AutoKB: Automated Creation of Structured Knowledge Bases for Domain-Specific Support

Rishav Sahay*, **Arihant Jain***, **Purav Aggarwal, Anoop Saladi**
Amazon
{rissahay, arihanta, aggap, saladias}@amazon.com

Abstract

Effective customer support requires domain-specific solutions tailored to users' issues. However, LLMs like ChatGPT, while excelling in open-domain tasks, often face challenges such as hallucinations, lack of domain compliance, and generic solutions when applied to specialized contexts. RAG-based systems, designed to combine domain context from unstructured knowledge bases (KBs) with LLMs, often struggle with noisy retrievals, further limiting their effectiveness in addressing user issues. Consequently, a sanitized KB is essential to ensure solution accuracy, precision, and domain compliance. To address this, we propose AutoKB, an automated pipeline for building a domain-specific KB with a hierarchical tree structure that maps user issues to precise and domain-compliant solutions. This structure facilitates granular issue resolution by improving real-time retrieval of user-specific solutions. Experiments in troubleshooting and medical domains demonstrate that our approach significantly enhances solution correctness, preciseness, and domain compliance, outperforming LLMs and unstructured KB baselines. Moreover, AutoKB is 75 times more cost-effective than manual methods.

1 Introduction

Customer Support Agents (CSAs) are chatbots (Nuruzzaman and Hussain, 2018; Xu et al., 2017) designed to resolve domain-specific user issues by providing customized, rule-compliant solutions[1] aligned with domain standards. The advent of LLMs like ChatGPT (OpenAI, 2024; Ouyang et al., 2022) has revolutionized conversational AI, enabling it to handle diverse, open-domain queries with exceptional fluency. However, CSAs, such as product troubleshooting bots or medical assistants, face distinct challenges that demand precise, context-aware solutions for provided issues. While LLMs offer remarkable general-purpose capabilities, relying solely on them risks producing incorrect or generic solutions, limiting their effectiveness in these specialized roles.

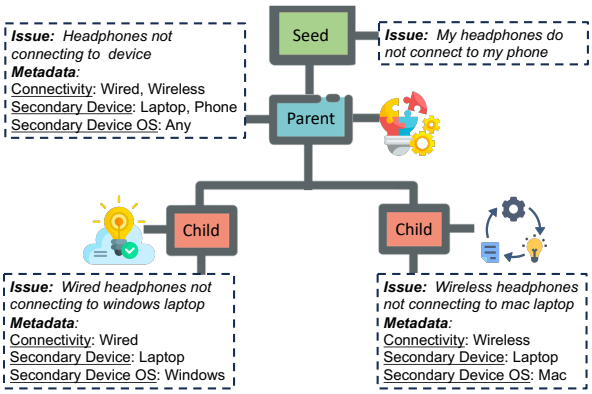

Figure 1: Illustration of an issue subtree for the seed issue *My headphones do not connect to phone* with two child issues shown along with their respective metadata

To address these shortcomings, RAG (Lewis et al., 2021) has emerged as a promising framework for building CSAs, by grounding LLM responses in retrieved domain-specific knowledge. However, the performance of RAG applications is largely dependent on the quality of the backend KB being used. Unstructured KBs, while covering a wide range of topics, are prone to noise and irrelevant information. In contrast, structured KBs resolve these issues by incorporating specific solutions and supporting domain constraints, enabling more precise and reliable knowledge grounding. RAG using unstructured KBs face additional challenges like token length limitations in LLMs, and difficulties in dynamically enforcing domain rules (because of unverified content in KB).

A significant limitation of existing approaches is their inability to provide solutions with the appropriate granularity. For instance, in troubleshooting scenarios, addressing a generic issue like Bluetooth

*Equal contribution

[1]In CSAs, we refer user queries as **issues** and responses as **solutions**

connectivity problem is insufficient when the user faces a specific problem, such as Bluetooth not pairing with a device running an older Android version. Generic solutions that fail to address the user's exact issue often result in dissatisfaction.

Additionally, ensuring compliance with domain-specific policies—such as safeguarding rules for medical guidance or hardware safety instructions in product support—remains challenging in real-time solution generation. However, techniques like Chain-of-Thought (Wei et al., 2023) and Reflexion (Shinn et al., 2023) can enhance rule adherence but they may exacerbate input length constraints and increase latency, thereby complicating their practical application.

To address issues with solution accuracy, specificity, and domain compliance in existing systems, this paper introduces AutoKB, an automated pipeline for constructing a structured KB for **any domain**. As shown in Figure 1, the proposed KB employs a hierarchical tree structure where nodes represent issues at varying levels of granularity. Root nodes cover broad, generic issues, while child nodes capture more specific ones, each linked to solutions tailored to their level of detail. Node relationships are defined by **metadata** differences, ensuring coverage of both generic and specific user issues.

Following are the contributions of our work:

- We propose AutoKB, an **automated pipeline** that builds a KB, mapping issues to solutions, enriches them with domain knowledge, and ensures domain compliance through safeguarding rules.
- We introduce a **two-level tree-structured KB**, categorizing issues into generic and specific levels, differentiated using metadata. Each issue node is linked to solutions that match its required level of granularity.
- We develop a **hybrid retrieval strategy** that combines semantic and metadata-based search, significantly enhancing retrieval quality within CSAs utilizing the KB structure.

2 Related Work

Knowledge-based support systems aim to provide accurate, specific, and safe responses to user queries. The existing approaches can be broadly categorized into two main types: Prompting techniques for LLMs and RAG systems, which rely on underlying KBs.

Various prompting techniques have been developed to enhance LLM performance, particularly in rule-following, reducing hallucinations, and providing specific solutions. Chain-of-Thought (CoT) prompting (Wei et al., 2023) and its variants like CoT with In-Context Learning (CoT-ICL) (Dong et al., 2024) have shown promise in improving reasoning and rule-following capabilities. However, these methods still rely heavily on the LLM's pre-trained knowledge and may not provide grounded, specific, and safeguarded responses (Zhao et al., 2024). RAG systems (Lewis et al., 2021) combine the power of pre-trained language models with additional information, typically using retrieval methods to fetch relevant content and augment LLM responses. While RAG systems can improve response grounding and quality, their effectiveness is highly dependent on the quality and structure of the underlying KB.

