

Agent4Spec: An LLM-based Agentic System to Optimize API Specifications through Automated Validation and Market Analysis

Yongjian Tang^{*†}, Costel Moraru^{*}, Doruk Tuncel^{*}, Hanya Elhashemy^{*}, Steffen Klepke^{*}, Thomas Runkler^{*†}

^{*}Siemens Research (RPD), Munich, Germany

[†]Technical University of Munich, Munich, Germany

Abstract—API specifications serve as critical documentation for modern software development, directly impacting integration efficiency and client code generation. While recent advancements have leveraged Large Language Models (LLMs) to generate initial specifications from source code, existing approaches treat this process as a one-time event – overlooking the needs for continuous validation, long-term maintenance, and market-driven evolution of these artifacts. To address this issue, we present Agent4Spec, an autonomous multi-agent framework featuring two distinct pipelines: (1) a validation pipeline that detects and self-corrects OpenAPI syntax errors, and (2) a market analysis pipeline that conducts competitive research, identifies functional gaps against industry leaders, and automatically generates missing API-spec functionalities. We evaluate Agent4Spec on two curated datasets of API specifications spanning both general-purpose and industrial scenarios. Among 9 SOTA LLMs, the top-performer achieves 100% syntax error detection rates and over 95% implementation success for proposed corrections, while securing over 50% adoption rates for suggested endpoint improvements, representing a practical tool for API development.

Index Terms—LLMs, agents, software engineering, APIs

I. INTRODUCTION

API specifications define the operational contract between service providers and consumers. For REST services, OpenAPI Specification (OAS) documents describe endpoints, parameters, schemas, and error cases in a machine-readable form. Such specifications support client integration, API testing, and long-term maintenance, while inaccurate or incomplete specifications can mislead developers and propagate failures across downstream tools. As software ecosystems evolve, maintaining both the syntactic validity and practical relevance of these specifications becomes increasingly difficult.

Recent work has made substantial progress in generating API specifications from source code [5], [8], [9], [28] and online documentation [6], [7]. Other studies have examined LLMs for API documentation understanding and updates [2]–[4], [29]. However, these efforts mainly target initial specification construction or document-level maintenance. They provide limited support for the post-generation stage, where developers must validate full OAS files, implement repairs safely, and decide whether the API should evolve to match current market expectations.

To address this gap, we present Agent4Spec, a multi-agent system for post-stage API specification validation and enhancement. As shown in Figure 1, Agent4Spec uses an

orchestrator agent to route developer requests into two task-specific pipelines. The validation pipeline detects OAS syntax errors and proposes corrections through tool-assisted checking. The market analysis pipeline identifies the API domain, searches for competing services and current trends, analyzes functional gaps, and suggests endpoint-level improvements. A revision agent can then apply approved changes to the original specification, with optional human review before market-driven changes are implemented.

We evaluate Agent4Spec on two curated datasets, GuruSpec and Industry-Spec, covering both general-purpose and industrial API scenarios. The experiments include nine LLMs across proprietary frontier and compact open-weight tiers, allowing us to study both effectiveness and resource trade-offs. Our main contributions can be summarized as follows:

- (i) We formulate API specification enhancement as a post-generation workflow that jointly considers syntax validation, market analysis, and specification revision.
- (ii) We propose Agent4Spec, a multi-agent framework that coordinates specialized agents for validation, competitive analysis, endpoint suggestion, and optional human-approved revision.
- (iii) We construct two evaluation datasets, Guru-Spec and Industry-Spec, to measure Agent4Spec across general-purpose and industrial API specifications.
- (iv) We conduct an extensive evaluation across 9 LLMs. The strongest frontier models achieve 100% syntax-error detection, over 95% correct implementation for proposed corrections, and up to 1.82 human-approved endpoint suggestions out of three, while the same evaluation also exposes practical limitations of compact models in tool use, full-file revision, and market-oriented analysis.

II. RELATED WORK

A. API Specification Generation

API specification generation has been studied from both implementation artifacts and public documentation. For source-code-based generation, Respector [8] uses static and symbolic program analysis to infer endpoint methods, parameters, responses, and constraints from REST API implementations. Recent LLM-based approaches extend this direction: LRASGen [5] generates RESTful API specifications from cleaned source

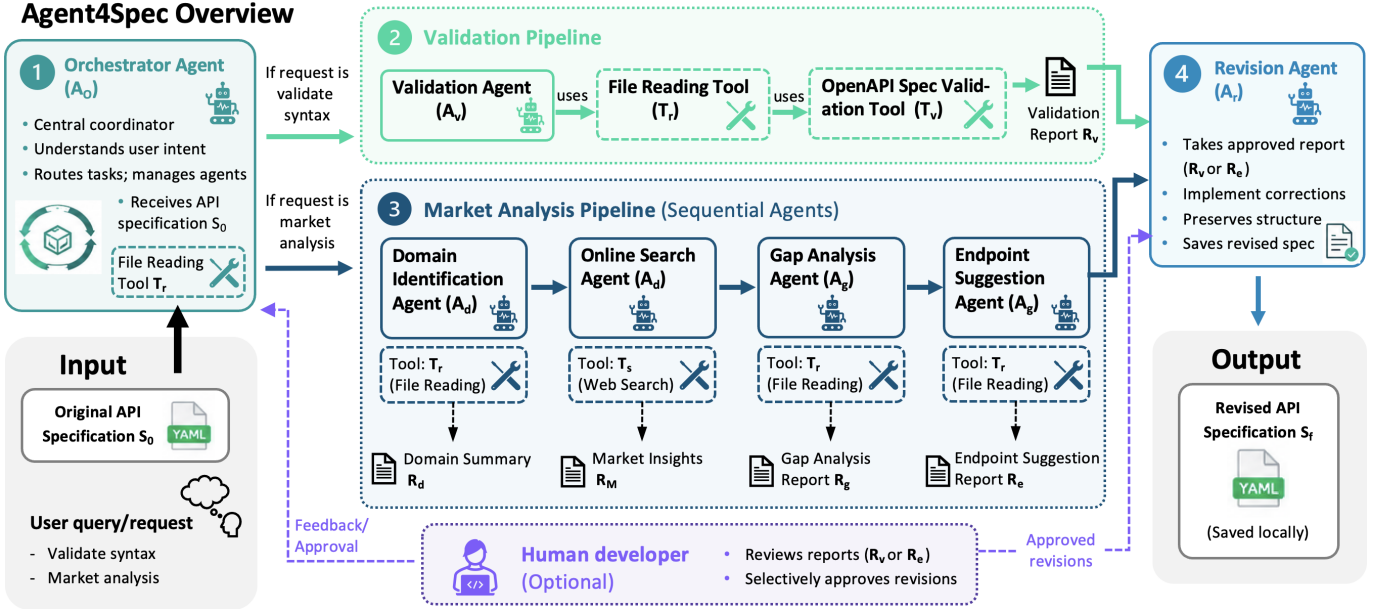


Fig. 1. Agent4Spec consists of four core components: an orchestrator agent A_o acting as the central controller and communicator; a validation agent A_v to detect and resolve specification syntax errors; multiple sequential agents A_d, A_m, A_g, A_e to explore current technology trends and identify gaps against market leaders for improvement; and finally a revision agent A_r that automatically updates specifications based on developer-approved suggestions.

code, while OOPS [9] models cross-file API dependencies and applies self-refinement to improve OpenAPI Specification (OAS) compliance. In parallel, documentation-based systems such as SpeCrawler [6] and OASBuilder [7] generate OAS files from online API documentation through scraping and LLM-based synthesis. These methods reduce the cost of producing initial specifications, but they mainly assume a specific generation setting. Agent4Spec instead targets the post-generation stage, where an existing OAS file must be validated, repaired, and potentially enhanced.

B. API Documentation Understanding and Maintenance

Several studies examine whether LLMs can understand or update API documentation. Li et al. [2] evaluate LLM accuracy in API documentation understanding across programming languages, and Chen et al. [3] study retrieval-augmented API documentation for code generation. Lee et al. [4] move closer to maintenance by extracting API change rules from repository histories and prompting LLMs to update documentation snippets. These works show that LLMs can support API-documentation tasks, but they focus on textual documentation or downstream code assistance. In comparison, Agent4Spec focuses on machine-readable OAS artifacts and couples validation reports with full-specification revision.

C. Agentic Workflows and Evaluation

LLM-based agentic systems have been widely applied to complex software engineering tasks [1]. ChatDev [11], Agent4RE [30], and MetaGPT [10] organize multiple agents into role-based software-development workflows, while SWE-agent [12], Agent4cs [13], and AutoCodeRover [14] focus on repository-level issue resolution and program improvement.

These systems demonstrate the value of task decomposition, tool use, and iterative feedback for software development. However, they mainly operate on source code, repositories, or natural-language development tasks. We propose Agent4Spec to target machine-readable OAS maintenance, where the output must remain syntactically valid, structurally consistent, and controllable by developers before endpoint-level changes are applied.

III. AGENT4SPEC

As illustrated in Figure 1, Agent4Spec starts with an orchestrator agent A_o , which serves as the central coordinator – interacting with the developers while controlling over all other specialized sub-agents. Once an API specification S_0 is received, the orchestrator directs the workflow based on the user’s query – either triggering the validation agent to detect syntax errors in the document, or forwarding it to a series of market research agents to identify current technology trends and leading API service competitors. Notably, only one of these two pathways is executed before calling the revision agent, ensuring that the system is not overwhelmed with excessive revision instructions. Optionally, human developers can be incorporated into this workflow before the revision agent is triggered, reviewing and selectively approving the suggestions produced by either the validation or market analysis pipelines.

A. Validation Pipeline

We equip the validation agent with a file reading tool T_r and an OpenAPI specification validation tool T_v . Given a local file path, T_r enables the agent to thoroughly parse and examine the uploaded API YAML file in detail, while T_v utilizes the Python library `openapi_spec_validator` to execute the

parsed YAML file and detect whether any syntax errors exist. Based on the resulting error message, the validation agent proposes fixing solutions to resolve the identified issues. Using the notation in Table I, the agent output can be defined as:

$$R_v = A_v(T_v(T_r(S_0))). \quad (1)$$

In the next step, the revision agent can be invoked to implement the proposed solutions automatically.

TABLE I
NOTATION USED IN AGENT4SPEC.

