AutoPKG: An Automated Framework for Dynamic E-commerce Product-Attribute Knowledge Graph Construction

Pollawat Hongwimol[1], **Haoning Shang**[1], **Chutong Wang**[1], **Zhichao Wan**[1], **Yi Gao**[1], **Yuanming Li**[1], **Lin Gui**[1], **Wenhao Sun**[2], **Cheng Yu**[1]
[1]Lazada, Alibaba International Digital Commerce Group
[2]Nanyang Technological University, Singapore
{pollwat.h, shanghaoning.shn, wangchutong.wct, wanzhichao.wzc, xiheng.gy}@alibaba-inc.com, yli056@e.ntu.edu.sg, lin.gui@u.nus.edu, wenhao006@ntu.edu.sg, yucheng.yc@lazada.com

Abstract

Product attribute extraction in e-commerce is bottlenecked by ontologies that are inconsistent, incomplete, and costly to maintain. We present **AutoPKG**, a multi-agent Large Language Model (LLM) framework that automatically constructs a Product-attribute Knowledge Graph (PKG) from multimodal product content. AutoPKG induces product types and type-specific attribute keys on demand, extracts attribute values from text and images, and consolidates updates through a centralized decision agent that maintains a globally consistent canonical graph. We also propose an **evaluation protocol** for dynamic PKGs that measures type/key validity and consolidation quality, as well as edge-level accuracy for value assertions after canonicalization. On a large real-world marketplace catalog **dataset from Lazada (Alibaba)**, AutoPKG achieves up to 0.953 Weighted Knowledge Efficiency (WKE) for product types, 0.724 WKE for attribute keys, and 0.531 edge-level F_1 for multimodal value extraction. Across three public benchmarks, we improve edge-level exact-match F_1 by 0.152 and yield a 0.208 precision gain on the attribute extraction application. Online A/B tests show that AutoPKG-derived attributes increase Gross Merchandise Value (GMV) in Badge (+3.81%), Search (+5.32%), and Recommendation (+7.89%), supporting AutoPKG's practical value in production.

1 Introduction

Product attributes serve as critical e-commerce infrastructure by powering faceted navigation, improving search relevance, and supporting recommendation and facilitating semantic product understanding at scale (Subramaniam, 2025; Magnani et al., 2022; Wang et al., 2019). However, industrial attribute pipelines remain bottlenecked by the *schema*, as product taxonomies and type-specific attribute keys are often inconsistent across markets and incomplete for long-tail inventory (Xu et al., 2019; Zhu et al., 2020). Furthermore, it is costly to maintain under continuous distribution shift and multilingual seller noise (Xu et al., 2019; Dong et al., 2020). Consequently, even strong Product Attribute Value Extraction (PAVE) models frequently operate under outdated or overly narrow attribute lists, which limits coverage and necessitates repeated human ontology work (Zheng et al., 2018; Putthividhya and Hu, 2011).

Table 1 highlights a gap in existing work. While prior e-commerce Knowledge Graph (KG) *frameworks* such as AutoKnow, AliCG, and COSMO demonstrate large-scale construction pipelines, they typically assume or rely on a governed schema while focusing less on *open-ended* type/key induction, multimodal evidence, and continual update. Similarly, open datasets and benchmarks including OpenTag, AE-110K, MEPAVE, MAVE, and ImplicitAVE have advanced PAVE models. However, these resources do not address dynamic schema evolution and canonical KG maintenance as a first-class problem. Furthermore, general-purpose LLM-based schema/KG induction frameworks such as AutoSchemaKG are not tailored to the specific constraints of e-commerce product attributes. Crucially, no prior *open* framework jointly supports **automatic type induction (AT)**, **automatic key discovery (AK)**, and **multimodal value extraction (MM)** within a continually updated PKG.

We present **AutoPKG**, an automated multi-agent LLM framework (Guo et al., 2024) that constructs and continually evolves a PKG from multimodal marketplace listings. Starting from an empty KG, AutoPKG induces PRODUCTTYPE nodes on demand and proposes type-specific ATTRIBUTEKEY nodes. It then extracts attribute values from text and multiple images, finally consolidating all updates into a PKG. A key design element is the centralized *Knowledge Graph Decision agent* (abbreviated as KGD throughout this paper), which acts as the *sole write interface* to

Findings of the Association for Computational Linguistics: ACL 2026, pages 15622–15650
July 2-7, 2026

Name	Type	Available	Domain	#Nodes	#Edges	#Types	#Attrs	AT	AK	MM
Frameworks / Systems *(construct & maintain a KG)*										
AutoKnow (Dong et al., 2020)	Framework	Closed	E-commerce	>1B	>1B	11K	>1K	✓	✓	✗
AliCG (Zhang et al., 2021)	Framework	Closed	E-commerce	>5M	13.5M	–	–	✓	✓	✗
FolkScope (Yu et al., 2023)	Framework	Open	E-commerce	1.3M	12.8M	–	–	✗	✓	✗
COSMO (Yu et al., 2024)	Framework	Closed	E-commerce	6.3M	29M	18	–	✓	✓	✗
AutoSchemaKG (Bai et al., 2025)	Framework	Open	General	900M	5.9B	–	–	✓	✓	✗
KG Resources *(released graphs)*										
AliCoCo (Luo et al., 2020)	KG Resource	Closed	E-commerce	>3B	>400B	–	20	✗	✗	✗
AliCoCo2 (Luo et al., 2021)	KG Resource	Closed	E-commerce	>160M	>800M	–	20	✗	✗	✗
MMpedia (Wu et al., 2023)	KG Resource	Open	General	2.6M	19.5M	–	–	✗	✗	✓
MedKGent (Zhang et al., 2025)	KG Resource	Open	Medical	156K	3.0M	–	–	✗	✓	✗
Datasets / Benchmarks *(PAVE)*										
OpenTag (Zheng et al., 2018)	Dataset	Partial	E-commerce	–	13K	3	5	✗	✓	✗
AE-110K (Xu et al., 2019)	Dataset	Open	E-commerce	–	110K	1	4	✗	✗	✗
MEPAVE (Zhu et al., 2020)	Dataset	Open	E-commerce	121K	87K	7	26	✗	✗	✓
MAVE (Yang et al., 2022)	Dataset	Open	E-commerce	>2M	3M	1.3K	2.5K	✗	✗	✗
ImplicitAVE (Zou et al., 2024a)	Dataset	Open	E-commerce	70K	–	5	25	✗	✗	✓
AutoPKG (Ours)	**Framework**	**Open**	**E-commerce**	**130K**	**560K**	**17K**	**16K**	✓	✓	✓

Table 1: Comparison of relevant frameworks, resources and datasets. **#Types** and **#Attrs** denote the number of product categories and unique attribute keys, respectively. **AT**: automatic type induction; **AK**: automatic key discovery; **MM**: multimodal value extraction.

the KG. Upstream agents propose edits, whereas KGD resolves them through a constrained action space (`ADD`/`MERGE`/`REPLACE`/`DISCARD`) using retrieved PKG context. This approach is similar in spirit to validator-based KG construction but operationalized as explicit edit decisions to ensure continual canonicalization (Boylan et al., 2024; Bian, 2025).

We also introduce an evaluation protocol for *dynamic PKGs* that measures validity and consolidation quality for induced product types and attribute keys, as well as edge-level precision/recall/F1 for value assertions after canonicalization. On our large real-world catalog dataset, AutoPKG achieves up to 0.953 WKE for product types, 0.724 WKE for attribute keys, and 0.531 edge-level F1 for multimodal value extraction. Online A/B tests further demonstrate production impact: AutoPKG-derived attributes improve GMV in Badge (+3.81%), Search (+5.32%), and Recommendation (+7.89%), while delivering no statistically significant change in Filter (+0.26%). Furthermore, across three public benchmarks, AutoPKG improves edge level exact-match F_1 by $+0.152$ and yields a $+0.208$ (weighted average) precision gain in the downstream PAVE application.

The contributions of this paper are as follows:

- We propose **AutoPKG**, the first automated multi-agent framework that unifies automatic type induction, key discovery, and multimodal value extraction within an incrementally evolving PKG, eliminating the need for a fixed, manually maintained taxonomy.
- We introduce a comprehensive **evaluation protocol for dynamic PKGs** that goes beyond standard knowledge extraction metrics, specifically measuring schema validity and consolidation quality alongside traditional edge-level correctness.
- We release a **large-scale multimodal dataset** comprising 37K Lazada online products and its corresponding KG with 130K nodes and 560K edges across 17K product types. This is the most diverse open-source dataset for e-commerce PAVE that supports multi-image reasoning and automatic schema induction.

2 Related Work

KG/PKG Construction and Schema Evolution. Industrial e-commerce KGs/PKGs are typically built via large-scale extraction plus canonicalization and governance, often emphasizing pipeline scalability rather than open-ended schema evolution (Dong et al., 2020; Zhang et al., 2021; Yu et al., 2024). In parallel, research benchmarks have accelerated extraction modeling but usually assume a predefined label space and do not evaluate continual schema growth (Zheng et al., 2018; Zou et al., 2024a). Work on LLM-based KG construction has recently explored schema induction and iterative

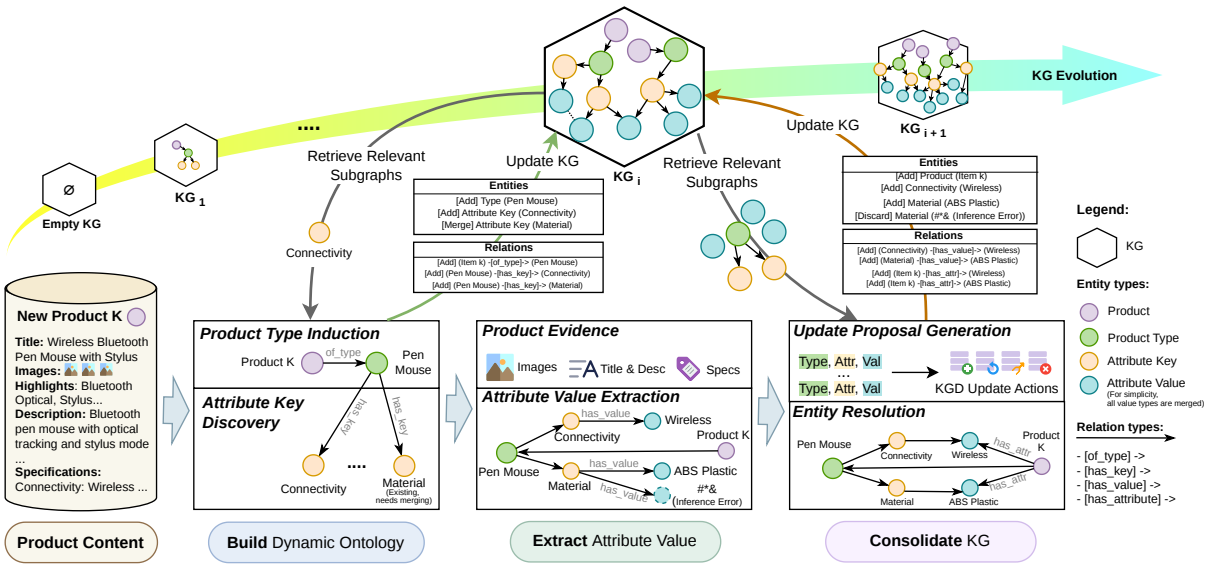

Figure 1: AutoPKG overview. From an initially empty KG, the system (1) induces product types and type-specific attribute keys from the listing text, (2) extracts attribute values from the text and images, and (3) consolidates all proposed edits through KGD to maintain a single canonical KG.

refinement, but is often batch-oriented and predominantly text-centric (Boylan et al., 2024). AutoPKG differs by tightly coupling on-demand type/key induction with multimodal value extraction and by treating canonicalization decisions as explicit constrained edits applied continually.

PAVE. Earlier approaches typically assume a curated taxonomy and train supervised taggers or classifiers, which are expensive to maintain and brittle under long-tail categories and evolving seller terminology (Putthividhya and Hu, 2011; Xu et al., 2019). Recent work reframes PAVE as instruction-following or generation with LLMs using schema-constrained prompting, category-aware prompts, and distillation/fine-tuning to improve robustness on noisy product text (Brinkmann et al., 2024; Gong and Eldardiry, 2024; Hongwimol et al., 2025). Graph and hypergraph-based approaches have also emerged, leveraging Graph Neural Networks to capture complex structural dependencies in text (Potta et al., 2024), or employing heterogeneous hypergraphs to model interrelations between text and image nodes (Hu et al., 2025a). For visually grounded or implicit attributes (e.g., color, style, pattern), multimodal extraction leverages product images to complement text-only signals (Yang et al., 2022; Zou et al., 2024a,b). Retrieval-augmented extraction further improves precision by grounding predictions in similar items or value inventories (Zou et al., 2025). However, most PAVE pipelines still treat the type–key schema as fixed (or manually governed); AutoPKG instead makes schema induction and consolidation first-class and continuous.

3 AutoPKG Framework

AutoPKG maintains a PKG ($\mathcal{G}$) and comprises four agents: type-induction agent, key-discovery agent, attribute value extraction agent, and KGD. Figure 1 illustrates the evolution process of $\mathcal{G}$. For each incoming product listing (title, description, specifications, and images), specialized agents propose (i) a product type, (ii) a type-specific attribute-key set, and (iii) item-level attribute values. All updates are mediated by KGD, which performs constrained PKG edits to ensure global consistency. See the Multi-Agent workflow in Appendix C. All prompts used in this section are provided in Appendix K.

