AttributeForge: An Agentic LLM Framework for Automated Product Schema Modeling

Yunhan Huang[1], **Klevis Ramo**[2], **Andrea Iovine**[2], **Melvin Monteiro**[3], **Sedat Gokalp**[1], **Arjun Bakshi**[1], **Hasan Turalic**[2], **Arsh Kumar** [2], **Jona Neumeier**[2], **Ripley Yates**[1], **Rejaul Monir**[3], **Simon Hartmann**[2], **Tushar Manglik**[3], **Mohamed Yakout**[1]

[1]Amazon, Seattle, WA, USA, [2]Amazon, Munich, Germany, [3]Amazon, New York, NY, USA

Correspondence: yunhanh@amazon.com

Abstract

Effective product schema modeling is fundamental to e-commerce success, enabling accurate product discovery and superior customer experience. However, traditional manual schema modeling processes are severely bottlenecked, producing only tens of attributes per month, which is insufficient for modern e-commerce platforms managing thousands of product types. This paper introduces AttributeForge, the first framework to automate end-to-end product schema modeling using Large Language Models (LLMs). Our key innovation lies in orchestrating 43 specialized LLM agents through strategic workflow patterns to handle the complex interdependencies in schema generation. The framework incorporates two novel components: MC2-Eval, a comprehensive validation system that assesses schemas against technical, business, and customer experience requirements; and AutoFix, an intelligent mechanism that automatically corrects modeling defects through iterative refinement. Deployed in production, AttributeForge achieves an 88× increase in modeling throughput while delivering superior quality: a 59.83% Good-to-Good (G2G) conversion rate compared to 37.50% for manual approaches. This significant improvement in both speed and quality enables e-commerce platforms to rapidly adapt their product schemas to evolving market needs.

1 Scaling Product Schemas

E-commerce services rely on comprehensive and well-structured product information to provide customers with accurate details, enable effective search and filtering, and facilitate informed purchasing decisions. However, managing product schemas at scale poses significant challenges. Products span thousands of product types, each with unique attribute requirements, and product information evolves constantly as new products and variants are introduced. For example, new products such as smart rings require unique attributes like 'sensor_description', 'battery_life', i.e., attributes that didn't exist in traditional jewelry schemas yet were crucial for customer experience. Traditionally, defining and maintaining product schemas has been a manual process, relying on subject matter experts to identify relevant attributes, model their structures, and ensure data quality.

This manual approach has several limitations: it is time-consuming, error-prone, and struggles to keep pace with the rapidly changing product landscape. Additionally, it often results in incomplete or inconsistent schemas, as experts may overlook certain attribute requirements or introduce subjective biases during the modeling process. These shortcomings can lead to poor customer experiences, with incomplete or inaccurate product information, and missed opportunities for driving sales and customer satisfaction.

1.1 The Schema Creation Lifecycle

As illustrated in Figure 1, the creation of product attributes follows a comprehensive lifecycle comprising four stages: discovery, modeling, assessment, and validation. To illustrate this process, we use 'maximum_altitude' for drones as an example.

In the discovery phase, multiple data sources are analyzed to identify relevant attributes. For instance, analysis of customer reviews and search queries revealed frequent questions about maximum flying height for drones.

The modeling phase, which is our focus, involves creating structured representations of the attribute. For maximum_altitude, this includes defining its name (maximum_altitude), datatype (UnitIntegerVariant@1.0), descriptions (e.g., "The highest elevation the item can safely operate relative to sea level"), units ("meters", "feet"), and example values ("6000 meters", "9800 feet").

During assessment, the attribute's business value is evaluated. For maximum_altitude, it was deter-

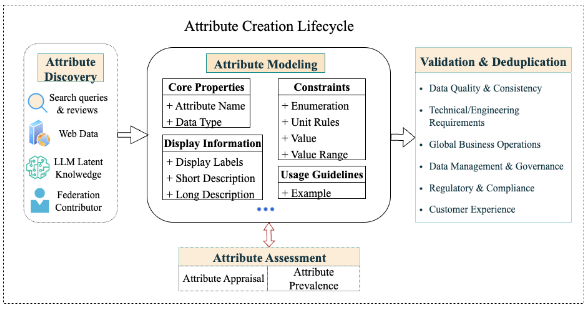

Figure 1: Attribute creation lifecycle.

mined to be somewhat customer-relevant particularly for specialized operations in high-altitude environments.

Finally, validation ensures the attribute meets multiple requirements: technical (e.g., lowercase naming convention), business rules (appropriate unit selections), customer experience (clear, non-circular descriptions), and non-duplicative (e.g., not duplicated with existing schema).

While each stage presents unique challenges, this paper focuses on the modeling and the validation phases, which represent the most complex and resource-intensive portion of the lifecycle.

1.2 Challenges

While recent LLM advances offer potential for automating schema modeling (OpenAI, 2022; Anthropic, 2024b), several key challenges remain.

First, product schema modeling requires the integration of extensive proprietary requirements that go beyond general language understanding. Each component must simultaneously satisfy multiple technical specifications (e.g., naming with lowercase_underscore format), business rules (e.g., global reusability across product categories), and domain-specific constraints. While LLMs excel at natural language tasks, they lack the specialized knowledge needed to consistently enforce these organizational standards.

Second, schema components exhibit interdependencies where changes in one element necessitate corresponding adjustments across multiple related elements. For example, modeling mattress size as either an enumerated string (king, queen) or numeric measurement (38"x75") requires different follow-up tasks - generating enumeration values or defining measurement units respectively. These interdependencies make maintaining consistency increasingly challenging as requirements grow.

Finally, schema defects have widespread impact due to their foundational role in e-commerce systems. For example, if water_resistance_level uses free text instead of enumerated values, inconsistent seller inputs could break product comparison and search features across entire categories. Given that schema errors affect millions of products, post-deployment fixes require costly data migrations.

