The American Journal of Applied Sciences
ISSN 2689-0992 | Open Access

TYPE Original Research
PAGE NO. 39-47
DOI 10.37547/tajas/Volume07Issue07-05

Swagger/OpenAPI Specification as a Governance Tool for Internal Data Products: Enabling Standardization, Transparency, and Control

Purva Desai
Data Analyst, USA

Sahil Fruitwala
Software Engineer, USA

OPEN ACCESS

SUBMITED 14 May 2025
ACCEPTED 29 June 2025
PUBLISHED 11 July 2025
VOLUME Vol.07 Issue 07 2025

CITATION

Purva Desai, & Sahil Fruitwala. (2025). Swagger/OpenAPI Specification as a Governance Tool for Internal Data Products: Enabling Standardization, Transparency, and Control. The American Journal of Applied Sciences, 7(07), 39–47. https://doi.org/10.37547/tajas/Volume07Issue07-05

Abstract: Modern businesses increasingly rely on internal data products, such as curated datasets or analytical services, to drive innovation and informed decisions. Despite substantial investments in data technologies, including a global Artificial Intelligence market valued at $230 to $280 billion in 2024, large organizations struggle with inconsistent API interfaces. This inconsistency hinders efficient data exchange and robust governance. This paper tackles this challenge by proposing a framework for mandatory OpenAPI Specification (OAS) adoption and automated enforcement for all internal data products. Our approach defines clear organizational standards and implements a twostep compliance checking mechanism. This involves Static Type Analysis (STA) for foundational rule enforcement and an AI agent for nuanced, contextual validation. Integrated within CI/CD pipelines, this automated system ensures continuous adherence to design standards, enhancing data product discoverability, interoperability, and overall data governance. This work provides a practical methodology for establishing standardized control over internal data product APIs, streamlining development, and fostering a resilient data ecosystem.

Keywords: OpenAPI Specification, Swagger, API Governance, Data Products, Standardization

I. Introduction:

The unimaginable adoption of data-driven strategies has created an exponential growth in data products creation and consumption across all modern enterprises. Data collected from users, from market research, from sales or from even external sources are foundation for innovation and strategic data-driven decision making. The increasing investment in AI and related data technologies such as data ware-houses highlights its importance, with the global Artificial Intelligence market alone valued at approximately $230-$280 billion in 2024, exhibiting significant growth from previous years [1]. However, this rapid advancement has introduced a critical challenge: large organizations struggle to establish cohesive, secure, and centrally governed solutions for sharing and consuming internal data products for effectively communicating and consuming these internal data products [2], [3]. While data collection and processing are vital, the efficacy of internal data exchange between teams is equally crucial for benefiting the value of these investments.

Modern systems are interconnected with each other in many way but, most common way how these systems communicate is through Application Programming Interfaces (APIs) or specifically Representational State Transfer (REST) APIs. Since RESTful APIs were introduced, it has seen sky rocketing growth and became a dominant paradigm for web APIs due to its simplicity, scalability, and alignment with HTTP protocols. Despite the widespread adoption and being the backbone of software architectures REST APIs face persistent challenges. Many organizations struggle with the standardization of RESTful APIs, particularly when they encapsulate data products [4], [5]. This lack of standardization frequently results into inconsistency in data representation, significant integration bottlenecks due to disparate interfaces, and heightened security and compliance risks across teams and systems [2], [6]. While REST APIs leverage standard protocols like Hypertext Transfer Protocol (HTTP) for communication, they do not mandate any specific guidelines or governance regulations for how an API should be designed. In other terms, REST APIs are not opinionated. They do not have any specific data structure, error handling, or naming conventions defined nor how to distribute them. This absence of a standardized rules is particularly problematic for data products, where clear and predictable interfaces are essential for efficient consumption and trustworthy exchange. Existing methods like SODA (Service Oriented Detection for Antipatterns) aim to identify antipatterns in Service-Based Systems but suffer from significant gaps, such as reliance on manual rule definitions, limited scalability, and a focus on post-deployment detection rather than real-time validation. Our proposed framework fills these gaps by leveraging automated tools and seamless integration into developer workflows, ensuring consistent adherence to OpenAPI specifications and addressing the critical need for standardized, secure, and centrally governed internal data products [15].