3.1 PKG Schema

The graph $\mathcal{G}$, maintained by AutoPKG, employs a minimal schema designed for continual expansion under noisy marketplace data (see Appendix A for an example subgraph). $\mathcal{G}$ contains three core node types: PRODUCT nodes represent marketplace listings, PRODUCTTYPE nodes represent canonical product types induced from listing content and consolidated by KGD, ATTRIBUTEKEY nodes represent canonical attribute keys that define the type-specific schema. Moreover, attribute values are represented as *key-typed value nodes*, each VALUE node is typed by exactly one ATTRIBUTEKEY and

is not treated as an independent ontology type.

AutoPKG uses four edge types that separate schema structure from item facts. Two schema edges represent applicable attribute keys for each product type and valid attribute values for each key: (PRODUCTTYPE, HAS_KEY, ATTRIBUTEKEY), and (ATTRIBUTEKEY, HAS_VALUE, VALUE). Two instance edges record product-level facts: (PRODUCT, OF_TYPE, PRODUCTTYPE) links an item to its inferred canonical type, while (PRODUCT, HAS_ATTRIBUTE, VALUE) asserts an extracted attribute value for that item. This design enforces type checking (values must be licensed by the product type via its keys) and enables canonical value reuse across items after KGD consolidation.

3.2 Dynamic Ontology Population (Types and Keys)

AutoPKG does not require a pre-defined taxonomy; instead, it populates the ontology in a dynamic manner. For each product, the type-induction agent proposes a canonical PRODUCTTYPE with a brief description from listing text. Conditioned on the inferred type, the key-discovery agent proposes a type-specific ATTRIBUTEKEY table with short definitions and representative value examples. All proposed types/keys are consolidated via KGD.

3.3 Multimodal Attribute Value Extraction

For a product p, KGD retrieves the most similar PRODUCTTYPE and corresponding ATTRIBUTEKEY table from $\mathcal{G}$. Then for each attribute key k in the table, the attribute value agent proposes value v assertions using title, description, specifications, and images, producing edges of the form (p, HAS_ATTRIBUTE, v) with implicit key type $\text{type}(v) = k$. KGD then canonicalizes and filters these proposals and writes approved edits to $\mathcal{G}$.

3.4 KGD: The Sole Write Interface

AutoPKG centralizes all KG updates in a single KGD. Upstream agents may propose new product types, attribute keys, values, or item-level edges, but only KGD is allowed to write. For each proposal, the agent is given (i) the candidate content and (ii) retrieved local KG neighborhood nodes, and it chooses one constrained edit action: `ADD` (create a new node/edge), `MERGE` (put the candidate into an existing canonical node as a synonym/variant), `REPLACE` (promote the candidate surface form as the new canonical label), or `DISCARD` (reject invalid, vague, or weakly grounded proposals).

To construct the neighborhood context, we perform dense retrieval over canonical KG node names within the same node type. We embed all canonical names using `Qwen3-Embedding-0.6B` and retrieve the top-k most similar nodes and provide KGD with their identifiers, names, and available metadata (e.g., descriptions, synonyms). This architecture makes KG growth controllable and consistent by forcing every update through canonicalization-aware, deduplication-focused decisions.

4 Evaluation Methodology

We evaluate AutoPKG as a *dynamic* PKG framework, where model outputs are not only extracted, but also *written into and consolidated within* an incremental evolving PKG. Therefore, our evaluation captures: (i) semantic validity of induced schema (types/keys), (ii) consolidation quality under canonicalization, and (iii) edge-level correctness of value assertions after normalization. Human verification is used to evaluate type/key validity and edge correctness. Sampling sizes and annotation guidelines are provided in Appendix D.

4.1 Dataset

We sample 323K product listings from a large Southeast Asian e-commerce marketplace by selecting up to 20 items per leaf category per country to maximize type and attribute diversity. Each listing contains (i) a product title, (ii) seller-provided description and highlights, (iii) structured specifications, and (iv) up to ten images. This setting reflects real marketplace noise that challenges continual schema induction and extraction. We will release a vetted subset and a corresponding PKG snapshot containing 130K nodes and 560K edges, derived from the 323K sampled listings after filtering and consolidation.[1] See details in Appendix H.

4.2 Models

We evaluate a diverse suite of instruction-tuned LLMs, categorizing them by modality and scale. For text-only tasks (Sections 3.2 and 3.4), we employ LLMs including Qwen3 (Team, 2025f), Gemma-3 and Gemma-3n (Team, 2025b), Llama-3.2 (Team, 2024), Kimi (Team, 2025d), GroveMoE (Wu et al., 2025), and SmolLM3 (Bakouch et al., 2025); Attribute value extraction from multimodal product listings is handled by MLLMs,

[1] https://github.com/Product-Understanding-Lazada-Alibaba/AutoPKG.git

specifically Qwen3-VL (Team, 2025g), Gemma-3 (Team, 2025b), and Kimi-VL (Team, 2025e). For a controlled comparison, all backbones are evaluated with an identical zero-shot prompt template and standardized hyperparameters. See implementation details in Appendix I.

4.3 KGD Decision Evaluation via Multi-LLM Consensus

We evaluate KGD model candidates using a panel of five frontier LLM judges: DeepSeek-V3.1 (DeepSeek-AI, 2025), Gemini-2.5-Pro (Team, 2025a), GPT-5 (OpenAI, 2025), Kimi-K2 (Team, 2025c), and Qwen3-Max (Team, 2025f). For each decision instance, judges select an edit action given the candidate and retrieved PKG context; the reference label is the majority vote ($\geq$3/5). Candidate model performance is reported as accuracy relative to this consensus, and we assess the reliability of the judging panel through pairwise Cohen's κ scores. We further validate the reliability of the LLM-judge consensus via human annotation on a subset, observing strong agreement (N = 2,000, $\kappa = 0.837$; see Appendix E, Table E.1).

Due to the absence of a pre-existing dataset for initial PKG states, we synthesize an experimental corpus by generating PRODUCTTYPE candidates using `Qwen3-Next-80B-A3B-Instruct`. These candidates are subsequently integrated into empty KGs via KGD under varied configurations. Then, we perform random sampling from this synthesized pool to construct a balanced dataset for evaluation.

4.4 Product Type Metrics

Let N be the number of products processed in evaluation, and V be the number of canonical PRODUCTTYPE nodes after KGD consolidation. We report: Acceptance (R_{acc}), the fraction of sampled product type predictions judged valid by humans; Compression ($R_{\text{comp}} = 1 - V/N$), the redundancy reduction achieved by canonicalization; and Coverage (R_{cov}), the fraction of products assigned to a valid canonical type. We summarize overall performance using Weighted Knowledge Efficiency (WKE$_{type}$), a weighted harmonic mean:

$$\text{WKE}_{type} = \frac{w_1 + w_2 + w_3}{\frac{w_1}{R_{\text{acc}}} + \frac{w_2}{R_{\text{comp}}} + \frac{w_3}{R_{\text{cov}}}}.$$

This formulation prevents over-aggressive merging from inflating scores. In practice, the weights $w_1 = 3, w_2 = 1, w_3 = 1$ yield the best balance between semantic validity, compression, and coverage, as detailed in Appendix B.

4.5 Attribute Key Metrics (Probabilistic)

Exhaustively enumerating ground-truth attribute keys for open-vocabulary, type-specific schemas is infeasible. So we adapt a probabilistic evaluation protocol, inspired by probabilistic precision/recall for incomplete labeling and multi-system pooling (Lamiroy and Sun, 2013; Park and Kim, 2023).

Let $\mathcal{K}_{\text{total}}$ be the pooled set of unique canonical keys obtained from the top-performing key-generation models (selected by acceptance) after consolidation via KGD. For each model M_j, we estimate a reliability prior $P(M_j)$ using human key acceptance on sampled outputs, i.e., $P(M_j) = R_{\text{acc}}(M_j)$.

For each canonical key $k \in \mathcal{K}_{\text{total}}$, let predictors$(k)$ be the set of models that proposed k. We estimate its probability of belonging to the latent ground truth under a Noisy-OR assumption:

$$P(k \in \text{GT}) = 1 - \prod_{M_j \in \text{predictors}(k)} (1 - P(M_j)).$$

We compute probabilistic true positives for M_j as $\text{TP}(M_j) = \sum_{k \in \mathcal{K}_j} P(k \in \text{GT})$, where $\mathcal{K}_j$ is the canonical key set produced by M_j. Let $|\widehat{\text{GT}}| = \sum_{k \in \mathcal{K}_{\text{total}}} P(k \in \text{GT})$. Then:

$$\text{P-Prec}(M_j) = \frac{\text{TP}(M_j)}{|\mathcal{K}_j|}, \quad \text{P-Rec}(M_j) = \frac{\text{TP}(M_j)}{|\widehat{\text{GT}}|}.$$

Similar to Section 4.4, We report WKE with weights $w_1 = 3, w_2 = 1, w_3 = 1$:

$$\text{WKE}_{key} = \frac{w_1 + w_2 + w_3}{\frac{w_1}{R_{\text{acc}}} + \frac{w_2}{\text{P-Prec}} + \frac{w_3}{\text{P-Rec}}},$$

where the weighting rationale is the same as that of WKE$_{type}$ (Appendix B).

4.6 Attribute Value Metrics (Edge-Level)

We evaluate attribute value extraction at *KG edge level* after KGD canonicalization. Each predicted assertion has the form (PRODUCT, HAS_ATTRIBUTE, VALUE). Because each VALUE node is typed by exactly one ATTRIBUTEKEY via (ATTRIBUTEKEY, HAS_VALUE, VALUE) (Section 3.1), an assertion is counted as correct only if (i) the value is supported by the listing evidence and (ii) the value is attached to the intended key type after canonicalization.

Single Canonical Evaluation Graph To avoid penalizing surface-form variation (e.g., "Grey" vs. "Gray") and to score all models in the same canonical space, we build an evaluation KG by taking the union of value-edge proposals from all evaluated extractors and letting KGD mediate all writes. Each model is then evaluated on the subset of canonicalized edges originating from that model. Human annotators label instantiated edges as Correct/Incorrect using full listing evidence. We report precision, recall, and F1 based on these labeled edges, with true/false positives/negatives defined over canonicalized (PRODUCT,HAS_ATTRIBUTE,VALUE) edges.

4.7 Public Dataset Evaluation: Edge-Level PKG Assessment

To assess AutoPKG beyond our primary benchmark, we conduct an edge-level evaluation on three public datasets: ImplicitAVE, MAVE, and AE-650k. Using the same canonicalized evaluation procedure described in Section 4.6, we construct a single evaluation KG per dataset by taking the value assertions from multiple high-performing LLMs from Section 5.4 and letting KGD determine the final node/edge instantiations. Subsequently, the flagship Qwen3-VL-235B-A22B-Instruct model is used as an AI annotator to generate high-quality reference graphs as silver ground truth. We validate the reliability of these annotations via human evaluation on a sampled subset, observing strong agreement (N = 1,000, Cohen's Kappa = 0.8297; see Appendix G). Finally, we compare subgraphs from public datasets and ours in the shared canonical space and report macro-averaged precision, recall, and F1 (details: Appendix F, Table F.2).

4.8 Public Dataset Evaluation: PKG-Augmented PAVE

We additionally evaluate AutoPKG in a PKG-augmented PAVE setting on the superset of three public datasets used in Section 4.7 (ImplicitAVE, MAVE, and AE-650k). For each dataset, we use the PKG produced in Section 4.7 as a retrieval source for graph retrieval-augmented generation (G-RAG) (Hu et al., 2025b) when performing the PAVE task. We then evaluate the PAVE outputs with Qwen3-VL-235B-A22B-Instruct serving also as an AI annotator to construct silver ground truth annotations, and report macro-averaged precision, recall, and F1. Full evaluation and metric computation details are provided in Appendix F.2. See human-verified silver GT judgments in Appendix G.

Model	Latency (↓)	Acc. (↑)
Qwen3-Next-80B-A3B-Instruct	4:37	**0.764**
Qwen3-30B-A3B-Instruct-2507	1:51	0.734
Qwen3-4B-Instruct-2507	1:38	0.666
Kimi-Linear-48B-A3B-Instruct	37:28	0.610
GroveMoE-Inst	4:39:28	0.594
Llama-3.2-3B-Instruct	2:14	0.384
Gemma-3n-E4B-it	3:06	0.233
SmolLM3-3B	3:23	0.123

Table 2: KGD backbone accuracy against a 5-LLM majority-vote reference (3/5). Time is wall-clock elapsed time; higher accuracy and lower latency indicate better performance.

5 Results

We report results for (i) KGD under multi-LLM judging, (ii) product-type induction and consolidation, (iii) attribute-key discovery under probabilistic evaluation, (iv) multimodal attribute value extraction at the item–value edge level, (v) edge-level evaluation on public datasets, (vi) PKG-augmented PAVE task in E-commerce, and (vii) an industrial online A/B deployment. Latency is reported as wall-clock elapsed time multiplied by the number of GPUs used, formatted as `HH:MM:SS`, under consistent hardware settings (Appendix I).

5.1 KGD: Decision Fidelity and Judge Agreement

Table 2 evaluates candidate backbones for KGD using a 5-judge panel and a 3/5 majority-vote consensus label. Qwen3-Next-80B-A3B achieves the highest consensus accuracy (0.764), followed by Qwen3-30B-A3B (0.734). The drop for smaller models is sharp (e.g., Llama-3.2-3B at 0.384 and SmolLM3-3B at 0.123), suggesting that constrained KG edits still require strong semantic discrimination: mistakes in `MERGE` vs. `ADD` (or overly confident `REPLACE`) have persistent downstream cost because they shape the canonical ontology used for retrieval and extraction.