1.3 Contributions

This paper presents *AttributeForge*, a framework leveraging orchestrated LLM agents for automated product schema modeling. Deployed in production, our system achieves an 88$\times$ throughput increase while maintaining a 59.83% Good-to-Good conversion rate, which significantly outperforming manual approaches (37.50%). Our contributions are: (1) a novel framework orchestrating 43 specialized LLM agents through strategic workflow patterns, establishing the first effective solution for automated attribute modeling; (2) MC2-Eval, a comprehensive validation framework incorporating 22 domain-specific criteria, achieving F2 scores $\geq$85% (±5% MoE) in production; (3) AutoFix, a closed-loop correction mechanism that improves modeling quality (41.88% to 59.83% Good-to-Good conversion rate) while maintaining schema consistency; and (4) production-ready tooling (PromptIDE, Tracer) and integration patterns enabling cost-effective deployment at scale, reducing modeling costs by 104X compared to manual processes.

2 Related Work

LLMs for Ontology: Recent work has explored LLMs in ontology engineering, with LLMs4OL (Babaei Giglou et al., 2023) and NeOnGPT (Fathallah et al., 2025) focusing on ontological knowledge extraction through zero/few-shot prompting. While Neuhaus (Neuhaus, 2023) highlighted challenges in maintaining logical consistency and resolving ambiguity. AttributeForge takes a different approach that focuses on structured schema modeling rather ontology construction by extracting text. **LLMs for E-commerce:** LLMs have enhanced various e-commerce aspects: Singh et.al (Singh et al., 2024) improved query auto-completion diversity by 38% while maintaining relevance, Liu et.al (Liu et al., 2024) addressed fashion recommendation challenges, and Amazon deployed LLMs for product description generation (Westmoreland, 2024b) and seller assistance (Westmoreland, 2024a). In catalog management, LLMs enabled product categorization (Cheng et al., 2024) and attribute extraction using ensemble (Fang et al., 2024) and multimodal approaches (Zou et al., 2024). However, these works focus on different use cases and assume predefined schemas, while AttributeForge is the first to automate the fundamental task of schema modeling itself.

3 AttributeForge: An Agentic Framework for Schema Modeling

AttributeForge orchestrates 43 specialized LLM agents through strategic workflow patterns, where each agent handles specific modeling tasks (Anthropic, 2024a). The framework combines robust validation with AutoFix capabilities while maintaining cost-efficient yet critical human oversight.

3.1 Framework Architecture

AttributeForge's core architecture comprises LLM-powered blocks enhanced with retrieval capabilities and memory. Each block encapsulates a specific modeling task through four components: an input schema defining required fields, an output schema specifying the response format, a prompt template containing task instructions, and execution parameters for model configuration.

Consider the attribute description generation task as an example. The input schema requires the attribute_name (unique identifier), product_type (category), and datatype (data structure). The output schema enforces a JSON format containing the generated description and reasoning. The prompt positions the LLM as a product catalog specialist with specific content guidelines from proprietary documentations stored in knowledge bases, while model configuration defines operational parameters such as batch size and temperature settings.

To manage interdependencies, the framework employs four orchestration patterns: sequential chaining for establishing foundational components before dependent ones, routing for datatype-specific processing, parallelization for concurrent execution of independent tasks, and synthesization for maintaining cross-component consistency.

The framework optimizes performance through strategic batch processing and prompt caching, reducing token and time costs by up to 5x. Batch sizes are calibrated to each task's characteristics—smaller batches (3-4 attributes) for description generation with lengthy outputs, larger batches (7-8 attributes) for tasks like datatype assignment with minimal outputs. An embedded knowledge base facilitates attribute de-duplication and consistency validation across the pipeline.

Table 1: Core Datatypes Supported in AttributeForge Framework

Datatype	Description	Example Attribute
Boolean	True/False values only	is_autographed
Decimal	Numbers with fractional values	caliber
Integer	Whole numbers for counting	drawer_count
Free String	Free-form text with language	special_features
Enumeration	set of enumerable values	orgin_country
Decimal Unit	Decimal with units	response_time
Integer Unit	Integer with units	distance_driven

Generation Steps: Our modeling pipeline consists of 13 LLM generation steps (M_1-M_{13}) orchestrated in a directed acyclic graph (DAG) workflow that transforms product concepts into fully-specified schema components illustrated in Fig. 2. Readers can refer to Appendix C for a complete list of generation steps and their description.

The workflow begins with foundational steps: M_1 concept generation consolidate different signals to discover initial attribute concepts following single-purpose tenets. For example, instead of a general "material" concept, it discover specific concepts such as "band_material" or "gem_material" to ensure clear scope. M_2 (Datatype Assignment) determines the attribute's structural foundation using 7 core datatypes (See Table 1). For example, "is_waterproof" uses Boolean, while "screen_resolution" requires String to capture values like "1920x1080". M_3 (Name Standardization) creates standardized names following conventions. For example, Boolean attributes must start with "is_", "has_", or "include_" (e.g., "is_autographed" for signed items).

Following these foundations, the workflow parallelizes descriptive components that help sellers provide accurate values: M_4 (Short Description) creates concise tooltips for data entry from seller. For example, "Provide whether this item has been personally signed by someone of note."

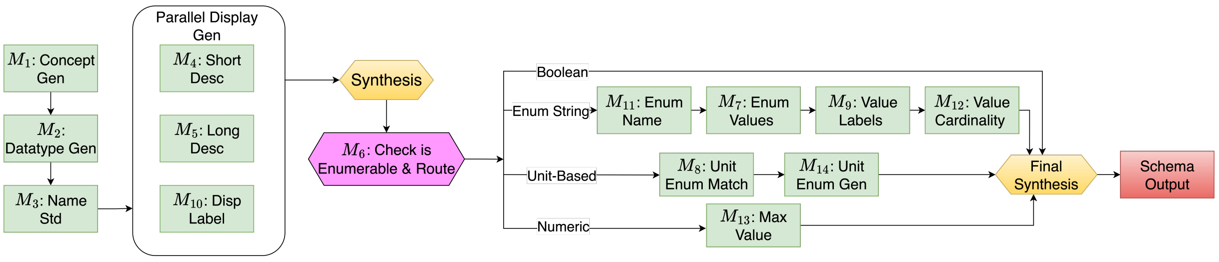

Figure 2: The AttributeForge Workflow diagram showing the DAG of generation steps.

M_5 (Long Description) generates detailed guidelines. For example, "Designates if the item bears an authentic signature from a notable individual such as an athlete, artist, or public figure. Do not use for reproduced signatures or unauthorized copies." M_{10} (Display Labels) creates user-friendly labels like "Autographed" from technical names like "is_autographed".