Many data products in organizations improve decision-making but raises challenges in finding, integrating, and managing them due to inconsistent and APIs [3], [4]. This paper proposes a comprehensive framework to address this critical issue through the mandatory adoption and automated enforcement of OpenAPI specifications for all data products. Similar existing approach such as Semantic Analysis of RESTful APIs (SARA) lacks the speed and fast response time as it could only be done post-deployment. SARA misses structural and type safety validation as well as validation for security and standard compliance [14]. Our solution begins with defining clear organizational OpenAPI standards, which promise substantial benefits including improved data product interoperability, simplified consumption, and strengthened data governance [7]. We laid out two primary methods for achieving automated compliance. First Static Type Analysis for precise structural validation, and second developing an AI agent capable of enforcing more nuanced, contextual organizational guidelines. Both approaches are designed for seamless integration into existing developer's workflow such as precommit hooks and other in CI/CD pipelines, enabling continuous and automated validation of data product APIs within each repository. This automated enforcement ensures consistent adherence to defined specifications, reducing integration overhead, and fortifying the organization's data ecosystem.

Proposed framework addresses challenges by advocating for the mandatory adoption and automated

enforcement of Open API Specification (OAS) for all internal data products. Paper will demonstrate how leveraging automated tools and integration of these tools in developer's workflow can achieve robust validation and reduce miscommunication between data producers and consumers. The core contribution of this work lies in outlining a practical methodology to establish standardized control over internal data product APIs, thereby significantly enhancing standardization, transparency, interoperability, and overall data governance within large organizations.

II. BACKGROUND AND RELATED WORK

The OpenAPI Specification (OAS), formerly known as Swagger, is a widely adopted industry standard for defining and documenting RESTful APIs [8]. It uses a machine-readable format, typically JSON or YAML, to precisely describe an API's operations, parameters, responses, and data models. This machine-readable nature is crucial as it enables not only human understanding of API design but also for automated tools to perform tasks such as generating client software development kits (SDKs), creating server stubs, and facilitating automated testing.

When it comes to internal data products, which are curated data into services for a company to use, OAS helps solve big problems with complex APIs. Companies often face issues like messy API designs, scattered documentation, and weak control over who can access what or follow rules. These problems make it hard to grow and slow down developers. For example, not having standard formats can lead to teams creating the same data models multiple times, and not knowing how APIs are used can increase the risk of security issues or breaking rules. OAS fixes this by providing a clear way to create consistent API designs, make detailed documentation, and work with tools like API gateways (such as AWS API Gateway or Apigee) to enforce rules and track usage. This organized approach helps turn scattered data into data products that are easy to find, use, and manage.

III. PROPOSED FRAMEWORK / METHODOLOGY

Every organization possesses unique operational needs and approaches to standardization. However, to effectively leverage the OpenAPI Specification (OAS) for its internal data products, a precise set of organizational standards must first be collaboratively devised. These standards must be formulated in a manner that ensures clarity, developer understanding, and broad consensus across teams. While individual teams within an organization may utilize varied technology stacks

