Agentic Prompt Optimization for E-Commerce Catalog Enrichment at Scale

Peng Gao[1], Athanasios N. Nikolakopoulos[1], Zhu Cheng[1], Andrea Scarinci[1], Umit Batur[1] and Suleiman A. Khan[1]

[1]*Amazon Catalog AI, Seattle, Washington, USA*

Abstract

Developing specialized large language model (LLM) prompts for domain-specific tasks at scale remains a significant hurdle, particularly for e-commerce applications managing tens of thousands of distinct product attributes. We introduce **CascadeAgent**, a multi-agent framework that automates prompt adaptation and specialization through error-driven semantic refinement. CascadeAgent employs a hierarchical architecture where a central Prompting Agent orchestrates four specialized counterparts—Writing, Generation, Evaluation, and Flaw Detection—that collaboratively analyze domain metadata, construct attribute-specific prompts, and enhance performance through iterative feedback. Our approach combines **Multi-pass Prompt Generation** (MPG) for modularity with textual-gradient-style optimization that refines instructions based on detected error patterns. We further use a simple formal model to characterize when this iterative refinement process is expected to reduce catalog loss under defined conditions.

In an internal production-scale case study, CascadeAgent generated and maintained over **27,000** product–attribute prompts. Compared with a universal-prompt baseline, it improved precision by **+21 to +33 percentage points** and coverage by **+14 to +18 percentage points** across multiple LLMs, with the largest gains observed on lower-cost models. We discuss the evaluation and deployment lessons from this setting, including precision–coverage trade-offs, negative labels for abstention, semantic verification of ambiguous cases, and common failure modes. These lessons are intended to help future work on agentic systems for product understanding and e-commerce search.

Keywords

e-commerce, product attributes, catalog enrichment, prompt optimization, language agents, information retrieval, evaluation

1. Introduction

High-quality product attributes are critical to e-commerce search, faceted navigation, recommendation, and customer trust. Automatic catalog enrichment can reduce seller burden and improve product discovery, but the task is difficult to scale. Product categories differ widely in relevant attributes, value constraints, evidence requirements, and abstention behavior. As a result, generic LLM prompts often fail to capture attribute-specific requirements, while manually writing and maintaining prompts for thousands of attributes is impractical.

To address this challenge, we introduce **Multi-pass Prompt Generation (MPG)**, a decomposition strategy that assigns dedicated prompts to individual product–attribute (PA) pairs. MPG avoids overly complex universal prompts and makes it possible to optimize each attribute independently. Building on MPG, we propose **CascadeAgent**, a multi-agent framework that creates and iteratively refines PA-specific instructions. CascadeAgent orchestrates five specialized agents—Prompting, Writing, Generation, Evaluation, and Flaw Detection—to generate initial instructions from catalog metadata, evaluate outputs, identify recurring errors, and rewrite instructions using textual-gradient-style feedback.

Prior work has studied prompt optimization, automatic prompt generation, and multi-agent refinement [1, 2, 3, 4, 5, 6], as well as e-commerce attribute extraction using sequence tagging, question answering, LLMs, and multimodal models [7, 8, 9, 10, 11, 12]. We focus on a practical gap between these lines: how to manage and improve prompt specialization at industrial catalog scale, where the system must balance precision, coverage, abstention, consistent quality, and deployment cost.

ACM SIGIR Workshop on eCommerce 2026, July 24, 2026, Melbourne, Australia

*Corresponding author.

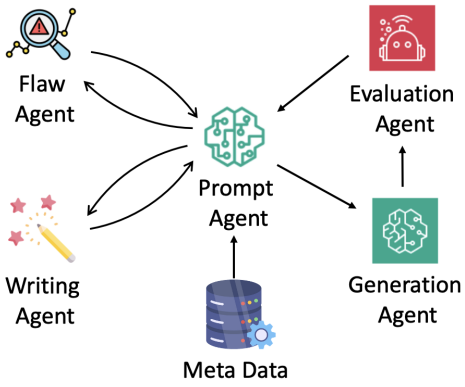

Figure 1: CascadeAgent workflow. A central Prompting Agent coordinates Writing, Generation, Evaluation, and Flaw Detection agents to create and iteratively refine product–attribute instructions.

In an internal production-scale case study, CascadeAgent generated and maintained over 27,000 PA-specific prompts. Compared with a universal-prompt baseline, it improved precision and coverage across multiple LLMs, with particularly large gains for lower-cost models. We also use a simple formal model to characterize when error-driven instruction updates are expected to reduce catalog loss, using the analysis as a guide for understanding the refinement loop rather than as a substitute for empirical validation.

The contributions of our work are:

1. **Multi-pass Prompt Generation (MPG):** A scalable decomposition strategy that assigns dedicated prompts to individual product–attribute subtasks.
2. **CascadeAgent:** A multi-agent framework that creates, evaluates, diagnoses, and refines attribute-specific instructions using domain metadata and error-driven feedback.
3. **Production-scale case study:** An internal e-commerce study involving over 27,000 PA-specific prompts, showing substantial gains over a universal-prompt baseline across multiple LLMs.
4. **Evaluation and deployment lessons:** Practical observations on precision–coverage trade-offs, negative labels for abstention, semantic verification, lower-cost model deployment, and common failure modes in agentic catalog enrichment.