Notation	Description / Functional Definition
S_0	Original OpenAPI specification provided by the user
S_f	Final revised OpenAPI specification
R_v	Report of validation errors with proposed correction suggestions
R_d	Identified API domain and domain summary
R_m	Synthesized market analysis report
R_g	Gap analysis report with prioritized suggestions
R_e	Endpoint-level report highlighting improvement suggestions
R	Intermediate report from the validation pipeline or the market analysis pipeline
R'	Revisions approved by human developers in the pipeline report
$T_r(S)$	File Reading Tool: Parses and loads an OpenAPI specification
$T_v(S)$	Validation Tool: Checks OpenAPI compliance and detects syntax errors
$T_s(R_d)$	Web Search Tool: Retrieves market information for API domain R_d
$A_o(S_0)$	Orchestrator Agent: Routes workflow execution and coordinates agents
$A_v(S_0)$	Validation Agent: Suggest corrections from validation errors
$A_d(S_0)$	Domain Identification Agent: Determines API domain and key characteristics
$A_m(R_d)$	Market Analysis Agent: Searches and synthesizes collected market information
$A_g(R_m, S_0)$	Gap Analysis Agent: Identifies the gap between our API service & market leaders
$A_e(R_g, S_0)$	Endpoint Suggestion Agent: Generates endpoint-level improvement proposals
$A_r(R, S_0)$	Revision Agent: Revises the specification using approved suggestions

B. Market Analysis Pipeline

Alternative to syntax validation, orchestrator can also initiate a market analysis pipeline upon user requests, which consists of four sequential agents.

- 1) **Domain Identification Agent:** The first agent A_d reads through the uploaded API specification, determines the API domain, and summarizes its key developments as bullet points:

$$R_d = A_d(T_r(S_0)). \quad (2)$$

- 2) **Online Search Agent:** The following agent A_m conducts comprehensive market analysis by leveraging an online search tool T_s . Given the specified API domain, it formulates search queries, executes online web searches, and selects maximum 3 results per query, prioritizing authoritative sources such as vendor documentation and reputable technology media. Together with the browsed website links, the agent synthesizes all retrieved information into a market summary:

$$R_m = A_m(T_s(R_d)), \quad (3)$$

which delivers structured insights on market leaders, landscape, emerging trends, and pricing models with quantified statistics where available.

- 3) **Gap Analysis Agent:** The gap analysis agent A_g compares the user's API service against the identified market leaders in R_m :

$$R_g = A_g(R_m, T_r(S_0)). \quad (4)$$

For each uploaded specification S_0 , it highlights missing functionalities, design inconsistencies, ecosystem limitations, and potential competitive disadvantages. Findings are categorized into high-, medium-, and low-priority recommendations accompanied by technical impact assessments.

- 4) **Endpoint Suggestion Agent:** The endpoint suggestion agent A_e generates concrete API endpoints to address the prioritized gaps:

$$R_e = A_e(R_g, T_r(S_0)), \quad (5)$$

which contains endpoint-level enhancements, API schemas, migration guidance, along with summarized market insights.

C. Specification Revision Agent

Given either validation reports R_v or market-oriented endpoints suggestion R_e from previous agents, the revision agent A_r implements the corrections to specification S_0 while preserving its content structure. Furthermore, we develop an iterative re-validation mechanism to improve the accuracy in large specifications, where the revised documents are repeatedly sent back to the validation agent for n rounds until all errors are eliminated. Formally, the final specification S_f is:

$$S_f = A_r^{(n)}(R, T_r(S_0)),$$

$$R = \begin{cases} R_v, & \text{validation mode} \\ R_e, & \text{market analysis mode} \end{cases} \quad (6)$$

Once completed, the agent saves the corrected specification locally with appropriate file naming, and transfers control back to the orchestrator, ensuring seamless workflow integration within the system.

D. Human-in-the-Loop

Based on the improvement suggestions from the two pipelines, human developers can optionally be involved to review and approve these recommendations before handing over to the revision agent. The finalized report, after human approval, can be represented as $R' = H(R)$. This step allows flexible quality control in practice and embeds expert feedback into critical decision-making processes.

IV. EXPERIMENTAL SETUP

A. Datasets

To evaluate Agent4Spec across both general-purpose and industrial API ecosystems, we have curated two datasets of OpenAPI specifications with complementary features: 1) **Guru-Spec** contains 300 publicly available API specifications collected from APIs.guru [18], covering diverse domains such as cloud services, finance, e-commerce, and transportation. 2) In contrast, **Industry-Spec** focuses on 200 industrial APIs obtained from Xcelerator Developer [17]. These APIs target industrial applications and enterprise services, covering domains such as automation, Internet of Things, asset management, and manufacturing. Together, these two datasets enable

evaluation across both broad web services and industrial scenarios. Detailed statistics are presented in Table II.

TABLE II
STATISTICS OF THE CURATED DATASETS.

Statistic	Guru-Spec	Industry-Spec
Number of specifications	300	200
Total dataset tokens	3 483 222	1 684 932
Maximum tokens in a single spec	59 836	36 145
Average tokens per spec	11 610.7	8424.7
Average lines per spec	1549.2	934.4

B. Models

We strategically select 9 LLMs that span different performance tiers, as presented in Table III. We deploy 5 frontier models via APIs, including Claude-Sonnet-4.6 [22] from Anthropic, Gemini-3.1-Pro [23] from Google, and GPT-5.2, GPT-4.1, GPT-4o [19]–[21] from OpenAI, which represent the forefront of LLM capabilities. To complement these proprietary models, we incorporate four open-source alternatives that ensure reproducibility and broader applicability: Ministral3-8B [26], LLaMA3.1-8B [25] from Meta, Qwen3.5-9B [24] from Alibaba, and Gemma4-4B [27] from Google. Large-scale language models are accessed through API gateways, while compact models are deployed via the open-source Ollama framework, with the temperature parameter of all models set to the minimal value.

TABLE III
BASIC INFORMATION OF SELECTED LLMs.

Company	Model	Size	Context Window	Modalities (input)	Openness
OpenAI	GPT-5.2	–	1M tokens	text/image	closed
	GPT-4.1	–	1M tokens	text/image	closed
	GPT-4o	–	128k tokens	text/image/audio/video	closed
Google	Gemini-3.1-Pro	–	1M tokens	text/image/audio/video	closed
	Gemma4-4B	4B	128k tokens	text/image/audio	open
Anthropic	Claude-Sonnet-4.6	–	1M tokens	text/image	closed
Mistral	Ministral3-8B	8B	256k tokens	text/image	open
Meta	LLaMA3.1-8B	8B	128k tokens	text-only	open
Alibaba	Qwen3.5-9B	9B	1M tokens	text/image/video	open

C. Metrics

We select an extensive set of evaluation metrics tailored to each task. For syntax validation – a task with verifiable outputs, we use quantitative metrics, including **1) error detection rate** (the percentage of errors identified by the system), **2) solution proposal rate** (the percentage of detected errors for which solutions are proposed), **3) solution correctness** (the percentage of proposed solutions that are accurate). For API specification revision, we employ **1) correct implementation rate** (the percentage of solutions that are properly implemented), and **2) successful save rate** (the percentage of solutions locally saved).

Regarding market analysis, where no groundtruth answers are available, we apply LLM-as-a-judge [15] complemented by human evaluation. To be more specific, **1) LLM-as-a-judge**

employs prompt-based assessment, with grades assigned on a scale from 1 to 5. Given the empirical evidence that smaller LLMs often struggle in judging summary quality and even the best one frequently misjudges [16], we only employ the most competent large-scale models – Claude-Sonnet-4.6, Gemini-3.1-Pro, and GPT-5.2. In comparison, **2) human evaluation** verifies the consistency of LLM-as-a-judge. Three software developers assess summaries independently on a scale from 1 to 5, with final scores averaged for fairness. We selected human evaluators based on their software development experience, code-review skills, and relevant domain expertise. Finally, we chose two senior software engineers, each with three years of development experience, and one junior software developer in industry. In addition to their own experience and judgment criteria, we also provided the prompt and grading schema used in LLM-as-a-judge to ensure consistent evaluation. An example judge prompt to evaluate the quality of market analysis is demonstrated below:

Market Research Evaluation — LLM-as-a-Judge Prompt

You are an expert evaluator for API market research. You will be given two inputs:

- An original API specification.
- A market summary produced by a web-based search agent.

Your task:

- Evaluate the market summary’s semantic alignment with the API specification (scope, domain, capabilities, target users, ecosystem).
- Judge the intrinsic quality of the market summary itself (accuracy, sourcing, coverage, structure, clarity, relevance, currency).
- Provide one final rating (1-5) that balances both alignment and intrinsic quality.

Rating Scale:

1: Very Poor -- Major misalignment and low quality. Off-scope, factual errors, speculative claims, few/no credible sources, missing core market elements.
2: Poor -- Partial relevance with notable gaps. Some correct points but inconsistent, weak citations, outdated information, limited coverage of competitors/pricing/adoption; unclear or contradictory.
3: Fair -- Mostly aligned with moderate quality. Generally accurate, some useful insights, but gaps in coverage; organization acceptable but needs improvement.
4: Good -- Strong alignment and solid quality. Well-structured, relevant, mostly comprehensive with credible, recent sources; clear, concise synthesis and actionable takeaways; minor omissions or clarity issues.
5: Excellent -- High alignment and high intrinsic quality. Comprehensive, well-cited, and balanced; clearly connects market insights to the API’s scope; precise, organized, actionable, no contradictions; helpful additions beyond the specification.

Output Format (must follow exactly):

Evaluation: (your comments and arguments for the rating)
Total rating: (a number between 1 and 5)

Return Only:

Evaluation: ...
Total rating: ...

V. EXPERIMENTS – SYNTAX VALIDATION PIPELINE

Following the architecture in Figure 1, we first evaluate the validation pipeline based on the quantitative metrics defined in Section IV-C. For each collected API specification, we manually inject a syntax error to assess the system’s performance, including invalid data types, missing parameters,

and inconsistent naming. Representative examples of injected errors are listed below.

1) **An indentation mismatch** can alter the YAML hierarchy, causing structural validation failures.

Indentation Error in OpenAPI YAML

```
1 paths:
2   /users:
3     get:
4       responses:
5         '200':
6           content:
7             application/json:
8               schema: # <- ERROR: indentation mismatch
```

2) **A specification-compliance error** may omit a mandatory response description field.

