Embedding-Aware Graph-RAG for Factual Question Answering in Product Support

T. T. Giang Nguyen[*] **Duc Hai Tran**[*]
Hanoi University of Science and Technology
{giang.ntt232336m, hai.td242025m}@sis.hust.edu.vn

Abstract

We present a **Graph-Retrieval-Augmented Generation (Graph-RAG)** system designed to enhance customer support in the smartphone domain. Our system constructs a heterogeneous knowledge graph from structured product specifications and explores two retrieval strategies: (1) a *baseline approach* that queries the graph using Cypher in Neo4j, and (2) a *proposed method* that leverages learned node embeddings for semantic subgraph retrieval.

Our experiments on a curated Vietnamese smartphone dataset demonstrate that the embedding-based retrieval approach significantly outperforms the Neo4j baseline in both subgraph relevance and response factuality. These results highlight the effectiveness of integrating structured knowledge with neural retrieval techniques for domain-specific question answering. The code is available at https://github.com/haitranduc4270/Graph.

1 Introduction

Customer support systems powered by large language models (LLMs) often struggle to provide accurate, consistent, and up-to-date responses in specialized domains. This limitation is especially critical in the consumer electronics sector, where user queries frequently require precise information about device specifications, feature comparisons, and purchasing options. While traditional Retrieval-Augmented Generation (RAG) methods enhance LLMs with external context, they typically rely on unstructured textual corpora and lack fine-grained entity relations.

To address this gap, we propose a Graph-Retrieval-Augmented Generation (Graph-RAG) framework specifically designed for the smartphone customer support domain. Our key insight is that structured product data—such as technical specifications, release timelines, and compatibility relations—can be naturally represented as a heterogeneous knowledge graph. This enables the system to retrieve not just isolated facts, but relational subgraphs that encode rich contextual dependencies.

In this work, we develop and evaluate a **Graph-Retrieval-Augmented Generation (Graph-RAG)** system tailored for the smartphone support domain. We construct a heterogeneous knowledge graph from structured product data, modeling smartphones and their specifications as typed entities and relations. We compare two retrieval approaches:

- **Neo4j-RAG Baseline:** A symbolic pipeline that translates natural language queries into Cypher queries to retrieve subgraphs directly from a Neo4j database.
- **Graph-Embedding-RAG (Proposed):** A semantic retrieval pipeline that encodes the graph using knowledge graph-aware embedding techniques and retrieves relevant subgraphs via embedding similarity.

[*]Equal contribution.

To obtain these embeddings, we implement and evaluate several graph encoding models, including knowledge graph embedding methods such as *TransE*[1] and *ConvE*[2], as well as graph neural network models like *R-GCN*[3], *CompGCN*[4], and *KBGAT*[5]. Our study investigates the impact of graph structure and model choice on retrieval quality and downstream answer generation. All models are trained on link prediction task, which involve predicting missing edges between entities in the knowledge graph. This training objective is well-suited because it encourages the models to learn meaningful representations of entities and relations that capture the graph's structural and semantic patterns. Consequently, these embeddings improve the quality of subgraph retrieval by enabling the system to infer relevant connections even when explicit edges may not be directly matched by query keywords.

Our experiments on a curated Vietnamese smartphone dataset demonstrate that graph-aware embedding retrieval approach significantly improves both subgraph relevance and the factual accuracy of generated answers. The results suggest that neural graph retrieval is a powerful complement to LLMs in domain-specific customer support scenarios.

2 Application Domain and Dataset

2.1 Application Domain

This work focuses on the domain of customer support for smartphone products, where user queries commonly involve device specifications, comparisons, promotions, and compatibility information. In this setting, accurate and context-aware responses are crucial, particularly when customers inquire about detailed technical features or seek to compare similar devices.

To address the need for structured, precise knowledge access, we developed a question-answering system grounded in a heterogeneous knowledge graph that captures multi-relational information among smartphones, their components, and related metadata. The system supports natural language queries and returns fact-grounded responses based on structured product knowledge.

2.2 Dataset

We constructed a custom dataset comprising **1,190 smartphone products** collected from *CellphoneS*, a leading Vietnamese e-commerce platform specializing in technology devices. Each product was represented as a structured JSON object containing **32 attributes** covering both hardware and software specifications.