2. Method

Figure 1 summarizes CascadeAgent. The system is designed to scale prompt specialization across thousands of product–attribute (PA) pairs. The method has two components: **Multi-pass Prompt Generation (MPG)**, which decomposes catalog enrichment into PA-specific prompting tasks, and an agentic refinement loop that improves each instruction using observed errors.

2.1. Multi-pass Prompt Generation

In standard prompting, a single instruction must cover many attributes, categories, value types, and edge cases. This quickly becomes difficult to maintain. MPG instead assigns a dedicated prompt to each PA pair. Each prompt starts from a shared base template containing product context such as title, description, bullet points, and available catalog fields, and is then specialized with attribute metadata such as definition, data type, allowed values, and abstention rules.

Each PA-specific prompt follows a shared structure: a system role, product context, task definition, attribute-specific instruction, and output schema. The product context includes available fields such as title, description, bullet points, category, and existing catalog attributes. The attribute-specific instruction then adds PA-level guidance, such as the attribute definition, valid values, expected data type, and abstention rules. This shared structure keeps prompts consistent while allowing the instruction

module to vary by PA pair.

This decomposition has three practical benefits. First, it keeps each prompt focused on one attribute, avoiding long universal prompts with complex conditional logic. Second, it allows underperforming attributes to be improved independently, without changing prompts for other attributes. Third, it supports parallel optimization across the catalog. In our internal case study, this design allowed us to manage over 27,000 PA-specific prompts.

2.2. CascadeAgent Refinement Loop

Building on MPG, CascadeAgent automates the creation and refinement of PA-specific instructions. For each PA pair, the system runs a loop with five specialized agents:

- **Prompting Agent:** Coordinates the refinement cycle and constructs the complete prompt from product context, attribute metadata, and the current instruction.
- **Writing Agent:** Drafts or rewrites attribute-specific instructions using catalog guidelines, metadata, and observed failure patterns.
- **Generation Agent:** Applies the current instruction to generate attribute values from product inputs.
- **Evaluation Agent:** Compares generated values against ground-truth labels or quality rules, including cases where the correct output is to abstain.
- **Flaw Detection Agent:** Summarizes recurring errors, such as hallucinated default values, omissions, over-generation, and schema violations.

The key design choice is that the system does not treat errors as isolated failures. Instead, the Flaw Detection Agent converts batches of errors into short diagnostic summaries, and the Writing Agent uses these summaries to produce revised instructions. This makes the refinement process interpretable: each prompt update can be traced back to observed error patterns.

2.3. Textual-Gradient-Style Optimization

Algorithm 1 summarizes the refinement process. Starting from an initial instruction I_0, CascadeAgent maintains a pool of candidate instructions. At each iteration, it samples error cases for the current top-performing instructions, summarizes the failure patterns, and generates revised instruction candidates. These candidates are then evaluated, added to the pool, and the top K instructions are retained for the next iteration.

This procedure is inspired by textual-gradient and prompt-optimization methods such as ProTeGi [13], but adapted to the e-commerce catalog setting. The Top-K pool avoids committing too early to a single rewrite, which is important because different error samples can suggest different fixes. In practice, this helped balance exploration and stability when optimizing prompts across heterogeneous attribute types.

2.4. Why Error-Driven Refinement Can Improve Prompts

We use a simple formal model to clarify when CascadeAgent's iterative refinement loop should improve an instruction. For a product–attribute instruction π, we define catalog loss as a weighted combination of three error types: value errors, omission errors, and commission errors. Each rewrite can fix existing errors, introduce new errors, or leave behavior unchanged. Under a natural *net-beneficial rewrite* condition—the probability of fixing an error is larger than the probability of introducing a new one—the expected loss decreases until reaching a floor determined by irreducible task difficulty and evaluation noise.

This analysis also motivates the Top-K candidate pool in Algorithm 1. Rather than greedily committing to a single rewrite, CascadeAgent keeps several high-performing instructions, reducing the chance of selecting a candidate that only appears good due to noisy minibatch evaluation. Appendix B provides formal propositions and proofs for the Top-K selection step and the expected loss-reduction behavior.

Algorithm 1 Instruction refinement with error-driven textual feedback

Input: Initial instruction I_0, training examples, minibatches m, error sample size n, Top-K size K, max iterations T
Output: Optimized instruction
Initialize candidate pool $P \leftarrow \{I_0\}$
Initialize *best_K* $\leftarrow \{I_0\}$
for $t = 1$ to T **do**
    new_instructions $\leftarrow \emptyset$
    for each instruction I in *best_K* **do**
        for each minibatch B_i **do**
            Sample n error cases S_i from B_i
            flaw_summary $\leftarrow$ `FlawAgent`(S_i)
            new_I $\leftarrow$ `WritingAgent`$(I, \textit{flaw\_summary})$
            Add *new_I* to *new_instructions*
        end for
    end for
    Evaluate *new_instructions* on validation examples
    $P \leftarrow P \cup$ *new_instructions*
    best_K $\leftarrow$ Top-K instructions from P
    if convergence criterion is met **then**
        break
    end if
end for
return best instruction from *best_K*

3. Experimental Design and Evaluation

3.1. Dataset and Sampling

We evaluate CascadeAgent using manually verified labels from an internal e-commerce catalog enrichment setting. The evaluation contains two levels: a catalog-level evaluation set for overall quality and PA-level evaluation sets for fine-grained analysis.