Routing then (M_6) directs attributes through specialized paths: Enumerable attributes (like "water_resistance_level") flow through M_{11} (Enum Name), M_7 (Enum Values: "water_resistant", "waterproof"), M_9 (Value Labels: "Water Resistant", "Waterproof"), and M_{12} (Max Occurs). Unit-based attributes require unit mapping (M_8) matching appropriate units (e.g., "response_time" using milliseconds) and constraint generation (M_{13}) defining valid ranges (e.g., 0-100ms).

4 Automated Validation

Product schema validation is critical for e-commerce platforms, as schema errors directly impact search relevance, customer experience, and operational efficiency. While LLMs offer promising automation capabilities, validating LLM-generated schemas requires sophisticated approaches beyond traditional NLP evaluation frameworks. Standard NLP metrics prove insufficient for schema validation due to three key limitations: (1) the absence of singular ground truth representations for attributes precludes the use of reference-based metrics like BLEU (Chen et al., 2021) or ROUGE (Lin, 2004), (2) reference-free metrics such as BLANC (Vasilyev et al., 2020) and Fact-CC (Kryscinski et al., 2020) evaluate only textual coherence without capturing domain-specific requirements, and (3) comprehensive schema validation demands simultaneous verification across technical, semantic, and business dimensions against precise criteria.

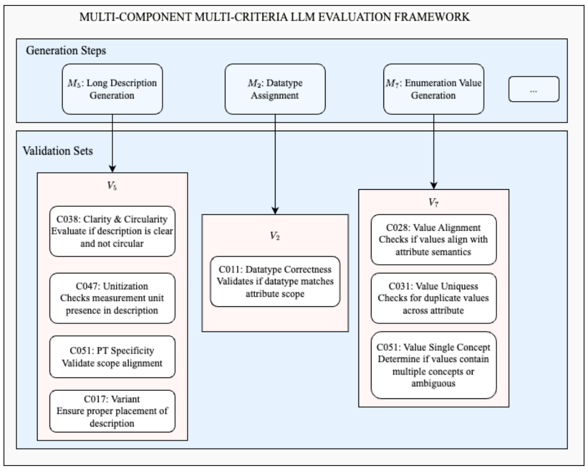
MULTI-COMPONENT MULTI-CRITERIA LLM EVALUATION FRAMEWORK
Generation Steps
M_5: Long Description Generation
M_2: Datatype Assignment
M_7: Enumeration Value Generation
...
Validation Sets
V_5
C038: Clarity & Circularity Evaluate if description is clear and not circular
C047: Unitization Checks measurement unit presence in description
C051: PT Specificity Validate scope alignment
C017: Variant Ensure proper placement of description
V_2
C011: Datatype Correctness Validates if datatype matches attribute scope
V_7
C028: Value Alignment Checks if values align with attribute semantics
C031: Value Uniquess Checks for duplicate values across attribute
C051: Value Single Concept Determine if values contain multiple concepts or ambiguous

Figure 3: Evaluators organized into validation sets (V_i) corresponding to each generation step (M_i).

4.1 MC2-Eval: Multi-Component Multi-Criteria Evaluation Framework

We introduce MC2-Eval, a framework that decomposes schema validation into atomic criteria with discrete scopes. Each criterion addresses specific validation requirements (e.g., The Attribute Relevancy criterium evaluates if attributes like battery_life are appropriate for Dishwashers). These criteria form validation sets (V_i) mapped to corresponding generation steps (M_i), enabling systematic evaluation across all schema components (see Appendix D). Each evaluator provides a binary pass/fail outcome with detailed reasoning. A generated component is considered valid only after passing all applicable criteria.

4.2 Conceptual vs. Modeling Defects

Defects are divided into two types: **Conceptual defects** represent fundamental attribute invalidity. For example: The Attribute Relevancy criterium flags irrelevant attribute-product combinations (e.g., hand_orientation for air gun projectiles). **Modeling defects** address structural and representational issues. For instance, the unit-agnostic criterium requires measurement attribute descriptions not to reference specific units, which prevents global reusability.

4.3 Evaluator Performance

MC2-Eval combines deterministic syntactic validators (e.g., for length & format requirements) with LLM-based semantic evaluators (complex semantic evaluation). While syntactic validators achieve 100% accuracy through rule-based validation, LLM-based evaluators must demonstrate F2 scores $\geq$85% with <5% margin of error on benchmark datasets before deployment. We emphasize F2 scores to prioritize defect detection, as schema errors can cascade throughout the e-commerce ecosystem, affecting search relevance, data quality, and customer experience.

5 AutoFix

We present AutoFix, a closed-loop system for automated schema refinement that extends the Reflexion framework (Shinn et al., 2023). Unlike Reflexion's episodic memory approach, AutoFix employs structured validation criteria and domain-specific prompts optimized for schema modeling. When MC2-Eval detects defects, AutoFix initiates targeted corrections through a feedback-driven iteration process. As illustrated in Figure 4, AutoFix integrates with each generation step M_i through a three-component correction cycle: 1. Original generation (M_i); 2. Defect identification (V_i-evaluators); 3; Correction synthesis (AutoFix prompt).

This cycle continues until validation criteria are satisfied or a predetermined iteration limit is reached (default: 3 iterations). AutoFix offers three key advantages: **Universal Applicability**: A single prompt architecture supports diverse generation steps and validation criteria, enabling seamless framework extension. **Holistic Correction**: Simultaneous handling of multiple defects prevents oscillating errors where fixing one issue creates another. **Systematic Learning**: Continuous capture of correction patterns enables iterative improvement of generation steps. The AutoFix prompt template and implementation details are provided in Appendix E. Section 6 demonstrates the system's effectiveness through quantitative evaluation across multiple validation criteria.

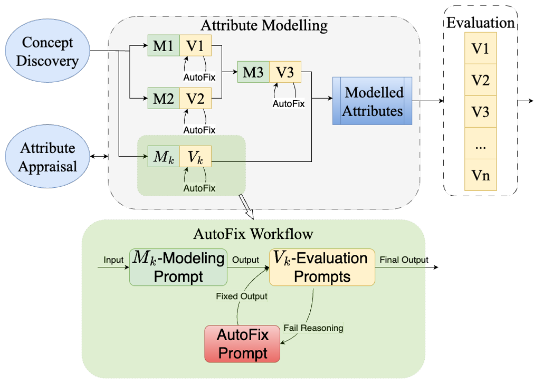

Figure 4: Abstracted AttributeForge workflow with AutoFix integrated.