5.2 Product Type Induction: Validity under Aggressive Consolidation

Table 3 (top block) reports type induction quality after KGD-mediated consolidation. Strong models produce highly valid canonical types, with acceptance around 0.94–0.95 for most candidates. Notably, the system collapses a very large space of raw type strings into a much smaller set of canonical

Product-type Induction						
Model	**Unique Types**	**Node Size**	**Accept.** (↑)	**Comp.** (↑)	**Cov.** (↑)	**WKE** (↑)
Qwen3-4B-Instruct-2507	80,806	16,692	0.952	0.948	0.960	**0.953**
Qwen3-30B-A3B-Instruct-2507	73,620	20,711	**0.954**	0.936	0.960	0.952
Qwen3-Next-80B-A3B-Instruct	81,565	21,608	**0.954**	0.933	0.951	0.949
GroveMoE-Inst	51,957	14,787	0.948	0.954	0.952	0.950
Kimi-Linear-48B-A3B-Instruct	101,062	25,311	0.938	0.946	0.922	0.940
Gemma-3n-E4B-it	37,645	10,603	0.936	**0.967**	**0.986**	0.952
Llama-3.2-3B-Instruct	55,317	17,470	0.628	0.946	0.945	0.726
SmolLM3-3B	198,814	21,649	0.646	0.933	0.613	0.682
Attribute-key Discovery						
Model	**Unique Keys**	**Node Size**	**Accept.** (↑)	**P-Prec.** (↑)	**P-Rec.** (↑)	**WKE** (↑)
Qwen3-235B-A22B-Instruct-2507	23,005	7,994	**0.956**	0.990	0.363	**0.724**
Qwen3-30B-A3B-Instruct-2507	34,738	10,634	0.794	0.934	**0.474**	0.718
Kimi-Linear-48B-A3B-Instruct	24,002	8,560	0.806	0.946	0.367	0.667
Qwen3-4B-Instruct-2507	23,689	7,007	0.822	0.950	0.353	0.664
Qwen3-Next-80B-A3B-Instruct	17,432	6,663	0.942	**0.991**	0.273	0.636
Gemma-3n-E4B-it	32,504	7,924	0.628	–	–	–
Llama-3.2-3B-Instruct	29,059	4,239	0.534	–	–	–
SmolLM3-3B	10,211	1,905	0.494	–	–	–

Table 3: KGD-based consolidation quality for two induction tasks. Top block: *Product-type induction* where *Unique Types* are raw predicted strings and *Node Size* is the number of canonical type nodes after consolidation; *Accept.* is the human validity rate of per-product type predictions (pre-consolidation); *Comp.* and *Cov.* are computed after consolidation; *WKE* aggregates these metrics. Bottom block: *Attribute-key discovery* where *Unique Keys* are raw generated keys and *Node Size* is the number of canonical key nodes after consolidation; *Accept.* is the human validity rate on sampled per-product generated keys (pre-consolidation); *P-Prec.*, *P-Rec.*, and *WKE* are computed using the probabilistic protocol with a pooled key set from the top models.

nodes, yielding high compression (0.933–0.967) while maintaining high coverage (0.945–0.986). This combination indicates that consolidation is not merely deleting long-tail types; rather, it is mostly removing surface-form fragmentation (pluralization, spelling variants, seller-specific phrasing) while still assigning almost all products to a valid type.

WKE summarizes this trade-off. Qwen3-4B achieves the highest WKE (0.953), narrowly ahead of Qwen3-30B (0.952). Interestingly, Gemma-3n achieves the strongest compression and coverage but slightly lower acceptance, consistent with a more aggressive merge tendency: when merges become too permissive, acceptance is penalized strongly by WKE. Overall, these results suggest that for type induction, moderate-size backbones can be sufficient if the KGD policy is stable, and that the dominant failure mode is not missing coverage but subtle semantic drift from over-merging.

5.3 Attribute Key Discovery: High Precision, Recall Limited by Long Tail

Table 3 (bottom block) evaluates induced attribute keys using the probabilistic precision/recall protocol in Section 4.5. Among the evaluated top models, probabilistic precision is consistently high (roughly 0.93–0.99), indicating that once KGD canonicalizes proposals, most surviving keys are meaningful and type-relevant. In contrast, probabilistic recall remains much lower (0.273–0.474), reflecting the open-vocabulary nature of e-commerce attributes: Many useful keys are rare, category-specific, or expressed idiosyncratically.

Qwen3-235B achieves the best WKE (0.724), driven by the strongest acceptance (0.956) and near-ceiling probabilistic precision, making it suitable when ontology growth must be conservative and stable. Qwen3-30B yields the best recall (0.474) and a comparable WKE (0.718), suggesting a complementary role as a higher-coverage proposer if guarded by a strong KGD. Taken together, the key results reinforce that schema evolution is bottlenecked less by filtering invalid keys than by capturing the long tail without inflating the ontology with weakly grounded concepts.

5.4 Attribute Value Extraction: Precision–Recall Trade-offs

Table 4 reports edge-level precision/recall/F1 for multimodal value extraction after KGD canonicalization (Section 4.6). Overall performance is

Model	Latency (↓)	Prec.	Rec.	F1 (↑)
Qwen3-VL-32B	2:59:04	0.483	**0.591**	**0.531**
Gemma-3-27b-it	4:43:12	**0.688**	0.420	0.521
Gemma-3-12b-it	4:15:01	0.626	0.396	0.485
Qwen3-VL-8B	1:50:52	0.455	0.513	0.482
Qwen3-VL-4B	1:50:51	0.436	0.462	0.448
Qwen3-VL-30B-A3B	2:17:22	0.389	0.522	0.446
Gemma-3n-E4B-it	4:56:59	0.521	0.348	0.417
Kimi-VL-A3B	5:58:20	0.394	0.258	0.312

Table 4: Attribute value extraction on the human-labeled set (edge-level) after KGD canonicalization.

Dataset	AVE Task Comparison	Attribute Key			Attribute Value		
		P	R	F1	P	R	F1
ImplicitAVE	Original Dataset	0.714	–	–	0.890	–	–
	Gemma-3-27b-it	**0.970**	0.894	**0.924**	**0.953**	0.876	0.907
	Qwen3-VL-32B	0.911	**0.951**	0.924	0.897	**0.932**	**0.908**
Mave	Original Dataset	0.774	0.187	0.282	0.625	0.175	0.250
	Gemma-3-27b-it	**0.855**	0.663	0.726	**0.895**	0.718	0.777
	Qwen3-VL-32B	0.767	**0.751**	**0.746**	0.802	**0.806**	**0.792**
AE650k	Original Dataset	0.805	0.291	0.407	0.774	0.238	0.344
	Gemma-3-27b-it	**0.826**	**0.597**	**0.671**	**0.856**	**0.631**	**0.707**
	Qwen3-VL-32B	0.466	0.462	0.442	0.487	0.507	0.475

Table 5: PAVE performance on three public datasets.

moderate, reflecting noisy marketplace evidence (incomplete specifications, ambiguous seller texts, and non-diagnostic images). Models exhibit a clear precision–recall trade-off: Qwen3-VL-32B achieves the best F1 (0.531) via the highest recall (0.591), while Gemma-3-27B is most conservative with the highest precision (0.688) but lower recall (0.420), yielding a comparable F1 (0.521).

Latency varies substantially, making model choice a throughput–quality decision. Smaller Qwen3-VL models (4B/8B) provide competitive F1 (0.448–0.482) at much lower runtime. Scaling within a family is not monotonic: the MoE Qwen3-VL-30B-A3B underperforms the dense Qwen3-VL-32B (0.446 vs. 0.531), suggesting structured extraction is sensitive to calibration and routing.

5.5 Public Datasets: Edge-Level Results

The AutoPKG-constructed KG_2 improves exact-match macro-F_1 over the strongest baseline KG_1, with Gemma-3-27b-it delivering the largest average gains: $+0.166$ on ATTRIBUTEKEY nodes and $+0.231$ on ATTRIBUTEVALUE nodes across ImplicitAVE/MAVE/AE–650K (absolute macro-F_1, averaged across datasets), up to $+0.207$ in precision on the multimodal ImplicitAVE dataset. See details in Table F.2. Overall, the gains align with the pipeline mechanism in AutoPKG.

5.6 Public Datasets: PKG-Augmented PAVE Results

Table 5 presents the PAVE results on three public datasets. PKG-augmented methods outperform the original dataset baselines. Gemma-3-27b-it yielding the best overall gains: improving exact-match macro-F_1 by $+0.391$ on ATTRIBUTEKEY and $+0.445$ on VALUE on average over the two text-only settings (MAVE and AE-650K), while further increasing ImplicitAVE precision by $+0.256$ (keys) and $+0.063$ (values). The gains are primarily driven by PKG augmentation: retrieving the relevant KG exposes canonical attribute schemas and candidate values, which constrains generation and improves coverage through graph-supported evidence aggregation.

5.7 Online Deployment: Filter, Badge, Search, and Recommendation

We deployed AutoPKG-derived attributes in four production applications and conducted A/B tests against the existing system (*Base*) to measure the effect of the AutoPKG-derived attributes (*Test*). Appendix L provides UI demos. In UI-facing settings, AutoPKG attributes are used for *Filter* facets and *Badge* display: *Filter* exposes selectable facet values to narrow result sets, while *Badge* provides attribute snippets on the listing card. Over 12/01–12/28, *Filter* shows no statistically significant change (GMV +0.26%), while *Badge* improves GMV by +3.81%. Badge likely benefits from higher effective coverage since partial attributes can still inform decisions without restricting retrieval.

We also deployed AutoPKG for *Attribute Backfilling* to enrich missing attributes on 110M items. *Base* relies on seller-provided attributes (46% fill rate), while *Test* adds AutoPKG backfill (70% fill rate). Over 11/21–12/22, this backfilling improves GMV by +5.32% in *Search* and +7.89% in *Recommendation*, suggesting that most value comes from long-tail coverage gains that strengthen downstream retrieval, matching, and ranking.

5.8 Cost–Quality Trade-off Analysis

To guide deployment decisions, we evaluate three configuration tiers ranging from latency-optimized to performance-maximized setups. Table 6 reports the aggregate quality score (harmonic mean of KGD accuracy, type/key WKE, and edge F_1) alongside inference costs estimated on AWS p5e.48xlarge instances ($4/GPU-hr). See configuration suggestions in Appendix J.

Config	Agent Backbones				Performance Metrics				Avg	Cost
	KGD	Type Ind.	Key Disc.	Value Extr.	Acc_{KGD}	WKE_{type}	WKE_{key}	Edge F_1	Qual.†	Est. ($/1M)
Minimal	Qwen3-30B	Qwen3-4B	Qwen3-30B	Qwen3-VL-8B	0.734	0.953	0.718	0.482	0.680	$12,012
Balanced	Qwen3-30B	Qwen3-4B	Qwen3-30B	Qwen3-VL-32B	0.734	0.953	0.718	0.531	0.703	$12,452
Full	Qwen3-Next-80B	Qwen3-4B	Qwen3-235B	Qwen3-VL-32B	**0.764**	**0.953**	**0.724**	**0.531**	**0.711**	$14,000

Table 6: Detailed cost–quality breakdown across three deployment configurations. †**Avg Qual.**: Harmonic mean of Acc_{KGD}, WKE_{type}, WKE_{key}, and Edge F_1. Costs are estimated based on batch processing at full GPU utilization on 8×NVIDIA H20 GPUs.

Model Variant	WKE_{type}	WKE_{key}	Edge F_1
Full AutoPKG (Ours)	**0.952**	**0.719**	**0.532**
– w/o retrieval context	0.886	0.680	0.490
– w/o visual evidence	0.952	0.719	0.523
– Both ablations	0.886	0.680	0.420

Table 7: Ablation study on the impact of KGD retrieval context and multimodal inputs.

5.9 Ablation Study: Impact of KGD Context and Multimodality

To quantify the contribution of key architectural components, we conduct an ablation study removing the retrieved neighborhood context in the KGD agent and multimodal inputs in the value extractor (Table 7). Removing the KGD retrieval context forces the decision agent to operate without graph consistency checks, leading to a significant drop in schema quality (WKE_{type} decreases by 0.066 and WKE_{key} by 0.039) as the agent fails to effectively merge synonyms or detect duplicates, resulting in a fragmented ontology. Disabling multimodal inputs while retaining text-only extraction causes a modest decline in edge-level F_1 (from 0.532 to 0.523); while text specifications often contain explicit values, visual evidence remains critical for resolving ambiguities, verifying implicit attributes (e.g., pattern, style), and compensating for noisy seller text. The combined removal of both components yields the lowest performance across all metrics, confirming that AutoPKG's effectiveness relies on the synergy between context-aware canonicalization and multimodal evidence aggregation.

6 Conclusion

We introduce AutoPKG, a production-oriented multi-agent framework for building and incrementally evolving product-attribute knowledge graphs from multimodal e-commerce listings. AutoPKG treats product types and attribute keys as dynamic rather than fixed, and centralizes KG updates to maintain global consistency during continual growth. On a large real-world marketplace dataset, we find that type induction can remain highly valid while aggressively consolidating surface variants, that key induction achieves high precision but is recall-limited in the long tail, and that multimodal value extraction exhibits a clear precision–recall trade-off across different backbones. Online A/B tests show heterogeneous production impact: *Badge*, *Search*, and *Recommendation* deliver clear GMV gains, while *Filter* shows no statistically significant change, motivating per-vertical gating and precision-oriented guardrails when promoting extracted attributes to ranking- or UI-facing features. Overall, AutoPKG offers a practical route to reducing manual taxonomy maintenance while keeping a product KG aligned with a fast-evolving catalog.

7 Limitations

AutoPKG writes value into a incremental evolving KG, early judgment errors can persist and propagate: an incorrect `MERGE` may collapse distinct concepts, an overly broad canonical node may distort retrieval context for future products, and an incorrect `ADD` may inflate the ontology with low-utility or noisy schema elements. Although KGD constrains edits to a limited action space, quality remains sensitive to the retrieved neighborhood and to subtle semantic distinctions (e.g., near-synonyms vs. true synonyms, or homographs across domains).