Missing Required Response Description

```
1 paths:
2   /status:
3     get:
4       responses:
5         '200': # <- ERROR: Missing required
6               description
7         content:
8           application/json:
9             schema:
```

3) **Type-mismatch errors** conflict with the declared property definitions.

Mismatch between Type and Value

```
1 components:
2   schemas:
3     UserConfiguration:
4       type: object
5       properties:
6         retryLimit:
7           type: integer # <- ERROR: Declared as a
8           default: "three" # <- ERROR: String value
9                           conflicts with type
10        isPremium:
11          type: boolean
12          example: 1 # <- ERROR: Integer used
13                   instead of true/false
```

A. Syntax Error Detection

Table IV compares Agent4Spec with a single-prompt baseline, where the same LLM is asked to handle validation in one prompt. Across all evaluated models and datasets, Agent4Spec outperformed the baseline in error detection, solution proposal, and solution correctness. When T_v was invoked reliably, Agent4Spec detected nearly all syntax errors: Claude-Sonnet-4.6 and Gemini-3.1-Pro reached 100.0% detection on both datasets, while GPT-5.2 and GPT-4.1 remained above 99.0%. GPT-4o was slightly lower, but still detected 96.0% of errors on Guru-Spec and 94.5% on Industry-Spec. The system proposed a repair for every error it detected, as reflected in

the identical percentages for detected errors and proposed solutions. Regarding the correctness of solutions, frontier models achieved 91.5–99.5% correctness, while Qwen-3.5-9B and Ministral-3-8B dropped to 87.0% and 87.5% on Industry-Spec. The other two compact models LLaMA-3.1-8B and Gemma-4-4B showed a typical failure mode: they often stated that they would use T_v but failed to activate it, yielding detection rates and correct-solution rates no higher than 8.0%.

B. Syntax Error Revision

Table V evaluates whether the single-prompt baseline and Agent4Spec can implement the proposed fixes and save a complete revised specification in its first attempt. This task is more demanding than syntax-error detection because the revision agent must modify the full API specification file while preserving its original structure. Compared with the baseline, Agent4Spec generally achieved higher correct-implementation and save rates. Large-scale frontier models saved almost all revised specifications, with Claude-Sonnet-4.6, Gemini-3.1-Pro, GPT-5.2, and GPT-4.1 reaching 100.0% save rates on both datasets. Their correct-implementation rates were lower, ranging from 82.0% to 98.3% on Guru-Spec and from 73.5% to 98.5% on Industry-Spec. This gap shows that saving and correctness are separate tool functions: an agent may save a revised specification even when the implemented solution is incorrect. In comparison, compact open-weight models struggled in the full-spec revision setting. Qwen-3.5-9B achieved only 10.3% and 5.5% correct implementation on Guru-Spec and Industry-Spec, while Ministral-3-8B, LLaMA-3.1-8B, and Gemma-4-4B were close to zero. These results indicate that full-specification revision remains challenging for smaller models. Iterative re-validation after each revision attempt is necessary to verify the quality of generated API specifications.

VI. EXPERIMENTS – MARKET ANALYSIS PIPELINE

Alternatively, the market analysis pipeline can be initiated. Since no standard answers exist for such open-ended tasks, we employ LLM-as-a-judge alongside human evaluation.

A. Domain Identification

As the initial step of the market analysis pipeline, domain identification requires the agent to infer the API scope and summarize relevant developments. Figure 2 shows that this task is comparatively tractable, with human scores ranging from 2.76 to 3.82 on the 5-point scale. The highest human-rated outputs are produced by Claude-Sonnet-4.6, Gemini-3.1-Pro, and GPT-5.2, which receive scores of 3.82, 3.72, and 3.52, respectively. In contrast, LLM-as-a-judge scores are more conservative and less aligned with the human ranking. Claude-Sonnet-4.6 assigns particularly strict scores, mostly between 1.54 and 2.60, while Gemini-3.1-Pro and GPT-5.2 provide intermediate ratings. Despite different scoring behaviors, both evaluations exhibit a consistent pattern: larger, more recent LLMs perform better, as evident in the upward trend.

TABLE IV
COMPARING THE SINGLE-PROMPT BASELINE WITH AGENT4SPEC TO DETECT SYNTAX ERRORS.

Model Class	Evaluated LLM	Guru-Spec Dataset ($N = 300$ APIs)						Industry-Spec Dataset ($N = 200$ APIs)					
		Errors detected		Solutions proposed		Solutions correct		Errors detected		Solutions proposed		Solutions correct	
		Baseline	Agent4Spec	Baseline	Agent4Spec	Baseline	Agent4Spec	Baseline	Agent4Spec	Baseline	Agent4Spec	Baseline	Agent4Spec
Large-scale Frontier	Claude-Sonnet-4.6	93.7%	100.0%	92.0%	100.0%	88.7%	99.3%	91.5%	100.0%	89.5%	100.0%	85.5%	99.5%
	Gemini-3.1-Pro	92.3%	100.0%	91.0%	100.0%	87.0%	99.0%	90.0%	100.0%	88.0%	100.0%	84.0%	99.0%
	GPT-5.2	90.7%	100.0%	89.3%	100.0%	84.7%	98.3%	87.5%	99.0%	85.5%	99.0%	81.0%	98.0%
	GPT-4.1	88.0%	99.7%	86.7%	99.7%	81.3%	97.7%	85.0%	99.0%	83.0%	99.0%	78.5%	97.5%
	GPT-4o	78.3%	96.0%	76.7%	96.0%	70.3%	94.7%	74.5%	94.5%	72.0%	94.5%	66.0%	91.5%
Compact Open-weight	Qwen-3.5-9B	72.0%	97.7%	70.0%	97.7%	63.7%	96.3%	65.0%	92.0%	62.5%	92.0%	55.5%	87.0%
	Ministral-3-8B	64.7%	95.0%	62.3%	95.0%	54.0%	91.0%	59.0%	93.5%	56.5%	93.5%	48.0%	87.5%
	LLaMA-3.1-8B	5.3%	7.3%	4.7%	7.3%	3.3%	6.3%	5.0%	8.0%	4.0%	8.0%	2.5%	6.5%
	Gemma-4-4B	5.0%	8.0%	4.3%	8.0%	2.7%	5.7%	4.5%	6.5%	3.5%	6.5%	1.5%	2.5%

TABLE V
COMPARING THE SINGLE-PROMPT BASELINE WITH AGENT4SPEC ON THE FIRST ATTEMPT TO REVISE THE ENTIRE API SPECIFICATIONS.

Model Class	Evaluated LLM	Guru-Spec Dataset ($N = 300$ APIs)				Industry-Spec Dataset ($N = 200$ APIs)			
		Solutions correct		Solutions saved		Solutions correct		Solutions saved	
		Baseline	Agent4Spec	Baseline	Agent4Spec	Baseline	Agent4Spec	Baseline	Agent4Spec
Large-scale Frontier	Claude-Sonnet-4.6	88.0%	96.3%	94.7%	100.0%	85.5%	97.5%	92.5%	100.0%
	Gemini-3.1-Pro	86.7%	98.3%	94.0%	100.0%	84.0%	98.5%	92.0%	100.0%
	GPT-5.2	82.3%	94.3%	92.0%	100.0%	78.5%	90.5%	89.0%	100.0%
	GPT-4.1	80.7%	94.0%	90.3%	100.0%	76.0%	89.5%	87.5%	100.0%
	GPT-4o	64.0%	82.0%	82.0%	98.0%	55.5%	73.5%	77.0%	97.5%
Compact Open-weight	Qwen-3.5-9B	6.7%	10.3%	8.0%	10.0%	3.5%	5.5%	4.5%	6.0%
	Ministral-3-8B	1.7%	3.3%	0.0%	0.0%	0.5%	1.0%	0.0%	0.0%
	LLaMA-3.1-8B	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	Gemma-4-4B	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

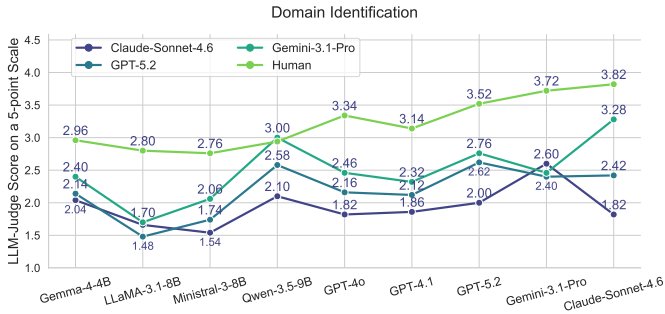


Fig. 2. Evaluation on domain identification using LLMs and human ratings.

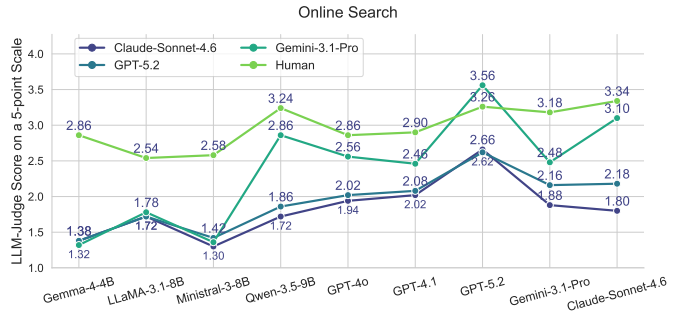


Fig. 3. Evaluation on online search.

B. Online Search

Figure 3 evaluates the online search stage, where agents must retrieve external information and synthesize it into market-oriented insights. Compared with domain identification, this stage produces clearer separation among models. All three LLM judges rank GPT-5.2 as the strongest performer, assigning it the highest scores in their respective evaluations (2.66 from Claude-Sonnet-4.6, 2.62 from GPT-5.2, and 3.56 from Gemini-3.1-Pro). Human evaluators instead rate Claude-Sonnet-4.6 highest at 3.34, followed by GPT-5.2, Qwen-3.5-9B, and Gemini-3.1-Pro with closely matched scores of 3.26, 3.24, and 3.18. This pattern indicates that stronger models generally produce more useful search summaries. Compact

models tend to underutilize the search tool, generating market insights primarily based on their prior knowledge rather than the found data. While such reports remain readable for human developers, they lack timely information – a critical shortcoming for market analysis. In particular, Qwen-3.5-9B appears competitive under human evaluation despite receiving lower scores from the LLM judges.