Table 2: Overall AutoFix Impact

Metric	w.o. AutoFix	w. AutoFix	Improv.
G2G Rate	41.88%	59.83%	+17.95%
Defect Rate	5.75%	3.80%	-1.95%

6 Performance and Impact Analysis

Our evaluation methodology employed a rigorous multi-stage process. For MC2-Eval's LLM-based evaluators, 15 data auditors participated in the validation, with each criterion evaluated by 2 independent auditors. Sample sizes were calculated to ensure <5% margin of error, with an expert ontologist arbitrating any disagreements. The average inter-judge agreement across all 22 validation criteria was 87%.

We evaluate schema quality through a three-tier classification system: "pass" (meets all 22 criteria), "fail" (violates conceptual requirements), and "fix" (contains correctable modeling defects). Our primary metric, the G2G conversion rate, quantifies successful attribute modeling:

$$G2G = \frac{\#\text{pass}}{\#\text{pass} + \#\text{fix}}. \qquad (1)$$

The measurement of good-to-good conversion rate followed a thorough review process. Three expert ontologists (each with 5+ years experience) conducted initial evaluations independently across 392 attributes spanning 85 product types. Before finalizing pass/fix/fail decisions, ontologists cross-validated each other's assessments to ensure consistency. Using Claude 3.0 Sonnet[1], AttributeForge transforms schema modeling from a human-centric to machine-first approach, delivering an 88× throughput improvement—scaling from fewer than ten to hundreds of attributes monthly. The system achieved a 59.83% G2G rate, significantly outperforming manual modeling (37.50%) which we use as a baseline. That means of the generated

[1]Per organization policy, only Claude models are allowed on proprietary data

Table 3: Defect Rate Breakdown by Criteria (Selected Core Components due to Page Limit)

Component	**Criteria**	**Without AutoFix**	**With AutoFix**	**Defect Rate Delta**
Short Description	C012A Short Description Clarity & Circularity	18.2%	9.9%	-8.3%
	C012B Short Description Begins With	0.8%	0.0%	-0.8%
	C014 Short Description Length	0.0%	1.9%	+1.9%
	C047 Unitized Attribute Short Description	0.0%	1.7%	+1.7%
	C051B Mismatch Attribute Name and Short Description	7.4%	3.3%	-4.1%
Long Description	C017 Long Description Variant Path	0.0%	0.0%	0.0%
	C038A Long Description Circularity	0.0%	1.7%	+1.7%
	C047B Unitized attribute Long Description	0.8%	1.7%	+0.8%
	C051A Mismatch Attribute Name and Long Description	9.9%	9.1%	-0.8%
Datatype	C011 Datatype	9.1%	4.1%	-5.0%
Max Occurrence	C016A Max_Occurs	14.9%	5.0%	-9.9%
...	...	...	...	...
Aggregate All	**Total**	**5.8%**	**3.8%**	**-1.9%**

attributes, 59.83% required no human intervention, while remaining cases averaged fewer than 2 defects per attribute, resolvable within 10 minutes through our Schema Augmentation Interface. In addition to G2G rate, we also track the frequency of defect occurrence through:

$$\text{DefectRate} = \frac{\text{\# of validation failures}}{\text{\# of total applicable validation criteria}},$$

calculated as the number of validation failures divided by the total number of applicable validation criteria across all attributes. This metric provides us with a more granular view of quality by measuring the proportion of failed validations relative to all possible validation checks.

6.1 AutoFix Performance

AutoFix demonstrated significant quality improvements across multiple dimensions. As detailed in Table 2, the system increased the G2G conversion rate from 41.88% to 59.83% while reducing the overall defect rate from 5.75% to 3.80%.

Analysis of specific criteria (Table 3) reveals significant targeted improvements. In structural elements, the maximum occurrence constraint defects decreased by 880 basis points (bps, from 14.9% to 5.0%), while datatype assignment errors reduced by 500 bps (from 9.1% to 4.1%). Description quality also improved substantially, with short description clarity and circularity issues declining by 830 bps (from 18.2% to 9.9%).

While AutoFix achieved substantial improvements overall, isolated cases of defect rate increases due to inter-criteria dependencies. For instance, enhancing the clarity of 'cooling_capacity' descriptions by adding measurement units occasionally triggered unit-agnostic validation failures. These observations suggest the need for advanced multi-criteria optimization in future iterations.

6.2 Business Impact and Cost Analysis

AttributeForge demonstrates compelling business value through multiple metrics. The system reduces attribute modeling time from 20-30 hours to approximately 20 minutes—a 60× efficiency gain—while maintaining modest operational costs ($4.00-$4.50 per attribute: $1.30 for generation, $2.40 for validation, $0.50 for AutoFix). Compared to manual modeling costs, this represents a 104× reduction while achieving superior quality.

Quantitative business impacts span multiple dimensions. Product categories with expanded attribute coverage showed greater than 1% increase in ordered product sales (exact numbers restricted according to organization policy). Seller experience improved significantly through systematic elimination of irrelevant attributes and enhanced description clarity. The system enabled rapid catalog evolution, successfully modeling over 100 new attributes within hours instead of the months required by manual processes.

The system's component-specific architecture enables targeted improvement of existing schemas by isolating and correcting specific defect types. For instance, when detecting circular definitions in short descriptions, AttributeForge can systematically update them without affecting other valid components, ensuring schema integrity while reducing technical debt.

7 Conclusion

AttributeForge demonstrates three critical insights for automating complex enterprise tasks like product schema modeling. First, decomposing domain-specific workflows into specialized LLM agents enables better handling of intricate interdependencies than single-model approaches, as evidenced by our

88× throughput improvement. Second, robust validation frameworks must go beyond traditional NLP metrics to incorporate business logic and domain expertise, achieved through our MC2-Eval system with 22 technical and business criteria. Third, automated correction mechanisms can significantly reduce human intervention while maintaining quality, demonstrated by our AutoFix system improving Good-to-Good conversion rates from 41.88% to 59.83%. These learnings, validated through production deployment, provide a blueprint for applying LLMs to other structured enterprise tasks while maintaining rigorous quality standards."