Catalog-level evaluation. CascadeAgent generated over **27,000** product–attribute (PA)-specific instructions from catalog metadata. For aggregate evaluation, we used a search-weighted sample of **10,000** products containing **304,847** attribute labels across **1,394** product categories. This sample reflects the real product distribution seen in the catalog.

PA-level evaluation. To understand performance at the attribute level, we curated **2,897** high-customer-relevance PA pairs. Each PA pair contained at least 100 test labels, resulting in approximately **304,000** labels.

Iterative optimization set. For iterative prompt refinement, we selected **1,879** high-customer-relevance PA pairs with sufficient verified labels. For each PA pair, we used 50 samples for training, 100 samples for validation and Top-K selection, and a separate holdout test set with at least 100 samples for final evaluation.

3.2. Models and Infrastructure

We used Claude 3.5 Sonnet for the Flaw Detection and Writing Agents because these steps require error analysis and instruction rewriting. For attribute generation, we evaluated both Mistral NeMo and Claude 3.5 Sonnet to compare a lower-cost model with a stronger premium model. Mistral NeMo was also used for semantic verification in inconclusive evaluation cases. Self-hosted Mistral NeMo inference and parallel orchestration were run on six AWS EC2 P5 instances, with 48 NVIDIA H100 GPUs in total, achieving approximately **613 PA pairs per hour** through parallel execution.

Table 1
Catalog-level comparison of universal prompting, MPG, and CascadeAgent.

Base Model	Prompt Type	Precision (%)	Coverage (%)
Mistral NeMo	Baseline	57.14	42.58
Mistral NeMo	MPG	76.19	58.10
Mistral NeMo	CascadeAgent + MPG	90.21	56.09
Claude 3.5 Sonnet	Baseline	72.73	68.47
Claude 3.5 Sonnet	MPG	87.32	86.02
Claude 3.5 Sonnet	CascadeAgent + MPG	93.55	86.49

3.3. Configuration

Based on internal ablation studies, we used 5 minibatches, 5 error cases per minibatch, and Top-$K = 3$ selection. This setting provided a practical balance between error diversity, generalization, and computational cost. Appendix A summarizes the ablation results.

3.4. Evaluation Methodology

We report three metrics:

- **Precision:** the fraction of generated non-empty values that match the verified label.
- **Recall:** the fraction of expected attribute values that are correctly generated.
- **Coverage:** the fraction of samples for which the model produces any non-empty value.

Coverage must be interpreted together with precision. For some attributes, such as *subject_character* for t-shirts, many products correctly have no value. In these cases, higher coverage is not always better: over-generation can indicate hallucination. To capture this behavior, the ground truth includes negative labels such as *Not Applicable* and *Not Obtainable*. Generated values are evaluated using string matching when possible, with LLM-based semantic verification for inconclusive cases such as synonyms, paraphrases, or ambiguous textual values. When the generator and verifier use the same LLM family, semantic verification is applied only to inconclusive string-matching cases rather than as the primary metric. This reduces the risk that evaluation rewards model self-consistency rather than correctness.

4. Results

4.1. Overall Performance

Table 1 compares three prompt settings: a universal baseline prompt, MPG without iterative agent refinement, and CascadeAgent with MPG. The comparison separates the value of PA-specific prompting from the additional value of agentic refinement.

CascadeAgent improves both precision and coverage over the universal-prompt baseline for both LLMs. For Mistral NeMo, precision increases from 57.14% to 90.21%, while coverage increases from 42.58% to 56.09%. For Claude 3.5 Sonnet, precision increases from 72.73% to 93.55%, and coverage increases from 68.47% to 86.49%. The gains are especially important for the lower-cost model: after CascadeAgent refinement, Mistral NeMo narrows the precision gap with Claude 3.5 Sonnet to approximately 3 percentage points. This suggests that PA-specific instruction refinement can make lower-cost models more viable for large-scale deployment.

For Mistral NeMo, CascadeAgent slightly lowers coverage relative to vanilla MPG, from 58.10% to 56.09%, while increasing precision substantially, from 76.19% to 90.21%. This reflects a useful precision–coverage trade-off in catalog enrichment settings where unsupported generations are costly.