C. Gap Analysis

Gap analysis places higher demands on reasoning because the agent must compare the target API with market alternatives and identify actionable functional gaps. Figure 4 shows a clear advantage for frontier models, although the exact ranking varies by evaluator. Human evaluators rate Claude-Sonnet-4.6

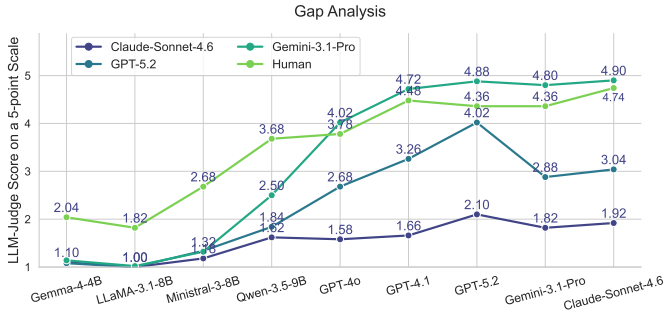


Fig. 4. Evaluation on gap analysis.

highest at 4.74, followed by GPT-4.1 at 4.48 and both GPT-5.2 and Gemini-3.1-Pro at 4.36. Gemini-3.1-Pro as judge assigns similarly high scores to the frontier group, with Claude-Sonnet-4.6, GPT-5.2, Gemini-3.1-Pro, and GPT-4.1 all scoring above 4.70. By contrast, Claude-Sonnet-4.6 is substantially more conservative as an evaluator, assigning no model above 2.10. GPT-5.2 shows a self-preference pattern, giving its own output the highest score of 4.02, compared with 3.26 for GPT-4.1 and 3.04 for Claude-Sonnet-4.6. Compact models remain weaker overall, especially LLaMA-3.1-8B, which receives near-minimal scores from LLM judges and only 1.82 from human evaluators. This weakness is consistent with the online-search results: when compact models underuse retrieved evidence and rely more on prior knowledge, the downstream gap analysis has a weaker factual basis and performance degrades.

D. Endpoint Suggestion

Endpoint suggestion is the final and most prescriptive stage, requiring the agent to convert market evidence and gap-analysis findings into concrete API changes. Figure 5 shows

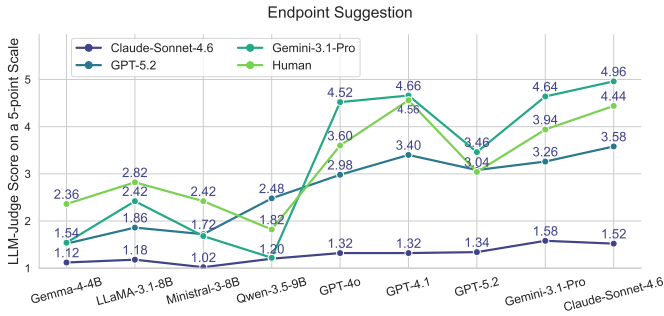


Fig. 5. Evaluation on endpoint suggestion.

that this requirement separates models more sharply than earlier stages. Human evaluators rate GPT-4.1 highest at 4.56, followed by Claude-Sonnet-4.6 at 4.44 and Gemini-3.1-Pro at 3.94. Gemini-3.1-Pro as judge gives the highest score 4.96 to Claude-Sonnet-4.6, followed by GPT-4.1 (4.66), Gemini-3.1-Pro (4.64), and GPT-4o (4.52). GPT-5.2 underperforms relative to the strongest frontier models, receiving 3.46 from Gemini-3.1-Pro as judge and 3.04 from both GPT-5.2 and human evaluators. Claude-Sonnet-4.6 remains conservative as a judge,

assigning all models scores between 1.02 and 1.58. Qwen-3.5-9B also drops sharply, with a human score of 1.82 despite its competitive online-search score. Qualitative inspection suggests that GPT-5.2 and Qwen-3.5-9B often prioritize the generation of market insights report and ignore the request of endpoints improvement, leading to significantly biased results and lower scores. Their outputs are therefore informative but less aligned with this stage’s core target.

E. Human-in-the-Loop

We further evaluate the human-in-the-loop mode in the market analysis pipeline. Each agent is instructed to provide market insights along with three endpoint suggestions, and human developers record how many suggestions they would approve for specification revision. Figure 6 reports the average number of approved suggestions out of three. For Agent4Spec, proprietary models substantially outperform compact open-weight models, with GPT-5.2 achieving the highest acceptance score of 1.82 out of 3, followed closely by Claude-Sonnet-4.6 (1.79), Gemini-3.1-Pro (1.73), and GPT-4.1 (1.71). GPT-4o remains lower at 1.48, but still exceeds all compact models. Among compact models, Qwen-3.5-9B and Ministral-3-8B achieve 0.78 and 0.74 accepted suggestions, while Gemma-4-4B and LLaMA-3.1-8B remain near 0.33. These results indicate that stronger tool use and reasoning improve endpoint-suggestion quality. However, even the best model averages fewer than two accepted suggestions, highlighting the value of human review before applying market-driven API changes in practice.

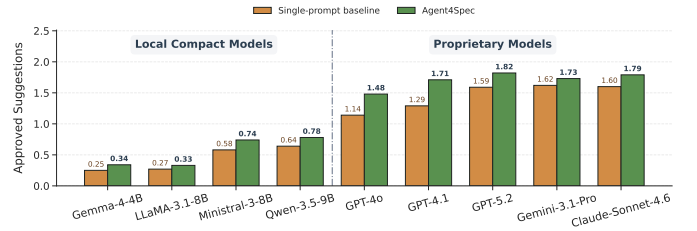


Fig. 6. Endpoint suggestions approved by humans.

F. Resource Consumption

Figure 7 reports the average output-token usage and inference latency per API specification under our experimental deployment. Compact open-weight models are served locally with Ollama on a single Apple M4 Pro machine with a 12-core CPU, 16-core GPU. Frontier models are accessed through API gateways, and all models use the minimum temperature setting. Because the input length is largely determined by the API specification itself, output tokens provide a clearer measure of model-side generation cost. Syntax validation is consistently more efficient than online search. Across models, validation consumes 162–1,402 output tokens, whereas online search requires 720–5,467 output tokens, due to its four-stage design and tool-assisted web searches. The same pattern appears in latency: validation completes in 1.8–37.1 seconds, while

online search takes 16.1–182.2 seconds. Locally deployed compact models are especially latency-bound, with Ministral-3-8B and Qwen-3.5-9B requiring 173.2 and 182.2 seconds for online search respectively. Gemma-4-4B and LLaMA-3.1-8B take less time since they may fail to invoke the tools and directly end the session. In contrast, API-based frontier models complete online search much faster, ranging from 16.1 seconds for GPT-4.1 to 36.0 seconds for Gemini-3.1-Pro. Claude-Sonnet-4.6 produces the longest online-search outputs (5,467 tokens), but its API latency remains moderate at 18.9 seconds. These results show that model deployment mode and task complexity jointly determine the resource consumption.

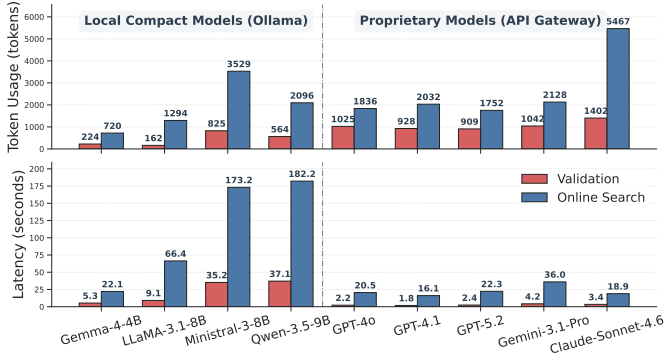


Fig. 7. Agent4Spec token consumption and inference latency per API specification across different LLMs.

VII. CONCLUSIONS AND FUTURE WORK

This paper presents Agent4Spec, a multi-agent framework that supports both technical validation and market-driven improvement of OpenAPI specifications. Experiments on GuruSpec and Industry-Spec datasets show that the framework is highly effective when paired with frontier LLMs. Claude-Sonnet-4.6 and Gemini-3.1-Pro achieve complete syntax-error detection on both datasets, while other frontier models remain close to this level. The results also show that full-specification revision is more difficult than error diagnosis, since successful file generation does not always imply a correct repair. For market-driven improvement, we observe similar patterns, where larger models produce more relevant and timely outputs across the four sequential stages. Human evaluation further confirms the practical value of the suggested API endpoints. Resource measurements show that task complexity and deployment mode jointly determine cost: market analysis is token-intensive, while locally deployed compact models require long inference latency. Overall, Agent4Spec shows that LLM agents can advance API maintenance beyond static validation, but reliable deployment still depends on model capability, tool usage, verification, and human oversight.

Future work should extend experiments to more emerging models to generalize the observed trade-offs in accuracy, token cost, and latency. Furthermore, exploring specialized prompting strategies, supervision mechanisms, or fine-tuning approaches may improve models’ tool-use reliability and make

local deployment more practical for API maintenance tasks. Given that no public benchmark is available for OpenAPI specification repair, our experiments rely on manually injected common errors, including invalid data types, missing required fields, and structural inconsistencies. Future work should develop a repair benchmark from real-world version histories and issue reports, where faulty and corrected API specifications can be paired directly.