The attributes cover a wide range of product details including manufacturer, chipset, CPU, operating system and version, display specifications (size, resolution, type, refresh rate), camera features (primary, secondary, video), charging technologies (wired and wireless), resistance capabilities (water and dust proof), connectivity options (SIM type, NFC, Bluetooth, Wifi, GPS), storage and memory specifications, physical dimensions and weight, special features, included accessories, and warranty information.

These structured data entries were converted into a heterogeneous knowledge graph consisting of **5,466 nodes** and **17,435 edges**, resulting in an average node degree of 6.38. This indicates a well-connected graph structure, well-suited for graph-based machine learning tasks such as relational learning, reasoning, and community-aware subgraph retrieval. The graph's rich connectivity and semantic diversity enable effective representation learning using graph neural networks.

Table 2: Graph Summary Statistics

Property	Value
Number of nodes	5,466
Number of edges	17,435
Average degree	6.38

Table 1: Important fields information in phone dataset

Key	Description
manufacturer	The device manufacturer
chipset	The main processing chipset
cpu	Central Processing Unit
operating_system	The operating system running on the device
os_version	Specific version of the operating system
display_size	Screen size measured in inches
display_resolution	Screen resolution (e.g., 1080x2400)
mobile_type_of_display	Display technology used
mobile_tan_so_quet	Screen refresh rate in Hz
camera_primary	Rear camera resolution
camera_secondary	Front camera resolution
camera_video	Video recording capability
mobile_cong_nghe_sac	Supported charging technology (e.g., fast charging)
sac_khong_day	Wireless charging support
mobile_khang_nuoc_bui	Water and dust resistance
sim	Supported SIM types (nano, eSIM, etc.)
storage	Internal storage capacity
mobile_ram_filter	Amount of RAM
mobile_nfc	NFC support
bluetooth	Supported Bluetooth version
wlan	Supported WiFi standards
gps	GPS support
dimensions	Physical dimensions of the device
product_weight	Weight of the device
mobile_tinh_nang_dac_biet	Other special features
included_accessories	Included accessories
warranty_information	Warranty information

3 Related Work

Retrieval-Augmented Generation (RAG). RAG[6] systems enhance language models by retrieving external knowledge to ground response generation. Traditional RAG relies primarily on unstructured textual corpora (e.g., Wikipedia), which introduces limitations in domains where knowledge is naturally structured, such as product specifications. These systems often struggle with factual grounding, especially in cases requiring precise attribute comparisons or reasoning over structured relationships.

Graph Neural Networks for Heterogeneous Graphs. To effectively represent multi-typed entities and relations, heterogeneous GNNs have been proposed. R-GCN [3] extends GCNs[7] with relation-specific transformations. HAN[8] introduces metapath-based attention to highlight relevant semantic paths. HGT[9] generalizes attention mechanisms to heterogeneous settings, enabling scalable and expressive learning. These models are particularly suited for knowledge graphs derived from e-commerce data, where node types (e.g., product, spec, review) and relation semantics (e.g., HAS_RAM, HAS_CAMERA) are essential.

Graph-RAG. Graph-RAG frameworks integrate structured knowledge graphs into retrieval-augmented generation to improve retrieval relevance and generation accuracy. These systems vary in how they leverage graph information. Some methods employ graph neural networks (GNNs) to learn continuous embeddings of nodes and edges, enabling semantic subgraph retrieval based on embedding similarity[10]. Others rely on symbolic or heuristic graph querying techniques, such as Cypher or SPARQL queries, to directly extract relevant subgraphs without learned embeddings[11]. Additionally, hybrid approaches enhance text-based retrieval by incorporating graph structure or link information into pretrained language models without explicitly encoding graphs via GNNs.