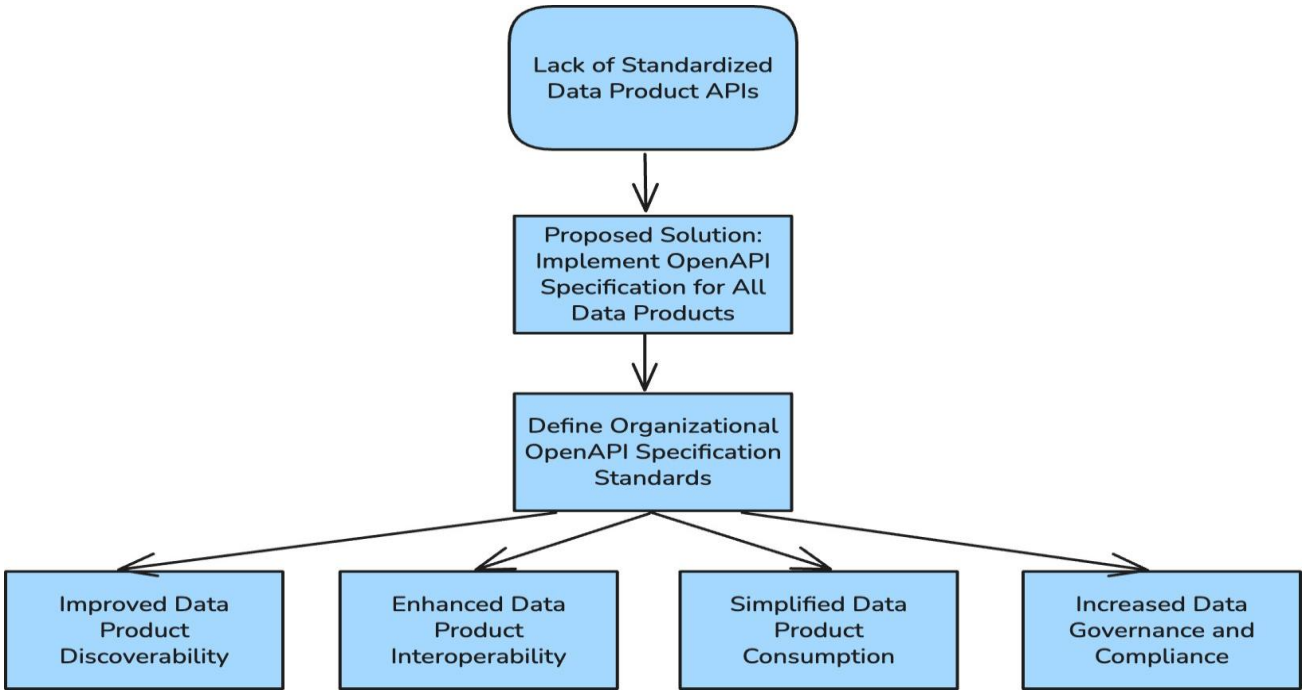

Fig. 1. Process of Defining Standard Policies and Compliance

and frameworks for developing RESTful APIs, the foundational API design for data products must remain consistent. Therefore, the initial step in defining these OpenAPI standards necessitates robust cross-functional collaboration, typically involving leaders and subject matter experts from various internal data product and engineering teams. This collaborative process is crucial for establishing a definitive scope of API specification requirements, outlining both mandatory inclusions and prohibited elements.

Once a commitment to standardization for all internal data products is established, the subsequent phase focuses on formalizing these comprehensive OpenAPI standards (as shown in Figure 1). This requires meticulously addressing several key considerations to ensure consistency, promote collaboration, and achieve alignment across the entire enterprise. The resulting OpenAPI specifications must be meticulously tailored to the overall needs of the organization, rather than being confined to the specific needs of individual teams or projects.

A. ***Standardization of API Design Rules***

- Schema Consistency: Agree on reusable schemas for common data models (e.g., customer, order, product) to prevent redundant or conflicting definitions across teams. Define naming conventions (e.g., *camelCase*, *snake_case*) and data types (e.g., ISO 8601 for dates).

- Endpoint Structure: Establish uniform patterns for endpoints (e.g., */resources/*{*id*} for RESTful APIs) and HTTP methods (e.g., GET, POST, PUT, DELETE) to ensure predictable API designs.

- Versioning Strategy: Decide how to handle API versioning within OpenAPI specs (e.g., using URL paths like */v1/resource* or headers) to maintain compatibility as APIs evolve.

Style Guides: Create an organization-wide OpenAPI style guide, specifying standards for parameters, error responses (e.g., consistent HTTP status codes like 400, 404), and documentation tags.

B. ***Checking API Rules:*** *Two Ways*

Our framework uses two main steps to check if an API follows the rules: Static Type Analysis (STA) and an AI Agent. Both are designed to make sure OpenAPI specifications meet our company's standards. Each method has its own strengths. A big plus of this two-step process is that STA can be run right on a developer's computer. This means they don't have to wait for the main CI/CD pipeline for some basic checks.