The schema evaluation is also limited by the open-vocabulary setting. In particular, attribute-key quality cannot be exhaustively labeled; our probabilistic protocol depends on system pooling and model-reliability priors, which may underrepresent rare but valuable long-tail keys and may favor keys proposed by multiple similar models. Attribute value extraction is additionally constrained by noisy marketplace evidence, such as inconsistent specifications, missing or low-quality images, and multilingual/code-mixed text, leading to both false positives (spurious assertions) and false negatives (missed implicit attributes).

8 Ethics and Risks

Ethics. This work uses marketplace catalog content and internal employee annotation for product understanding and quality evaluation. It does not involve collection of user data or intervention with human subjects. The study has been reviewed under internal research governance procedures and deemed exempt from external IRB review under applicable policies.

Potential Risks. As an automated framework powered by large language models, AutoPKG may produce incorrect, biased, or misleading attributes that harm user experience (e.g., wrong facet filters or badges), disadvantage sellers (e.g., mischaracterized product properties), or degrade ranking and recommendation when used as features. Quality may vary across languages, regions, and long-tail categories, leading to uneven coverage and disparate impact. While our released subset undergoes internal approval, includes PII redaction where applicable, and is subject to content-safety screening for both listings and images, it may still contain sensitive commercial information (e.g., brands or seller-provided identifiers). We therefore recommend conservative deployment: gate high-impact keys with precision-oriented thresholds, monitor drift and error patterns by vertical and language, and periodically audit canonicalization decisions that affect large fractions of traffic.

References

Jiaxin Bai, Wei Fan, Qi Hu, Qing Zong, Chunyang Li, Hong Ting Tsang, Hongyu Luo, Yauwai Yim, Haoyu Huang, Xiao Zhou, Feng Qin, Tianshi Zheng, Xi Peng, Xin Yao, Huiwen Yang, Leijie Wu, Yi Ji, Gong Zhang, Renhai Chen, and Yangqiu Song. 2025. AutoSchemaKG: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora. *Preprint*, arXiv:2505.23628.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, and 4 others. 2025. SmolLM3: Smol, multilingual, long-context reasoner. `https://huggingface.co/blog/smollm3`.

Haonan Bian. 2025. Llm-empowered knowledge graph construction: A survey. *Preprint*, arXiv:2510.20345.

Jack Boylan, Shashank Mangla, Dominic Thorn, Demian Gholipour Ghalandari, Parsa Ghaffari, and Chris Hokamp. 2024. Kgvalidator: A framework for automatic validation of knowledge graph construction. *Preprint*, arXiv:2404.15923.

Alexander Brinkmann, Roee Shraga, and Christian Bizer. 2024. Extractgpt: Exploring the potential of large language models for product attribute value extraction. *Preprint*, arXiv:2310.12537.

DeepSeek-AI. 2025. DeepSeek-V3 technical report. *Preprint*, arXiv:2412.19437.

Xin Luna Dong, Xiang He, Andrey Kan, Xian Li, Yan Liang, Jun Ma, Yifan Ethan Xu, Chenwei Zhang, Tong Zhao, Gabriel Blanco Saldana, Saurabh Deshpande, Alexandre Michetti Manduca, Jay Ren, Surender Pal Singh, Fan Xiao, Haw-Shiuan Chang, Giannis Karamanolakis, Yuning Mao, Yaqing Wang, and 3 others. 2020. Autoknow: Self-driving knowledge collection for products of thousands of types. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20, page 2724–2734. ACM.

Jiaying Gong and Hoda Eldardiry. 2024. Multi-label zero-shot product attribute-value extraction. *Preprint*, arXiv:2402.08802.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *Preprint*, arXiv:2402.01680.

Pollawat Hongwimol, Dong Sheng, Li Zhang, Kai Liu, and Xiufei Wang. 2025. GAVEL: Generative attribute-value extraction using LLMs on LLM-augmented datasets. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pages 81–90, Albuquerque, New Mexico, USA. Association for Computational Linguistics.

Jiazhen Hu, Jiaying Gong, Hongda Shen, and Hoda Eldardiry. 2025a. Hypergraph-based zero-shot multimodal product attribute value extraction. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 4853–4862, New York, NY, USA. Association for Computing Machinery.

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2025b. GRAG: Graph retrieval-augmented generation. *Preprint*, arXiv:2405.16506.

Bart Lamiroy and Tao Sun. 2013. Computing precision and recall with missing or uncertain ground truth. In *Graphics Recognition. New Trends and Challenges*, pages 149–162, Berlin, Heidelberg. Springer Berlin Heidelberg.

Yuxing Lu and Jinzhuo Wang. 2025. Karma: Leveraging multi-agent llms for automated knowledge graph enrichment. *Preprint*, arXiv:2502.06472.

Xusheng Luo, Le Bo, Jinhang Wu, Lin Li, Zhiy Luo, Yonghua Yang, and Keping Yang. 2021. Alicoco2: Commonsense knowledge extraction, representation and application in e-commerce. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 3385–3393, New York, NY, USA. Association for Computing Machinery.

Xusheng Luo, Luxin Liu, Yonghua Yang, Le Bo, Yuanpeng Cao, Jinghang Wu, Qiang Li, Keping Yang, and Kenny Q. Zhu. 2020. Alicoco: Alibaba e-commerce cognitive concept net. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 313–327, New York, NY, USA. Association for Computing Machinery.

Alessandro Magnani, Feng Liu, Suthee Chaidaroon, Sachin Yadav, Praveen Reddy Suram, Ajit Puthenputhussery, Sijie Chen, Min Xie, Anirudh Kashi, Tony Lee, and Ciya Liao. 2022. Semantic retrieval at walmart. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 3495–3503, New York, NY, USA. Association for Computing Machinery.

OpenAI. 2025. GPT-5 System Card. https://openai.com/index/gpt-5-system-card/.

Dogyun Park and Suhyun Kim. 2023. Probabilistic precision and recall towards reliable evaluation of generative models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20099–20109.

Rohan Potta, Mallika Asthana, Siddhant Yadav, Nidhi Goyal, Sai Patnaik, and Parul Jain. 2024. AttriSage: Product attribute value extraction using graph neural networks. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 89–94, St. Julian's, Malta. Association for Computational Linguistics.

Duangmanee Putthividhya and Junling Hu. 2011. Bootstrapped named entity recognition for product attribute extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Yunxiao Shi, Haoning Shang, Xing Zi, Wujiang Xu, Yue Feng, and Min Xu. 2025. Answering narrative-driven recommendation queries via a retrieve–rank paradigm and the OCG-agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13181–13202, Suzhou, China. Association for Computational Linguistics.

Yunxiao Shi, Wujiang Xu, Tingwei Chen, Haoning Shang, Ling Yang, Yunfeng Wan, Zhuo Cao, Xing Zi, Dimitris N. Metaxas, and Min Xu. 2026. Agentselect: Benchmark for narrative query-to-agent recommendation. *Preprint*, arXiv:2603.03761.

Shivaramakrishnan Kalpetta Subramaniam. 2025. Ai-based e-commerce search optimization. *Journal Of Engineering And Computer Sciences*, 4(6):307–316.

Gemini Team. 2025a. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

Gemma Team. 2025b. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Kimi Team. 2025c. Kimi K2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Kimi Team. 2025d. Kimi Linear: An expressive, efficient attention architecture. *Preprint*, arXiv:2510.26692.

Kimi Team. 2025e. Kimi-VL technical report. *Preprint*, arXiv:2504.07491.

Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Qwen Team. 2025f. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Qwen Team. 2025g. Qwen3-VL technical report. *Preprint*, arXiv:2511.21631.

Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 950–958, New York, NY, USA. Association for Computing Machinery.

Haoyuan Wu, Haoxing Chen, Xiaodong Chen, Zhanchao Zhou, Tieyuan Chen, Yihong Zhuang, Guoshan Lu, Junbo Zhao, Lin Liu, Zenan Huang, Zhenzhong Lan, Bei Yu, and Jianguo Li. 2025. GroveMoE: Towards efficient and superior MoE LLMs with adjugate experts. *Preprint*, arXiv:2508.07785.

Yinan Wu, Xiaowei Wu, Junwen Li, Yue Zhang, Haofen Wang, Wen Du, Zhidong He, Jingping Liu, and Tong Ruan. 2023. MMpedia: A large-scale multi-modal knowledge graph. In *The Semantic Web – ISWC 2023: 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023, Proceedings, Part II*, page 18–37, Berlin, Heidelberg. Springer-Verlag.

Huimin Xu, Wenting Wang, Xin Mao, Xinyu Jiang, and Man Lan. 2019. Scaling up open tagging from tens to thousands: Comprehension empowered attribute value extraction from product title. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5214–5223, Florence, Italy. Association for Computational Linguistics.

Li Yang, Qifan Wang, Zac Yu, Anand Kulkarni, Sumit Sanghai, Bin Shu, Jon Elsas, and Bhargav Kanagal. 2022. Mave: A product dataset for multi-source

attribute value extraction. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 1256–1265, New York, NY, USA. Association for Computing Machinery.

Changlong Yu, Xin Liu, Jefferson Maia, Yang Li, Tianyu Cao, Yifan Gao, Yangqiu Song, Rahul Goutam, Haiyang Zhang, Bing Yin, and Zheng Li. 2024. COSMO: A large-scale e-commerce common sense knowledge generation and serving system at amazon. In *Companion of the 2024 International Conference on Management of Data*, SIGMOD '24, page 148–160, New York, NY, USA. Association for Computing Machinery.

Changlong Yu, Weiqi Wang, Xin Liu, Jiaxin Bai, Yangqiu Song, Zheng Li, Yifan Gao, Tianyu Cao, and Bing Yin. 2023. FolkScope: Intention knowledge graph construction for E-commerce commonsense discovery. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1173–1191, Toronto, Canada. Association for Computational Linguistics.

Duzhen Zhang, Zixiao Wang, Zhong-Zhi Li, Yahan Yu, Shuncheng Jia, Jiahua Dong, Haotian Xu, Xing Wu, Yingying Zhang, Tielin Zhang, Jie Yang, Xiuying Chen, and Le Song. 2025. Medkgent: A large language model agent framework for constructing temporally evolving medical knowledge graph. *Preprint*, arXiv:2508.12393.

Ningyu Zhang, QiangHuai Jia, Shumin Deng, Xiang Chen, Hongbin Ye, Hui Chen, Huaixiao Tou, Gang Huang, Zhao Wang, Nengwei Hua, and Huajun Chen. 2021. AliCG: Fine-grained and evolvable conceptual graph construction for semantic search at Alibaba. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 3895–3905, New York, NY, USA. Association for Computing Machinery.

Guineng Zheng, Subhabrata Mukherjee, Xin Luna Dong, and Feifei Li. 2018. OpenTag: Open attribute value extraction from product profiles. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1049–1058, New York, NY, USA. Association for Computing Machinery.

Tiangang Zhu, Yue Wang, Haoran Li, Youzheng Wu, Xiaodong He, and Bowen Zhou. 2020. Multimodal joint attribute prediction and value extraction for E-commerce product. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2129–2139, Online. Association for Computational Linguistics.

Henry Peng Zou, Vinay Samuel, Yue Zhou, Weizhi Zhang, Liancheng Fang, Zihe Song, Philip S. Yu, and Cornelia Caragea. 2024a. ImplicitAVE: An open-source dataset and multimodal LLMs benchmark for implicit attribute value extraction. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 338–354, Bangkok, Thailand. Association for Computational Linguistics.

Henry Peng Zou, Gavin Heqing Yu, Ziwei Fan, Dan Bu, Han Liu, Peng Dai, Dongmei Jia, and Cornelia Caragea. 2024b. 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.

Huike Zou, Haiyang Yang, Yindu Su, Liyu Chen, Chengbao Lian, Qingheng Zhang, Shuguang Han, and Jufeng Chen. 2025. Multi-value-product retrieval-augmented generation for industrial product attribute value identification. *Preprint*, arXiv:2509.23874.

A Product Knowledge Graph Example

This appendix provides a concrete example of the Product-attribute Knowledge Graph (PKG) produced by AutoPKG. Figure A.1 visualizes a local neighborhood after KGD canonicalization.

In the figure, large green nodes denote canonical PRODUCTTYPE concepts (e.g., *Smartwatch*, *Desktop Computer*). Orange nodes denote ATTRIBUTEKEY slots that are valid for a given product type (e.g., *Brand*, *GPU*, *RAM Capacity*, *Storage Type*). Purple nodes denote individual PRODUCT items. The remaining nodes correspond to canonical VALUE entities (e.g., *SSD*, *16GB*, *NVIDIA GeForce*), which are reused across items when surface forms refer to the same meaning.

Edges follow the KG schema defined in Section 3.1. At the schema level, `(ProductType, has_key, AttributeKey)` specifies which keys apply to a type, and `(AttributeKey, has_value, Value)` assigns each value node to exactly one attribute key (the value typing constraint). At the instance level, `(Product, of_type, ProductType)` links an item to its canonical product type, and `(Product, has_attribute, Value)` encodes item-specific attribute assertions.