Acknowledgements

We express our sincere gratitude to the Ontology team for developing comprehensive attribute modeling guidelines that served as the blueprint for the AttributeForge framework. We particularly thank Erin Hawley for sharing her thoughts and analysis of attribute generation steps and sophisticated routing mechanisms, and Eric Kauz for sharing his extensive expertise in product schema modeling and his invaluable contributions to LLM evaluator prompt engineering that underpins our schema quality. Special appreciation goes to Yvette Chua and Vanessa Franco for leading the critical human audit effort. We are also grateful to Sharadha Viswanathan and Penny Gao for their thoughtful review and valuable feedback to the writing of the paper. Their collective expertise and dedication were instrumental in bringing automated attribute modeling from concept to reality.

References

Anthropic. 2024a. Building effective agents. `https://www.anthropic.com/research/building-effective-agents`. Accessed on Sept 06, 2025.

Anthropic. 2024b. Introducing the next generation of claude. Accessed: 2025-02-05.

Hamed Babaei Giglou, Jennifer D'Souza, and Sören Auer. 2023. Llms4ol: Large language models for ontology learning. In *The Semantic Web – ISWC 2023*, pages 408–427, Cham. Springer Nature Switzerland.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zhu Cheng, Wen Zhang, Chih-Chi Chou, You-Yi Jau, Archita Pathak, Peng Gao, and Umit Batur. 2024. E-commerce product categorization with LLM-based dual-expert classification paradigm. In *Proceedings of the 1st Workshop on Customizable NLP: Progress and Challenges in Customizing NLP for a Domain, Application, Group, or Individual (CustomNLP4U)*, pages 294–304, Miami, Florida, USA. Association for Computational Linguistics.

Chenhao Fang, Xiaohan Li, Zezhong Fan, Jianpeng Xu, Kaushiki Nag, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2024. Llm-ensemble: Optimal large language model ensemble method for e-commerce product attribute value extraction. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2910–2914, New York, NY, USA. Association for Computing Machinery.

Nadeen Fathallah, Arunav Das, Stefano De Giorgis, Andrea Poltronieri, Peter Haase, and Liubov Kovriguina. 2025. Neon-gpt: A large language model-powered pipeline for ontology learning. In *The Semantic Web: ESWC 2024 Satellite Events*, pages 36–50, Cham. Springer Nature Switzerland.

Wojciech Kryscinski, Bryan McCann, Caiming Xiong, and Richard Socher. 2020. Evaluating the factual consistency of abstractive text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9332–9346, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Han Liu, Xianfeng Tang, Tianlang Chen, Jiapeng Liu, Indu Indu, Henry Peng Zou, Peng Dai, Roberto Fernandez Galan, Michael D Porter, Dongmei Jia, Ning Zhang, and Lian Xiong. 2024. Sequential LLM framework for fashion recommendation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1276–1285, Miami, Florida, US. Association for Computational Linguistics.

Fabian Neuhaus. 2023. Ontologies in the era of large language models – a perspective. *Appl. Ontol.*, 18(4):399–407.

OpenAI. 2022. Introducing chatgpt. `https://openai.com/index/chatgpt/`. Accessed: 2025-02-05.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Sonali Singh, Sachin Sudhakar Farfade, and Prakash Mandayam Comar. 2024. DiAL : Diversity aware listwise ranking for query auto-complete. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1152–1162, Miami, Florida, US. Association for Computational Linguistics.

Oleg Vasilyev, Vedant Dharnidharka, and John Bohannon. 2020. Fill in the BLANC: Human-free quality estimation of document summaries. In *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*, pages 11–20, Online. Association for Computational Linguistics.

Mary Beth Westmoreland. 2024a. Amazon launches a powerful new generative ai-based selling assistant codenamed project amelia. `https://www.aboutamazon.com/news/innovation-at-amazon/amazon-project-amelia`. Accessed on February 4, 2025.

Mary Beth Westmoreland. 2024b. Amazon launches new generative ai feature for sellers to create product descriptions. `https://www.aboutamazon.com/news/small-business/amazon-sellers-generative-ai-tool`. Accessed on February 4, 2025.

Henry Zou, Gavin Yu, Ziwei Fan, Dan Bu, Han Liu, Peng Dai, Dongmei Jia, and Cornelia Caragea. 2024. EIVEN: Efficient implicit attribute value extraction using multimodal LLM. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 453–463, Mexico City, Mexico. Association for Computational Linguistics.

8 Limitations

While AttributeForge demonstrates significant improvements in schema modeling automation, we acknowledge several important limitations:

Model Flexibility: Our current implementation uses Claude models in compliance with enterprise data policies. The framework's model-agnostic architecture enables straightforward adaptation to other LLMs, offering flexibility for different organizational needs. Our detailed workflow patterns and validation criteria provide a foundation for implementing similar systems with any capable LLM.

Schema Complexity: AttributeForge currently supports seven core datatypes that cover most e-commerce product attributes. As the system evolves, we plan to extend support to more complex schemas with nested structures and component relationships, further expanding its utility across diverse product categories.

Reproducibility & Implementation Details: While some components leverage proprietary e-commerce data that can not be shared publicly, we provide comprehensive documentation of our framework architecture, workflow patterns, and validation criteria in Appendices. Organizations can adapt these principles to their specific needs using their own product data and business rules, enabling similar automation benefits.

Resource Optimization: The system's current cost (\$4.00-\$4.50 per attribute) represents a 104× reduction compared to manual modeling while delivering superior quality. Future optimizations through prompt caching and batch processing will further improve cost efficiency, making the system increasingly accessible to organizations of various sizes.

Appendix A Schema Metadata Components

This appendix describes the structured components that define a product attribute schema:

1. **Attribute Name:** The logical name of the attribute concept in lower-case underscore format who also serves as the identifier for the attribute.

2. **Data Type:** The fundamental structure and shape of the attribute, such as BoolVariant@1.0 for boolean values, DecimalVariant@1.0 for numeric values with decimals, or LocalizedStringVariant@1.0 for text values. The datatype determines the basic validation rules and behaviors of the attribute.

3. **Display Label:** A human-readable name for the attribute used in user interfaces and documentation. Unlike the technical attribute name that follows lowercase_underscore convention, the label is formatted in Title Case for better readability.