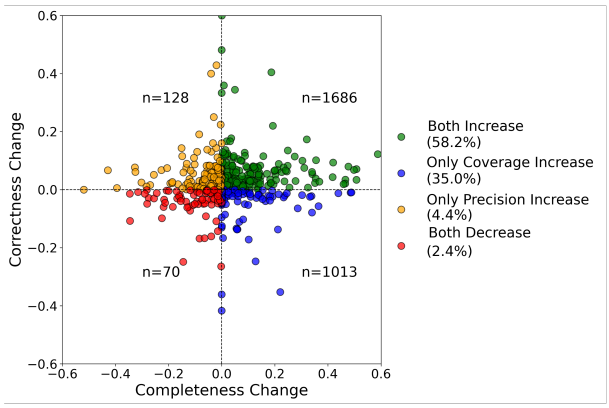

Figure 2: PA-level changes in precision and coverage after applying PA-specific instructions.

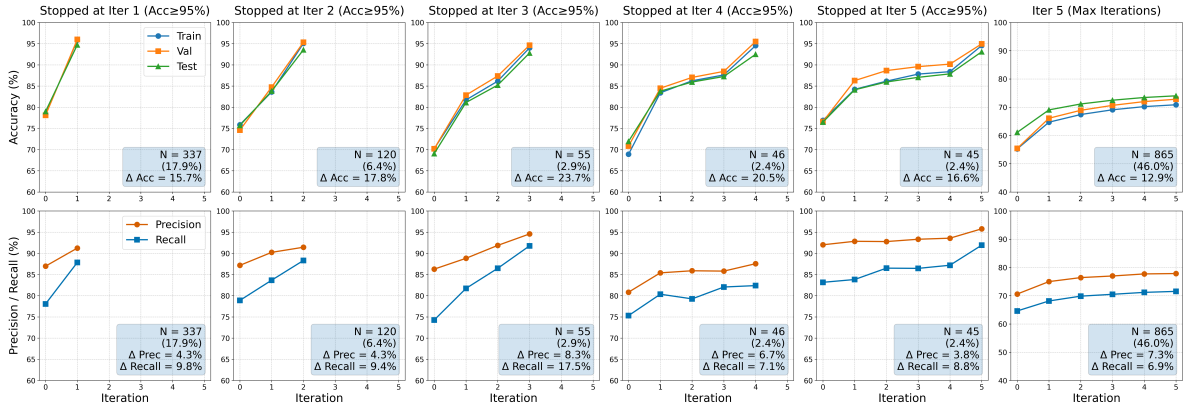

Figure 3: Accuracy, precision, and recall changes over iterative refinement, grouped by stopping behavior.

4.2. PA-Level Precision–Coverage Trade-offs

We next analyze changes across the **2,897** high-impact PA pairs. As shown in Figure 2, **58.2%** of PA pairs showed a consistently positive effect, either improving both precision and coverage or improving one metric without degrading the other. Only **2.4%** declined in both metrics. The remaining **39.4%** showed mixed behavior, most commonly increased coverage with reduced precision.

This trade-off is important for catalog enrichment. Higher coverage can be useful when the model correctly fills missing attributes, but it can also increase hallucination when product evidence is insufficient. A common failure mode is default-value generation, such as predicting *plastic* for a computer mouse when the material is not supported by the product text. This motivates the use of negative labels and explicit abstention rules in the refinement loop.

4.3. Iterative Refinement with Textual Feedback

We evaluate the iterative refinement loop on the **1,879** high-fidelity PA pairs. After the initial CascadeAgent run, **22%** of PA pairs (409 PAs) reached 95% accuracy. For the remaining **1,470** PA pairs, we applied Algorithm 1 for up to five iterations, stopping early when both training and validation accuracy reached 95%.

Iterative refinement produced an average **+15 percentage point** improvement in holdout test accuracy, including **+6 points** in precision and **+8 points** in recall. The improvements were statistically significant under a paired t-test ($p \leq 1 \times 10^{-10}$). For the PA pairs that required refinement, outcomes fell into three groups: **27.2%** of the full high-fidelity set reached the stopping criterion within three iterations, **4.8%** continued improving in the final two iterations, and **46.0%** improved early before plateauing.

The remaining difficult cases reveal useful deployment lessons. Among PA pairs requiring all five iterations, approximately **40%** involved image-dependent attributes, where textual product evidence was often insufficient. Numeric attributes were another recurring challenge, often requiring unit handling, conversion, or stronger numerical constraints. These results suggest that some failure modes should be routed to multimodal models, numeric guardrails, or abstention policies rather than repeated text-only prompt refinement.

5. Lessons Learned

Our case study suggests several practical lessons for agentic catalog enrichment.

Prompt decomposition matters. Universal prompts are difficult to maintain because attributes differ in semantics, value constraints, and abstention behavior. PA-specific prompts make failures easier to diagnose and enable targeted refinement.

Coverage must be evaluated with abstention. Higher coverage is not always better. For sparse or optional attributes, over-generation can reduce catalog quality by introducing unsupported values. Negative labels such as *Not Applicable* and *Not Obtainable* are necessary for measuring whether the system knows when to abstain.