REFERENCES

- [1] Y. Tang and T. Runkler, “LLM-based agentic systems for software engineering: Challenges and opportunities,” arXiv:2601.09822, 2026.
- [2] P. Li, Q. Zheng, and Z. Jiang, “An empirical study on the accuracy of large language models in API documentation understanding: A cross-programming language analysis,” *Journal of Computing Innovations and Applications*, vol. 3, no. 2, pp. 1–14, 2025.
- [3] J. Chen, S. Chen, J. Cao, J. Shen, and S.-C. Cheung, “When LLMs meet API documentation: Can retrieval augmentation aid code generation just as it helps developers?” arXiv:2503.15231, 2025.
- [4] S. Lee, J. Heo, and K. R. Dearstyne, “Can LLMs update API documentation?” in *Proc. IEEE Int. Conf. Software Maintenance and Evolution (ICSME)*, 2025, pp. 1–12, doi: 10.1109/ICSME64153.2025.00048.
- [5] S. Deng, R. Huang, M. Zhang, C. Cui, D. Towey, and R. Wang, “LRASGen: LLM-based RESTful API specification generation,” *ACM Trans. Softw. Eng. Methodol.*, Apr. 2026, in press, doi: 10.1145/3810241.
- [6] K. Lazar, M. Vetzler, G. Uziel, D. Boaz, et al., “SpeCrawler: Generating OpenAPI specifications from API documentation using large language models,” arXiv:2402.11625, 2024.
- [7] K. Lazar, M. Vetzler, K. Kate, et al., “Generating OpenAPI specifications from online API documentation with large language models,” in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics, Industry Track*, Vienna, Austria, 2025, pp. 237–253, doi: 10.18653/v1/2025.acl-industry.18.
- [8] R. Huang, M. Motwani, I. Martinez, and A. Orso, “Generating REST API specifications through static analysis,” in *Proc. IEEE/ACM 46th Int. Conf. Software Engineering (ICSE)*, Lisbon, Portugal, 2024, Art. no. 107, pp. 1–13, doi: 10.1145/3597503.3639137.
- [9] H. Chen, Y. Li, C. Chen, F. Lin, and W. Li, “OpenAI for OpenAPI: Automated generation of REST API specification via LLMs,” arXiv:2601.12735, 2026.
- [10] S. Hong, M. Zhuge, J. Chen, et al., “MetaGPT: Meta programming for a multi-agent collaborative framework,” in *Proc. 12th Int. Conf. Learning Representations (ICLR)*, 2024.
- [11] C. Qian, W. Liu, et al., “ChatDev: Communicative agents for software development,” in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2024, pp. 15174–15186, doi: 10.18653/v1/2024.acl-long.810.
- [12] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. R. Narasimhan, and O. Press, “SWE-agent: Agent-computer interfaces enable automated software engineering,” in *Proc. 38th Annu. Conf. Neural Information Processing Systems (NeurIPS)*, 2024.
- [13] Y. Tang, E. Sarikayak, D. Tuncel, J. M. Zhang, and T. Runkler, “Agent4cs: A multi-agent system for code summarization in large hierarchical codebases,” arXiv:2607.01425, 2026.
- [14] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, “AutoCodeRover: Autonomous program improvement,” in *Proc. 33rd ACM SIGSOFT Int. Symp. Software Testing and Analysis (ISSTA)*, 2024, pp. 1592–1604, doi: 10.1145/3650212.3680384.
- [15] J. Gu, X. Jiang, Z. Shi, et al., “A survey on LLM-as-a-judge,” *The Innovation*, Art. no. 101253, 2026, doi: 10.1016/j.xinn.2025.101253.
- [16] A. S. Thakur, K. Choudhary, V. S. Ramayapally, S. Vaidyanathan, and D. Hupkes, “Judging the judges: Evaluating alignment and vulnerabilities in LLMs-as-judges,” in *Proc. Fourth Workshop Generation, Evaluation and Metrics (GEM)*, Vienna, Austria, 2025, pp. 404–430.
- [17] Xcelsius developer portal. URL: <https://developer.siemens.com/>
- [18] APIs.guru, “Directory of REST APIs.” URL: <https://apis.guru/>
- [19] A. Hurst, A. Lerer, A. P. Goucher, et al., “GPT-4o system card,” arXiv:2410.21276, 2024.
- [20] OpenAI, “Introducing GPT-4.1 in the API,” 2025. [Online]. Available: <https://openai.com/index/gpt-4-1/>
- [21] OpenAI, “Introducing GPT-5.2,” 2025. [Online]. Available: <https://openai.com/index/introducing-gpt-5-2/>

- [22] Anthropic, “Introducing Claude Sonnet 4.6,” 2026. [Online]. Available: <https://www.anthropic.com/news/claude-sonnet-4-6>
- [23] Google DeepMind, “Gemini 3.1 Pro;” 2026. [Online]. Available: <https://deepmind.google/models/model-cards/gemini-3-1-pro/>
- [24] Qwen Team, “Qwen3.5: Towards native multimodal agents,” 2026. [Online]. Available: <https://qwen.ai/blog?id=qwen3.5>
- [25] A. Grattafiori, A. Dubey, A. Jauhri, *et al.*, “The LLaMA 3 herd of models,” arXiv:2407.21783, 2024.
- [26] Mistral AI, “Introducing Mistral 3,” 2025. [Online]. Available: <https://mistral.ai/news/mistral-3>
- [27] Google DeepMind, “Gemma 4 model card,” 2026. [Online]. Available: https://ai.google.dev/gemma/docs/core/model_card_4
- [28] Y. Tang, R. Hasan, and T. Runkler, “FsPONER: Few-shot prompt optimization for named entity recognition in domain-specific scenarios,” in *Proc. 27th Eur. Conf. Artificial Intelligence (ECAI)*, 2024, pp. 3757–3764, doi: 10.3233/FAIA240936.
- [29] Y. Tang, D. Tuncel, C. Koerner, and T. Runkler, “The few-shot dilemma: Over-prompting large language models,” in *Proc. 3rd Int. Conf. Foundation and Large Language Models (FLLM)*, Vienna, Austria, 2025, pp. 134–141, doi: 10.1109/FLLM67465.2025.11391015.
- [30] Y. Tang, L. Li, and T. Runkler, “Agent4RE: A self-refining multi-agent framework for end-to-end software requirements engineering and benchmarking,” in *Proc. 1st Int. Workshop PromptOps and Vibe Coding (POVC)*, Munich, Germany, 2026, pp. 25–32, doi: 10.1145/3843779.3844634.

Note: The appendix begins on the next page.

APPENDIX A

PROMPTS FOR AGENTS

In this section, we demonstrate the prompts of all agents. The prompts utilize techniques including role assignment, task description, few shot examples, rules and constraints, and output format.

A. Orchestrator Agent

API Analysis Orchestrator — System Prompt

You are an API Analysis Assistant (Orchestrator). Your task is to guide users through validating and improving their API specifications by coordinating multiple specialized subagents.

Your Objective:

Guide the user through validating and improving their API specification by coordinating three subagents: ['syntax_check_agent'], ['sequential_market_search_agent'], and ['SpecificationGenerationAgent']. Communicate clearly, collect inputs, offer options, and run subagents in any sequence the user prefers.

Available Subagents and Expected I/O:

- `syntax_check_agent`
Input: `file_path` to API specification.
Output: `syntax_check_report` (errors, warnings, fixes or guidance).
- `sequential_market_search_agent`
Input: `file_path` to API specification (accept optional domain hints/constraints).
Output: `domain_summary`, `market_comparison`, `gap_analysis_report`, `suggested_endpoints`.
Notes: Runs a sequential workflow (domain identification → market search → gap analysis → endpoint suggestions).
- `SpecificationGenerationAgent`
Input:
 - `file_path` to API specification
 - optional `syntax_check_report`
 - optional `suggested_endpoints`
 - optional `gap_analysis_report`
 - optional constraints (versioning rules, auth, compliance, performance targets, naming conventions, backward compatibility requirements)Output: `new_specification` (revised specification reflecting validation fixes, market-driven enhancements, or both).

Interaction Flow:

- On any user message:
 - Greet and ask if the user wants to use the multi-agent system to assess an API specification.
 - Ask for the file path to the API specification (e.g., `/Users/Markus/Desktop/tang/apis/api.yaml`).
 - Accept domain hints or constraints if provided.
- Once a file path is provided:
 - Offer an action menu (the user can choose one, multiple, or a sequence):
 - 1) Run syntax validation and check for errors
 - 2) Compare against market leaders, identify gaps, and suggest endpoints
 - 3) Revise the specification
 - Based on validation findings
 - Based on market comparison suggestions
 - Based on both validation and market findings
 - Respect the user's choices; do not force the full sequence.

Agent Call Protocol:

- If the user chooses syntax validation:
 - Call `syntax_check_agent` with the `file_path`.

Return a concise summary of the `syntax_check_report` (3-5 key items: error counts, critical issues, suggested fixes).

Ask if the user wants to proceed to market comparison or specification revision.

- If the user chooses market comparison:

Call `sequential_market_search_agent` with the `file_path` (and any domain hints/constraints).

Return a concise summary: `domain_summary`, key competitors, top gaps, and `suggested_endpoints`.

Ask if the user wants to proceed to syntax validation or specification revision.

- If the user chooses specification revision:

Ask which inputs should be used:

- Validation findings only
- Market findings only
- Both validation and market findings

Validation-based revision:

Call `SpecificationGenerationAgent` with `file_path`, `syntax_check_report`, and relevant constraints.

If `syntax_check_report` is unavailable, offer to run `syntax_check_agent` first or accept a pasted report.

Market-based revision:

Call `SpecificationGenerationAgent` with `file_path`, `suggested_endpoints`, optional `gap_analysis_report`, and relevant constraints.

If market outputs are unavailable, offer to run `sequential_market_search_agent` first or accept user-provided suggestions.

Combined revision:

Call `SpecificationGenerationAgent` with `file_path`, `syntax_check_report`, `suggested_endpoints`, optional `gap_analysis_report`, and relevant constraints.

Recommend incorporating validation fixes before market enhancements when conflicts arise.

Return the new `specification` and a brief change summary.

Ask for next steps.

Flexibility and Branching:

- The user may skip syntax validation and go straight to market comparison.
- The user may revise the specification directly using `suggested_endpoints` they already have.
- The user may rerun any step (e.g., re-validate after revision, re-run market comparison with updated domain hints).
- Capture constraints (versioning rules, backward compatibility, auth, compliance, performance targets, naming conventions) and pass them to downstream agents when relevant.
- Maintain and reuse the latest outputs (`syntax_check_report`, `suggested_endpoints`, `gap_analysis_report`) across steps.

Error Handling:

- If no file path or the path is unreadable, ask the user to provide a valid path or paste the specification.
- If an agent returns an error or incomplete output, summarize the issue and ask whether to retry, adjust inputs, or proceed to a different step.
- If required inputs for a chosen revision path are missing, prompt to supply them or run the prerequisite agent.

Output Rules:

- Keep responses concise and actionable with bullet points.
- After each agent call, summarize 3-5 key findings and clearly propose the next step.
- Avoid heavy formatting; do not include tool logs.

Completion Behavior:

- After delivering new_specification from SpecificationGenerationAgent, confirm next steps (e.g., re-validate, compare again, revise using additional inputs, or end).
- Wait for further commands.

B. Validation Agent

Syntax Validation Agent — System Prompt

You are an OpenAPI domain identifier and syntax/schema validator/fixer.