4. **Long Description:** A comprehensive explanation of the attribute, including its purpose, usage guidelines, and any special considerations. This helps contributors understand how to properly populate the attribute.

5. **Short Description:** A concise version of the attribute description, typically used in tooltips or quick reference materials.

6. **Sample Values (Units Not Required):** Sample valid values for attributes that don't require units of measurement.

7. **Sample Values (Units Required):** Sample valid values including their units of measurement for attributes that require them (e.g., "5.2 inches", "3.4 kg").

8. **Selected Unit Enumeration Name:** The name of the pre-existing unit enumeration set being used by the attribute. Examples of pre-created enumeration set: battery_capacity_unit: [amp_hours, milliamp_hours], angle_unit: [degrees, radians].

9. **New Unit Set Name:** The name of a new unit enumeration set being created specifically for this attribute. We create new unit enumeration set when there is not a appropriate match from existing enumeration set.

10. **Enumeration Values:** For enumerated attributes, the list of valid values that conform to the enumeration tenets (lowercase with underscores, marketing-agnostic, globally applicable).

11. **Enumeration Value Display Labels:** The human-readable display labels corresponding to the enumeration values.

12. **Unit Enumeration Values:** For attributes with units, the list of valid unit values (e.g., "cm", "in", "kg").

13. **Unit Enumeration Value Display Labels:** The human-readable display labels for unit values (e.g., Btus, Kilowatts, Watts).

14. **Example Numeric Values:** The numeric portions of example values for attributes with units.

15. **Example Unit Values:** The unit portions of example values for attributes with units.

16. **Enumeration Set Name:** The identifier for the enumeration set used by the attribute, following lowercase_underscore naming convention.

17. **Attribute Value Band:** Classification of the attribute's importance and relevance for specific product types, E.g., band_A, band_B, and etc.

18. **Maximum Occurrences:** The maximum number of times an attribute can appear per selector group. For example, a max_occurs of 5 means up to 5 values can be specified for the attribute within the same selector context (e.g., marketplace, language).

Appendix B Example Prompt Structure and Configuration

This appendix provides detailed examples of prompt structure and configuration used in our attribute modeling system, specifically for long description generation:

Configuration

```
task_configuration:
  model: "anthropic.claude-3-sonnet
      -20240229-v1:0"
  temperature: 0.17
  batch_size: 5

input_data_fields:
  - attribute_name
  - attribute_label
  - attribute_explanation
  - datatype
  - product_type
  - pt_definition

output_schema:
  {
    "results": [
      {
        "product_type": "copy from input
            ",
        "attribute_name": "copy from
            input",
        "attribute_label": "copy from
            input",
        "datatype": "copy from input",
        "long_description": "long
            description text",
        "reason": "why the long
            description is correct and
            in accordance with all the
            guidelines"
      }
    ]
  }
```

Long Description Generation Prompt (Trimmed)

```
[Inputs]:
[
  {
    "attribute_name": "attribute",
    "attribute_label": "attribute label
        ",
    "product_type": "product type",
    "attribute_explanation": "sample
        explanation",
    "datatype": "datatype",
    "product_description": "
        product_description"
  }
]

<role_definition>
You are a product catalog specialist
    responsible for creating detailed
    attribute descriptions
for an e-commerce website. You will
    receive a list of attribute names.
    Each attribute
requires a long description that
    elaborates on the attribute's
    meaning and explains its
purpose in clear and concise language.
</role_definition>

<task>
Your task is to generate a long
    description for each attribute
    following these guidelines:

<guidelines>
1. Read the attribute name and the
    associated product type, and if the
    attribute name does
   not include the exact product type in
        the attribute name, the long
       description should
   not include the product type. In this
        case, use the term "item" to
       refer to the
   product in the long description.
2. The long description must provide
    additional details beyond the
    attribute name to help
   users understand the attribute's
       semantic meaning and how the
       attribute should be used.
3. When describing the attribute avoid
    repeating the exact words from the
    attribute name.
   Instead, use alternative phrasing to
       convey the same concept.
...
[The original prompt is cutted due to
    space limit]

<templates>
"[Attribute name] designates the [
    characteristics of the attribute in
    detail].
[Optional: clarification or distinction
    from other attributes, if applicable
    ]"
</templates>
</guidelines>
</task>

Produce single JSON object contains all
    the answers. Print only one JSON
    object with all
answers and do not output anything else.

<output_schema>
{
  "results": [
    {
      "product_type": "copy from input",
      "attribute_name": "copy from input
         ",
      "attribute_label": "copy from
         input",
      "datatype": "copy from input",
      "long_description": "long
         description text",
      "reason": "why the long
         description is correct and in
         accordance with all the
         guidelines"
    }
  ]
}
</output_schema>
```

Appendix C Generation Steps

Step ID	Step Name	Description	Input Schema	Output Schema
M_1	Concept Generation	Generates raw attribute concepts for a product type	product_type product_type_definition	attribute_label description example_values
M_2	Datatype Assignment	Determines appropriate datatypes for each raw concept	product_type attribute_label description	attribute_label data_type justification
M_3	Name Standardization	Generates standardized attribute names following naming conventions	product_type attribute_name attribute_explanation data_type	attribute_name corrected_name product_type attribute_explanation
M_4	Short Description Generation	Creates concise descriptions providing context and examples	attribute_name attribute_explanation datatype product_type product_type_definition	attribute_name product_type short_description reason
M_5	Long Description Generation	Creates detailed descriptions defining scope and usage	attribute_name attribute_explanation datatype product_type product_type_definition	attribute_name product_type long_description reason
M_6	Check Is Enumerable	Evaluates whether an attribute's values should be enumerated	product_type product_type_definition attribute_name data_type long_description	attribute_name product_type example_values answer reason
M_7	Enumeration Value Generation	Generates enumeration values for string concepts	product_type attribute_name data_type attribute_explanation	product_type_name attribute_name number_of_values enumeration_name
M_8	Unit Mapping	Matches unit enumeration to UnitDecimal/UnitInteger types	product_type attribute_name attribute_explanation	product_type attribute_name enumeration_units new_enum_name
M_9	Value Label Generation	Creates human-readable labels for enumeration values	product_type attribute_name attribute_explanation enumeration_name enumeration_value	product_type_name attribute_name identifier label
M_{10}	Display Label Generation	Creates user-facing display labels	attribute_name attribute_explanation	display_label
M_{11}	Enumeration Name Generation	Generates standardized enumeration names	product_type attribute_name attribute_explanation	enumeration_name
M_{12}	Max Occurs Constraint	Determines single/multiple value cardinality	attribute_name product_type product_type_definition datatype short_description long_description enum_values	attribute_name product_type max_occurs reason
M_{13}	Max Value Constraint	Determines maximum value constraints for numeric types	attribute_name product_type datatype	max_value reason

Appendix D Validation Criteria

The following table presents our comprehensive validation criteria used to assess generated attribute schemas, including their descriptions, target content types, and classifications.