This spectrum of strategies reflects a growing interest in combining symbolic and neural methods for grounding language models in structured knowledge. Our work contributes to this landscape by

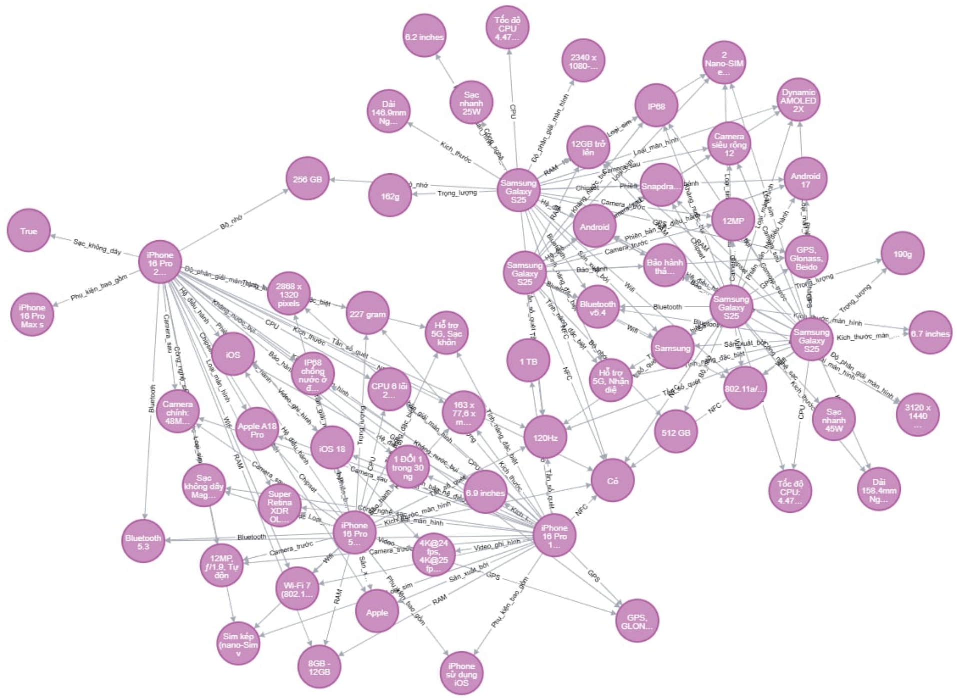

Figure 1: Graph data in Neo4j.

designing a domain-specific Graph-RAG framework that leverages graph embedding for heterogenous knowledge graph for product-centered question answering in Vietnamese. Unlike prior efforts focused on open-domain or biomedical QA, we target the under-explored area of structured customer support grounded in product knowledge graphs, demonstrating the benefits of neural graph encoding in a consumer technology setting.

4 Methodology

4.1 Knowledge Graph Construction

We begin by constructing a heterogeneous knowledge graph from structured smartphone product data scraped from the CellphoneS platforms. Each product entry is normalized and mapped to a schema consisting of typed entities (e.g., `Phone`, `Chipset`, `Battery`, `Brand`) and directed relations (e.g., `camera_primary`, `camera_video`, `manufacturer`). This results in a multi-relational graph $G = (V, E, R)$, where V denotes the set of nodes (entities), E the set of directed, labeled edges, and R the relation types. Nodes are initialized with type-specific embeddings and optionally augmented with textual features such as name or description.

The constructed knowledge graph is saved and managed using the Neo4j graph database platform, which enables efficient storage and querying of complex multi-relational data. Additionally, Neo4j's built-in visualization tools allow for interactive exploration and visual inspection of the graph structure, facilitating qualitative analysis and validation of the extracted knowledge.

4.2 Graph Encoding Models

To support effective subgraph retrieval and representation, we investigate several models designed for encoding graph-structured data, including both knowledge graph embedding methods and graph neural networks. The node embeddings generated by these models serve as the foundation for retrieval and grounding in our Graph-RAG system.

Translational Embeddings (TransE) TransE is a knowledge graph embedding method that models relations as translations in the embedding space. For a given triple (h, r, t), TransE optimizes:

$$f(h, r, t) = \|h + r - t\|_2$$

Despite its simplicity and lack of explicit graph message passing, TransE performs well in entity retrieval and link prediction tasks, serving as a strong and efficient baseline in our structured product setting.

Convolutional Embeddings (ConvE) ConvE is another knowledge graph embedding approach that applies 2D convolutional neural networks over reshaped entity and relation embeddings to capture complex interactions. Given a triple (h, r, t), ConvE learns to score its plausibility via convolutional feature extraction, improving link prediction performance especially in multi-relational graphs.