Error summaries are more useful than raw failures. The Flaw Detection Agent helps convert individual mistakes into reusable patterns, such as default hallucinations, missing evidence rules, or schema violations. These summaries make prompt rewriting more targeted and interpretable.

Not all failures should be solved by more prompt refinement. Some hard cases reflect missing input evidence rather than poor instructions. Image-dependent attributes may require multimodal models, while numeric attributes may require dedicated validation, unit normalization, or rule-based guardrails.

6. Related Work

Product attribute extraction has been studied for many years in e-commerce and information retrieval. Early work used rule-based systems and semi-supervised extraction methods for product descriptions and catalog fields [14, 15, 16, 17]. Later neural approaches formulated attribute extraction as sequence tagging or question answering, including OpenTag [7], AdaTag [8], and MAVE [9]. These methods improved extraction quality but often required task-specific training or retraining when new attributes, categories, or schemas were introduced.

Recent e-commerce systems have explored LLMs and multimodal models for attribute generation and implicit value extraction [10, 11, 18, 12, 19]. These systems reduce some dependence on manually engineered extractors, but scaling instruction design and maintenance across tens of thousands of product–attribute pairs remains a practical challenge.

Our work is also related to prompt optimization and agentic refinement methods, including AutoPrompt [1], ProTeGi [13], TextGrad [4], Reflexion [6], and broader work on LLMs as optimizers [2]. CascadeAgent adapts this line of work to the e-commerce catalog setting, where each prompt must handle attribute-specific semantics, abstention behavior, schema constraints, and deployment cost.

7. Limitations, Reproducibility, and Ethical Considerations

This work is based on an internal production-scale e-commerce catalog enrichment setting. The specific LLMs used in this case study reflect the deployment setting at the time of experimentation; the main

Table 2
High-level comparison of product attribute extraction approaches.

Approach	Needs retraining	Implicit values	Scale
OpenTag [7]	Yes	No	Small
AdaTag [8]	Yes	No	Small
MAVE [9]	Yes	Limited	Medium
SAGE [20]	Yes	Yes	Large
LLM-Ensemble [10]	No	Yes	Medium
EIVEN [18]	Yes	Yes	Medium
ViOC-AG [12]	No	Yes	Medium
MXT [19]	Yes	Yes	Large
CascadeAgent	**No**	**Yes**	**Large**

contribution is the prompt-specialization and refinement framework rather than a benchmark of the latest model releases. Due to contractual and policy constraints, we cannot release the proprietary dataset, production prompts, or implementation. We therefore position the paper as a practice-oriented case study and design report rather than a fully reproducible benchmark.

Several limitations remain. First, the evaluation is conducted on internal catalog data, so the absolute metrics may not transfer directly to other marketplaces or product taxonomies. Second, our semantic verification step uses an LLM for inconclusive cases, which can introduce judge bias despite the use of manually verified labels and negative labels. Third, some attributes require evidence not available in text, especially image-dependent attributes, and are better handled by multimodal models or abstention policies. Finally, numeric attributes remain challenging when they require unit conversion, range reasoning, or normalization.

CascadeAgent operates on product catalog content rather than personally identifiable customer data. The main risk is incorrect or unsupported attribute generation, which can mislead shoppers or degrade trust. We mitigate this risk by evaluating against verified labels, including negative labels for abstention, and analyzing common hallucination patterns. In production settings, generated attributes should be monitored and subject to quality controls before being used in customer-facing experiences.

Despite these constraints, we provide detailed descriptions of the system design, optimization loop, evaluation protocol, and observed failure modes to support comparison and future work on agentic catalog enrichment systems.

8. Conclusion

We presented **CascadeAgent**, a multi-agent framework for scaling product–attribute prompt specialization in e-commerce catalog enrichment. CascadeAgent combines **Multi-pass Prompt Generation (MPG)** with an error-driven refinement loop in which specialized agents generate, evaluate, diagnose, and rewrite attribute-specific instructions.

In an internal production-scale case study, CascadeAgent generated and maintained over 27,000 PA-specific prompts and improved both precision and coverage over a universal-prompt baseline across multiple LLMs. The gains were especially useful for lower-cost models, suggesting that prompt specialization can improve the cost–quality trade-off for large-scale catalog enrichment. Iterative refinement further improved difficult PA pairs, while also revealing recurring failure modes such as hallucinated default values, image-dependent attributes, and numeric reasoning errors.

Overall, the results suggest that large-scale catalog enrichment benefits from PA-level prompt decomposition, explicit abstention handling, error-pattern-based rewriting, and evaluation protocols that jointly measure precision, coverage, and negative-label behavior. We hope these observations help future work on agentic systems for product understanding and e-commerce search.

Declaration on Generative AI

The authors used generative AI tools for grammar, clarity, and language polishing. The authors reviewed and edited the content and take full responsibility for the publication.

References

[1] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, S. Singh, Autoprompt: Eliciting knowledge from language models with automatically generated prompts, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, 2020, pp. 4222–4235.

[2] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, X. Chen, Large language models as optimizers, 2024. URL: https://arxiv.org/abs/2309.03409. arXiv:2309.03409.