The example can be read as follows. Items typed as *Desktop Computer* connect to type-relevant keys such as *GPU* and *RAM Capacity*; those keys in turn type their value nodes (e.g., *NVIDIA GeForce* for *GPU*, *8GB*/*16GB* for *RAM Capacity*). Item-level edges then link specific products to the values supported by their evidence (title/description/specifications/images). This design enables type checking (preventing values whose key is not defined for the product type) and canonicalization (merging synonymous surface variants

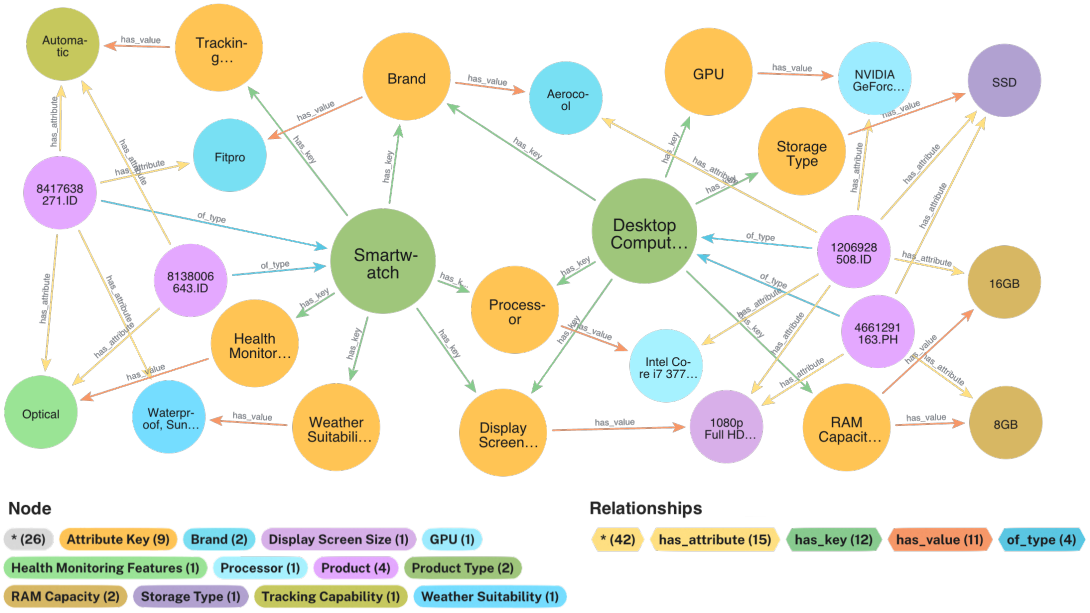

Figure A.1: Example subgraph of a PKG constructed from extracted product attributes.

into shared nodes), which is critical for continual KG growth under noisy marketplace data.

B WKE Weighting and Sensitivity

We use a 3:1:1 weighting in WKE to prioritize semantic validity. Acceptance is human-labeled and directly measures whether induced types/keys are correct; in contrast, Compression/Coverage (and P-Prec/P-Rec for keys) can be improved by policies that change ontology size without improving correctness (e.g., aggressive `MERGE` decisions). In a continually written KG, invalid schema nodes are high-cost because they persist and can misguide retrieval and downstream extraction, whereas lower compression or coverage is typically recoverable in later updates.

We performed an offline sensitivity check with alternative weights (2:1:1 and 4:1:1). Using 2:1:1 occasionally elevated models/policies that achieved higher compression primarily via over-merging, despite lower human acceptance. Using 4:1:1 produced similar rankings to 3:1:1 but reduced the influence of efficiency/coverage. Overall, 3:1:1 best aligned the WKE ranking with our qualitative assessment and human annotations by favoring models that are both accurate and reasonably efficient.

C AutoPKG Workflow Design

To support automatic and dynamic product-KG construction, AutoPKG adopts a modular, centralized multi-agent orchestration mechanism that delegates core subtasks—ontology population, attribute–value extraction, KG update, and optional refinement procedures—to specialized Agents, as illustrated in Figure C.1 (Guo et al., 2024; Lu and Wang, 2025; Shi et al., 2025, 2026).

Formally, agent communication follows a centralized protocol. Given a product input x, each task-specific agent i produces an intermediate output

$$a_i = \pi_i(h_i, x), \tag{1}$$

where π_i denotes the agent-specific policy implemented by an LLM, and h_i represents the agent's internal state. All agent outputs are transmitted to the KGD agent, which aggregates them and determines the subsequent KG operations.

Each agent operates as an autonomous functional module while coordinating through the KGD agent, which serves as the sole read–write interface to the KG. The Ontology Population agent block and the Attribute Value Extraction Agent performs as described in Sections 3.2, 3.3, 3.4, respectively.

The Ontology Population agent block performs a two-stage schema alignment process (product-type induction with canonical type-description generation, and key-attribute discovery) and hands the resulting schema to the Decision Agent for controlled integration. The Attribute Value Extraction Agent retrieves type-specific attribute tables via the Decision Agent and extracts attribute values under a RAG-style workflow, proposing updates to the

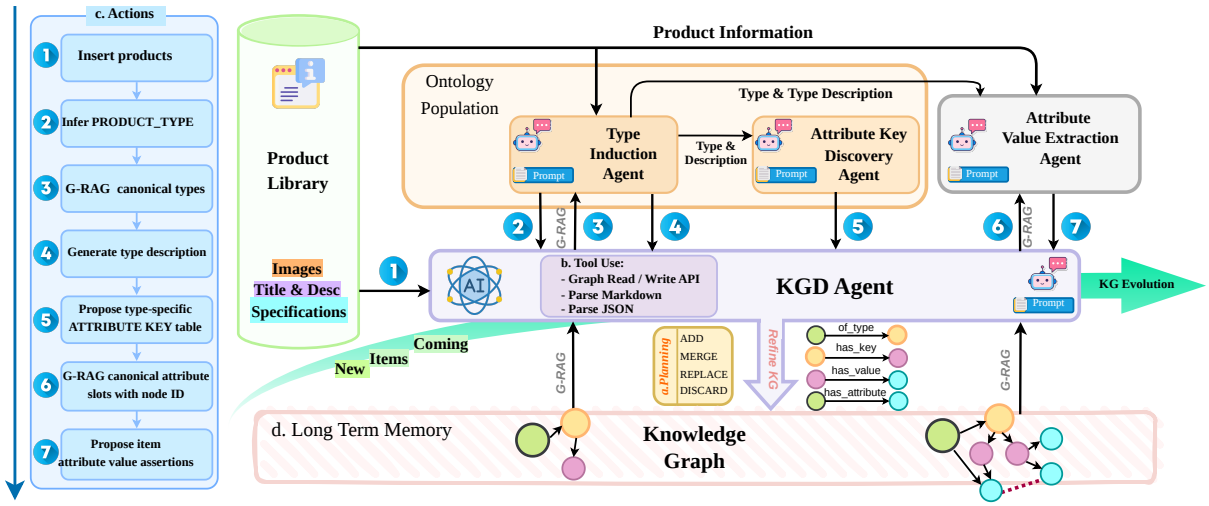

Figure C.1: Multi-Agent Framework.

KGD Agent when necessary.

The overall objective of the system is to incrementally construct a consistent and saturated KG, which can be expressed as

$$\hat{G} = \arg\max_{G} R(G), \tag{2}$$

where $R(G)$ measures KG completeness and structural consistency under the saturation criterion.

Long-term memory and retrieval-coupled generation From a multi-agent perspective, we treat $\mathcal{G}$ as a long-term memory that stores canonicalized types/keys/values and their local structural neighborhoods. At each step, G-RAG retrieves context from $\mathcal{G}$ (e.g., nearest canonical nodes and type-conditioned key tables) to condition proposal generation and to constrain subsequent decisions. As a result, the pipeline is inherently *non-linear*: each accepted update changes the retrieval landscape, which in turn reshapes future proposal distributions and progressively stabilizes the induced schema.

KGD: A planner with a Constrained Action Space While multiple agents contribute distinct competencies, AutoPKG centers on a *Knowledge Graph Decision* agent (KGD) as the critical planner and *sole write interface* to $\mathcal{G}$. Rather than directly committing open-ended model outputs, KGD resolves every upstream proposal through a constrained edit space—`ADD`, `MERGE`, `REPLACE`, or `DISCARD`—thereby projecting unconstrained generation into an explicit, auditable update language. This centralization makes continual canonicalization operational: KGD arbitrates deduplication, synonym consolidation, and label promotion using retrieved local neighborhoods as decision context, ensuring that graph growth remains controlled as coverage expands.

Tool-grounded execution and emergent bootstrapping. KGD is tool-grounded: it performs graph read and write via explicit APIs and parses structured LLM outputs (e.g., JSON/Markdown) into executable KG transactions (node/edge creation and insertion statements). The resulting retrieval–decision–write feedback loop enables *self-bootstrapping* and emergent behavior: schema elements induced early become anchors for later extraction, repeated consolidation induces attractor-like canonical nodes, and the system continuously refines both ontology (types/keys) and facts (values) **without manual curation**. In aggregate, AutoPKG realizes a fully automated, continually updating PKG construction process whose reliability derives from the synergy between specialized proposal agents and KGD's constrained, memory-conditioned planning.

D Human Annotation Protocol and Guidelines

We use trained in-house annotators to evaluate (i) product type validity, (ii) attribute key validity, and (iii) item–value edge correctness after KGD canonicalization. Annotation is performed as paid work by professional annotators as part of their employment. The released dataset subset described in Section 4.1 has been approved for release by the company under internal governance.

D.1 Annotation Materials and Setup

Annotators are shown a single product listing consisting of a title, description/highlights, structured specifications when available, and up to ten images. They are also shown the specific model output to be judged. Annotators are instructed to base decisions only on the provided listing evidence and not to use external resources.

D.2 Labels

For product type and attribute key evaluation, annotators assign Accept or Reject. For attribute value evaluation, annotators assign Correct or Incorrect to each instantiated edge of the form `(Product, has_attribute, Value)` after KGD canonicalization.

D.3 General Guidelines

Annotators judge semantic correctness rather than surface form. Spelling variants, formatting differences, and clear synonyms are acceptable (e.g., "Gray" vs. "Grey"). When evidence is insufficient, annotators should not guess and instead choose Reject or Incorrect. Outputs that are subjective or promotional in nature (e.g., "High Quality", "Best", "Premium") are labeled Reject or Incorrect. When different fields conflict (e.g., title vs. specifications), annotators label Incorrect unless one source clearly dominates through repeated mentions, internal consistency, and/or strong visual support.

D.4 Task 1: Product Type Validity

A product type should describe what the item is, using a canonical category name that generalizes across sellers. A prediction is labeled Accept if it is a valid, unambiguous product type supported by the listing, and if it does not include brand names, model identifiers, marketing language, or attribute phrases. A prediction is labeled Reject if it is incorrect for the listing, too vague to be actionable (e.g., "Item", "Accessory"), primarily an attribute value (e.g., "Red"), or an over-specific seller phrasing that does not generalize.

D.5 Task 2: Attribute Key Validity

An attribute key should represent an inherent, type-relevant characteristic that a buyer can use to compare products, rather than logistics or commercial metadata. A key is labeled Accept if it is applicable to the product type, refers to a stable product characteristic (e.g., brand, material, capacity, compatibility), and is not a redundant paraphrase of another key at the same meaning. A key is labeled Reject if it corresponds to shipping, seller, pricing, warranty administration, SEO/marketing claims, or if it is too vague or inapplicable to the type.

D.6 Task 3: Attribute Value Edge Correctness

Each predicted assertion is evaluated at the KG edge level after canonicalization. An edge is labeled Correct only if the value is supported by evidence in the title, description, specifications, and/or images, and if the value matches the intended attribute meaning under its key type after canonicalization. An edge is labeled Incorrect if the value is not supported, is only weakly inferred, conflicts with stronger evidence, or is attached to the wrong key type (for example, a battery form factor being recorded under a brand key).

D.7 Sampling and Annotation Volume

For product type validity, we sample 500 products and evaluate per-product type predictions from 8 models, yielding 4,000 judgments. For attribute key validity, we sample a set of 500 product types, each model generates a type-specific key table and we uniformly sample one key per product type per model, again yielding 4,000 judgments. For attribute value edges, annotators label instantiated edges `(Product, has_attribute, Value)` as Correct or Incorrect using the full listing evidence of the same set, containing 9,040 judgments.

D.8 Recruitment and Payment

Annotators are full-time professional annotators employed by the organization operating the marketplace. Annotation is conducted as part of paid work at rates aligned with local regulations and internal compensation standards.

D.9 Consent and Data Use

The annotated product listings are seller-provided catalog content. Annotators are instructed to use only the provided listing materials and not to access external resources. The study does not involve interaction with end users.

E LLM Judge Agreement for KGD Evaluation

In Section 4.3, we evaluate candidate KGD backbones against a reference label derived from majority vote among five frontier LLM judges. To assess the reliability of this reference—and to justify its use as a scalable evaluator for KGD edit

	GPT-5	Gemini-2.5-Pro	Qwen3-Max	Kimi-K2	DeepSeek-V3.1	Consensus	Human
GPT-5	1.000	**0.812**	0.687	0.695	0.573	0.798	0.799
Gemini-2.5-Pro	**0.812**	1.000	0.710	0.730	0.580	0.827	0.794
Qwen3-Max	0.687	0.710	1.000	**0.812**	**0.730**	0.861	0.764
Kimi-K2	0.695	0.730	**0.812**	1.000	0.696	**0.876**	0.750
DeepSeek-V3.1	0.573	0.580	**0.730**	0.696	1.000	0.721	0.635
Consensus	0.798	0.827	0.861	**0.876**	0.721	1.000	**0.837**
Human	0.799	0.794	0.764	0.750	0.635	**0.837**	1.000

Table E.1: Pairwise Cohen's κ agreement among the LLM judges and human annotators used in Section 4.3.

actions—we analyze inter-judge agreement, including alignment with human annotations. Specifically, we compute pairwise Cohen's κ for all judge pairs, including human annotators on sample of 2,000 KGD decision instances (10,000 total judgments from 5 AI judges), on the same set of KGD decision instances used to form the 3/5 consensus labels. Cohen's κ quantifies agreement beyond chance for categorical decisions, with higher values indicating greater consistency. As shown in Table E.1, the LLM judges exhibit substantial mutual agreement, and—critically—their consensus aligns closely with human judgments (e.g., $\kappa = 0.837$ between the Consensus panel and Human). This strong alignment supports the use of majority-vote LLM consensus as a reliable and scalable surrogate for human evaluation in KGD edit assessment.