Relational Graph Convolutional Network (R-GCN) R-GCN is a graph neural network model that extends traditional GCNs to handle labeled, directed edges by associating each relation type $r \in R$ with a distinct transformation matrix W_r. The representation of a node v at layer $l + 1$ is computed as:

$$h_v^{(l+1)} = \sigma\left(\sum_{r \in R} \sum_{u \in \mathcal{N}_r(v)} \frac{1}{c_{v,r}} W_r^{(l)} h_u^{(l)} + W_0^{(l)} h_v^{(l)}\right)$$

where $\mathcal{N}_r(v)$ denotes the neighbors of v under relation r, $c_{v,r}$ is a normalization constant, and W_0 handles self-loops. R-GCN leverages graph structure and relation types to produce expressive node embeddings suitable for retrieval and inference.

We also explore **R-GAT**, which introduces relation-aware attention into the message passing framework, allowing the model to dynamically weigh the importance of different neighbors based on both structural and relational context.

Composition-based Graph Convolutional Network (CompGCN) CompGCN unifies knowledge graph embedding and GNN-based message passing by applying composition operations (e.g., circular correlation, element-wise multiplication) between entities and relations before aggregation. This enables richer semantic modeling across diverse relation types, improving generalization in link prediction and entity encoding tasks.

Knowledge-Based Graph Attention Network (KBGAT) KBGAT extends GAT by incorporating relation-specific attention mechanisms tailored for knowledge graphs. It computes attention over neighbors considering both entity and relation features, allowing richer representation learning that explicitly captures relational semantics. This makes KBGAT especially suitable for multi-relational graphs like our structured smartphone product knowledge graph.

Model Selection Justification

We selected models that balance efficiency and expressiveness for knowledge graph retrieval. TransE and ConvE serve as baselines: TransE is simple and fast but lacks relational depth, while ConvE captures more interaction patterns via convolution. For richer relational modeling, we used R-GCN and CompGCN, both suited for heterogeneous graphs with typed entities and relations — common in smartphone product data. R-GAT and KBGAT incorporate attention mechanisms, allowing the model to focus on more relevant neighbors, which is important in queries where only a few attributes (e.g., display size or battery) are critical. This diverse set of models allows us to systematically evaluate the impact of structural complexity and semantic expressiveness on retrieval quality within a Graph-RAG framework.

4.3 Subgraph Extraction Approach

Given a set of candidate nodes within a large knowledge graph, we first perform a controlled neighborhood expansion by iteratively traversing the graph outward from each candidate node. Specifically, starting from the initial seed set, we conduct a breadth-first search over adjacent edges, continuing for a fixed number of hops. All nodes and edges discovered within this traversal radius are included in the expanded neighborhood, thus capturing both immediate and multi-hop relational context.

```python
seed_ids = set()
for ent in entities:
    match = self._match_entities(ent)
    if not match:
        continue
    matched_name, score, matched_idx = match
    query_emb = self.embedder.embeddings[matched_idx].reshape(1, -1)
    _, I = self.embedder.search(query_emb, top_k=1)
    for i in I[0]:
        seed_ids.add(i)

if len(seed_ids) == 0:
    print(f"Not enough nodes found for question: {question}")
    retriever_data = []
else:
    edge_index_sub = self._extract_subgraph(seed_ids)
    retriever_data = []
    for i in range(edge_index_sub.shape[1]):
        src = edge_index_sub[0, i].item()
        dst = edge_index_sub[1, i].item()
        rel = self.kg.relation_map.get((src, dst), "related_to")
        src_name = self.kg.id_entity_map.get(src, f"node_{src}")
        dst_name = self.kg.id_entity_map.get(dst, f"node_{dst}")
        retriever_data.append(f"{src_name} {rel} {dst_name}")
```

Listing 1: Seed node selection and subgraph extraction

Following neighborhood expansion, we construct the induced subgraph by selecting all nodes identified during traversal along with every edge interconnecting them. To ensure compatibility with downstream graph processing frameworks, particularly those that rely on contiguous indexing, we optionally remap the node identifiers to form a compact zero-based index range. The resulting subgraph thus faithfully preserves the local structural and semantic relationships inherent in the original graph, while significantly reducing the scale of the data required for downstream reasoning or embedding tasks.