1) Step 1 - Static Type Analysis: The first step, Static Type Analysis (STA) (as shown in Figure 2), validates OpenAPI specifications clearly using a programmatic approach. This involves creating a special tool just for checking OpenAPI compliance.

- What it does: The STA tool carefully looks at the OpenAPI specification file (usually a YAML or JSON file) without actually running the API. Its main job is to strictly enforce the clear, defined rules that come straight from the company's OpenAPI standards.
- How it works: This tool is designed to find problems related to:
  - Syntax and Structure: It makes sure that OpenAPI document itself is correctly written or generated according to the OpenAPI Specification. For example, it checks if all required fields are there and if data types are correct.
  - Naming: It verifies that names for paths, operations, parameters, and data properties follow the company's rules. Such as using *camelCase* for fields or *snake_case* for paths.
  - Mandatory Elements: It checks for specific required parts of the API, consistent ways to show errors, and correct security settings.
  - Style: It ensures the API follows the defined Ope- nAPI style guide for documentation, tags, and overall presentation [11].
- Why it's good: STA gives quick, clear, and easy-to-understand results. This makes it perfect for finding common and obvious errors early in the development process.

2) Step 2 - AI Agent: The second step, the AI Agent (also in Figure 2), handles more complex, context-based, or somewhat subjective parts of API compliance that a simple rule-based checker might miss. This process starts by" Training the AI Agent with Organizational API Standards."

- What it does: The AI Agent learns to understand complex patterns and unwritten best practices found in the com- pany's API standard documents and from past examples of good and bad APIs. Its goal is to give smart feedback on areas where an API might not meet desired quality or consistency, even if it passes the basic STA checks.
- How it works:

Training: The AI Agent learns from different types of data. This includes official company API design guides and policy documents. It also uses examples of OpenAPI specifications that people have reviewed