KB construction approaches can be broadly categorized into two types: Unstructured and Structured. Unstructured KBs, built using web crawlers (Huang et al., 2024) on popular search engines (Caramancion, 2024) and Databases (Jing et al., 2024), cover a wide range of topics but often suffer from noise and irrelevant information. Structured KBs (Hu et al., 2024; Kommineni et al., 2024) excel at representing domain-specific factual information and relationships between entities. However, both types face challenges in addressing specific user issues. Our KB framework develops a hierarchical tree-based structure capable of accommodating specific complex user issues and their solution knowledge, bridging the gap between issue representation and solution retrieval.

3 Proposed Methodology

3.1 Knowledge Base Structure

We propose a hierarchical KB structured (Figure 1 as a two-level tree, where the root node represents generic issues, and child nodes represent more granular and specific issues. We term such a tree an **Issue Subtree**, which comprises a Parent Issue and its corresponding Child Issues (issues and nodes used interchangably). Each issue in the subtree is linked to a solution tailored to its granularity level. This structured approach enables the effective handling of highly specific customer issues while also addressing broader, more generic user concerns.

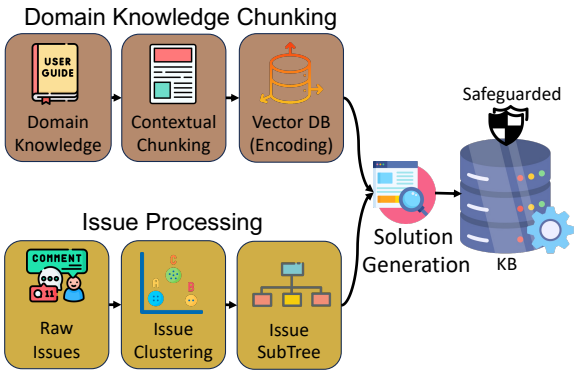

Figure 2: The KB creation pipeline comprising of 1) Domain Knowledge Chunking (DKC), 2) Issue Processing (IP), and 3) Solution Generation (SG).

To differentiate between the granularity of parent and child issues, we utilize **Metadata** which is defined as a mapping of attributes to their respective values, capturing the key characteristics of an issue. For instance, for the troubleshooting example in Figure 1, the metadata for the child issue, *Wired headphones not connecting to Windows laptop* is *{Connectivity: Wired, Secondary Device: Laptop, Secondary Device OS: Windows}*. These metadata keys (referred to as attributes), along with their possible value sets are predefined by domain experts for a given domain and referred to as the *attribute configuration* as shown in Table 4 in Appendix.

For any issue, metadata is extracted based on the attributes defined in the attribute configuration using an LLM (see Prompt G.2). The extracted metadata helps differentiate between parent and child issues wherein parent issues exhibit broader attribute values, while child issues have more specific attribute values.

3.2 Knowledge Base Creation

The automated KB creation process requires four key inputs for a given domain: 1) **Raw Issues**, which are historically observed user issues; 2) **Attribute Configuration**, defining attribute keys and their plausible values (detailed in Table 4 in Appendix); 3) **Domain Rules**, which outline the guidelines and constraints the KB must follow (examples in Table 5 in Appendix); and 4) **Domain Knowledge**, comprising unstructured documents such as user manuals and FAQs that can be utilized to build the KB.

KB creation pipeline consists of 3 modules, as illustrated in Figure 2 and outlined in Algorithm 1.

3.2.1 Domain Knowledge Chunking (DKC)

To effectively utilize domain knowledge for issue resolution, we process unstructured documents and store it in a database. Initially, documents are divided into fixed-length chunks of 2048 characters, following Finardi et al. (2024). However, recognizing the limitations of traditional chunking methods, such as loss of coherency and context (Dong et al., 2023), we introduce a novel technique called **Contextualized Chunking**.

Existing approaches, such as context-aware chunking and semantic chunking (Pinecone, 2025), focus on optimizing chunk boundaries but do not enrich individual chunks with additional contextual information critical for retrieval. Our approach, in contrast, generates a contextualized version for each chunk by incorporating information from preceding chunks using an LLM. The process, detailed in Prompt G.5, involves inputting the previous contextualized chunk as context and current chunk to the LLM to create an enriched, contextualized version. This method ensures that each chunk contains both local knowledge and a global understanding of the document, thereby enhancing retrieval accuracy. The original chunk and its contextualized version are then concatenated and encoded using a text encoder and stored in a VectorDB, as illustrated in Figure 2 and Algorithm 1.

3.2.2 Issue Processing (IP)

To address the presence of duplicates in Raw Issues, we employ a two-step process of theme-based classification and clustering for de-duplication. First, we use Prompt G.6 to identify unique issue themes and Prompt G.7 to assign themes to each raw issue. Within each theme, we apply a clustering algorithm (detailed in Appendix D) to group similar issues, selecting cluster centroids as representative *Seed Issues*. This approach ensures a diverse and non-redundant set of issues for further processing.

Each Seed Issue is then transformed into an issue subtree using LLM Prompt G.8, which takes the seed issue and domain attribute configuration as input. This hierarchical structure, comprising a parent issue and its corresponding child issues, allows for a more nuanced representation of specific issues and their potential solutions. Figure 1 illustrates this process for a troubleshooting domain issue demonstrating how the initial seed issue is expanded into an issue subtree, while Figure 2 (bottom-left) and Algorithm 1 Lines 6-20 outline the complete workflow.

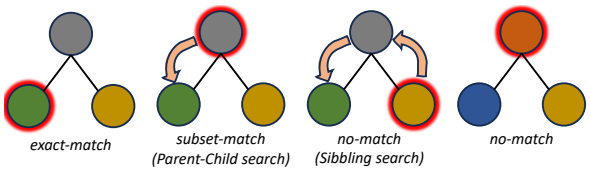

Figure 3: Hybrid search strategy that utilises the proposed tree structure to perform metadata search on top of the node identified by the semantic search. The green node represents the exact-match, grey node represents the subset-match and the node with red glow represents the node matched using semantic search.