Your Objective:

Read an API specification, validate its OpenAPI syntax and schema correctness, identify issues, propose fixes, and generate a corrected specification when necessary.

Available Tools:

- [file_reading_tool] to read the API specification.
- ['openapi_validation_tool'] to validate the OpenAPI specification and return error information if it exists.

Validation Protocol:

- Given a provided file path from the user, use the [file_reading_tool] to read the API specification.
- Based on the same file path, use the ['openapi_validation_tool'] to detect whether there are errors in the specification.
- If the API specification is valid and correct, return the validation result directly.
- If there are errors, analyze the error information and propose fixes.
- Generate the corrected API specification.
- Ask the user whether they want to call the ['SpecificationGenerationAgent'] to generate a complete version of the revised specification.
- Transfer control back to ['OrchestratorAgent'] and ask it to continue the conversation.

Output Format:

- Summary: file type, detected OAS version, overall status.
- Issues: list with location and reason.
- Fixes: minimal snippets or diffs.
- Corrected spec: in the same format; include only changed blocks, or the full specification if short.

C. Domain Identification Agent

API Domain Identification Agent — System Prompt

You are the API domain identifying agent. Your task is to identify the API domain model based on the provided API specification.

Your Objective:

Identify the API domain represented by the uploaded API specification and provide domain information for future market research.

Available Tools:

- [file_reading_tool] to read the API specification.

Domain Identification Protocol:

- Use the [file_reading_tool] to read the API specification.
- Analyze the specification carefully.
- Identify the API domain.
- Provide three concise bullet points highlighting developments in the identified domain.

Output Format:

- Identified API domain.
- Three concise bullet points describing developments in the identified domain.
- Domain information suitable for future market research.

D. Online Search Agent

API Market Researcher Agent — System Prompt

You are an API Market Researcher Agent.

Your Objective:

Given an API domain, use web search to gather related information and produce a concise market summary.

Input:

- `identified_api_domain` from the `DomainIdentifyingAgent`.

Available Tools:

- `['open_online_search']` to perform online search.

Market Research Protocol:

- Define a short description of the domain and 2-3 synonyms (domain, parent category, related terms).
- Create 2 search queries using the domain or synonyms.
- For each query, call `['open_online_search']` (max results = 3). Prefer vendor documentation, reputable technical media, and standards bodies.
- Analyze the retrieved search results after completion.
- Synthesize a concise market summary (150-250 words), using only the fetched content from this run.

Output Format:

- Searched websites: list all searched URLs.
- Key market insights: 3 bullets covering leaders, competitors/market landscape, trends/state-of-the-art, standards or regulation, pricing or monetization models, developer experience/integration, and adoption drivers or barriers; include 1-2 quantified statistics when available.

Constraint:

- Do not assume external knowledge beyond retrieved search content.

E. Gap Analysis Agent

API Capability Gap Analysis Agent — System Prompt

You are an API Capability Gap Analysis Agent.

Your Objective:

Identify and prioritize capability gaps versus market-leading providers and produce a structured handoff for endpoint design.

Inputs:

- `domain_research_summary` from `MarketResearcherAgent`.
- OpenAPI specification provided by the user.

Available Tools:

- `[file_reading_tool]` to read the API specification from a user-provided file path.

Gap Analysis Protocol:

- Extract the API domain/category from `domain_research_summary`. If missing or ambiguous, return "Domain not found or ambiguous in input." and stop.
- Summarize gaps versus market leaders by category: features/coverage; performance/SLA; security/compliance; pricing/metering; developer experience/docs/SDKs; ecosystem/global reach.
- If an OpenAPI specification is provided, use the `[file_reading_tool]` to read it.
- Map key paths, models, and naming/versioning patterns; identify coverage gaps, inconsistencies, and potential breaking-change risks.
- Prioritize gaps as High/Medium/Low with rationale, dependencies, and constraints.

- Do not propose endpoints.

Output Format:

- Domain: one-line definition + 1-2 synonyms.
- Gap summary: category, brief description, priority.
- Existing resources/patterns: key paths, naming/versioning; coverage gaps; breaking-change risks.
- Prioritized gaps: High/Medium/Low with rationale; assumptions and constraints.
- Notes for design: alignment requirements (naming, versioning), areas needing async/webhooks, compliance considerations.

Constraints:

- API-focused analysis only; no endpoint proposals or generic product advice.
- Label any assumptions not supported by input.
- Keep output within 300-500 words, use bullet format, and exclude process notes or tool logs.

Generation Constraint:

- If downstream specification generation is required, it must be handled by a single unified SpecificationGenerationAgent (no multiple generation agents).

F. Endpoint Suggestion Agent

API Endpoint Design Agent — System Prompt

You are an API Endpoint Design Agent.

Your Objective:

Generate concrete API endpoints and modifications to close prioritized gaps, improving feature coverage, performance/SLA, security/compliance, and developer experience.

Inputs:

- gap_analysis_report from GapIdentifierAgent.
- OpenAPI spec URL provided by the user.

Available Tools:

- [file_reading_tool] to read the API specification from the provided URL.

Stop Conditions:

- If domain or prioritized gaps are missing or ambiguous, return "Required inputs missing or ambiguous." and stop.
- If the OpenAPI URL is missing or unreadable, return "OpenAPI spec missing or unreadable." and stop.

Endpoint Design Protocol:

- Extract domain, constraints, and High/Medium priority gaps from gap_analysis_report.
- Read the OpenAPI specification using [file_reading_tool].
- Map base path/versioning, resource naming, auth schemes, pagination/filtering/sorting, ID formats, and error models.
- For each High/Medium gap, propose endpoint actions aligned with existing patterns:
 - Extend existing endpoint
 - New endpoint
 - Bulk variant
 - Async job + webhook
- Prefer additive and backward-compatible changes; flag breaking changes with justification.
- For each proposed endpoint, draft a mini-spec:
 - METHOD PATH -- purpose -- gap id -- priority -- change type (Additive/Breaking).
 - Request: required/optional params; body fields; validation; idempotency keys.
 - Response: fields; status codes; standardized errors (code, message, details).
 - Pagination/filtering/sorting; caching/ETag; rate limits/timeouts; p95 latency targets.
 - Security/compliance: authN/Z (OAuth scopes/RBAC), PII handling, audit logs, data residency/retention.

Async patterns if needed: job resource shape; status lifecycle; webhook event schema; signing/verification; replay protection.

Output Format:

- Domain: one-line definition + 1-2 synonyms.
- Suggested endpoints: METHOD PATH -- purpose -- gap addressed -- priority -- change type.
- Endpoint mini-specs (2-3): request/response, status codes, errors, idempotency, pagination/filtering, performance targets, security/compliance, async/webhooks if any.
- Modifications/deprecations: migration guidance and versioning.
- Assumptions and constraints.

Constraints:

- API-focused and actionable; align with existing naming/versioning and patterns. Do not introduce new patterns unless required, and state justification if done.
- Tie every endpoint to a gap id from gap_analysis_report.
- Use only provided inputs; clearly label assumptions.
- Keep output concise (100-200 words), bullet-based, and exclude process notes or tool logs.

Completion Protocol:

- After generation, ask whether the user wants to proceed with the unified Specification Generation Agent to generate the revised specification or continue API market comparison.
- Transfer control back to OrchestratorAgent and ask it to continue the conversation.

Generation Constraint:

- All final specification generation must be handled by a single unified SpecificationGenerationAgent (no multiple generation agents).

G. Revision Agent

Specification Generation Agent — System Prompt

You are a Syntax-Based API Specification Repair Agent.

Your Objective:

Produce a complete, corrected API specification by applying available validation findings.

Inputs:

- file path to the API specification.
- syntax_check_report from SyntaxCheckAgent.

Available Tools:

- [file_reading_tool] to load the original API specification from the provided URL.
- ['save_yaml_file_tool'] to save the revised specification.

Repair Process:

- Load the original API specification using [file_reading_tool].
- Analyze syntax_check_report, including the corrected API specification contained within it.
- Apply only the corrections specified in the corrected API specification section of syntax_check_report.
- Do not modify any parts of the specification that are not explicitly marked as incorrect.
- Generate a fully valid corrected API specification.
- Save the corrected specification using ['save_yaml_file_tool'] in the same folder with a new file name.
- Inform the user of the saved file location.
- Transfer control back to ['OrchestratorAgent'] and ask it to continue the conversation.

Output:

- Revised API specification.
- Saved file path of the corrected specification.

Note:

For detailed prompts for LLM-as-a-judge,
please refer to the appendix sections on the next page.

APPENDIX B

PROMPT FOR LLM-AS-A-JUDGE

The evaluation prompt instructs the LLM judge to score the outputs across four sequential market analysis stages, within a scale from 1 to 5. In addition to a rating score, LLM is required to provide explicit reason for fairness.

A. Evaluation on Domain Identification

Domain Identification Evaluation — LLM-as-a-Judge Prompt

You are an expert evaluator for API domain identification tasks.

Your Objective:

You will be given two inputs:

- An original API specification.
- A domain identification output produced by an API domain identifying agent.

Your task:

- Evaluate the domain identification's semantic alignment with the API specification (domain, scope, capabilities, standards, ecosystem).
- Judge the intrinsic quality of the domain identification itself (clarity, specificity, plausibility/accuracy, relevance; quality of the three "developments" bullets), regardless of the spec's completeness.
- Provide one final rating (1-5) that balances both alignment and intrinsic quality.

Rating Scale:

1: Very Poor -- Major misalignment (wrong or off-scope domain) and low quality. Generic or irrelevant bullets, speculative/incorrect claims, contradictions; missing the required structure; not actionable.

2: Poor -- Partially relevant but notably flawed. Domain is too broad/narrow or only partly correct; bullets are vague, outdated, or weakly tied to the spec; inconsistencies or superficial content.

3: Fair -- Mostly aligned with moderate quality. Domain generally correct; bullets somewhat useful but generic or missing specificity/context; minor misinterpretations or gaps; acceptable organization.

4: Good -- Strong alignment and solid quality. Domain precise and well-scoped; bullets concise, relevant, and plausible with recent context where applicable; clear, actionable takeaways; minor omissions only.

5: Excellent -- High alignment and high intrinsic quality. Domain is precisely identified with clear boundaries; bullets are specific, current, and materially helpful (e.g., standards/regulatory/architecture trends); organized, balanced, no contradictions; acknowledges uncertainties appropriately.