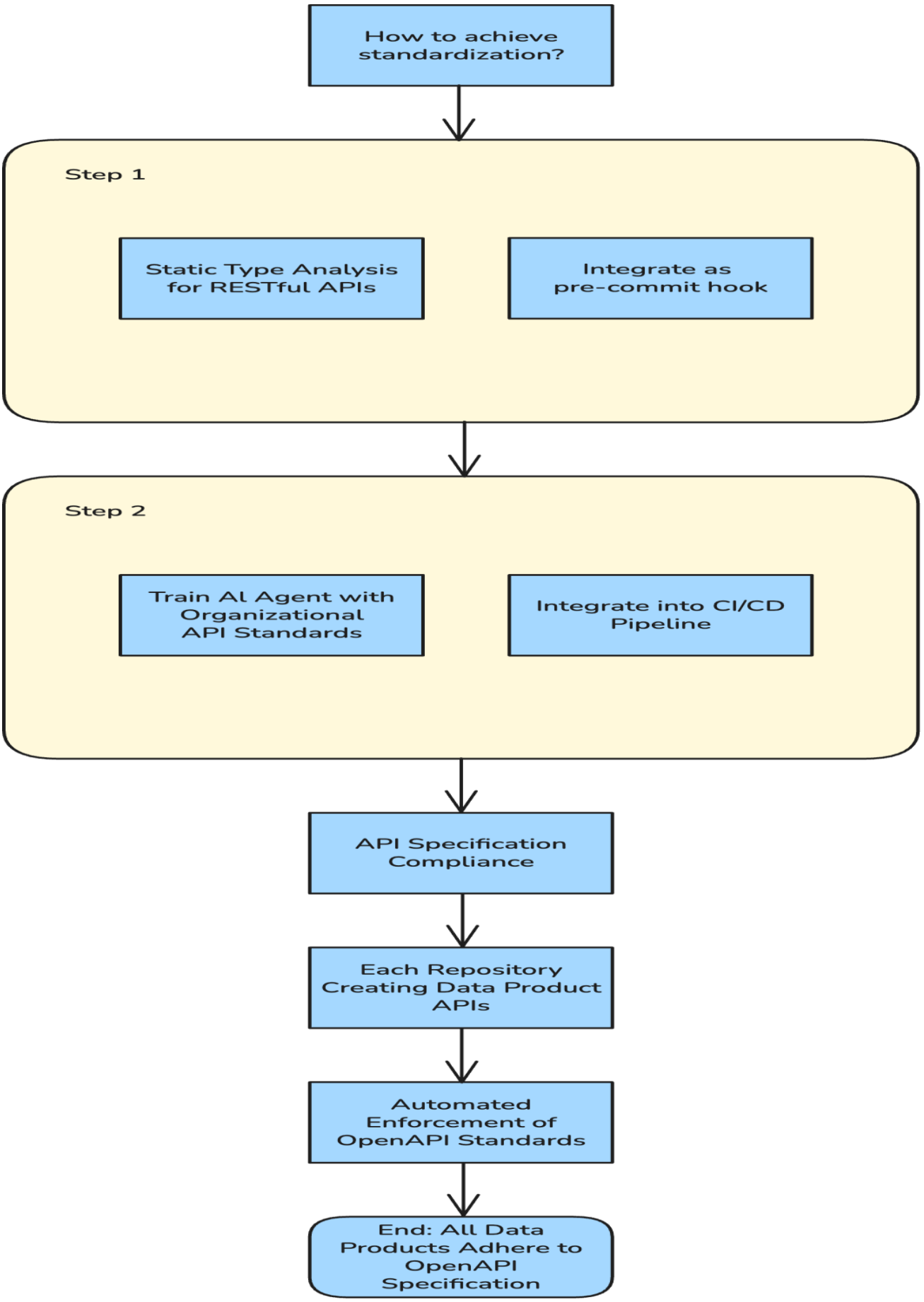

Fig. 2. Process of Achieving Standardization

and clearly marked as compliant or not, along with specific reasons for issues. Feedback from developers on previous checks helps it learn and reduce mistakes over time.

– Automated Review (Happens in CI/CD): Once trained, the AI Agent reviews new or updated OpenAPI specifications. It can find:

∗ Hidden Issues: Problems that a regular STA tool might miss, like a field named *customer _ id* that isn't a universally unique identifier (UUID), even if company policy says all IDs should be UUIDs.

∗ Style and Readability: Deviations from learned "best practices" or how easy it is to read the API description.

∗ Security Risks: Possible vulnerabilities or data exposure risks based on patterns it has learned [12]. Agent can also compare OpenAPI Specification against OWASP API Security for issues like broken auth or excessive data exposure.

- Why it's good: The AI Agent is flexible, can adapt as standards change, and can handle complex, judgement- based rules. Such as when *GET* api is called on */products* API and parameter is named as *customer_id* AI agent can detect is easily.