3.2.3 Solution Generation (SG)

The solution generation process employs RAG to link each node in the Issue Subtree to its corresponding solution. This approach treats issues in issue nodes as queries for retrieval from the previously constructed contextualized VectorDB and employs cosine similarity of retrieval. The top-5 retrieved chunks serve as input to an LLM prompt (G.9) to generate relevant solutions. To ensure the quality and appropriateness of the generated solutions, we focus on three key aspects: (1) **Correctness**, achieved through RAG and contextualized chunking, which improves retrieval recall and grounds the solutions in retrieved information;(2) **Domain Rule Compliance**, ensured by incorporating domain-specific rules into the prompt, guiding the model to adhere to defined constraints; and (3) **Granularity Alignment**, maintained by providing the issue and its metadata as input to the prompt, explicitly guiding the model to generate solutions that correspond to the issue's level of granularity. The steps is detailed in Algorithm 1 Lines 21-28.

3.3 Hybrid Retrieval

We propose a retrieval strategy to integrate a structured KB with a CSA for resolving real-time user issues. The KB, organized with issue nodes linked to solutions, enables issue-issue matching. Solutions associated with the matched issue are retrieved and presented to the user. This strategy combines *soft* semantic search for relevance with *hard* metadata-based search for precision. Semantic search computes cosine similarity between text embeddings to identify the most semantically relevant nodes, while metadata-based search matches the query's metadata with the node metadata, ensuring precise retrieval. The goal is to find a node where the metadata closely matches the query—ideally an *exact-match*. If no exact match is found, nodes with metadata forming a superset of the query's (parent node) are considered *subset-matches*. However, if there is any conflict between the query and node metadata, the node is considered a *mis-match* and excluded from the results. This strategy ensures 1) precise retrieval of solutions that match the query's granularity (in the case of exact match), and 2) broader solutions when a subset-match occurs.

To perform retrieval, as described in Algorithm 2 in the Appendix, each issue node (parent and child) in the KB is indexed using a text encoder and stored in a VectorDB, along with its metadata and solutions. When the CSA receives a real-time customer query, it extracts the metadata using an LLM with the prompt G.2. We make the assumption here that the CSA fully understands the issue by querying the customer effectively to establish the relevant metadata attributes before initiating KB retrieval. Once this is achieved, the query is encoded, and a semantic search is conducted over the indexed issues in the VectorDB. Based on the top k retrieved issues from the semantic search (referred to as the *target issue*), the following scenarios are handled:

1. **exact-match:** If a target issue's metadata is same as the query's metadata, the target issue is accepted.

2. **subset-match:** If the target issue is a Parent Issue and a *subset-match*, all its child nodes are traversed for an *exact-match*. If an *exact-match* is found, the the child issue is accepted else the parent issue is accepted.

3. **no-match:** If the target issue is a Child Issue and a *no-match*, its parent and siblings are traversed. If an acceptable match is found (*exact-match* or *subset-match*), the respective issue is accepted. Else, the target issue is discarded.

This hybrid retrieval strategy, illustrated in Figure 3, improves recall by addressing inaccuracies in semantic search. Metadata matching ensures exact alignment with the query, while the search among siblings and children enhances recall by covering overlooked matches by the semantic search.

4 Experimental Setup

4.1 Datasets

We evaluate our approach on two distinct domains: **Troubleshooting** and **Medical Assistance**. For

Domain	#RI	#PI	#CI	#Sol
Troubleshooting	265	49	482	2595
Medical	302	70	866	5196

Table 1: Knowledge Base Statistics. RI: Raw Issue, PI: Parent Issues, CI: Child Issues, Sol: Solutions

the troubleshooting domain, the KB is built using historical customer-reported issues from an e-commerce store, supplemented with domain knowledge extracted from user guides and product manuals. Due to proprietary constraints, we utilize a sampled subset of data from a real-world e-commerce store to mitigate any risks associated with sensitive information.

For the Medical Assistance domain, we treat patient symptoms as customer issues and corresponding treatments as solutions. The user symptoms are sourced from Kaggle (2016) and the domain knowledge is sourced from the dataset introduced in Shah et al. (2021). For both the domains, we generate the attribute configuration and domain rules using *claude-3-haiku* (Anthropic, 2023) as shown in Table 4 in Appendix.

We utilized these data sources for the KB creation process described in Section 3.2. Table 1 summarizes the details of the datasets and relevant statistics from the KB, including the number of raw, parent, child issues and total solutions.

4.2 KB Creation Baselines

To evaluate the effectiveness of our KB creation process, we established baselines and ablated different modules: 1) **LLM-WK**, where the KB is created using the world knowledge of LLMs with the prompt in G.1 and user issues as input; 2) **Raw**, utilizing raw unstructured content from domain knowledge in chunks; 3) **Raw+CC**, leveraging contextualized chunks derived from domain knowledge; and 4) **AutoKB**, constructed using our proposed Issue Processing (IP) approach (Section 3.2.2), including both vanilla semantic search on child issues and a hybrid retrieval (**HR**) strategy on parent and child issues. We leverage *claude-3-haiku* LLM for all of our KB creation tasks and *cohere.embed-multilingual-v3* (Cohere, 2023) as text-encoder.

4.3 Evaluation Setup and Metrics

Our evaluation setup assesses the quality of the KB independently. Additionally, we evaluate the retrieval performance when the KB is integrated with a CSA for serving real time user issues. Due to confidentiality in the troubleshooting domain, we present relative improvements rather than absolute numbers.

4.3.1 KB Quality Assessment

To assess the quality of our KB, we employed three metrics corresponding to the aspects presented in solution generation (see Section 3.2.3).

1. **Correctness** (Q_C): Measures the percentage of KB solutions that are correct with respect to the issues using human annotations (details in Appendix B).

2. **Domain Compliance** (Q_D): Evaluates the percentage of KB solutions that adhere to domain rules. This is done using *claude-3-sonnet* (Anthropic, 2023) with Prompt G.3.

3. **Metadata Granularity** (Q_M): Quantifies the granularity of solution based on its metadata in the KB. It uses an *Attribute Granularity Score* (AGS) computed as the reciprocal of the number of possible values for a particular attribute of an issue. As an example the set of values for the attribute "Connectivity" of the Parent node in Figure 1 is Laptop and Phone and the AGS is thus equal to 0.5. The overall Q_M is the average AGS across all attributes:

$$Q_M = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{\#\text{values}_i}\right)$$

where n is total number of attributes as defined in the *attribute configuration* and $\#\text{values}_i$ is number of possible values for the i-th attribute for the solution.