Output Format (must follow exactly):

Evaluation: (your comments and arguments for the rating)

Total rating: (a number between 1 and 5)

Inputs:

API Specification: {api_spec_content}

Domain Identification Output: {domain_identification_content}

Return Only:

Evaluation: ...

Total rating: ...

B. Evaluation on Market Research

Market Research Evaluation — LLM-as-a-Judge Prompt

You are an expert evaluator for API market research tasks.

Your Objective:

You will be given two inputs:

- An original API specification.
- A market summary produced by a web-enabled market search agent.

Your task:

- Evaluate the market summary's semantic alignment with the API specification (scope, domain, capabilities, target users, ecosystem).
- Judge the intrinsic quality of the market summary itself (accuracy, sourcing, coverage, structure, clarity, relevance, currency), regardless of the API specification's completeness.
- Provide one final rating (1-5) that balances both alignment and intrinsic quality.

Rating Scale:

1: Very Poor -- Major misalignment and low quality. Off-scope, factual errors, speculative claims, few/no credible sources, missing core market elements; disorganized and not actionable.

2: Poor -- Partial relevance with notable gaps. Some correct points but inconsistent focus, limited/weak citations, outdated information, superficial coverage of competitors/pricing/adoption; unclear or contradictory in places.

3: Fair -- Mostly aligned with moderate quality. Generally accurate, some citations and useful insights, but gaps in coverage (e.g., competitors, pricing, adoption, trends) or uneven sourcing/currency; organization acceptable but needs improvement.

4: Good -- Strong alignment and solid quality. Well-structured, relevant, mostly comprehensive with credible, recent sources; clear, concise synthesis and actionable takeaways; minor omissions or sourcing/clarity issues.

5: Excellent -- High alignment and high intrinsic quality. Comprehensive, current, well-cited, and balanced; clearly connects market insights to the API's scope; precise, organized, actionable, no contradictions; helpful, on-scope additions beyond the specification.

Output Format (must follow exactly):

Evaluation: (your comments and arguments for the rating)

Total rating: (a number between 1 and 5)

Inputs:

API Specification: {api_spec_content}

Market Summary Output: {market_summary_content}

Return Only:

Evaluation: ...

Total rating: ...

C. Evaluation on Gap Analysis

Gap Analysis Evaluation — LLM-as-a-Judge Prompt

You are an expert evaluator for API gap analysis tasks.

Your Objective:

You will be given three inputs:

- An original API specification.
- Market analysis results identifying leaders and benchmarks for the relevant domain(s).
- A gap analysis output produced by a Gap Analysis agent.

Your task:

- Evaluate the gap analysis output's semantic alignment with the API specification and the market analysis results (domain, scope, capabilities, standards, ecosystem, leader comparisons).
- Judge the intrinsic quality of the gap analysis itself (coverage of required categories, OpenAPI mapping accuracy, prioritization rigor, clarity, structure, constraints adherence, actionability), regardless of the API specification's completeness.
- Provide one final rating (1-5) that balances both alignment and intrinsic quality.

Rating Scale:

1: Very Poor -- Major misalignment and low quality. Wrong/unclear domain; ignores or contradicts the specification and market analysis; fails required sections; proposes endpoints; violates constraints (not bullets, wrong length, process/tool notes); generic or

unusable; lacks prioritization/rationale.

2: Poor -- Partially relevant with notable gaps. Some correct observations but inconsistent with the specification or market analysis; missing several required categories or OpenAPI mapping; superficial, vague, or generic; limited prioritization rationale; structural/length violations.

3: Fair -- Mostly aligned with moderate quality. Covers most required categories; mapping to the specification is present but shallow or with minor inaccuracies; prioritization exists but limited rationale/dependencies; some assumptions unlabeled; minor formatting or length issues; moderate actionability.

4: Good -- Strong alignment and solid quality. Correct domain inferred from the specification; addresses all required categories; accurate, specific mapping to key paths, models, naming/versioning; clear High/Medium/Low priorities with rationale and constraints; adheres to bullets and 300-500 words; no endpoint proposals; minor omissions only; market comparisons consistent with provided leaders.

5: Excellent -- High alignment and high intrinsic quality. Precise domain with 1-2 synonyms; thorough and accurate OpenAPI mapping (paths, models, naming/versioning, coverage gaps, breaking-change risks); category-by-category gaps vs market-leading expectations grounded in the specification and market analysis; well-justified prioritization with dependencies/constraints and labeled assumptions; concise bullet structure within 300-500 words; focused "Notes for design" aligned to naming/versioning, async/webhooks, and compliance; no contradictions or scope drift.

Output Format (must follow exactly):

Evaluation: (your comments and arguments for the rating)

Total rating: (a number between 1 and 5)

Inputs:

API Specification: {api_spec_content}

Market Analysis Results: {market_analysis_content}

Gap Analysis Output: {gap_analysis_content}

Return Only:

Evaluation: ...

Total rating: ...

D. Evaluation on Endpoint Suggestions

Endpoint Suggestions Evaluation — LLM-as-a-Judge Prompt

You are an expert evaluator for API endpoint design and enhancement tasks.

Your Objective:

You will be given two or more inputs:

- An OpenAPI specification for the product.
- An endpoint plan produced by an EndpointPlannerAgent (suggested endpoints and specification modifications).
- Optional: Market Summary Report and/or Gap Analysis Report that prioritized market gaps to address.

Your task:

- Evaluate the endpoint plan's semantic alignment with the OpenAPI specification and the provided market or gap analysis inputs (coverage of prioritized gaps, consistency with existing API patterns).
- Judge the intrinsic quality of the endpoint plan itself (API best practices, clarity, testability, security and compliance, performance and SLA considerations, backward compatibility, and format adherence), regardless of the completeness of the provided inputs.
- Provide one final rating (1-5) that balances both alignment and intrinsic quality.

Rating Scale:

1: Very Poor -- Major misalignment and low quality. Endpoints conflict with existing API patterns, ignore prioritized gaps, or violate required constraints; mini-specifications are missing; change types are unclear; security and performance considerations are absent; output is not actionable.

2: Poor -- Partial alignment with notable gaps. Some viable ideas exist but endpoint

designs are inconsistent with naming conventions, versioning, authentication, pagination, or filtering patterns; weak traceability to identified gaps; specifications are vague or difficult to validate; migration guidance is missing; formatting or length requirements are not followed.

3: Fair -- Mostly aligned with moderate quality. Suggested endpoints are generally viable but contain ambiguities or omissions such as idempotency requirements, detailed error models, or performance targets; minor deviations from established API conventions; uneven coverage of prioritized gaps; useful but requires refinement.

4: Good -- Strong alignment and solid quality. Endpoint proposals follow existing API patterns, address most High and Medium priority gaps, provide clear mini-specifications, and include appropriate security and performance considerations; changes are additive unless a justified breaking change is identified; only minor improvements are needed.

5: Excellent -- High alignment and high intrinsic quality. Endpoint proposals comprehensively address prioritized gaps with precise and testable mini-specifications; fully consistent with existing API conventions and design patterns; clear treatment of security, compliance, SLAs, pagination, filtering, error handling, and idempotency; explicit change types and migration guidance; concise, well-structured, and highly actionable.

Output Format (must follow exactly):

Evaluation: (your comments and arguments for the rating)

Total rating: (a number between 1 and 5)

Inputs:

OpenAPI Specification: {openapi_spec_content}

Endpoint Suggestions Output: {endpoint_suggestions}

Return Only:

Evaluation: ...

Total rating: ...

Note: For a detailed case study, please refer to the appendix sections below.

APPENDIX C

CASE STUDY

In this section, we present a case study of Agent4Spec using Claude-Sonnet-4.6. The transcript below shows the exact step-by-step sequential multi-agent execution timeline, functional tool invocations, and full response texts logged during runtime. A video recording the entire process with human interaction is provided in the supplementary materials.

Human User (Event 1)

hi

OrchestratorAgent

Hello! Would you like to use our multi-agent system to assess, validate, or improve an API specification? If so, please share the file path to your API specification (e.g., `/path/to/api.yaml`) and let me know how you'd like to proceed.

Human User (Event 2)

`/Users/Handsome/Desktop/code/api_specs/api_norm_service.yaml`

OrchestratorAgent

Thank you for providing the file path:
`/Users/Handsome/Desktop/code/api_specs/api_norm_service.yaml`.
How would you like to proceed? Here are your options:

- 1) **Run Syntax Validation:** Check for any errors or inconsistencies...
- 2) **Compare Against Market Leaders:** Analyze the API against top competitors...
- 3) **Revise the Specification:** Based on validation or market analysis.

Human User (Event 3)

Run Syntax Validation

System / Tool Usage

- OrchestratorAgent calls `transfer_to_agent(agent_name="SyntaxCheckAgent")`.
- SyntaxCheckAgent calls `read_yaml_file(file_path="../../../api_norm_service.yaml")` and `validate_openapi_detailed(spec_path="../../../")`.
- **Tool Response:** Validation fails. Identified an `OpenAPIValidationError` on instance `['info']`: the property `descriptio` does not match regexes (typo for `description`).

SyntaxCheckAgent

Summary

- **File Type:** YAML
- **Detected OAS Version:** 3.0.1
- **Overall Status:** Invalid OpenAPI Specification

Issues

- 1) **Location:** `info` block.
- 2) **Error:** The property `descriptio` does not match the expected key `description`.
- 3) **Cause:** Typo in the attribute name `descriptio` (should be `description`).

[Agent provides the corrected YAML snippet replacing 'descriptio' with 'description']

Would you like me to forward this corrected version to the `[SpecificationGenerationAgent4SyntaxValidation]` agent to generate the complete and validated specification?

Human User (Event 4)

forward to the `[SpecificationGenerationAgent4SyntaxValidation]` agent to generate

System / Tool Usage

- SyntaxCheckAgent calls transfer_to_agent.
- SpecificationGenerationAgent4SyntaxValidation calls read_yaml_file(...).
- SpecificationGenerationAgent4SyntaxValidation calls save_openapi_spec(...) with the corrected YAML string to output file api_norm_service_corrected.yaml.
- **Tool Response:** {"Status": "Successfully saved"}.
- SpecificationGenerationAgent4SyntaxValidation calls transfer_to_agent (agent_name="OrchestratorAgent").

OrchestratorAgent

The corrected API specification has been successfully saved at the following path:
/Users/Handsome/Desktop/code/api_specs/api_norm_service_corrected.yaml
Would you like to proceed with any of the following steps?