4.4 Graph-RAG Base Line Pipeline

We implement and evaluate two complementary Graph-RAG pipelines for product-centered question answering, each leveraging structured knowledge graphs differently:

Baseline Graph-RAG Pipeline (Neo4j + Cypher) This baseline system tightly integrates a graph database with a large language model to retrieve and ground answers.

1. **Graph Construction in Neo4j:** We build a heterogeneous knowledge graph from structured smartphone product data and load it into the Neo4j graph database. Entities and their relations are represented as nodes and directed edges respectively.
2. **Query-to-Cypher Generation:** Given a user query in natural language, a pretrained LLM is employed to generate the corresponding Cypher query, which can be executed on the Neo4j database.

```
    CYPHER_GENERATION_TEMPLATE = """
    Task: Generate a Cypher statement to query a graph database
        .
    Instructions:
    - Analyze the question and extract relevant graph
        components dynamically. Use this to construct the Cypher
         query.
    - Use only the relationship types and properties from the
        provided schema. Do not include any other relationship
        types.
    - The schema is based on a graph structure with nodes and
        relationships as follows:
    relations: {schema}
    entities: {entities}
    - Return only the generated Cypher query in your response.
        Do not include explanations, comments, or additional
        text.
    - Ensure the Cypher query directly addresses the given
        question using the schema accurately.
    - If you can't generate it, return "0"
    The question is:
    {question}
    """
```

Listing 2: Prompt template for Cypher generation

3. **Graph Query Execution:** The generated Cypher query is run on the Neo4j graph database to retrieve relevant subgraphs or entity sets that match the query intent.
4. **Contextual Information Fusion:** The retrieved graph data is combined with the original user query to form a coherent context, providing structured factual grounding for downstream processing.
5. **Answer Generation:** The combined context is fed into the LLM to generate a natural language answer grounded in the queried graph knowledge.

This pipeline leverages Neo4j's expressiveness for accurate retrieval while relying on LLMs for query interpretation and response generation.

Graph-RAG Pipeline This pipeline integrates graph-based retrieval with neural text generation to support product-centered question answering. The pipeline consists of the following steps:

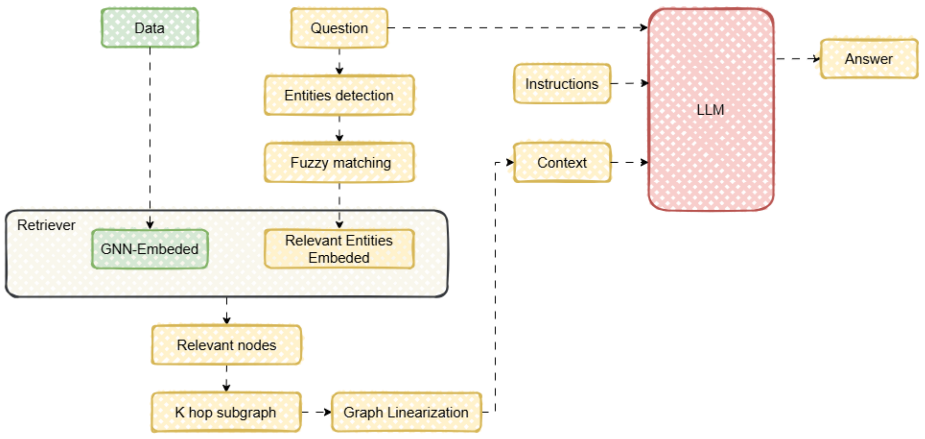

Figure 2: Graph-RAG Pipeline .

1. **Embedding and Indexing:** We compute node embeddings using one of the knowledge graph embedding methods described above and index them using FAISS for efficient retrieval.

```
    def _build_faiss_index(self):
        """
        Build FAISS index from the loaded embeddings.
        Returns:
            faiss.Index: FAISS index object
        """
        dim = self.embeddings.shape[1]
        index = faiss.IndexFlatL2(dim)
        index.add(self.embeddings)
        return index
```

Listing 3: FAISS index building function

2. **Entity Matching:** Given a user query, we apply a fuzzy matching algorithm to identify the most relevant entity mentions in the graph based on string similarity to node names.
3. **Subgraph Expansion:** For each anchor node, we perform k-hop expansion to gather a connected subgraph capturing related entities.
4. **Graph Linearization:** The subgraph is converted into natural language using a rule-based triple verbalizer or learned graph-to-text model. This step transforms structured facts into fluent context sentences.
5. **Answer Generation:** The linearized subgraph and original query are input to a pretrained language model, which generates the final response.