F Open-Source Dataset Comparison Experimental Setup

We adopt widely used public AVE task datasets as benchmarks to ensure transparent, reproducible comparison and to validate generalization beyond any single marketplace snapshot. Concretely, we construct evaluation splits by sampling high-quality product–attribute instances from three influential datasets that collectively cover both multimodal and text-only listing regimes—ImplicitAVE (text+images), MAVE (text), and AE-650K (text)—thereby spanning diverse categories, attribute taxonomies, and annotation styles. These benchmarks have broad community uptake and are among the most frequently cited product-attribute resources in recent years, which makes them a strong reference point for positioning performance. The resulting evaluation set sizes (70,214 / 275,049 / 39,505 instances, respectively) provide substantial coverage while keeping the protocol consistent across settings. Table F.1 lists the details of these datasets.

F.1 Fine-grained KG Node Quality Comparison Experiment

Rationale for Edge-level Precision and Recall We evaluate product–attribute KGs using edge-level precision and recall at both the attribute key and attribute value granularity. This choice reflects the core objectives of dynamic product KG construction and aligns with standard evaluation practices in knowledge extraction and KG population.

At the attribute key level, edge-level precision measures the correctness of induced attribute schemas, indicating whether predicted attribute nodes correspond to semantically valid and product-relevant attributes. Recall measures schema coverage, capturing whether salient attributes present in a reference KG are successfully recovered. Together, they quantify the quality of attribute induction and canonicalization.

At the attribute value level, edge-level metrics assess the correctness and completeness of atomic key–value facts, abstracting away from graph topology and focusing on factual content. High precision reflects accurate value extraction, while high recall indicates effective population of attribute values.

Crucially, edge-level performance also reflects graph construction quality beyond extraction accuracy. In AutoPKG, all updates are mediated by the centralized Knowledge Graph Decision (KGD) agent. High precision implies effective filtering of noisy proposals, while high recall indicates that consolidation and alignment decisions do not suppress valid knowledge. Thus, edge-level metrics jointly capture extraction quality and the effectiveness of knowledge fusion and canonicalization.

Data Subsets and Input Modalities To balance computational cost and dataset diversity, we evaluate on stratified subsets from three public benchmarks:

ImplicitAVE (multimodal): 10,000 products
MAVE (text-only): 55,000 products

Dataset	Release Year	# Products	# Categories	Avg. #Keys per Product	Avg. #Values per Key	Lan-guage	Multi-modal	Applicable Fields & Introduction
ImplicitAVE	ACL 2024 Findings	70,214	167	1.000	1.000	English	✓	Sampled from MAVE.
MAVE	WSDM 2022	275,049	950	1.629	3.395	English	✗	Amazon.
AE–650k	ACL 2019	39,505	558	2.250	1.000	English	✗	AliExpress.

Table F.1: Open-source dataset statistics.

Datasets	KG Node Quality Comparison	Attribute Key Node			Attribute Value Node		
		P	R	F1	P	R	F1
ImplicitAVE	KG_1`raw`	**0.8264**	0.1028	0.1789	0.6945	0.0828	0.1449
	KG_2`google/gemma-3-27b`	0.7289	0.6265	0.6641	**0.9031**	0.7854	**0.8285**
	KG_3`Qwen/Qwen3-VL-32B`	0.6858	**0.7132**	**0.6909**	0.7607	**0.8014**	0.7707
Mave	KG_1`raw`	0.6339	0.1677	0.2458	**0.7669**	0.178	0.2736
	KG_2`google/gemma-3-27b`	**0.6908**	**0.4107**	**0.4909**	0.453	0.2472	0.3018
	KG_3`Qwen/Qwen3-VL-32B`	0.1665	0.1569	0.1476	0.3653	**0.3817**	**0.3471**
AE–650k	KG_1`raw`	**0.8392**	0.2997	**0.4127**	**0.8002**	0.2423	**0.3447**
	KG_2`google/gemma-3-27b`	0.4057	0.128	0.1793	0.6359	0.2399	0.325
	KG_3`Qwen/Qwen3-VL-32B`	0.5138	**0.3231**	0.3633	0.4452	**0.2611**	0.2987

Table F.2: Fine-grained KG edge-level quality evaluation results on three datasets.

AE-650k (text-only): 39,505 products

These subsets preserve original category distributions while remaining tractable for large-scale LLM-based KG construction. For text-only datasets, we use the concatenation of product titles and detailed descriptions as model input, reflecting realistic e-commerce settings where images may be unavailable.

Knowledge Graph Construction We construct three KGs from the same product subsets:

KG_{raw}. A baseline KG built directly from dataset triplets. The graph is implemented in Neo4j with global node deduplication. Two relation types are used: HAS_KEY (product–attribute) and HAS_VALUE (attribute–value).

KG_{Gemma} / $KG_{Qwen\text{-}VL}$. Two predicted KGs constructed using the full AutoPKG framework, starting from an empty graph. All schema induction, extraction, and updates are proposed by agents and executed exclusively through the KGD agent. The two variants differ in the Multimodal Attribute Value Extraction model: isolating model effects from graph-level consolidation.

Structural statistics of KGs constructed under different variants are reported in Table F.3.

Reference KG Construction As no dataset provides a fully canonicalized product KG, we construct a reference graph (KG*) using a G-RAG-based annotation pipeline with the Qwen3-VL-235B LLM annotator. The prompting enforces canonical attribute naming, value normalization, and duplicate suppression, yielding a globally consistent **reference KG** as our silver level ground truth. Example prompts and outputs are shown in Figure K.7

Calculation Method For each product i, let K_i and $\hat{K}_i$ denote the sets of distinct attribute keys in $KG^\star$ and a predicted KG, respectively, and let V_i and $\hat{V}_i$ denote the sets of distinct key–value facts (k, v).

Key-level scoring We compute per-product precision and recall as

$$P_K(i) = \frac{|\hat{K}_i \cap K_i|}{|\hat{K}_i|}, \qquad R_K(i) = \frac{|\hat{K}_i \cap K_i|}{|K_i|}.$$

Value-level scoring Value true positives require key agreement and a substring match:

$$(k, v) \in \mathrm{TP}_V(i) \iff \exists (k, v^\star) \in V_i \text{ s.t. } v^\star \supseteq v,$$

yielding

$$P_V(i) = \frac{|\mathrm{TP}_V(i)|}{|\hat{V}_i|}, \qquad R_V(i) = \frac{|\mathrm{TP}_V(i)|}{|V_i|}.$$

Results Explained We report macro-averaged precision, recall, and F1 over the intersection of

Dataset	kg version	Node Size				Edge Size			
		Product	Type	Key	Value	Has Type	Has Key	Has Value	Has Attribute
ImplicitAVE	KG_1$_{\texttt{raw}}$	10000	–	23	152	–	10000	152	–
	KG_2$_{\texttt{gemma}}$	10000	779	1178	2604	10000	9561	2612	51049
	KG_3$_{\texttt{Qwen}}$	10000	779	1178	3692	10000	9561	3703	106680
MAVE	KG_1$_{\texttt{raw}}$	55010	–	491	16379	–	55010	16379	–
	KG_2$_{\texttt{gemma}}$	55010	1462	2780	15181	55010	19241	15183	175656
	KG_3$_{\texttt{Qwen}}$	55010	1462	2780	18498	55010	19241	18498	330545
AE–650k	KG_1$_{\texttt{raw}}$	39505	–	2584	12497	–	39505	12497	–
	KG_2$_{\texttt{gemma}}$	39505	558	1344	2652	39505	7013	2652	83076
	KG_3$_{\texttt{Qwen}}$	39505	558	1344	3530	39505	7013	3530	177849

Table F.3: Structural statistics of KGs constructed under different variants. For $\mathrm{KG}_{\mathrm{raw}}$, type nodes and the corresponding `has_attribute` and `of_type` relations are intentionally omitted, as this design represents a deliberate trade-off between storage footprint and computational efficiency.

predicted and reference product IDs, and report the results in Table F.2.

Across three datasets (ImplicitAVE, MAVE, and AE-650K), our LLM-constructed PKG (KG_2) improves exact-match macro-F_1 over the strongest baseline KG_1 (raw triplet KG directly converted from the original product -> attribute name and attribute -> attribute value data.), with `google/gemma-3-27b-it` delivering the largest average gains: $+0.166$ on ATTRIBUTEKEY nodes and $+0.231$ on ATTRIBUTEVALUE nodes across ImplicitAVE/MAVE/AE–650k (absolute macro-F_1, averaged across datasets), up to $+0.684$ F1 gain on multimodal ImplicitAVE. The improvements are most pronounced on the multimodal ImplicitAVE setting, where KG_2 substantially increases both key and value F_1 relative to KG_1, indicating markedly better node coverage under exact matching. On MAVE, KG_2 increases key-level F_1 and yields modest but consistent gains on value nodes, suggesting that graph-guided consolidation reduces missing and redundant schema elements. On AE-650k, key-node F_1 can drop for some extractors, while value-node F_1 remains close to the baseline, reflecting a harder consolidation regime with stricter canonicalization requirements. Overall, the gains align with the G-RAG-style decision mechanism in KGD: retrieving the local KG neighborhood during edit validation provides schema context that suppresses spurious node additions and improves recall through consistency-aware merging.

F.2 Downstream PAVE Task Assessment

G-RAG has been demonstrated to be an effective paradigm for enhancing the generative capabilities of large language models (LLMs). Within the AutoPKG framework, the methodological design presented in Section 3.3, together with the experimental results reported in Section 5.4, empirically demonstrates the performance gains of G-RAG in attribute value extraction tasks.

Following the same implementation protocol described in Figure 1 and Section 3.3, we extend the framework to realize G-RAG in three external datasets as illustrated in Appendix F, ensuring a fair and controlled comparison across retrieval paradigms. Let $TP_i = \{(k, v) \in \hat{Y}_i : \exists (k, v^\star) \in Y_i \text{ s.t. } v^\star \supseteq v\}$, requiring key agreement and substring value matching. Per-product precision and recall are

$$P(i) = \frac{|TP_i|}{|\hat{Y}_i|}, \qquad R(i) = \frac{|TP_i|}{|Y_i|}.$$

Below, we illustrate how retrieved subgraph content applied to the PAVE generative task, showcasing how graph-structured knowledge can be effectively integrated into the generation process.

G-RAG for Attribute Keys The product type and the candidate attributes, together with their descriptions, examples, and synonyms, are retrieved as a text-based subgraph from the existing KG and provided as reference context and candidate guidance for attribute-value generation for a specific product. This reference content is incorporated into the prompt for the generative LLM in the same manner as in Figure K.5

G-RAG for Attribute Values Type, Attribute Candidates with Description, Examples and Synonyms are retrieved as a text-based sub-graph from the existing KG and provided as a reference and candidates for potential attribute value generation, for a specific product. The reference content if put into the prompt for generative LLM in the same way as Figure K.6

Results We report macro-averaged P, R, F_1 over product IDs in Table 5. All runs are inference-only; the system input is $(x_i, \mathcal{G})$ and the output is a deduplicated set of (pid, k, v) facts.

G Human Validation of LLM-based Annotations for Sections 4.7 and 4.8

To assess the reliability of the LLM-generated annotations used for constructing silver reference KGs on public datasets, we conduct a human validation study on a randomly sampled subset of 1,000 instances drawn from ImplicitAVE, MAVE, and AE-650k. Human annotators independently annotate these instances following the same protocols as those used in the Section 4.7 edge-level assessment and the Section 4.8 PKG-augmented PAVE task.

We measure the agreement between LLM-based annotations (produced by Qwen3-VL-235B-A22B-Instruct) and human labels using Cohen's Kappa and exact agreement rate. The results indicate strong and consistent alignment between LLM and human annotations across both tasks:

- **KG Edge-Level Assessment (Section 4.7):**

  Cohen's Kappa = 0.8297, agreement rate = 86.71%;

- **PKG-Augmented PAVE (Section 4.8):**

  Cohen's Kappa = 0.8255, agreement rate = 84.09%.

These results indicate substantial agreement and support the reliability of the LLM-based annotations as a scalable proxy for human judgment in both edge-level KG evaluation and PKG-augmented reasoning tasks, enabling substantial reductions in annotation effort (approximately 2,560 annotator-hours).

H Data Release, Privacy, and Content Safety

We plan to release, upon acceptance, a vetted subset of our product-listing sample together with a snapshot of the constructed product–attribute knowledge graph. The release will include (i) product listing text (title/description/specifications), (ii) associated product image URLs, and (iii) the PKG snapshot.

The data is derived from seller-provided catalog content and is not intended to identify individual people. It does not include end-user data (e.g., user profiles, messages, or behavioral logs). Nevertheless, to reduce privacy and safety risks prior to release, we will apply content-safety screening to exclude listings flagged for potentially unsafe content (e.g., adult or violent content). We will also remove or redact obvious personally identifying information (PII) patterns in text when present (e.g., phone numbers, email addresses, and messaging handles). Images will be screened using automated detectors, followed by manual spot checks on random samples to validate screening quality.

I Hardware and Inference Details

Hardware. All experiments were run as containerized batch-inference jobs on Alibaba Cloud with NVIDIA H20 GPUs (141 GB GPU memory), using a CUDA 12.8 / PyTorch 2.8.0 runtime image (Python 3.10.13, GCC 13).