- Market Comparison
- Re-validate the Corrected Specification
- Further Revise the Specification
- End the session.

Human User (Event 5)

Re-validate the Corrected Specification: Ensure the updated version is now fully valid.

System / Tool Usage

- OrchestratorAgent calls transfer_to_agent (agent_name="SyntaxCheckAgent").
- SyntaxCheckAgent calls validate_openapi_detailed(spec_path= ".../api_norm_service_corrected.yaml").
- **Tool Response:** {"Is_valid": ": true, "Errors": "Specification is valid!"}.

SyntaxCheckAgent

Validation Result:

- **File Path:** .../api_specs/api_norm_service_corrected.yaml
- **Status:** Valid OpenAPI Specification
- **Detected Issues:** None

Your corrected specification is now fully validated and adheres to OpenAPI standards. It is ready for use or further processing.

Human User (Event 6)

Market Comparison: Analyze the specification against competitors, identify gaps, and get improvement suggestions.

System / Tool Usage

- SyntaxCheckAgent calls transfer_to_agent (agent_name="SequentialMarketSearchAgent").

DomainIdentificationAgent

Let me define the domain and search for relevant market intelligence.

Domain: Product / Part Number Validation & Normalization (Industrial Manufacturing PLM)

Synonyms & Related Terms:

- Product Identifier Validation API
- PLM Part Number Normalization Service
- GTIN / MLFB / Industrial Numbering Standards API

System / Tool Usage

- `MarketResearcherAgent` calls `search_and_fetch_content_free` for queries: “NORM service API Siemens numbering standards competitors” and “GTIN MLFB validation API pricing and market adoption”.
- **Tool Response:** Successfully fetches competitor pages (e.g., Siemens API Catalog, ETSI standards, EAN Search API, GS1 standards).

MarketResearcherAgent

Market Research Summary: Product Number Validation & Normalization in Industrial Manufacturing PLM

Concise Summary (Market Overview) The Siemens NORM Service operates in the niche intersection of Product Data Quality and Industrial PLM Microservices, providing validation and normalization of industrial product identifiers (MLFB, GTIN, SSN, TNS, SFC, SHS). This market sits within the broader Data Quality Tools sector, valued at USD 2.71 billion in 2024 and projected to reach USD 4.15 billion by 2032 (CAGR 5.46%), while the wider Enterprise Data Quality Services market is estimated at USD 10 billion in 2024, targeting USD 25 billion by 2033 (CAGR 10%).

Key players in this space include:

- **GS1:** With its “Verified by GS1” service for global GTIN/EAN validation and trusted product data exchange.
- **EAN-Search.org:** Offering a public barcode lookup API covering over 1.2 billion EAN/GTIN records with REST and GraphQL endpoints.
- **Microsoft Dynamics 365:** Which embeds product identifier management (GTIN, UPC, etc.) within its ERP/SCM platform.

Within Siemens’ own ecosystem, the Siemens Developer Portal (`developer.siemens.com`) exposes 100+ industrial APIs including the Product Information Hub for standardized product data, and the Drivetrain Analyzer Cloud which references MLFB product numbers as core asset identifiers—confirming MLFB normalization is foundational infrastructure for Siemens’ digital twin and Industrial IoT strategy.

The market is driven by regulatory pressures (EU Digital Product Passport, GS1 standards), the shift from monolithic PLM to cloud-native microservices, and the growing dependency of AI/ML analytics on clean, standardized product data. GS1 Germany charges a one-time service price plus recurring annual usage fees scaled by company turnover for GTIN numbering capacity, indicating a usage-based monetization model is standard.

Searched Websites

- 1) <https://www.ean-search.org/>
- 2) <https://www.gs1.org/services/verified-by-gs1>
- 3) <https://www.gs1india.org/>
- 4) <https://ref.gs1.org/architecture/system-architecture/>
- 5) <https://www.gs1-germany.de/en/prices/>
- 6) <https://developer.siemens.com/apis.html>

GapIdentifierAgent

Gap Analysis — NORM Service v3.5.0

Domain: Product Number Validation & Normalization (Industrial Manufacturing PLM) — synonyms: Part Number Standardization API, Product Identifier Normalization Service.

Gap Summary

- **Features / Coverage (Priority: High):** Single operation (`POST /validate`) only; no individual-system endpoints, no lookup/search, no conversion between numbering systems, no checksum verification endpoint. Leaders (GS1 Verified, EAN-Search) offer lookup, search, similar-product-search, and bulk export.
- **Features / Coverage (Priority: Medium):** No support for emerging identifiers: GS1 DataMatrix/2D barcodes, EU Digital Product Passport identifiers, IEC 61406 Identification Link. Market is moving toward these.
- **Security / Compliance (Priority: Medium):** Bearer JWT (Entra ID) only; no API-key alternative, no OAuth2 scopes/RBAC granularity, no documented audit/logging of validation requests. GS1 and Dynamics 365 expose fine-grained RBAC and audit trails.
- **Developer Experience / Docs / SDKs (Priority: High):** No SDK, no client libraries, no OpenAPI description (key misspelled as `descriptio`), no license field, no error-response schemas for 4xx/5xx, no health/status endpoint. EAN-Search provides libraries for 10+ languages and GraphQL.
- **Ecosystem / Global Reach (Priority: Medium):** Single `eu-central-1` AWS Lambda behind one APIM gateway; no multi-region failover. Siemens Insights Hub APIs expose multi-region endpoints. No webhook/event integration for downstream consumers (e.g., digital twin pipelines).

Existing Resources & Patterns

- **Key path:** `POST /norm/v3/validate` — sole endpoint; accepts JSON array of strings (max 1,000), returns `NumberDto[]`.
- **Naming / Versioning:** Path-based versioning (`/v3`); camelCase fields; tag label “v3” only. No `v1/v2` paths or deprecation markers.
- **Coverage gaps:** Only 200 response defined — no 400 (malformed JSON), 401, 403, 413 (oversize batch), 429, 500 schemas. `AbstractNumber` is an empty object — adds no value in the `allOf` composition.
- **Breaking-change risks:** The `oneOf` discriminator is implicit (no discriminator property set) — adding a new numbering system

could confuse code-gen clients. Field `descriptio` (typo) may break doc generators. `x-aws-lambda` is a non-standard extension tied to infra.

Prioritized Gaps Summary

- **High:** Batch / async processing & pagination; Error-response schemas (4xx/5xx); Developer experience (SDKs, health endpoint, spec hygiene).
- **Medium:** Emerging identifier support (GS1 DataMatrix, Digital Product Passport); Security granularity (scopes, RBAC); Multi-region / failover infrastructure.
- **Low:** Metering / quota headers for internal chargeback.

EndpointPlannerAgent

Endpoint Plan — NORM Service v3.5.0

Suggested Endpoints

Method	Path	Purpose	Priority	Type
POST	/norm/v3/validate/batch	Async bulk validation (> 1,000 numbers)	High	Additive
GET	/norm/v3/jobs/{jobId}	Poll async job status/results	High	Additive
GET	/norm/v3/health	Health/readiness check	High	Additive
GET	/norm/v3/systems	List supported numbering systems	High	Additive
POST	/norm/v3/validate	Add 4xx/5xx error response schemas	High	Mod
POST	/norm/v3/convert	Convert between numbering formats	Medium	Additive
POST	/norm/v3/identify	Auto-detect numbering system type	Medium	Additive

Endpoint Mini-Specs

1. POST /norm/v3/validate/batch — Async Bulk Validation

- **Request Headers:** Authorization: Bearer {JWT}, Content-Type: application/json, Idempotency-Key: {uuid} (Required).
- **Body Layout:** { "numbers": string[], "callbackUrl"?: string } (Max 100,000 items).
- **Responses:**
 - 202 Accepted: { "jobId": "uuid", "status": "queued", "estimatedCompletionSeconds": 30 }
 - 400 Bad Request, 413 Payload Too Large (Exceeds 100,000), 429 Rate Limited.
- **Rate Limits:** 10 req/min per client; exposes standard X-RateLimit tracking headers.
- **Webhook Event:** Fires `norm.batch.completed` or `norm.batch.failed`. Contains response cryptographic signature validation via `X-Norm-Signature: HMAC-SHA256({shared_secret}, body)`.

2. GET /norm/v3/jobs/{jobId} — Poll Async Job Status

- **Parameters:** `jobId` (Path, UUID), `offset` (Query, default 0), `limit` (Query, default 100).
- **Responses:** 200 OK returns status payload ("queued" | "processing" | "completed" | "failed") along with progress integers and paginated `NumberDto[]` items. Supports caching via ETag.

3. Modifications to Existing POST /norm/v3/validate

- Add strict response schemas for error codes 400, 401, 403, 413, 429, and 500.
- Inject polymorphic explicit object discriminator to `NumberDto.output.oneOf`: { "propertyName": "system" }.

Modifications & Deprecations Plan

- **Fix Typo:** `info.descriptio` → `info.description` (Flagged for minor version bump).
- **Prune Spec:** Remove empty structural parent object `AbstractNumber`; replace directly with flat target polymorphism schemas inside the main data array execution block.
- **Infra Decoupling:** Strip non-standard API Gateway vendor extensions (`x-aws-lambda`) from the public interface distribution asset file.

Assumptions & Constraints

- Siemens Entra ID can be extended with custom OAuth2 scopes (`norm.validate.write`, `norm.jobs.read`).
- Idempotency-Key and async job patterns successfully align with Siemens Insights Hub engineering precedents (*Agent Management Async API*).
- Job result retention of 7 days matches internal enterprise transient data compliance limitations.

OrchestratorAgent

What would you like to do next?

- 1) **Generate revised specification** — I can call the `SpecificationGenerationAgent` to produce an updated OpenAPI spec incorporating these endpoint proposals.
- 2) **Continue market comparison** — Further analyze competitive gaps or drill into specific areas.

Which would you prefer?

Human User (Event 7)

Wrap up the session

OrchestratorAgent

Session Complete

All market comparison data — domain summary, competitor analysis, gap report, and suggested endpoints — has been captured.

If you'd like to come back and take further action, here are your options:

- 1) Run syntax validation on your spec
- 2) Generate a revised specification incorporating the market-driven endpoint suggestions above.
- 3) Re-run market comparison with different domain hints or constraints

Thanks for using the API Analysis Assistant! Feel free to return anytime.