach. The same cannot yet be said for UI testing, where human judgment remains a practical necessity.

Looking Ahead

The patterns in this dataset point to a clear directional shift: API testing is evolving from a discrete engineering activity into a continuously running quality layer embedded directly in the delivery pipeline.

Several trends are likely to accelerate this transition.

API-level end-to-end testing will increasingly displace UI regression automation.
As backend systems become more API-driven, teams are discovering that API workflows can validate the same critical paths as UI tests with significantly less setup and maintenance overhead. We expect API-level automation to absorb a growing share of regression coverage that UI testing currently handles.

AI-assisted test generation will become the default.
Within growth-stage and enterprise organizations, manually authored baseline tests will become the exception rather than the norm. The question will shift from whether to use AI for test creation to how much of the testing workflow to automate end-to-end.

Schema drift monitoring will move from reactive to proactive.
Currently, most teams discover schema drift only after failures appear in CI pipelines. As drift detection matures, the expectation will shift toward systems that surface contract changes before they break downstream tests or integrations.

Business logic validation will overtake schema-only validation as the primary measure of test quality.
With foundational assertions increasingly automated, engineering effort will concentrate on validating system behavior under complex conditions: state dependencies, cross-service logic, and failure mode handling.

GraphQL-specific tooling will mature rapidly.
Its embedded failure rate of 13.5%, more than double that of REST, indicates significant unmet tooling demand, particularly around resolver validation and nested schema assertion.

Conclusion

API testing is undergoing a structural transition. For most of software's history, API testing was treated as a supporting activity, useful but secondary to UI automation and manual QA. The data in this report suggests that characterization is becoming obsolete.

As systems grow more API-driven, the API layer is becoming the most reliable place to validate system behavior at scale. API tests are faster to write, cheaper to maintain, and more stable than their UI counterparts. They can validate complete backend workflows without the brittleness of browser-based automation. And with AI-assisted generation now handling baseline coverage automatically, the barrier to comprehensive API testing has dropped significantly.

What this creates, for the first time, is the conditions for truly autonomous testing at the API layer. AI agents can generate tests, execute them continuously, detect schema drift, and surface failures with minimal human intervention. This is not yet the reality for most organizations, but the teams at the leading edge of this dataset are getting close.

UI testing, by contrast, still requires meaningful human involvement. Selectors break. Flows change. Visual context matters in ways that are difficult to automate reliably. The human-in-the-loop remains a practical necessity for UI automation in a way it no longer is for API testing.

The teams with the lowest regression rates in this dataset share three characteristics: they test write operations as rigorously as read operations, they validate failure responses as thoroughly as success responses, and they treat schema drift as a continuous monitoring problem rather than a periodic maintenance task.

But the broader pattern is simpler than any individual practice. The teams that are pulling ahead have stopped treating API testing as a task to complete and started treating it as a system to run.