This embedding-based pipeline enables scalable graph retrieval and allows the language model to ground generation in structured knowledge, improving factual accuracy and relevance, especially for complex attribute queries.

5 Experiments

5.1 Problem Formulation

The primary task addressed in this work is *knowledge-grounded question answering* (KG-QA) over structured smartphone product data. Given a natural language query q, the objective is to retrieve and aggregate relevant information from a knowledge graph $G = (V, E, R)$, and generate a fluent, informative, and factually grounded answer a. Formally, we aim to learn a mapping:

$$q \to a \quad \text{such that} \quad a = \text{LM}\big(\text{Linearize}(\text{Retrieve}(q, G))\big)$$

Here, $\text{Retrieve}(\cdot)$ denotes a subgraph retrieval step, implemented via similarity search over pretrained node embeddings, and $\text{Linearize}(\cdot)$ transforms the resulting subgraph into natural language context for the language model (LM).

The final goal is to evaluate the effectiveness of different graph encoding models for knowledge-grounded question answering (KG-QA) in the smartphone customer support domain. Specifically, we assess how well various encoders support subgraph retrieval, which serves as the foundation for conditioning large language models in a Graph-RAG setting.

We benchmark six graph encoding models introduced in Section 4: TransE, ConvE, R-GCN, R-GAT, CompGCN, and KBGAT. These models span multiple architectural families including translational embeddings, convolutional approaches, and graph neural networks with attention or compositional reasoning capabilities.

Training Objective All models are trained on a link prediction task, which involves learning to predict missing edges in the knowledge graph. This task encourages the embeddings to capture the underlying relational structure and semantic compatibility between entities and relations. Link prediction is a suitable proxy for retrieval performance, as it reflects how well a model understands the graph connectivity, which is critical for accurate subgraph retrieval.

5.2 Evaluation Metrics

To assess overall system performance, we evaluate both retrieval and generation metrics:

Retrieval Quality

- **Recall@K**: Measures the proportion of relevant nodes retrieved in the top K candidates.
- **Mean Reciprocal Rank (MRR)**: Evaluates the average inverse rank of the first relevant node retrieved.

Answer Generation

- **ROUGE**: Evaluates lexical overlap and phrase-level matching between generated and reference answers.
- **METEOR**: Assesses semantic similarity, including synonyms and word order, between generated and reference answers.

5.3 Results

Table 3 summarizes the performance of several graph encoding models on the link prediction task, measured by Recall@k and Mean Reciprocal Rank (MRR). Overall, we observe that models incorporating attention mechanisms and relational modeling tend to outperform simpler baselines, particularly in the context of heterogeneous graphs.

Model	Recall@3	Recall@5	Recall@10	MRR
TransE	0.3214	0.4352	0.5677	0.3425
ConvE	0.5923	0.6614	0.7267	0.6121
R-GCN	0.5247	0.6451	0.7309	0.5441
R-GAT	0.4982	0.6247	0.7152	0.5041
CompGCN	0.5612	0.6934	0.7781	0.5689
KBGAT	**0.6315**	**0.7358**	**0.7903**	**0.6938**

Table 3: Retrieval performance of graph encoding models on link prediction task with Recall@3, @5, @10 and MRR.

Table 4 presents the quality of generated answers when integrating these graph embeddings into a retrieval-augmented generation (Graph-RAG) pipeline, evaluated by ROUGE and METEOR scores against gold-standard answers. The baseline pipeline using Neo4j Cypher queries demonstrates the lowest scores, indicating limited answer generation quality without learned embeddings.

Model	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
Graph-RAG Baseline	0.3651	0.1497	0.2236	0.2104
TransE	0.4205	0.1984	0.2807	0.2652
ConvE	0.4681	0.2502	0.3272	0.3291
R-GCN	0.4905	0.2756	0.3528	0.3554
R-GAT	0.4803	0.2650	0.3401	0.3450
CompGCN	0.5033	0.2891	0.3675	0.3687
KBGAT	**0.5210**	**0.3054**	**0.3832**	**0.3859**

Table 4: Answer generation quality for different Graph-RAG pipelines. Metrics are based on comparison with gold answers using ROUGE and METEOR.