[3] H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, G. Song, Reevo: Large language models as hyper-heuristics with reflective evolution, in: Advances in Neural Information Processing Systems, 2024.

[4] M. Yuksekgonul, F. Bianchi, J. Boen, S. Liu, Z. Huang, C. Guestrin, J. Zou, Textgrad: Automatic differentiation via text, 2024. URL: https://arxiv.org/abs/2406.07496. arXiv:2406.07496.

[5] K. Chang, S. Xu, C. Wang, Y. Luo, X. Liu, T. Xiao, J. Zhu, Efficient prompting methods for large language models: A survey, 2024. URL: https://arxiv.org/abs/2404.01077. arXiv:2404.01077.

[6] M. Shinn, C. Cassano, E. Berman, R. Gopinath, K. Narasimhan, P. Yao, Reflexion: Language agents with verbal reinforcement learning, arXiv preprint arXiv:2303.16119 (2023).

[7] G. Zheng, S. Mukherjee, X. L. Dong, F. Li, Opentag: Open attribute value extraction from product profiles, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2018, pp. 1049–1058. doi:10.1145/3219819.3219839.

[8] J. Yan, N. Zalmout, Y. Liang, C. Grant, X. Ren, X. L. Dong, Adatag: Multi-attribute value extraction from product profiles with adaptive decoding, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, 2021, pp. 4694–4705. doi:10.18653/v1/2021.acl-long.362.

[9] L. Yang, Q. Wang, X. Quan, F. Feng, Y. Chen, M. Khabsa, S. Wang, Z. Xu, D. Liu, Mave: A product dataset for multi-source attribute value extraction, in: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, Association for Computing Machinery, 2022, pp. 1256–1265. doi:10.1145/3488560.3498377.

[10] C. Fang, X. Li, Z. Fan, J. Xu, K. Nag, E. Korpeoglu, S. Kumar, K. Achan, 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, Association for Computing Machinery, 2024, pp. 2910–2914. doi:10.1145/3626772.3661357.

[11] T. Zhang, C. Zhang, X. Li, J. Shang, H. Nguyen, P. S. Yu, Stronger, lighter, better: Towards life-long attribute value extraction for e-commerce products, in: Findings of the Association for Computational Linguistics: ACL 2024, 2024, pp. 8631–8643.

[12] J. Gong, M. Cheng, H. Shen, P.-Y. Vandenbussche, J. Jenq, H. Eldardiry, Visual zero-shot e-commerce product attribute value extraction, in: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, Industry Track, 2025, pp. 460–469.

[13] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, M. Zeng, Automatic prompt optimization with gradient descent and beam search, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023, pp. 7957–7968.

[14] R. Ghani, K. Probst, Y. Liu, M. Krema, A. Fano, Text mining for product attribute extraction, SIGKDD Explorations Newsletter 8 (2006) 41–48. doi:10.1145/1147234.1147241.

[15] K. Probst, R. Ghani, M. Krema, A. Fano, Y. Liu, Semi-supervised learning of attribute-value pairs

from product descriptions, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2007, pp. 2838–2843.

[16] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, S. Vaithyanathan, Domain adaptation of rule-based annotators for named-entity recognition tasks, in: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, 2010, pp. 1002–1012.

[17] D. Vandic, J.-W. van Dam, F. Frasincar, Faceted product search powered by the semantic web, Decision Support Systems 53 (2012) 425–437. doi:10.1016/j.dss.2012.02.010.

[18] H. Zou, G. Yu, Z. Fan, D. Bu, H. Liu, P. Dai, D. Jia, C. Caragea, 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, Industry Track, 2024, pp. 453–463. doi:10.18653/v1/2024.naacl-industry.40.

[19] A. Khandelwal, H. Mittal, S. Kulkarni, D. Gupta, Large scale generative multimodal attribute extraction for e-commerce attributes, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, Industry Track, 2023, pp. 305–312. doi:10.18653/v1/2023.acl-industry.29.

[20] A. N. Nikolakopoulos, S. Kaul, S. K. Gade, B. Dubrov, U. Batur, S. A. Khan, Sage: Structured attribute value generation for billion-scale product catalogs, arXiv preprint arXiv:2309.05920 (2023).

A. Ablation Studies

We conducted internal ablation studies to choose the refinement configuration used in the main experiments. Each study varied one parameter while keeping the others fixed. We evaluated three parameters: Top-K selection, the number of sampled error examples per minibatch, and the number of minibatches per refinement iteration. All ablations were run on a sample of 10 product–attribute pairs over three refinement iterations.

A.1. Top-K Selection

Figure 4 compares $K = 1, 2, 3$. Top-2 and Top-3 produced more consistent improvement than Top-1, which often plateaued after the first iteration. This suggests that keeping only the single best instruction can lead to premature convergence, while retaining a small candidate pool improves exploration. We therefore used Top-$K = 3$ in the main experiments.