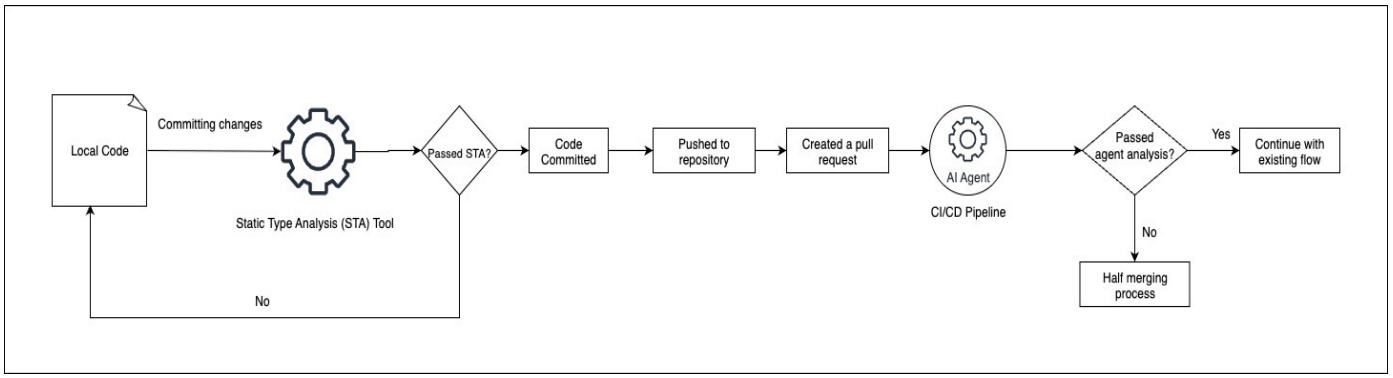

Fig. 3. Standardization Process in Developer's Workflow

- Working Together: While you can use either method alone, combining them is very powerful. STA acts as a quick first check for basic errors. Then, the AI Agent does a deeper, smarter review, often pointing out warnings or areas for people to look at, rather than completely stopping the process.

3) ***How They Fit In:*** How well these checking methods work depends on where they are used in the development process. Integrating them turns occasional checks into a continuous, automatic enforcement system.

a) ***Why Integration Matters:*** Using Static Type Analysis as a pre-commit hook (before code is committed) and the AI agent in the CI/CD pipeline acts like a gatekeeper (as shown in Figure 3). This ensures that every proposed change to a data product's API is automatically checked against company standards before it can be committed, deployed, or added to the main branch code.

b) ***How Static Type Analysis Works:***

- When it runs: Whenever a developer tries to commit new changes, a pre-commit hook automatically starts. This check includes the Static Type Analysis of the API.
- What it does: During the pre-commit check, a specific program or outside service is called, and the OpenAPI specification is sent to it for analysis.
- Feedback and Action: The OpenAPI specifications are carefully checked against all the company's defined and standardized rules. If everything passes, the new changes are committed. If any new API specification doesn't meet the standards, the pre-commit check will show an error message explaining the problem, stopping the developer from committing non-compliant changes.

c) ***How the AI Agent Works in CI/CD:***

- When it runs: Whenever a developer pushes new changes and creates a pull-request for a data product, the CI/CD pipeline automatically starts.
- What it does: Inside the CI/CD pipeline, the AI Agent is activated. It receives the OpenAPI specification file and performs its validation checks.
- Feedback and Action: If the OpenAPI specification passes all defined checks, the pipeline continues, allowing the

changes to be integrated and deployed. If it fails, the pipeline is set up to stop the build or prevent the pull request from being combined with the main code. This automated enforcement gives immediate, useful feedback to the developer, stopping non-compliant APIs from going into production. Developers get clear reports showing any issues, helping them quickly find and fix problems.

c. *What You Get: Consistent APIs*

Using this method consistently and automatically leads to the desired result:" All Data Products Adhere to OpenAPI Specification" (this is the final goal shown in Figure 3). By building these compliance checks directly into the development process, organizations can achieve several key benefits:

- Consistent Design: All data product APIs will follow a single set of standards.
- Less Old Code Debt: Problems are found and fixed early, reducing expensive re work later on.
- Faster Development: Clear APIs and automated checks make the process smoother for everyone creating and using data.
- Stronger Governance: A solid and traceable system is in place to ensure APIs always meet company and legal rules for data sharing.

IV. Understanding Our Approach

This section explores what it takes to put our proposed system into action. This system automatically checks OpenAPI specifications for data products. We'll look closely at the strengths of Static Type Analysis (STA) and the AI Agent, understanding how they work best alone and together. We'll also cover the big hurdles companies face when adopting and maintaining such a strong system.

A. *How We Check Rules: STA versus AI Agent*

Choosing the right ways to check compliance is key to our framework's success. Our system uses a two-step approach: Static Type Analysis (STA) and an AI Agent. Each has its own pros and cons, which decide when and how they are best used.

a) *Static Type Analysis (STA):* STA tools check OpenAPI compliance using fixed rules. They are great at making sure the API's structure, data types, and specific formats are correct according to the company's OpenAPI rules. For example, STA is perfect for checking that all required fields are present (like a description for every action), that data types match what's expected (like ISO 8601 for dates), and that naming rules are strictly followed (like using *snake_case* for paths). Its strong points are being precise, fast, and easy to understand. When a check fails, it clearly shows exactly which rule was broken and where. This makes STA very useful for finding basic errors early on, and developers can even run it on their own computers for quick feedback [11].