5.4 Discussion

The results show that KBGAT achieves the best overall performance, both in retrieval and answer generation metrics, demonstrating the effectiveness of combining knowledge-aware attention with graph neural networks for modeling complex entity and relation interactions. Its high Recall@10 (0.7903) and MRR (0.6938) indicate strong ability to retrieve and rank relevant candidates, while superior ROUGE and METEOR scores reflect more accurate and fluent answer generation.

CompGCN also performs competitively, especially in Recall metrics (e.g., Recall@10 = 0.7781), highlighting the benefit of compositional relational modeling. Compared to R-GCN and R-GAT, CompGCN consistently retrieves more relevant candidates, although its MRR remains slightly lower than that of ConvE and KBGAT.

Among the GNN-based models, R-GCN slightly outperforms R-GAT across all metrics. While R-GAT introduces attention mechanisms that can theoretically enhance representation learning by weighting neighbor importance, its lower performance suggests that incorporating attention into relational message passing requires more careful architectural design or parameter tuning, especially in heterogeneous graph settings. Alternatively, the characteristics of our dataset—such as the diversity and sparsity of relation types—may limit the effectiveness of basic attention mechanisms, making structured relational aggregation, as employed by R-GCN, more robust in this context.

ConvE exhibits strong performance in MRR (0.6121), suggesting high ranking precision. However, it lags behind graph-based models in Recall, due to its limited ability to model multi-hop structural dependencies.

As expected, the translational embedding model TransE yields the lowest performance across all metrics, indicating that it lacks the representational power required to model the complex relational patterns present in our knowledge graph.

The answer generation results align well with retrieval performance, indicating that better retrieval embeddings contribute directly to higher-quality generated answers. This confirms that improving graph embedding quality is essential not only for retrieval accuracy but also for enhancing downstream generation in retrieval-augmented pipelines.

6 Conclusion

This work presents a Graph-Retrieval-Augmented Generation (Graph-RAG) framework tailored for the smartphone customer support domain. By constructing a heterogeneous knowledge graph from structured e-commerce data and leveraging both symbolic and neural retrieval pipelines, we demonstrate the potential of combining structured knowledge with language models for domain-specific question answering. Empirical results confirm that embedding-based retrieval—particularly with models like KBGAT and CompGCN—substantially improves both the relevance of retrieved subgraphs and the factual accuracy of generated answers compared to traditional symbolic query approaches.

Despite these promising results, several limitations remain. First, our entity matching strategy is primarily string-based and does not leverage contextual cues from the query, potentially reducing recall in cases of implicit references or paraphrased mentions. Second, the knowledge graph is static and reflects a snapshot of product data; it does not yet incorporate updates or user-generated content such as reviews or FAQs, which could enrich the knowledge base and improve response diversity. Additionally, while we evaluate retrieval and generation performance with automated metrics, human-centric evaluations—such as answer helpfulness or user satisfaction—are not yet included.

Future work will explore more advanced graph neural architectures, to enhance the contextual representation of retrieved subgraphs. We also aim to integrate community detection and graph summarization techniques to enable more scalable and semantically coherent subgraph retrieval. Finally, we plan to develop a joint training framework that learns retrieval and generation in a unified pipeline, allowing for tighter alignment between the retrieved knowledge and the generated response.

By advancing both retrieval and reasoning capabilities, this research contributes to the growing efforts in grounding language models with structured knowledge and demonstrates the value of graph-augmented generation in real-world customer support scenarios.

A Model Exploration and Design Choices

Throughout model development, we experimented with multiple architectures and training strategies to identify effective configurations for our Graph-RAG system. This section summarizes our observations from evaluating different model variants, score functions, and initialization techniques. Although these results are not part of a formal ablation study, they provide valuable insights into which design choices contributed to improved retrieval and generation performance.