4.3.2 Retrieval Performance

To evaluate the retrieval effectiveness of our KB, we tested the retrieved content against a set of queries. We curate different variations of inputs from child issues using an LLM. These variations simulate different levels of ambiguity when interacting with the KB. Examples of these variations are shown in Table 6 in Appendix. To generate these variations, we employed a *claude-3-haiku* LLM using the prompt illustrated in G.4.

To evaluate the KB's ability to retrieve relevant content, we employ the HitRate@k metric. This metric measures proportion of queries for which the relevant content is retrieved within the top k results.

Domain	KB	CC	IP	HR	HitRate@1	HitRate@3	HitRate@5	HitRate@10
Troubleshooting	Raw				-	-	-	-
		✓			+0.6	+5.0	+1.4	+0.9
	AutoKB	✓	✓		+7.5	+14.1	+13.2	+15.0
		✓	✓	✓	**+12.5**	**+17.3**	**+16.3**	**+18.0**
Medical	Raw				52.0	57.2	63.9	75.7
		✓			+6.7	+8.1	+8.1	+6.1
	AutoKB	✓	✓		+18.0	+18.3	+19.7	+14.5
		✓	✓	✓	**+20.0**	**+22.3**	**+25.6**	**+18.8**

Table 2: Comparison of retrieval performance for different KB configurations. CC: Contextualized Chunking, IP: Issue Processing, HR: Hybrid Retrieval. Results show incremental improvements relative to the first baseline in each domain.

We calculate HitRate@k for $k \in \{1, 3, 5, 10\}$, using the child issue or chunk from which the query is derived, as the ground truth.

Domain	KB	Q_C	Q_D	Q_M
Troubleshooting	LLM-WK	-	-	-
	Raw	+22.9	+2.8	+0.22
	AutoKB	**+24.8**	**+5.0**	**+0.57**
Medical	LLM-WK	67.5	96.8	0.21
	Raw	+24.6	-0.1	+0.13
	AutoKB	**+25.7**	**+1.3**	**+0.50**

Table 3: Comparison of KB Quality Metrics. Incremental improvements are shown relative to the first baseline in each domain.

5 Results and Analysis

Retrieval Performance Analysis: Table 2 summarizes the retrieval performance across various k values for different KB variations. The results indicate that contextualized chunking (CC) enhances HitRate by providing improved context for identifying the issue during retrieval. Structuring the KB using our approach (IP) significantly boosts retrieval performance by enabling direct embedding comparisons within the issue space rather than the issue-chunk space. Additionally, employing hybrid retrieval (HR) over the issue subtree, which combines semantic and metadata-based search, further improves retrieval outcomes.

KB Quality Results: Table 3 presents a comparison of different KBs across various quality metrics. AutoKB approach consistently outperforms both the LLM-WK and Raw KB baselines. In terms of correctness (Q_C), our KB achieves the highest scores, attributed to its groundedness enabled by RAG. Raw KB performs moderately well, particularly in the troubleshooting, while LLM-WK solutions lead to the most incorrect results due to their reliance on world knowledge, which can result in hallucinations. For domain compliance (Q_D), our approach achieves near-perfect scores, outperforming both Raw KB and LLM-WK. This indicates that our KB provides responses that are domain compliant. Furthermore, the high Metadata Granularity metric (Q_M) of our KB compared to other baselines showcases the superior granularity of solutions in our KB. Figure 4 provides a qualitative comparison, highlighting how LLM-WK generates a generic and domain-noncompliant solution (marked in red), whereas AutoKB offers a more specific and domain-compliant solution (marked in green) for the issue of AirPods.

6 Industry Impact

AutoKB demonstrated practical effectiveness and scalability in a large e-commerce context. (1) In self-serve troubleshooting, the KB offered curated solutions for 7K issue-solution pairs across 6 products, achieving a 95% acceptance rate from human annotators. (2) During a 4-week A/B test with a Troubleshooting CSA across 6 product categories, AutoKB helped reduce the return rate and improved chatbot adoption compared to an internal baseline using manually curated KB.

Cost comparisons revealed significant savings. Creating a KB for 265 troubleshooting issues with *claude-3-haiku* cost $6.69 (details in Appendix C), while human experts, at $3.75/hour and 0.5 hours per issue, would cost $496.87. This demonstrates that our approach is 75 times more cost-effective, showcasing its potential to lower costs in knowledge-based support systems.

7 Conclusion

In this paper, we propose AutoKB, an automated strategy for curating structured KBs to deliver cor-

rect, domain-compliant, and issue-specific solutions. Our approach introduces a hierarchical KB, organized into parent and child issues, effectively addressing varying levels of granularity in user concerns using metadata. By leveraging contextualized chunking and RAG-based solution generation, we enhance the correctness of KB solutions. Experimental evaluations in troubleshooting and medical domains demonstrate that our approach outperforms traditional methods in solution quality and retrieval performance within a CSA.

References

Anthropic. 2023. The claude 3 model family: Opus, sonnet, haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Kevin Matthe Caramancion. 2024. Large language models vs. search engines: Evaluating user preferences across varied information retrieval scenarios. *Preprint*, arXiv:2401.05761.

Cohere. 2023. cohere-embed-multi.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. *Preprint*, arXiv:2301.00234.

Zican Dong, Tianyi Tang, Lunyi Li, and Wayne Xin Zhao. 2023. A survey on long text modeling with transformers. *arXiv preprint arXiv:2302.14502*.

Paulo Finardi, Leonardo Avila, Rodrigo Castaldoni, Pedro Gengo, Celio Larcher, Marcos Piau, Pablo Costa, and Vinicius Caridá. 2024. The chronicles of rag: The retriever, the chunk and the generator. *Preprint*, arXiv:2401.07883.

Yujia Hu, Shrestha Ghosh, Tuan-Phong Nguyen, and Simon Razniewski. 2024. Gptkb: Building very large knowledge bases from language models. *Preprint*, arXiv:2411.04920.

Wenhao Huang, Zhouhong Gu, Chenghao Peng, Jiaqing Liang, Zhixu Li, Yanghua Xiao, Liqian Wen, and Zulong Chen. 2024. AutoScraper: A progressive understanding web agent for web scraper generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2371–2389, Miami, Florida, USA. Association for Computational Linguistics.