b) *AI Agent:* On the other hand, the AI Agent handles more flexible or context-based rules that are hard to put into strict STA rules. An AI Agent learns from company guidelines and examples of good and bad API specifications. It can then figure out best practices and spot small problems. For instance, it might check if an API's overall design is user friendly, if its documentation is easy to grasp, or if its data models accidentally share sensitive information based on broader company policies that are tough to write as specific rules. Its strengths are its flexibility and its ability to learn from changing rules and complex patterns, giving deeper insights beyond just correct structure [12].

c) *Using Both Together:* Instead of choosing between STA and the AI Agent, the most powerful approach is to combine them. STA can be the first, quick check for basic errors. It efficiently catches all syntax mistakes, structural problems, and clear rule breaking. If an API specification passes these first, definite checks, it can then go to the AI Agent for a deeper, more context aware review. This layered method makes the compliance process better. STA gives immediate, clear feedback on fundamental issues, while the AI Agent offers smart insights for higher level compliance. It might flag warnings or suggest areas for people to review instead of outright stopping the process. This balance prevents unnecessary headaches for developers while keeping standards high.

V. Challenges and What to Consider

Putting a strong, automated OpenAPI checking system into place comes with several real world and organizational challenges that need careful thought.

A. *Effort for Tools and Training*

The initial time and money spent on creating tools and training can be significant. For Static Type Analysis (STA), this means either heavily customizing existing tools (like Spectral) with many companies specific rules, or building entirely new tools from scratch [11]. For an AI Agent, the effort is even greater. It involves gathering and preparing high quality training data (including both compliant and non-compliant API specifications with expert notes), picking and fine tuning the right AI models, and continuously updating the training to stay accurate. This initial cost can be a major barrier for companies with limited resources or expertise in AI and machine learning.

B. *5.2 Keeping Up with Evolving Standards*

Company OpenAPI standards aren't set in stone. They must change as technology advances, business needs shift, and we learn from real world use. Managing these changes and making sure the checking tools stay current is an ongoing challenge. Every change to the standards means updating the STA rules or retraining the AI Agent. Without a clear plan for maintaining these checkers and keeping them in sync with standard changes, the system could become outdated, give unhelpful feedback, or miss new types of non compliance, making it less effective.

C. *5.3 Getting People to Adopt It*

A big obstacle is getting everyone in the organization to adopt the system and changing how they work [13]. Developers, who are used to more freedom in API design, might see automated checks as an extra burden or something that slows them down. For the system to be widely accepted, there needs to be clear communication about its benefits, thorough training, easy to use tools, and a supportive culture that truly values consistent API rules. Making the checkers fit smoothly into existing developer routines (like adding them to programming environments or providing clear feedback in CI/CD) and showing how automated checks actually make their work simpler can greatly help this change.

D. *5.4 Scaling Across Many Teams and Products*

Making our proposed system work for a growing number of different data products and spread-out teams is a complex challenge. It's crucial to ensure that the checking tools can efficiently process many specifications without causing big delays in the CI/CD pipeline. Also, setting up consistent ways to deploy and manage these checkers across possibly different team specific tools and infrastructure needs solid planning and automation. This prevents the system from becoming a roadblock instead of a help.

VI. CONCLUSIONS

The huge growth of data products in modern companies is vital for innovation driven by data. However, it has also brought big challenges with finding, using, and managing these products because their API interfaces aren't consistent. This paper has tackled this key issue by suggesting a full framework. This framework focuses on making it mandatory to use and automatically enforce the OpenAPI Specification (OAS) for all internal data products.

Our framework starts with the crucial step of defining clear organizational OpenAPI standards. These standards are specifically designed to meet the needs of those using data products, especially for later applications and data processing. This standardization offers great benefits. It helps people find data products more easily, makes them work better together, simplifies their use for developers, and ultimately strengthens data governance and compliance.