ConvE with Adversarial Sampling. We compared uniform negative sampling with Sample Adversarial Negative Sampling (SANS). The SANS configuration consistently led to improved retrieval accuracy, highlighting its effectiveness in generating informative negatives.

```python
logits = model(h, r)  # shape: [B, num_entities]

# Step 1: Sample negatives excluding true tails
true_tails_list = [
    label[i].nonzero(as_tuple=False).view(-1).tolist()
    for i in range(batch_size)
]
neg_tails = sample_negatives(
    batch_size, num_negatives, num_entities, true_tails_list
)  # shape: [B, num_neg]

# Step 2: Retrieve logits of sampled negatives
batch_indices = torch.arange(batch_size).unsqueeze(1)  # shape: [B, 1]
neg_logits = logits[batch_indices, neg_tails]          # shape: [B, num_neg]

# Step 3: Compute self-adversarial weights
neg_weights = F.softmax(neg_logits * temperature, dim=1).detach()
```

Listing 4: Self-Adversarial Negative Sampling (SANS)

R-GCN Variants. We experimented with different configurations of the R-GCN model:

- **Score Function:** We tested both TransE-style and DistMult scoring functions for link prediction. DistMult yielded better performance, particularly on symmetric relation types.
- **Normalization & Dropout:** Adding layer normalization and dropout improved training stability and generalization, leading to better retrieval metrics.
- **Skip Connections:** Introducing residual connections led to faster convergence and a slightly boost in performance.
- **Pretrained Embeddings:** Initializing node embeddings with pretrained TransE vectors leads to faster improvement in the early training steps, although the final performance upon convergence is similar. This early speedup can be beneficial in resource-constrained settings.

KBGAT Initialization Strategy. For KBGAT, we evaluated random initialization versus using pretrained TransE embeddings for both nodes and relations. The pretrained configuration achieved the best performance overall in both retrieval (Recall@10, MRR) and answer generation (ROUGE, METEOR), suggesting that initializing with semantically meaningful embeddings benefits attention-based graph models.

These results demonstrate that model design decisions—such as residual connections, scoring functions, sampling strategies, and pretraining—can significantly influence both retrieval effectiveness and training efficiency. In particular, KBGAT with pretrained embeddings achieves the best overall performance, while R-GCN benefits notably from architectural refinements.

Table 5: Ablation results for selected model variants (Recall@10, MRR).

Model Variant	**Recall@10**	**MRR**
R-GCN (TransE Score)	0.6154	0.4226
R-GCN (DistMult Score)	0.6954	0.4887
Add LayerNorm + Dropout	0.7221	0.5210
Add Skip Connection	**0.7309**	0.5441
Using Pretrained TransE Init	0.7287	**0.5483**
ConvE + Uniform Sampling	0.7151	0.5817
ConvE + SANS	**0.7267**	**0.6121**
KBGAT (Random Init)	0.7258	0.6132
KBGAT + TransE Init	**0.7903**	**0.6938**

References

[1] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” *Advances in neural information processing systems*, vol. 26, 2013.

[2] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, “Convolutional 2d knowledge graph embeddings,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[3] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, “Modeling relational data with graph convolutional networks,” in *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*, Springer, 2018, pp. 593–607.

[4] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, “Composition-based multi-relational graph convolutional networks,” *arXiv preprint arXiv:1911.03082*, 2019.

[5] D. Nathani, J. Chauhan, C. Sharma, and M. Kaul, “Learning attention-based embeddings for relation prediction in knowledge graphs,” *arXiv preprint arXiv:1906.01195*, 2019.

[6] P. Lewis, E. Perez, A. Piktus, *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[7] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[8] X. Wang, H. Ji, C. Shi, *et al.*, “Heterogeneous graph attention network,” in *The world wide web conference*, 2019, pp. 2022–2032.

[9] Z. Hu, Y. Dong, K. Wang, and Y. Sun, “Heterogeneous graph transformer,” in *Proceedings of the web conference 2020*, 2020, pp. 2704–2710.

[10] A. Talmor, O. Yoran, A. Catav, *et al.*, “Multimodalqa: Complex question answering over text, tables and images,” *arXiv preprint arXiv:2104.06039*, 2021.

[11] M. Yasunaga, J. Leskovec, and P. Liang, “Linkbert: Pretraining language models with document links,” *arXiv preprint arXiv:2203.15827*, 2022.