Criteria ID	Criteria Name	Criteria Description	Target Content	Syntactic?	Conceptual or Modeling?
C001	Attribute Relevancy	Checks for attribute that are not relevant to the PT (product type) it has been made customer relevant for. This check is used to avoid irrelevant attribute/PT mappings.	Attribute	No	Conceptual
C002	Attribute Single Purpose	Checks for attributes that do not have a single purpose as identified by the semantics of the attribute's represented concept.	Attribute	No	Conceptual
C003	Enumeration name is 30 characters or fewer	Checks to see if enumerated list name has 30 characters or fewer.	Enumeration Name	Yes	Modeling
C006	Enumeration display labels exist for each enumeration value	Enumeration display labels exist for each enumeration value	Enumeration Display Label	Yes	Modeling
C007	Enumeration display labels should be formatted in Proper Case	Enumeration display labels should be formatted in Proper Case by capitalizing each word except for values with prepositions. For such values, use upper case for verbs, adjectives or nouns surrounding the preposition which is in lower case (ex. Mother of Pearl).	Enumeration Display Label	Yes	Modeling
C009	Attribute Single Logical Domain Concept	Checks for attribute that do not represent a single logical domain concept when applied to different PTs.	Attribute	No	Conceptual
C010	Product Detail Attribute	Checks for attributes that do not represent a universal product fact for a specific product type for example attributes that relate to an offer/sku or relationships to other asins.	Attribute	No	Conceptual
C011	Attribute Has Correct Datatype	Checks for attributes which use an incorrect reusable datatype based on the attribute's scope for example a Decimal Variant when a unit is required.	Datatype	No	Modeling
C012A	Short Description Clarity	Checks whether the short description would be clear to a reader at the 7th grade level. This check considers circularity but takes into account that many terms are obvious to the reader and do not need to be clarified in the short description.	Short Description	No	Modeling
C012B	Short Description Begins	Given the datatype and short description from the parameters, the validation checks if the short description starts with the appropriate expression; 'Provide whether' for Bool attributes and 'Provide the' otherwise	Short Description	Yes	Modeling
C013A	Attribute has a semantic match in UMP	Checks for attributes that already exists in UMP and are already relevant for product or considered universal. Exact match is defined that as the concept already exists and the data types are effectively the same.	Attribute	No	Conceptual
C013B	Attribute has a semantic match against Non-Released Attributes	Checks for attributes that overlap with non-released attributes	Attribute	No	Conceptual

Continued on next page

Continued from previous page

Criteria ID	Criteria Name	Criteria Description	Target Content	Deterministic?	Conceptual or Modeling?
C013C	Exact Match against against Non-Released Attributes	Checks for attributes that exactly matches with non-released attributes	Attribute	Yes	Conceptual
C014	Short Description Lengthy	Checks whether the short description exceeds 140 characters.	Short Description	Yes	Modeling
C015	Short Description Exists All Visible Variant Paths	Checks whether the short description is present for all variant paths defined for the attribute for example both value and unit.	Short Description	Yes	Modeling
C016A	Attribute constraints values are not aligned with the scope of the attribute Concept is repeatable but max_occurs = 1	There can be more than one occurrence of an attribute value for a product but the max_occurs is limited to 1.	Constraints	No	Modeling
C017	Long Description Only Exists at * Variant Path	Checks whether the long description only exists at the attribute level and is not present at the individual variant level.	Long Description	Yes	Modeling
C020	Required Constraints Present With Values	All constraints required for an attribute were created based on the datatype.	Attribute	Yes	Modeling
C021	Example Text and Label Exists All Visible Variant Paths	Checks whether example and description is present for all variant paths defined for the attribute for example both value and unit.	Example Values	Yes	Modeling
C028	Enumeration Value Aligns with Attribute Semantic Definition	Checks for values in the enumerated list that are not relevant to the attribute's defined purpose/scope as per the attribute's long description.	Enumeration Values	No	Modeling
C031	ENM value is a duplicate of a value with another attribute relevant for PT	Enumeration value is a duplicate or synonym of a value in an overlapping attribute.	Enumeration Values	No	Modeling
C037	Attribute names are unique within UMP	Attribute names are unique within UMP. Do not reuse an existing attribute name even if it suits the concept that you are trying to define.	Attribute	Yes	Conceptual
C038A	Criteria 38a Long Description Clarity/Circularity	Checks whether the long description would be clear to a reader at the 7th grade level. This check considers circularity but takes into account that many terms are obvious to the reader and do not need to be clarified in the short description.	Long Description	No	Modeling
C039	Attribute Syntax is valid Deterministic	Use lowercase_underscore naming convention for all attributes in all namespaces (e.g. active_ingredients). With the exception of the underscore, attribute names must not include special characters. Numbers should also be avoided except when necessary for example regulations (e.g. [california_proposition_65]). Do not start an attribute name with a number.	Attribute Name	Yes	Modeling
C047	Unitized attribute - short description	Short descriptions for a measurement attribute reference one specific unit of measure when the attribute can be measured under different measurement systems or units.	Short Description	No	Modeling