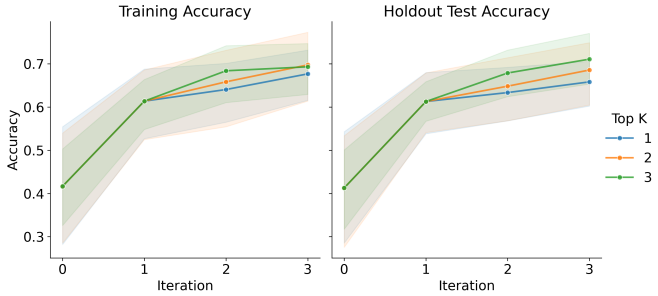

Figure 4: Effect of Top-K selection on iterative refinement performance.

A.2. Number of Error Examples

Figure 5 compares 1, 3, and 5 sampled error examples per minibatch. A single error example produced narrow feedback and weaker holdout improvement. Using 3 or 5 examples gave the Flaw Detection Agent more representative error patterns and improved refinement stability. We used 5 error examples per minibatch in the main experiments.

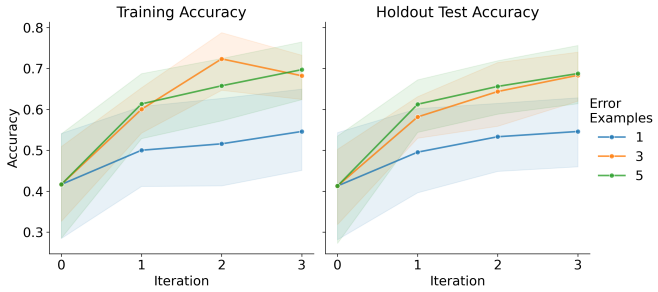

Figure 5: Effect of the number of sampled error examples per minibatch.

A.3. Number of Minibatches

Figure 6 compares 1, 3, and 5 minibatches per refinement iteration. Using more minibatches exposed the Writing Agent to a broader set of failure patterns and reduced overfitting to a single error sample. In our ablation, 3 and 5 minibatches converged faster and more reliably than a single minibatch. We used 5 minibatches in the main experiments.

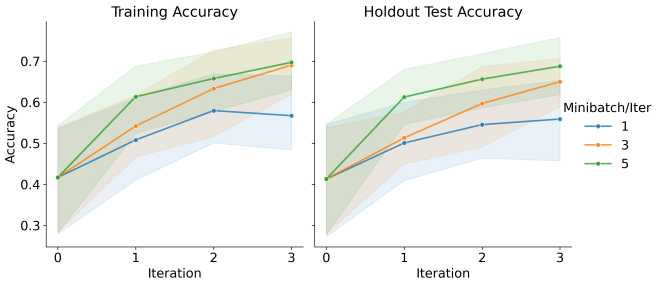

Figure 6: Effect of the number of minibatches per refinement iteration.

B. Formal Analysis

We provide a compact formalization of the two design choices used in CascadeAgent: retaining a Top-K pool of candidate instructions and applying error-driven rewrites. The analysis is intended to characterize when the refinement loop is expected to improve catalog loss, rather than to model every implementation detail.

B.1. Top-K Selection

Proposition 1 (Empirical Top-K selection retains the best instruction)**.** *Fix an iteration t. Let $\mathcal{P}_t \subseteq \Pi$ be the current pool of instructions and define*

$$\pi_t^\star := \arg\min_{\pi \in \mathcal{P}_t} \mathcal{L}(\pi), \qquad \Delta_t := \min_{\pi \neq \pi_t^\star} [\mathcal{L}(\pi) - \mathcal{L}(\pi_t^\star)] > 0.$$

Assume $0 \leq \mathcal{L}(\pi) \leq 1$ for every $\pi \in \mathcal{P}_t$. Draw an i.i.d. evaluation batch $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^n$ and define

$$\widehat{\mathcal{L}}_t(\pi) := \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[f_\theta(x_i, \pi) \neq y_i] \,.$$

For confidence level $\vartheta \in (0, 1)$, if

$$n \geq \left\lceil \frac{2}{\Delta_t^2} \log \frac{2|\mathcal{P}_t|}{\vartheta} \right\rceil,$$

then, with probability at least $1 - \vartheta$, the empirical Top-K set contains $\pi_t^\star$ for any $K \geq 1$. Consequently, if $\mathcal{P}_{t+1} := \mathcal{P}_t \cup \widehat{\mathcal{P}}_{t,K}$, then

$$\min_{\pi \in \mathcal{P}_{t+1}} \mathcal{L}(\pi) \leq \min_{\pi \in \mathcal{P}_t} \mathcal{L}(\pi).$$

Proof. For any fixed instruction π, $\widehat{\mathcal{L}}_t(\pi)$ is an empirical mean of Bernoulli variables in $[0, 1]$. By Hoeffding's inequality,

$$\Pr\left[\left|\widehat{\mathcal{L}}_t(\pi) - \mathcal{L}(\pi)\right| \geq \varepsilon\right] \leq 2\exp(-2n\varepsilon^2).$$