Serving stack. We use vLLM for inference when supported by the model and deployment environment; otherwise we fall back to standard HuggingFace inference with equivalent decoding settings. We enable prefix caching when available to reduce repeated prompt overhead in batched evaluation.

Inference details. All LLM/MLLM backbones are evaluated in a zero-shot setting using fixed prompt templates (Appendix K) and *non-thinking* inference (i.e., without explicit test-time reasoning modes). Unless otherwise noted, decoding hyperparameters are held constant across models: temperature $= 0.7$, top-p $= 0.8$, top-k $= 20$, and max_new_tokens $= 6400$. For attribute value extraction, we include up to 10 images per product (`limit_mm_per_prompt`$= 10$). We set GPU memory utilization to 0.9 for batch serving (`gpu_memory_utilization`$= 0.9$). For neighborhood context construction in Section 3.4, we set the retrieval hyperparameter $k = 10$.

J Configuration Suggestions

The *Minimal* config leverages smaller backbones to achieve the lowest cost but suffers in multimodal

Dataset	Section 4.7: Edge-Level Assessment				Section 4.8: PKG-Augmented PAVE			
	KG Attribute Key Nodes		KG Attribute Value Nodes		Attribute Key		Attribute Value	
	κ	Agree (%)	κ	Agree (%)	κ	Agree (%)	κ	Agree (%)
ImplicitAVE	0.8251	86.41	0.8015	84.32	0.8572	85.90	0.7871	80.18
MAVE	0.8408	89.73	0.8392	84.69	0.8429	87.12	0.8128	84.08
AE650k	0.8583	87.58	0.8135	87.52	0.8515	86.66	0.8014	80.62

Table G.1: Agreement between LLM-based annotations and human labels on public datasets for Section 4.7 (edge-level assessment) and Section 4.8 (PKG-augmented PAVE). We report Cohen's Kappa (κ) and exact agreement rate.

extraction fidelity (Edge F_1: 0.482). The *Balanced* config upgrades only the value extractor; this yields a substantial quality leap (+3.4% in Avg Quality) by maximizing Edge F_1 to 0.531—matching the Full config's extraction performance—for a negligible cost increase (+3.7%). The *Full* config further employs maximum-capacity models for KGD and Key Discovery, improving average quality by an additional +1.1% (total +4.6% over Minimal) at a moderate premium (+16.5%). These results suggest that for most production scenarios, the *Balanced* tier offers the optimal return on investment, securing high-fidelity attribute extraction without the overhead of larger schema induction models.

K Prompts Used in AutoPKG

K.1 KGD Prompt

KGD is the sole write interface to the PKG. For each proposed node (or edge), KGD is provided with (i) a set of top-k retrieved *relevant KG nodes* of the same node type (e.g., PRODUCTTYPE, ATTRIBUTEKEY, VALUE), and (ii) a *candidate* node generated by upstream agents (e.g., product type suggestion, schema induction, or value extraction). KGD then selects one of four actions—`ADD`, `MERGE`, `REPLACE`, or `DISCARD`—to ensure global consistency and prevent redundancy. In particular, `MERGE` collapses surface-form variants into an existing canonical node, `REPLACE` updates the canonical label to a more standard naming, and `DISCARD` filters vague or weakly grounded proposals. To enable lightweight post-processing, the output is constrained to a single-line decision (action only, or action plus matched `node_id`).

Figure K.1 shows an example prompt instance for product type canonicalization, where the candidate *Wall Anchor* is compared against retrieved neighbors such as *Wall Anchors* and *Concrete Anchor* to decide whether to add a new type or merge with an existing one. Figure K.2, K.3 shows further versions.

K.2 Product Type Induction

Figure K.4 shows a lightweight prompt to normalize each raw product listing into a *canonical product type* string. The prompt instructs the model to remove non-type information (e.g., brand names, model numbers, attribute phrases, and marketing language) and to abstract the input to the most general category that remains unambiguous. This design encourages consistent, reusable type nodes (improving mergeability across sellers and countries) while avoiding over-generalization that would harm downstream attribute schema induction. If the input is not a valid product or lacks sufficient evidence, the model is required to abstain by returning `None`. The resulting type proposal is then passed to KGD (Section 3.4) for graph insertion or canonicalization via `ADD`/`MERGE`/`REPLACE`/`DISCARD`.

K.3 Attribute Key Discovery

Figure K.5 shows the prompt used to generate a type-specific attribute-key table (standard keys, definitions, and representative value examples) conditioned on the inferred product type and its short type description. The model must output *only* a Markdown table, with attributes sorted by buyer importance, starting with `Brand`. Values in the `Examples` column are constrained to proper title case and 5–10 representative examples when applicable.

K.4 Attribute Value Extraction

Figure K.6 presents the multimodal prompt used for attribute value extraction once a product type has been determined and its attribute schema (IDs, names, descriptions, and example values) is available. The model is instructed to read the product title, highlights, description, structured specifications, and the associated image, and to populate a JSON dictionary keyed *only* by attribute IDs. To support downstream deterministic ingestion, the

output format is tightly constrained: attributes with multiple mentions are returned as lists, and any attribute that is not explicitly stated or cannot be inferred with high confidence is set to `null`.

L Online Deployment: Filter and Badge Demos

Figures L.1 and L.2 illustrate how AutoPKG-derived attributes are surfaced in real user flows. For *Filter*, attribute values are promoted to faceted options (e.g., *Skin Type*) that let users narrow result sets with semantically meaningful constraints. For *Badge*, the same PKG attributes are rendered as concise highlights on product cards (e.g., *Material*, *Style*), improving attribute visibility and reducing the effort needed to compare items.

KGD prompt example (Basic Version)

Instruction: You are a product knowledge graph decision agent.

Relevant nodes from KG:
[{'node_id': 4587, 'node_name': 'Wall Anchors', 'description': 'A category of fastening devices designed to securely attach objects to hollow or brittle walls ...'}, {'node_id': 3762, 'node_name': 'Tie-Down Anchor', 'description': 'A specialized hardware component ...', 'synonyms': ['Tie-down loops']}, ... (truncated) ...]

Node Type: Product Type

Candidate: {'node_name': 'Wall Anchor', 'description': 'Wall Anchor: A hardware product type used to securely fasten objects to hollow walls or masonry surfaces ...'}

Choose one action from:
- ADD: Candidate is not in the KG and not a synonym of an existing node.
- MERGE: Candidate already exists in KG or is a synonym of an existing node, and the existing node name is the preferred form.
- REPLACE: Candidate is a synonym of an existing node, but the candidate name is the preferred or more standard form, so it should replace the existing node name.
- DISCARD: Candidate is invalid, vague, or unreliable.

Preferred Naming Rules:
- Use the most common international or scientific name unless the local name adds significant value.
- If candidate is clearer, more standard, or better aligned with naming conventions, choose REPLACE.
- If candidate is less standard or too narrow, use MERGE.
- Always preserve other name variants in synonyms.

Output format:
- For ADD or DISCARD: output only the action in uppercase.
- For MERGE or REPLACE: output the action in uppercase, followed by a space, then the ‘node_id‘ of the matched existing node.

Example outputs:
- MERGE 1749
- REPLACE 3943
- ADD
- DISCARD

Example output: MERGE 4587

Figure K.1: KGD prompt example (basic version). Given retrieved KG context (top-k relevant nodes) and a candidate node proposal, KGD selects a single constrained action—`ADD`, `MERGE`, `REPLACE`, or `DISCARD`—to control KG growth and enforce canonicalization. This is a minimum viable version.

KGD prompt example (Strict Version)

Instruction: You are a product knowledge graph decision agent.

Goal: Decide whether to ADD a new node, MERGE with an existing node, REPLACE an existing node name, or DISCARD the candidate.

Relevant nodes from KG: {pretty_nodes}

Node Type: {node_type}

Candidate: {pretty_candidate}

Choose one action from:
- ADD: Candidate is a distinct concept not present in the KG.
- MERGE: Candidate is the SAME concept as an existing node (true synonym), and the existing node name is the preferred form.
- REPLACE: Candidate is the SAME concept as an existing node (true synonym), but the candidate name is the preferred/standard form and should replace the existing node name.
- DISCARD: Candidate is invalid, too vague, unreliable, or not a real concept for this Node Type.

STRICT SYNONYM POLICY (required for MERGE/REPLACE):
- Same meaning and scope (no broader/narrower relationship)
- Same real-world referent (can be substituted in a sentence without changing meaning)
- Same measurement basis/units/standards if applicable (e.g., ratings, units, dimensions)

DO NOT MERGE/REPLACE if any of the following is true:
- Candidate is related but not identical (association $\neq$ synonymy)
- Candidate is a subtype/supertype, variant, feature level, or category of the existing node
- Candidate differs by metric vs attribute (e.g., "size" vs "dimensions"; "weight" vs "mass")
- Candidate differs by standard/rating/threshold (e.g., "splash resistant" vs a specific IP rating)
- Candidate differs by mechanism (e.g., over-voltage vs over-current vs surge vs thermal protection)
- Candidate is ambiguous while the existing node is specific, or vice versa

DISCARD RULES:
- Candidate is not a valid term for this Node Type
- Candidate is overly vague or marketing-only with no clear definition
- Candidate conflicts with common technical meaning or is unreliable

Preferred Naming Rules:
- Use the most common international or scientific name unless a local name adds value
- If candidate is clearer or more standard, choose REPLACE; otherwise MERGE
- Preserve other variants as synonyms (handled outside this decision)

Decision procedure:
- 1) Validate candidate (otherwise DISCARD)
- 2) Check each relevant KG node for strict synonymy
- 3) If a strict synonym match exists:
  - If existing name preferred $\rightarrow$ MERGE with that node_id
  - If candidate name preferred $\rightarrow$ REPLACE that node_id
- 4) If no strict synonym match exists $\rightarrow$ ADD

Output format:
- For ADD or DISCARD: output only the action in uppercase
- For MERGE or REPLACE: output action + space + node_id

Example outputs:
- MERGE 1749
- REPLACE 3943
- ADD
- DISCARD

Example output: MERGE 4587

Figure K.2: KGD prompt example (strict version). This version has the strictest filtering rules: it lists in detail the situations that cannot be merged (such as different mechanisms, metrics vs. attributes, different standards, etc.).

KGD prompt example (Remove `Discard` Version)

Instruction: You are a product knowledge graph decision agent.

Goal: Decide whether to ADD a new node, MERGE with an existing node, or REPLACE an existing node name.

Relevant nodes from KG:
{pretty_nodes}

Node Type: {node_type}

Candidate: {pretty_candidate}

Choose one action from:

- ADD: Candidate is a distinct concept not present in the KG.
- MERGE: Candidate is the SAME concept as an existing node (true synonym), and the existing node name is the preferred form.
- REPLACE: Candidate is the SAME concept as an existing node (true synonym), but the candidate name is the preferred/standard form and should replace the existing node name.

STRICT SYNONYM POLICY (required for MERGE/REPLACE):

- Same meaning and scope (no broader/narrower relationship)
- Same real-world referent (can be substituted in a sentence without changing meaning)
- Same measurement basis/units/standards if applicable (e.g., ratings, units, dimensions)

DO NOT MERGE/REPLACE if any of the following is true (choose ADD instead):

- Candidate is related but not identical (association $\neq$ synonymy)
- Candidate is a subtype/supertype, variant, feature level, or category of the existing node
- Candidate differs by metric vs attribute (e.g., "size" vs "dimensions"; "weight" vs "mass")
- Candidate differs by standard/rating/threshold (e.g., "splash resistant" vs a specific IP rating unless explicitly defined as equivalent)
- Candidate differs by mechanism (e.g., over-voltage vs over-current vs surge vs thermal protection)
- Candidate is ambiguous while the existing node is specific, or vice versa

Preferred Naming Rules (apply only after synonymy is confirmed):

- Use the most common international or scientific name unless a local name adds significant value.
- If candidate is better to be a representative of synonyms, choose REPLACE; otherwise choose MERGE.
- Preserve other name variants as synonyms (handled outside this decision).

Decision procedure:

1. Evaluate candidate validity and relevance to Node Type.
2. Check each relevant KG node for strict synonymy using the STRICT SYNONYM POLICY.
3. If a strict synonym match exists:
    - If existing name preferred $\rightarrow$ MERGE with that node_id
    - If candidate name preferred $\rightarrow$ REPLACE that node_id
4. If no strict synonym match exists $\rightarrow$ ADD

Output format:

- For ADD: output only `ADD`
- For MERGE or REPLACE: output the action in uppercase, followed by a space, then the `node_id` of the matched existing node

Example outputs:

- MERGE 1749
- REPLACE 3943
- ADD

Example output: MERGE 4587

Figure K.3: KGD prompt example (remove discard version). The `DISCARD` option has been removed from the action space: it is now assumed that all input candidate terms are valid, with the focus shifting solely to deduplication and name standardization. Logical Focus: The logic now centers on determining whether a candidate term is more standardized than the existing node name (thereby triggering a `REPLACE` operation); otherwise, it simply performs a `MERGE`.

Product type suggestion prompt

Instruction: Return the simplest, most general product type that accurately represents the item while ensuring clarity and avoiding ambiguity. Follow these rules:

- Remove brand names, model numbers, attributes, and marketing language.
- Eliminate redundant or situational descriptors.
- Standardize technical terms using widely accepted industry vocabulary.
- Abstract to the highest-level accurate category (e.g., "Chair" instead of "Office Chair").
- Use singular form, unless the product is commonly referred to in plural (e.g., "Scissors", "Pants", "Shoes").
- Return 'None' for inputs that are unclear, ambiguous, or not valid products.
- If the input already meets all criteria, return it unchanged.