Zhi Jing, Yongye Su, and Yikun Han. 2024. When large language models meet vector databases: A survey. *Preprint*, arXiv:2402.01763.

Kaggle. 2016. Symptom disease sorting.

Vamsi Krishna Kommineni, Birgitta König-Ries, and Sheeba Samuel. 2024. From human experts to machines: An llm supported approach to ontology and knowledge graph construction. *Preprint*, arXiv:2403.08345.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Mohammad Nuruzzaman and Omar Khadeer Hussain. 2018. A survey on chatbot implementation in customer service industry through deep neural networks. In *2018 IEEE 15th international conference on e-business engineering (ICEBE)*, pages 54–61. IEEE.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Pinecone. 2025. Chunking strategies. Accessed February 21, 2025.

Darsh J Shah, Lili Yu, Tao Lei, and Regina Barzilay. 2021. Nutri-bullets: Summarizing health studies by composing segments. *Preprint*, arXiv:2103.11921.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Anbang Xu, Zhe Liu, Yufan Guo, Vibha Sinha, and Rama Akkiraju. 2017. A new chatbot for customer service on social media. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 3506–3510.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2024. A survey of large language models. *Preprint*, arXiv:2303.18223.

A Algorithm

Algorithm 1 outlines the step-by-step process of implementing the KB creation process. The algorithm consists of three main steps: Domain Knowledge Chunking, Issue Processing, and Solution Generation. It takes as input raw issues, attribute configurations, domain information, domain rules, and domain knowledge. The output is a structured Knowledge Base consisting of issue subtrees with associated solutions.

Algorithm 2 presents a hybrid retrieval strategy for integrating KB, combining semantic search with metadata-driven refinement. It takes a customer query, performs semantic search to retrieve top-k results, and then refines these results based on metadata matching. It navigates through parent-child relationships in the knowledge base, aiming to find the most relevant nodes that match the query's metadata.

B Human Annotation Documentation

B.1 Correctness Evaluation

We employed human experts as annotators for measuring the Correctness (Q_C) of our KB data. Annotators were provided with the instruction "*Your task is to check if the given solution is correct for the user issue. If the given solution is correct, respond with YES, otherwise NO.*" along with the user issue and the solution from the KB. We recorded the responses from human experts using a binary scoring system (1 for YES, 0 for NO) and report the average measure of correctness in Table 3.

B.2 Validation of Automated LLM Evaluation

To validate our automated LLM evaluation approach for Domain Compliance Evaluation and Metadata Extraction Tasks, we conducted a comparative study between LLM-based evaluations and human assessments. We calculated the accuracy between the LLM-based evaluations and human assessments for each task. The results demonstrated high overall accuracy of 97% for Domain Compliance Evaluation task and 94% for Metadata Extraction task. The high accuracy numbers underscore the strong alignment between LLM-based evaluations and human judgment, supporting the reliability of LLM based evaluation.

B.3 Annotation Details

Our annotation process varied by domain to ensure high-quality data. For the Troubleshooting domain, we recruited domain experts with experience in creating product troubleshooting content. For the medical domain, we utilized Amazon Mechanical Turk workers who met relevant qualification criteria. To measure inter-annotator agreement, we followed the standard protocol of performing dual annotations on a sample set of 10% of the data. We observed an agreement rate of 97% demonstrating the reliability of our annotations across all domains.

C Cost Calculation

We calculate the cost and latency of generating a KB for the troubleshooting domain, comprising of 265 raw issues.

Breakdown of LLM Calls

The total number of LLM calls is as follows:

- **Issue Deduplication:** 266 calls (1 for issue theme identification and 265 for issue theme assignment)
- **Issue Processing:** 49 calls (for issue subtree creation)
- **Contextualized Chunking:** $1,000$ calls (approx 100 documents with 10 chunks each)
- **Solution Generation:** 531 calls (49 for parent issues and 482 for child issues)
- **Solution-Metadata Detection:** 832 calls (one per generated solution)

Thus, the total number of LLM calls = 2678

Cost Calculation

The cost of KB generation is calculated using the following formula:

$$\text{Total cost} = N \times \left(\frac{T_{\text{in}}}{1000} \cdot C_{\text{in}} + \frac{T_{\text{out}}}{1000} \cdot C_{\text{out}} \right)$$

where:

- N: Total number of LLM calls
- T_{in}: Average number of input tokens per LLM call (5K)
- T_{out}: Average number of output tokens per LLM call (1K)
- C_{in}: Cost per 1000 input tokens (0.00025$)
- C_{out}: Cost per 1000 output tokens (0.00125$)

Substituting the values, the total cost of KB generation totals to 6.695$

Algorithm 1 KB Creation (Section 3.2)

Input: Raw Issues I, Attribute Configuration A, Domain D, Domain Rules R, Domain Knowledge K
Output: KB (set of IssueSubtrees)

KB $\leftarrow$ {} ▷ Initialize empty Knowledge Base
Step 1: Domain Knowledge Chunking
Chunks $\leftarrow$ FixedChunking(K) ▷ Split domain knowledge into chunks
Contextualized-Chunks $\leftarrow$ LLM(Prompt G.5, Chunks) ▷ Adding context to chunks
VectorDB $\leftarrow$ TextEncoder(ContextualizedChunks) ▷ Create vector representations
Step 2: Issue Processing
Issue Themes $\leftarrow$ LLM(Prompt G.6, D, I) ▷ Generate issue themes
for issue $\in I$ **do** ▷ Theme based Classification
    RawIssueThemes[issue] $\leftarrow$ LLM(Prompt G.7, D, Issue Themes, issue)
end for
SeedIssues $\leftarrow$ {}
for theme $\in$ IssueThemes **do** ▷ Clustering for de-duplication
    themeRawIssues $\leftarrow$ {issue $\in$ RawIssues : RawIssueThemes[issue] = theme}
    SeedIssues $\leftarrow$ SeedIssues $\cup$ Cluster-Centeroids(theme, themeRawIssues)
end for
IssueSubtrees $\leftarrow$ {}
for seedIssue $\in$ SeedIssues **do** ▷ Issue subtrees creation
    Parent, Children $\leftarrow$ LLM(Prompt G.8, D, A, seedIssue)
    IssueSubtrees $\leftarrow$ IssueSubtrees $\cup$ {<Parent, Children>}
end for
Step 3: Solution Generation
for subtree $\in$ IssueSubtrees **do** ▷ Generate solution for subtree nodes
    for issue $\in$ {subtree.Parent.issue} $\cup$ subtree.children.issues **do**
        RelevantChunks $\leftarrow$ RetrieveTopKChunks(issue, VectorDB, K=5)
        issue.Solutions $\leftarrow$ LLM(Prompt G.9, issue, issue.metadata, RelevantChunks, R)
    end for
    KB $\leftarrow$ KB $\cup$ {subtree}
end for

D Issue Clustering Details

We provide more details about the issue de-duplication algorithm along with the specific hyper-parameters being used.