Set

$$\varepsilon = \sqrt{\frac{\log(2|\mathcal{P}_t|/\vartheta)}{2n}}.$$

A union bound over all $\pi \in \mathcal{P}_t$ gives

$$\left|\widehat{\mathcal{L}}_t(\pi) - \mathcal{L}(\pi)\right| < \varepsilon \quad \forall \pi \in \mathcal{P}_t$$

with probability at least $1 - \vartheta$. Under the assumed sample size, $2\varepsilon \leq \Delta_t$. Therefore, for every competitor $\pi \neq \pi_t^\star$,

$$\widehat{\mathcal{L}}_t(\pi) - \widehat{\mathcal{L}}_t(\pi_t^\star) \geq \mathcal{L}(\pi) - \mathcal{L}(\pi_t^\star) - 2\varepsilon \geq 0.$$

With deterministic tie-breaking, $\pi_t^\star$ is retained in the empirical Top-K set. Since $\pi_t^\star \in \mathcal{P}_{t+1}$, the best true loss in the updated pool cannot be worse than the best true loss in the previous pool. □

B.2. Loss Decay Under Error-Driven Rewrites

Proposition 2 (Geometric loss decay under marginal churn)**.** *For a product–attribute example at refinement step t, let $E_t \in \{0, 1\}$ indicate whether the instruction makes any error. Let the per-example catalog loss be*

$$\mathcal{L}_t = w_{\text{val}} E_{\text{val},t} + w_{\text{omis}} E_{\text{omis},t} + w_{\text{commis}} E_{\text{commis},t},$$

where the three error types correspond to value errors, omission errors, and commission errors. Assume the error types are mutually exclusive, the weights are positive and sum to one, and define

$$\kappa_{\min} := \min(w_{\text{val}}, w_{\text{omis}}, w_{\text{commis}}), \qquad \kappa_{\max} := \max(w_{\text{val}}, w_{\text{omis}}, w_{\text{commis}}).$$

Then

$$\kappa_{\min} E_t \le \mathcal{L}_t \le \kappa_{\max} E_t.$$

Assume the rewrite step follows the marginal-churn dynamics

$$\Pr[E_{t+1} = 0 \mid E_t = 1] = p_{\text{fix}}, \qquad \Pr[E_{t+1} = 1 \mid E_t = 0] = p_{\text{break}},$$

where $p_{\text{fix}} > p_{\text{break}} \ge 0$ and $p_{\text{fix}} + p_{\text{break}} \in (0, 1]$. Let

$$\lambda := 1 - (p_{\text{fix}} + p_{\text{break}}), \qquad r := \frac{\kappa_{\max}}{\kappa_{\min}}, \qquad \phi := r\lambda.$$

If $\phi \in [0, 1)$, then

$$\mathbb{E}[\mathcal{L}_{t+1}] \le \phi\, \mathbb{E}[\mathcal{L}_t] + \kappa_{\max} p_{\text{break}}.$$

For any horizon $T \ge 0$,

$$\mathbb{E}[\mathcal{L}_T] \le \phi^T \mathbb{E}[\mathcal{L}_0] + \kappa_{\max} p_{\text{break}} \frac{1 - \phi^T}{1 - \phi}.$$

Consequently,

$$\lim_{T \to \infty} \mathbb{E}[\mathcal{L}_T] \le \frac{\kappa_{\max} p_{\text{break}}}{1 - \phi}.$$

Proof. By the marginal-churn dynamics,

$$\mathbb{E}[E_{t+1}] = (1 - p_{\text{fix}})\mathbb{E}[E_t] + p_{\text{break}}(1 - \mathbb{E}[E_t]).$$

Rearranging gives

$$\mathbb{E}[E_{t+1}] = \lambda \mathbb{E}[E_t] + p_{\text{break}}.$$

Using $\mathcal{L}_{t+1} \le \kappa_{\max} E_{t+1}$ and $E_t \le \mathcal{L}_t / \kappa_{\min}$, we obtain

$$\mathbb{E}[\mathcal{L}_{t+1}] \le \kappa_{\max} \mathbb{E}[E_{t+1}] = \kappa_{\max} \left(\lambda \mathbb{E}[E_t] + p_{\text{break}}\right) \le \lambda \frac{\kappa_{\max}}{\kappa_{\min}} \mathbb{E}[\mathcal{L}_t] + \kappa_{\max} p_{\text{break}}.$$

Since $\phi = r\lambda$, this proves the one-step bound:

$$\mathbb{E}[\mathcal{L}_{t+1}] \le \phi \mathbb{E}[\mathcal{L}_t] + \kappa_{\max} p_{\text{break}}.$$

Let $x_t = \mathbb{E}[\mathcal{L}_t]$ and $c = \kappa_{\max} p_{\text{break}}$. The recurrence $x_{t+1} \le \phi x_t + c$ unrolls to

$$x_T \le \phi^T x_0 + c \sum_{k=0}^{T-1} \phi^k = \phi^T x_0 + c \frac{1 - \phi^T}{1 - \phi}.$$

Taking $T \to \infty$ gives the stated asymptotic floor. □