Additional Guidance:

- Contextual Clarity: Ensure the abstracted term maintains enough context to avoid ambiguity. For example, if the original product is clearly related to vehicles, use terms like "Vehicle Part" instead of just "Part."
- Specificity Check: If the most general term could refer to multiple unrelated categories (e.g., "Panel" could mean a house panel or a motorcycle panel), provide a more specific term that still fits the criteria (e.g., "Motorcycle Panel").

Your goal is to return the most common specific type that is still universally understood and accurately descriptive. Provide only the product type in English, no extra text or explanation.

Title: Wireless Mouse Pen 2.4G Bluetooth Optical Pocket Pen Mouse with Stylus Function for Laptop Tablet Phone Stylus Mouse

Description: Experience a new level of convenience with our innovative 2.4G Bluetooth Wireless Pen Mouse... [truncated for brevity] ...Package Contents: 1 * Wireless Mouse Pen.

Specifications: {"brand": ["Universal"], "connectivity": ["Bluetooth"], ...}

Example output: Pen Mouse

Figure K.4: Product type suggestion prompt. The model is instructed to strip non-type information and output a concise, canonical product type that is as general as possible while remaining unambiguous; it must output `None` when evidence is insufficient.

Attribute key discovery prompt

Instruction: Generate a comprehensive table containing key product attributes.

The attributes must meet these criteria:

- Essential for both sellers and buyers
- Focused on inherent product characteristics (not logistics, packaging, or SEO)
- Based on industry standards or common e-commerce practices
- Includes visual and functional attributes that influence purchasing decisions

Table format:

- Attribute Name: Standard Attribute Name
- Description: What the attribute represents
- Examples: Examples of Standard Values (comprehensive)

Output constraints:

- Provide only the table, no extra text or explanation.
- Sort ALL attributes by importance from the buyer's perspective.
- Start with Brand as the top row, followed by other attributes in importance order.
- Use proper title case capitalization for all values in Attribute Name and Examples (e.g., `Universal`, `Infrared`, `Lithium-Ion`, not `universal`, `infrared`, `lithium-ion`).
- Provide values in examples at least 5 if applicable.
- Keep example values less than 10.

Product Type: Pen Mouse

Product Type Description: A hybrid input device that combines the form factor and precision of a stylus pen with the functionality of a computer mouse, designed for enhanced control in graphic design, digital note-taking, and precision navigation on touch-enabled surfaces.

Example output:

Attribute Name	Description	Examples
Brand	The Manufacturer Or Company That Produces The Product	Wacom, Apple, Huion, XP-Pen, Logitech
Precision	The Level Of Accuracy And Responsiveness In Tracking Movement And Input	0.01 Mm, 0.05 Mm, 0.1 Mm, 0.2 Mm, 0.5 Mm
Pressure Sensitivity	The Number Of Levels The Device Can Detect To Vary Line Thickness Or Opacity Based On Pen Pressure	2048, 4096, 8192, 16384, 20480

(truncated for brevity)

Figure K.5: Attribute key discovery prompt. The model generates a buyer-centric, type-specific attribute schema as a Markdown table (Attribute Name, Description, Examples) sorted by importance and constrained to standardized naming/value formats.

Attribute value extraction prompt example (multimodal).

Instruction: Given the table of attributes as a reference, extract relevant attributes from the provided input text and image. Return a JSON object containing only the attribute ids and their corresponding attribute values.

- Extract JSON of attribute id and its corresponding attribute value.
- For attributes that have multiple values, return a list of values.
- If an attribute is not mentioned or cannot be determined with confidence, return null for that attribute.
- You are not restricted to the specific values in the example table — if another value makes sense based on the input, feel free to use it.
- Cross-validate conflicting or ambiguous information between the text and image (e.g., if the text says "100% cotton" but the image shows a shiny fabric, consider whether it might be polyester or a blend).
- Resolve ambiguities or duplicate values by selecting the most appropriate one based on context and consistency (e.g., choosing between "L" and "Large" based on other entries or general conventions).
- Ensure the JSON output follows the structure and naming conventions shown in the example attribute table.
- Provide only the JSON, no extra text or explanation.

Simple Example:

- Input text: Brand: Philips. The machine body is made of ABS plastic and is available in White, Black, and Silver. It has an IP44 rating for basic dust and splash protection.
- Output JSON: {"123": "Philips", "124": "IP44", "125": "ABS Plastic", "126": ["White", "Black", "Silver"]}

Product Type: Battery Holder

Product Type Description: A product type designed to securely hold and connect batteries within electronic devices, providing electrical contact and physical support for standard battery sizes such as AA, AAA, or coin cells. Commonly used in consumer electronics, remote controls, and portable devices.

Attribute ID	Attribute Name	Description	Examples
————	——————-	—————-	————-
18392	Brand	The Manufacturer Or Company That Produces The Battery Holder	Duracell, Energizer, Panasonic, Anker, Varta
18597	Battery Type	The Type Of Battery The Holder Is Designed To Accommodate	AA, AAA, C, D, 9V
19903	Number of Batteries Supported	The Total Number Of Batteries The Holder Can Support	1, 2, 3, 4, 6
(truncated for brevity)

Title: DIY 6-Pin Battery Holder for 3x 18650 Lithium-Ion Batteries

Highlight: Compact and lightweight design for easy installation and portability. - Dimensions: 78 x 60 x 21 mm / 3.1ẍ 2.4ẍ 0.83(̈L*W*H) - Weight: 24g - Material: Hard plastic and metal - Color: Black Package Include: 1 x 6-Pin Battery Holder for 18650 Batteries

Description: Features: 1. This DIY 6-pin battery holder is designed for 3x 18650 lithium-ion batteries (batteries not included). 2. Made from durable hard plastic and metal, ensuring long-lasting use. 3. Compact and lightweight design for easy installation and portability. Specification: - Dimensions: 78 x 60 x 21 mm / 3.1" x 2.4" x 0.83" (L*W*H) - Weight: 24g - Material: Hard plastic and metal - Color: Black Package Include: 1 x 6-Pin Battery Holder for 18650 Batteries

Specifications: {"batteries_type": ["D"], "warranty_type": ["No Warranty"], "Plug_Type": ["Universal"]}

Expected Output:
{ "18392": "BLM", "18597": "18650", "19903": 3, "27091": "Series", "18809": "Nickel-Plated Brass", "18407": "Snap-In", "20066": "Positive Front", "18427": "-20°C to 70°C", "18724": null, "19542": "ABS Plastic" }

Figure K.6: Attribute value extraction prompt example (multimodal). Given a product type and its attribute table (with IDs), the model extracts values from the listing text and image and returns a JSON object keyed by attribute ID, using lists for multi-valued fields and `null` when evidence is insufficient.

KG ground-truth construction prompt

Based on Product Description, extract relevant attribute name and its corresponding attribute value of this product, and Refer to the Hypotheses list when providing the answer. Be very accurate!
A. Task Description:
You are given **2** information sources. 1: Original Product Description; 2. A **Hypotheses list of attributes and specifications under review**, it is provided for you to review and judge after the Step1, at the Step2.
Step1: You should first independently determine the attributes of this product in an e-commerce context based on the product Description, providing this information to both buyers and sellers. Then, you should fill in the values for these attributes based on the product information.
- Do not rely on ***Hypotheses list of attributes and specifications under review*** at this step, as they may be inaccurate and incomplete.
- Analyze only based on the Description in Step1.
Step2: Based on the **Hypotheses list of attributes and specifications under review** and the names/values you determained on the Step1:
- If the existing attribute names and values are correct → Adopt them directly. Using the same words for correct name/values in Hypotheses list!
- If they are inaccurate → Correct them.
- If they are incomplete → Add new ones.
Step3: Output Only in Markdown table format like:
For a prodcut Pen:
Attribute Name	Value
——————-	———————
Writing Mechanism	Retractable
Ink Color	Black
etc.	etc.
B. Criterias:
The **attributes names** must meet these criterias: - Focused on **inherent product characteristics** (not logistics, packaging, or SEO) - Based on **industry standards or common e-commerce practices** - Includes **visual and functional attributes** that influence purchasing decisions
The corresponding **attributes values** meet these criterias:
- For attributes that have multiple values, return a list of values.
C. Provided information:
1. Real Product Image and the Product Description you need to use:
Product Description: [Title] Tommy Hilfiger Big Boys' Short Sleeve 85 Shirt Product image 1 (if have):

2. **Hypotheses list of attributes and specifications under review**, it is provided for your reference:
Attribute name 1:
Neckline
Attribute value 1:
Henley
Rest of attributes:
- Brand (71009): Tommy Hilfiger
- Care Instructions (71057): Machine Washable
- Closure Type (71018): Button
- Color (71006): Gray
- Drape (71176): Soft
- Dress Length (71446): Tunic
- Elasticity (71071): Slight Stretch
- Fit Type (71095): Regular Fit
- Material (71010): Cotton
- Size (71007): 85
- Sleeve Style (71049): Short Sleeve (truncated for brevity)

Expected Output:
{ color:[0:"grey"], brand:"Tommy Hilfiger", material:"cotton", closure type:"button", pattern:"solid", sleeve style:"short sleeve", neckline:"Henley", fit type:"fitted", lining material:"unlined" }

Figure K.7: Prompt for KG ground-truth construction. Optional visual inputs are used only for multimodal tasks. Both raw and AutoPKG-generated KGs are normalized into a unified node–edge schema, after which LLM outputs are parsed into triples and integrated into the KG by the KGD.

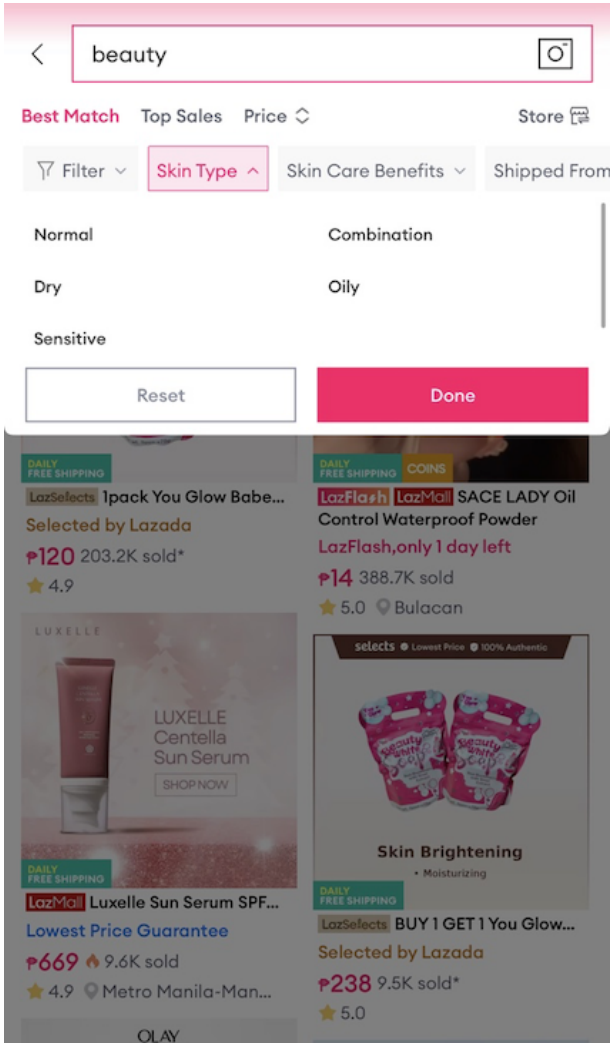

Figure L.1: Demo of *Filter*: AutoPKG-derived attribute facets surfaced in the UI to refine result sets.

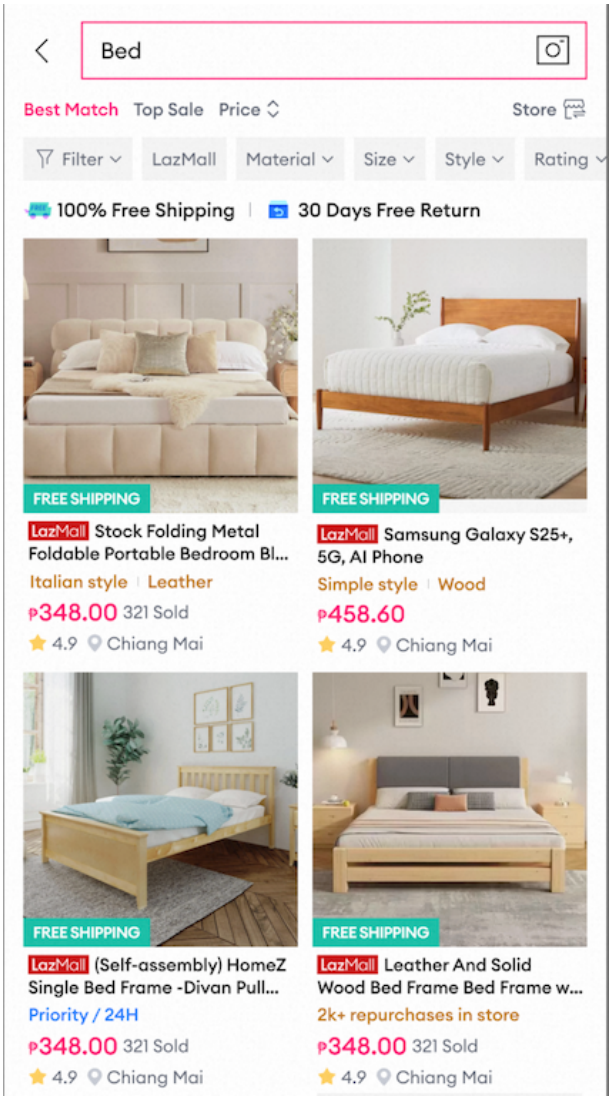

Figure L.2: Demo of *Badge*: AutoPKG-derived attribute badges shown on product cards.