To make sure these standards are followed widely and consistently, we've shown two different, yet potentially com- plementary, automated ways to check compliance: Static Type Analysis (STA) and an AI Agent. STA provides quick and definite checks of structure and syntax, giving immediate feedback. The AI Agent, on the other hand, can enforce more subtle, context based, and changing company guidelines through what it learns. Both methods are made to fit seam- lessly into CI/CD pipelines. This allows for continuous and automatic checks of data product APIs at every development stage. This automatic gate-keeping ensures that only compliant data product interfaces are put into use, encouraging a culture of designing data with an "API first" mindset.

In short, putting this OpenAPI focused framework into action for data products is more than just about documentation. It's a fundamental change toward treating data as a well defined, usable product. By using

automated enforcement, companies can become more efficient, reduce integration prob- lems, improve data quality, and build a strong foundation for their developing data ecosystems.

REFERENCES

[1] "Artificial Intelligence [AI] Market Size, Growth & Trends by 2032." Accessed: Jul. 01, 2025. [Online]. Available: https://www.fortunebusinessinsights.com/industry-reports/artificial- intelligence-market-100114

[2] "OWASP API Security Project — OWASP Foundation." Accessed: Jul. 01, 2025. [Online]. Available: https://owasp.org/www-project-api-security/

[3] "4 data product challenges and solutions." Accessed: Jul. 01, 2025. [On- line]. Available: https://www.starburst.io/blog/data-product-challenges/

[4] "REST API Standards and Guidelines - AppSentinels." Accessed: Jul. 01, 2025. [Online]. Available: https://appsentinels.ai/blog/rest-api-standards-and-guidelines/

[5] L. Singh, "Ultimate Guide to Resolving REST API Performance Issues," Medium. Accessed: Jul. 01, 2025. [Online]. Available: https://medium.com/@lakhwinder.chdit/ultimate-guide-to-resolving- rest-api-performance-issues-331a47c38ab7

[6] A. Mehta, "API Governance: The key to Digital harmony," Medium. Accessed: Jul. 01, 2025. [Online]. Available: https://medium.com/@m.anurag08/api-governance-the-key-to-digital- harmony-5428ecfe09df

[7] "Theneo Blog - Understanding Open API Specifica- tions." Accessed: Jul. 01, 2025. [Online]. Available: https://www.theneo.io/blog/understanding-the-benefits-of-open-api- specifications

[8] "OpenAPI Specification - Version 3.1.0 — Swagger." Accessed: Jul. 01, 2025. [Online]. Available: https://swagger.io/specification/

[9] C. Brinson, "7 Key Principles of API Design for 2025," Jitterbit. Accessed: Jul. 01, 2025. [Online]. Available: https://www.jitterbit.com/blog/api-design-principles/

[10] "Gateway-enforced API Authorization," Gateway-enforced API Authorization. Accessed: Jul. 01, 2025. [Online]. Available: https://www.aserto.com/blog/gateway-enforced-api-authorization

[11] T. Sevenich, "API Linting with Spectral [From Basic Rules to Enterprise-Wide Standards]," Axway Blog. Accessed: Jul. 01, 2025. [Online]. Available: https://blog.axway.com/learning-center/apis/api-design/api-linting-with-spectral

[12] "How AI Can Help Automate API Governance and Compli- ance - Treblle." Accessed: Jul. 01, 2025. [Online]. Available: https://treblle.com/blog/ai-api-governance-compliance

[13] "Why AI Adoption Fails Without Cultural Alignment and Governance Support." Accessed: Jul. 01, 2025. [Online]. Available: https://www.allganize.ai/en/blog/resistance-to-ai-governance-and- cultural-challenges

[14] F. Palma, J. Gonzalez-Huerta, M. Founi, N. Moha, G. Tremblay, and Y.-

G. Gue´he´neuc, "Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns," *Int. J. Coop. Info. Syst.*, vol. 26, no. 02, p. 1742001, Jun. 2017, doi: https://doi.org/10.1142/ S0218843017420011.

[15] N. Moha *et al.*, "Specification and Detection of SOA Antipatterns," in *Service-Oriented Computing*, P. P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds., Lecture Notes in Computer Science, vol. 6470, Berlin, Heidelberg: Springer, 2012, pp. 1–16. doi: https://doi.org/10. 1007/978-3-642-34321-6 1