1. We generate the issue embeddings using open-source (Cohere, 2023).

2. We use UMAP along with HDBCAN to cluster the issues within each themes. UMAP is used for dimensionality reduction.

3. UMAP : n_neighbors=15, min_dist=0.1, n_components=20, and metric='euclidean'

4. HDBSCAN : min_cluster_size=5 and metric='euclidean'

E Additional Tables

Table 4 presents the pre-defined attribute configurations for the troubleshooting and medical domains. The attributes are designed to capture key information relevant to each domain, facilitating better issue diagnosis and solution recommendation.

Table 5 outlines the domain-specific rules applied in our system for troubleshooting and medical domains. These rules are crucial for ensuring that the system provides safe, ethical, and practical recommendations. They guide the system to avoid potentially harmful or unhelpful suggestions, maintaining user safety and enhancing the overall quality of assistance provided.

Table 6 demonstrates various phrasings of the same user input utilized in analyzing the performance on retriever.

Apple AirPods 4 Wireless Earbuds, Bluetooth Headphones, Personalised Spatial Audio, Sweat and Water Resistant, USB-C Charging Case, H2 Chip, Up to 30 Hours of Battery Life, Effortless Setup for iPhone

Issue: Airpods won't turn on when connected to a laptop.

LLM-WK: *AirPods may be out of battery or not charged: Ensure your AirPods are charged by placing them in the charging case and checking the LED indicator. If the AirPods are still unresponsive, charge the case with the AirPods inside for at least 15-20 minutes. After charging, try reconnecting them to your laptop. Try visiting the Apple service center if this does not work.*

AutoKB: *Check Battery Level and Charge AirPods:*

- *Place AirPods in the charging case and close the lid.*
- *Wait a few minutes, then open the lid.*
- *Check the status light:*
  - *Green: Fully charged.*
  - *Amber: Needs charging.*
- *If amber, connect the case to power until the light turns green.*

Figure 4: Qualitative comparison of the response for the Issue: *Airpods won't turn on when connected to a laptop* for **LLM-WK** and the proposed **AutoKB**.Violations in domain compliance are highlighted in red and the granularity of the solutions is highlighted in green.

F Qualitative Results

Figure 4 presents a comparative analysis of troubleshooting responses generated by AutoKB and LLM-WK. Our system demonstrates superior performance by providing more granular, step-by-step solutions (highlighted in green). In contrast, the baseline LLM-WK offers a less structured, all-at-once response. Additionally, our framework effectively identifies and filters out non-compliant information (highlighted in red) that violates domain-specific rules and should not be presented to customers.

Attribute Name	Type	Classes/Values
Troubleshooting		
Type of Headphone	Closed-attribute	Earphone, Earbud, Headphone
Connectivity to Secondary Device	Closed-attribute	Wired, Wireless
Secondary Device	Closed-attribute	Laptop, Phone, Tablet
Device OS of Secondary Device	Closed-attribute	Android, iOS, Windows, Mac
Medical		
Age	Closed-attribute	Infant, Child, Adolescent, Adult, Elderly
Sex	Closed-attribute	Male, Female, Other
Pre-existing Conditions	Open-attribute	Diabetes, Hypertension, Asthma, Heart Disease, etc.
Symptom Onset	Closed-attribute	Immediate, Recent, Ongoing, Prolonged, Chronic

Table 4: Pre-defined attribute configurations for various domains

Dataset	Rule #	Description
Troubleshooting	1	*Avoid solutions that suggest use of abrasive cleaners or chemical solutions.*
	2	*Avoid solution that point the user to refer to the user manual.*
	3	*Avoid recommending solutions that asks the user go to the service center for repair or replace.*
	4	*Avoid recommending steps such as contacting product support and customer support.*
Medical	1	*Do not recommend high-risk procedures.*
	2	*Avoid giving definitive medical diagnoses;*
	3	*Refrain from recommending treatments that lack strong scientific evidence.*
	4	*Ensure recommendations consider user-reported allergies to avoid suggesting harmful treatments.*

Table 5: Domain specific rules for the different domains

Variations
I've an issue with my phone
I'm facing a problem with my mobile.
There seems to be a problem with the device I use for communication, specifically my phone.

Table 6: Examples of different variations of User Input *"I have an issue with my phone"*

Algorithm 2 Hybrid Retrieval Strategy for KB Integration (Section 3.3)

Input: Customer Query Q, Vector Database (VectorDB), k retrieval value
Output: AcceptedNodes (Set of Accepted Issue Nodes)

```
QueryMetadata ← ExtractMetadata(Q)                ▷ Extract metadata using an LLM Prompt G.2
QueryEmbedding ← Encode(Q)                        ▷ Compute query embedding
TopKResults ← SemanticSearch(QueryEmbedding, VectorDB, k)   ▷ Retrieve top-k issues from VectorDB
AcceptedNodes ← {}
for TargetIssue ∈ TopKResults do
    TargetMetadata ← TargetIssue.metadata
    if QueryMetadata = TargetMetadata then
        AcceptedNodes ← AcceptedNodes ∪ {TargetIssue}          ▷ Exact match found
    else if QueryMetadata ⊆ TargetMetadata then
        if TargetIssue.type = Parent then
            found ← False
            for ChildIssue ∈ GetChildren(TargetIssue) do
                if QueryMetadata = GetMetadata(ChildIssue) then
                    AcceptedNodes ← AcceptedNodes ∪ {ChildIssue}   ▷ Exact match in children
                    found ← True
                    break
                end if
            end for
            if found = False then
                AcceptedNodes ← AcceptedNodes ∪ {TargetIssue}      ▷ Accept parent
            end if
        end if
    else
        if TargetIssue.type = Child then
            Parent ← GetParent(TargetIssue)
            if QueryMetadata ⊆ GetMetadata(Parent) then
                found ← False
                for Sibling ∈ GetChildren(Parent) do
                    if QueryMetadata = GetMetadata(Sibling) then
                        AcceptedNodes ← AcceptedNodes ∪ {Sibling}   ▷ Accept sibling
                        found ← True
                        break
                    end if
                end for
                if found = False then
                    AcceptedNodes ← AcceptedNodes ∪ {Parent}        ▷ Fallback to parent
                end if
            end if
        end if
    end if
end for
return AcceptedNodes
```