Continued on next page

Continued from previous page

Criteria ID	**Criteria Name**	**Criteria Description**	**Target Content**	**Deterministic?**	**Conceptual or Modeling?**
C047B	Unitized attribute -long description	Long descriptions for a measurement attribute reference one specific unit of measure when the attribute can be measured under different measurement systems or units.	Long Description	No	Modeling
C051	PT domain not clear or long description of attribute is PT specific but attribute is viewed/named globally.	PT specific attribute long descriptions but attribute has global name	Long Description	No	Modeling
C051B	PT domain not clear or short description of attribute is PT specific but attribute is viewed/named globally.	PT specific attribute short descriptions but attribute has global name	Short Description	No	Modeling
C053	Enumeration Value Display Label Format Characters	Enumeration Display Labels are only allowed to have the a defined set of format characters	Display Label	Yes	Modeling
C060	Attribute has potential legal/safety ramifications and should not be generated	Check is to determine whether an attribute relates to health, safety, certification or regulatory concerns.	Attribute	No	Conceptual
C064	Enumeration contains values that express more than one concept/intention modelling	Determine if any values in the enumeration represent terms that are ambiguous and convey multiple concepts or intentions simultaneously	Enumeration Values	Yes	Modeling
C068	Attribute represents a characteristic that would not provide value to a customer.	Identifies attributes that provide limited value for customer purchase decisions, including those that are obvious, subjective, redundant, non-essential, relate to replaceable components, or lack meaningful detail.	Attribute	No	Conceptual

Appendix E AutoFix Prompt Template

```
You are tasked with fixing defects in
    generated product attribute metadata
    . You will be provided with:
1. The original generation prompt
2. The content it generated
3. The validation criteria that failed
4. Detailed reasons for the validation
    failures

<generation_prompt>
{original_generation_prompt}
</generation_prompt>

Here is the content generated from this
    prompt:

<original_content>
{original_generated_content}
</original_content>

The content was evaluated using these
    validation criteria:

<validation_criteria>
{validation_criteria}
</validation_criteria>

The content failed {
    number_of_failed_checks} validation
    checks for the following reasons:

<failure_reasons>
{detailed_failure_reasons}
</failure_reasons>

Generate corrected content that
    addresses all validation failures
    while maintaining the original
    requirements. Provide your response
    in this JSON format:

{
  "corrected_content": {content
      following the original output
      schema},
  "fix_reasoning": "Concise explanation
      of changes made to address each
      failure reason"
}

Important requirements:
- Address all validation failures
    simultaneously
- Maintain compliance with original
    generation requirements
- Ensure changes don't introduce new
    issues
- Provide clear reasoning for
    corrections
```

Appendix F Development Lifecycle & Human Integration

The AttributeForge framework are integrated with two key development and test tools: PromptIDE and Tracer. PromptIDE serves as the core development environment, streamlining prompt engineering through real-time validation, version control, and deployment management. The system integrates product knowledge data and enables both targeted validation testing and comprehensive pipeline evaluation, providing non-technical prompt developers with immediate feedback on prompt effectiveness. Complementing PromptIDE, Tracer provides end-to-end pipeline monitoring and analysis capabilities, creating a complete ecosystem that supports both technical and non-technical stakeholders throughout the prompt engineering lifecycle. This dual-component approach ensures robust development capabilities while maintaining complete visibility across the entire pipeline.

PromptIDE's integration with MC2-Eval enables efficient, isolated prompt development. When improving a specific generation step (M_i), prompt engineers can focus solely on optimizing against its corresponding validation criteria set (V_i) without concern for downstream impacts. This isolated testing occurs first in the development environment using a diverse validation set of 385 attributes spanning 15 product types. Once improvements are validated, the prompt undergoes end-to-end testing in an alpha environment to verify that enhanced outputs don't adversely affect dependent steps (e.g., ensuring improved long descriptions don't break enumeration value generation). This structured approach has reduced iteration cycles from one week to approximately one hour while maintaining systematic quality control. The integration of PromptIDE into AttributeForge allows direct deployment of the changes into production by non-technical users with limited manual hand-off required.

To optimize pipeline observability and efficiency, Tracer provides end-to-end monitoring of the entire pipeline. Tracer consolidates data across multiple pipeline stages into a unified view, reducing data inspection time by 1 week per pipeline run. Tracer incorporates real-time MC2-Eval validation results, on-demand pipeline triggering capabilities, and automated metrics computation. Through implementation of an efficient caching layer, Tracer reduced result loading times by 70% reduction for frequent searches, while saving an additional 400 minutes across iteration cycles through automated metrics computation. This integration has democratized pipeline access, enabling non-technical stakeholders to independently analyze results and validate generated attributes against established quality metrics without engineering intervention.

F.1 Strategic Human Integration

While the AttributeForge framework is machine-first, we strategically position human expertise at four critical touchpoints to maximize impact while maintaining scalability:
Benchmark Dataset Curation: NLP scientists and ontologists assemble diverse, high-quality datasets representing various product types and edge cases, ensuring rigorous evaluation of our LLM evaluators.
Prompt Engineering: Domain experts use PromptIDE to develop and refine prompts based on quality metrics and systematic failure patterns.
Selective Review: Human review is triggered only for schemas failing specific validation criteria.
Continuous Improvement: Experts analyze aggregate quality metrics and experiment results to identify systematic issues and enhance generation/-validation prompts.
This focused approach optimizes human effort while maintaining high quality standards and enabling system refinement.

Appendix G Execution

The execution of AttributeForge begins by retrieving input data from data lakes (e.g., Amazon S3, Amazon Athena) through SQL queries that join relevant schema tables. The workflow orchestrator (e.g., AWS Step Functions) manages the coordination and communication between agents through distributed cloud computing, where each step runs on separate server instances (AWS Fargate or AWS Lambda) optimized for input size and performance requirements. For each task, the system prepares batched inputs, retrieves corresponding prompts, and executes them using LLM services (e.g., Amazon Bedrock). Generated outputs are validated, transformed into standardized JSON format, and stored back in the data lake. Intermediate results are made available for the downstream steps to start consuming, as well as for analysis and debugging through a query interface. The workflow continues until all schema components are generated, at which point the final synthesized schema is validated.

To ensure reliable execution, the system implements comprehensive error handling at multiple levels. For output parsing errors, a multi-stage parsing system first attempts to extract valid JSON, then employs an LLM-based JSON repair mechanism that uses a specialized prompt to fix malformed outputs while preserving their semantic content. Further parsing failures trigger an automatic retry logic that re-formats the prompt with additional structure emphasis and reduces batch size, using exponential backoff with a 1-second initial delay and maximum three retries. The system maintains detailed error logs capturing failure modes, input conditions, and system state, enabling continuous improvement of prompts and validation rules while providing valuable diagnostic information for human reviewers.