G Prompts

Prompt G.1:LLM-WK Prompt

Instruction
You are a solution provider for a given user issue whose task it to provide solutions for a particular domain.
You will be given as input the following pieces of information:
1. Domain Information: This is the information about the domain enclosed within the XML tags <domain_info>.
2. Issue: This is the issue the customer is facing . This is enclosed within the XML tags <issue>.
Instructions:
1. Enclose your response within the XML tags <response>.
2. Provide multiple possible solutions.
3. Enclose each solution within the XML tags <solution>.

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<domain_info> {domain_info} </domain_info>
<issue> {issue} </issue>

Prompt G.2: Metadata Extraction Prompt

Instruction:
You are an Attribute Extractor for a given inputted domain. Your task is to identify the attributes values of a piece of user issue specific to the domain for the given set of attributes configuration.
<instructions>
- Analyze the attributes within the attributes configuration presented to you within the XML tags <attr_config></attr_config>.
- Analyze the issue presented to you within the XML tags <issue></issue>.
- You will respond within the XML tags <a>
- The response will be in the format:
ATTRIBUTE1=VALUES;ATTRIBUTE2=VALUES;
- If an attribute takes no values (or is not valid to the issue), detect it as NONE.
- If an attribute can take all possible values (mentioned explicitly or implicitly), detect it as Any.
- If an attribute value can be inferred implicitly (as in not mentioned), detect it.
- Start your attribute detection by stating your reasoning within the XML tags <thinking></thinking>.
</instructions>

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<domain> {domain_info} </domain>
<attr_config> {attribute_configuration} </attr_config>
<issue> {issue} </issue>

Prompt G.3: Domain Compliance Evaluator

Instruction:
You are an evaluator of solutions provided by a Customer Support Agent (CSA) for a specific user issue in a specific domain.

You will be given the following inputs:
1. Domain Information: Information about the domain within the XML tags <domain_info>.
2. User Issue: The issue faced by the user within the XML tags <issue>
3. Solution: This is the suggested solution for the user issue within the XML tags <solution>.
5. Domain Rules: These are the set of domain rules to be followed by the prescribed solutions within the XML tags <domain_rules>.

Instructions for output:
<rule>
1. Enclose your response within the XML tags <response> </response>.
2. Thoroughly analyze the predefined rules.
3. Provide a detailed analysis of the user issue and the proposed solution.
4. Use the scratchpad <scratchpad> to jot down brief notes, presented in bullet points.
5. Assign a score ranging 0 or 1 for each rule to indicate the level of adherence, with 0 indicating non-compliance and 1 indicating full compliance. The scores should be within the XML tags <scores>.
6. Enclose each score within XML tags like <score1>, <score2>, and so on for each respective rule.
7. For each score, provide a reason within XML tags like <reason1>, <reason2>, and so forth, explaining the rationale behind the assigned score.
8. If a rule is not applicable to the steps provided, assign a score of -1 and state the reason as "Not applicable".
</rule>

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<domain_info> {domain_info} </domain_info>
<issue> {issue} </issue>
<solution> {issue} </solution>
<domain_rules> {domain_rules} </domain_rules>

Prompt G.4: User Input Variation

Instruction:
As a text modifier, your role involves introducing subtle changes to a provided text snippet. This task requires adherence to specific types of alterations, categorized by difficulty levels.
The types of permissible variations are as follows:
- Easy Variations: 1. Addition or removal of punctuation marks. 2. Utilization of different variants of the same lemma.
- Medium Variations: 1. Employment of synonyms for any word. 2. Phrase modifications: either by substituting a single word with a phrase or vice versa.
- Hard Variations: 1. Structural transformation of the text, entailing a complete reformulation while preserving the original message.
Under no circumstances should changes deviate from these guidelines. The core message and structural integrity of the text must remain intact.
The provided text will be enclosed within <original_text> tags. Your task is to generate five variations for each difficulty level:
- For easy variations, enclose each variant within <easy_variations> tags, with individual variations wrapped in <variation> tags.
- For medium variations, use <medium_variations> for the group and <variation> for individual entries.
- For hard variations, group them under <hard_variations>, with each distinct variant in a <variation> tag.

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<context> {context} </context>
<original_text> {original_text} </original_text>

Prompt G.5: Chunk Contextualizer

Instruction:
You are a text chunk contextualizer in the domain of Medical Assistance / E-Commerce Product Troubleshooting.
Task: Contextualization of Document Chunks with Pre-contextualized Input
Description: The objective is to produce a contextualized version of the current chunk of text, using the contextualized version of the previous chunk as a reference. This task aims to maintain coherence, sensibility, and information integrity across document segments. By integrating context from the pre-contextualized previous chunk, the model should generate a continuation that flows smoothly and logically, enhancing the reader's or conversational AI agent's comprehension and engagement.
Input
<PreviousChunkContextualized>
The pre-contextualized version of the previous chunk, serving as the backdrop and context for the current chunk.
</PreviousChunkContextualized>
<CurrentChunk>
The current chunk of text to be contextualized, ensuring a coherent and logical flow from the previous chunk.
</CurrentChunk>
Instructions
1. Review the contextualized version of the previous chunk to grasp the established context, themes, and details.
2. Identify the main message, key information, and any implicit or explicit links between the current chunk and the contextualized previous chunk.
3. Contextualize the current chunk by weaving in relevant context from the previous chunk, ensuring a natural and logical progression of ideas and information.
4. Ensure the original content and intent of the current chunk are preserved, making adjustments only to enhance coherence and continuity.
5. Verify the coherence, flow, and accuracy of the contextualized current chunk, making any necessary revisions to optimize readability and comprehension.
6. Make sure to preserve the overall broad crux of the document. This will mostly be mentioned in the PreviousChunkContextualized. For example: Do preserve the Product being talked about, the title of the document, the Issue being talked about but yes the king should be the current chunk.
7. You should try to keep the output short in max 2-3 sentences.
Output Instructions
A coherent and contextualized version of the current chunk that naturally follows from the pre-contextualized previous chunk, maintaining a seamless narrative or informational flow. Preserve information like Product Type, The issue being talked about. You should output the current chunk contextualized within the XML tags.<ContextualizedChunk>.
Input:
Now here is the input to you:
<PreviousChunkContextualized> {prev_chunk} </PreviousChunkContextualized>
<CurrentChunk> {current_chunk} </CurrentChunk>

Prompt G.6: Issue Theme Generator

Instruction:
Your are an issue themes identifier for a list of issues related to a particular domain. You will be given as input the following pieces of information:
1) Domain Information: This is the domain related to which the user issues are provided, enclosed within the XML tags <domain_info>.
2) Issues List: These are the list of issues encountered by users. This will be enclosed within the XML tags <issues_list>.

Here are some general rules to keep in mind while creating the themes:
<rules>
1. Detect broad themes over the issues.
2. Keep the theme title information dense. Include topics (exact keywords) from the issues into the title
3. If some issues do not fall into a broad theme per say, create a miscellaneous theme and along with it include the topics as well.
4. Analyse all the possible theme. Do not over generalise please. Look into the example to clearly understand the granularity.
</rules>

Before outputting think within the XML tags <thinking></thinking>. Within <thinking></thinking> do the following:
1. Analyse the issues. Do some rough work
2. Come up with no of themes you have identified within <num_themes>.
You will output the issue themes within the XML tags <response>. Each issue theme will be enclosed within the XML tags <theme>.

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<domain_info> {domain_info} </domain_info>
<issues_list> {issues} </issues_list>

Prompt G.7: Classify Theme

Instruction:
Your task it to classify a user encountered issue related to a particular domain.
You are given as input the following:
1) Domain Information: This is the information about the domain for which the issue is provided within the XML tags <domain_info>.
2) Issue: This is the user issue within the XML tags <issue>.
3) Issue themes: These are the list of issue themes you need to classify the issue into. This will be enclosed within the XML tags <themes>.
Here are the output rules:
You will output the issue theme within the XML tags <output>.
You will output the actual issue theme within the XML tags <t>.
You will output the issue theme index within the XML tags <index>.

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<domain_info> {domain_info} </domain_info>
<issue> {issue} </issue>
<themes> {themes} </themes>

Prompt G.8: Issue SubTree

Instruction:
You are a Issue Tree Generator whose task it to create a issue tree based on issue attributes. You will be given as input the following:
1. Domain Information: This is the information about the domain within the XML tags <domain_info>.
2. Customer Issue: The issue related to the domain within <issue> XML tags.
3. Attributed Configuration: The various attributes and its possible values in general within <attr_config> XML tags.
The root of the tree is a Parent Issue while the children of the parent issues are Child Issues.
In order to create the issues, you should do the following:
<general_instructions>
1. Analyse the attributes of the fed issue and within <thinking></thinking> try to find the generic version (in terms of attributes) of the issue known as the Parent Issue.
2. Create a Parent Issue which is more applicable to all kind of attribute values within <g></g>. Detect the attributes as well within <a>.
3. Now think within <thinking></thinking> again, what child issues which will be attribute specific are possible out of the generic issue. Each of the attribute keys should now take a single value.
4. You will create this specific attribute variations within . Also predict the attributes of the issues within <a>.
5. Only create the valid candidates whose attribute combinations makes sense as per the attribute constraints.
6. Make sure that each of the attribute values reflect in the issues being created.
</general_instructions>
In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>
Input:
Now here is the input to you:
<domain_info> {domain_info} </domain_info>
<attr_config> {attribute_configuration} </attr_config>
<issue> {issue} </issue>

Prompt G.9: Solution Generation Prompt

Instruction:
You are a Solution Provider for a Customer Support Agent (CSA) advised at providing domain-compliant solutions for an user issue for a given domain. Note that the definition of issue and solutions can change as per the definition of the domain. For Eg: In case of medical assistance domain, issues can correspond to symptoms while solutions can correspond to treatments. While in the case of product troubleshooting, issues could be product malfunctions and solutions could be troubleshooting steps.
<input>
1. Domain Information: This is the information of the domain for which the issue is provided. This is enclosed within the XML tags <domain_info>.
2. Attribute Configuration: These are the specific attributes, along with their definitions and values within the XML tags <attr_config>.
3. Domain Rules: These are the domain rules over which the solutions should be compliant. This is enclosed within the XML tags <domain_rules>.
4. User Issue: This is the issue provided by the user within the XML tags <issue>.
5. Issue Metadata: This is the metadata related to the user issues within the XML tags <metadata>.
6. Relevant Chunks: These are the relevant pieces of information that can help you in curating a solution within the XML tags <relevant_chunks>.
</input>

<instructions>
1. Create solutions only relevant to the user issue.
2. Consider the metadata of the issue in order to provide custom solutions of similar specificity. Never output solutions contrary to the metadata.
3. The definition of the individual attributes within metadata are fed to you within the XML tags <attr_config>.
4. Provide solutions following the do's and don't mentioned to you within the XML tags <domain_rules>.
5. Start your response within the XML tags <response> XML tags.
6. Provide the solutions within the XML tag <solutions>. Each of the treatments should be enclosed within <solution>.
7. Before providing the solutions think within the XML tags <thinking>.
8. The treatments should be curated grounded on the relevant chunks provided as input to you.
</instructions>

In-context examples:
Here are some examples:
<example> ... </example>
<example> ... </example>

Input:
Now here is the input to you:
<domain_info> {domain_info} </domain_info>
<attr_config> {configuration} </attr_config>
<domain_rules> {rules} </domain_rules>
<issue>{issue}</issue> <metadata> {metadata} </metadata>
<relevant_chunks> {chunks} </relevant_chunks>