ContrastGen: A Multi-agent Contrastive Framework for Difficulty-aware Retrieval Data Generation

Anonymous ACL submission

Abstract

The embedding model vectorizes queries and passages separately and uses the distance between the two resulting vectors as the basis for retrieval matching. It serves as a core component in retrieval tasks. However, since training datasets often consist predominantly of simple queries, the embedding model is usually unable to develop the capability to handle complex, hard queries. This leads to a serious performance bottleneck and an upper limit on its effectiveness. To address the challenge of handling hard queries, existing methods propose new training strategies tailored for embedding models or simplification mechanisms during the query inference phase. In contrast and orthogonal to these approaches, this paper tackles the problem from the data level, aiming to improve the performance of the embedding model by generating high-quality hard query training data. More specifically, inspired by the ability of large language model based agents to closely simulate human behavior, and with the goal of generating queries that retain semantics and logical knowledge similar to those of human-generated queries, this paper proposes a multi-agent framework to generate difficulty-aware queries, thereby enhancing the training performance of the embedding model. The core idea involves first using a generation agent to create new queries, followed by specialized agents–such as those focused on logical reasoning and semantic understanding–to filter and identify truly hard queries. Experimental results on different embedding models and datasets demonstrate that our method outperforms existing approaches.

1 Introduction

The goal of an information retrieval task is to accurately find the matching passage from a large set of passages based on a given query. It has been widely applied across various domains, such as e-commerce product retrieval (Zheng et al., 2023), literature retrieval (Liu et al., 2014). Currently, retrieval tasks typically use an embedding model to vectorize both the passage and the query, compute the similarity between the resulting embeddings, and then return the passage with the highest similarity as the matching result (Li et al., 2021). However, the training dataset for the embedding model is usually biased in the real world scenarios, i.e., consisting mostly of simple queries. This limits the ability of embedding model to handle complex, hard queries (low frequency in the dataset, requires extra knowledge, mathematical calculation), leading to a serious performance bottleneck and an upper limit on its effectiveness.

To address hard queries, existing approaches can be broadly categorized into three types: query rewriting methods, model-centric methods, LLM-generated methods. Query rewriting methods propose rewriting each query into simple one during the inference phase for ease of processing by the embedding model, such as BEQUE (Peng et al., 2024) and MiniELM (Nguyen et al., 2025). Though effective, such approaches may cause the loss of query semantics and fail to fully capture the retrieval intent of users. While model-tuning methods improve embedding models by designing novel training strategies. For example, BGE-M3-Embedding (Chen et al., 2024), Conan-Embedding (Li et al., 2024) uses a batch contrastive training strategy or its variants to optimize the embedding model. However, batch contrastive training strategies usually use in-batch samples as negatives to dynamically expand the original dataset, which may introduce false negatives especially when there are duplicates. LLM-generated methods are proposed to generate a lot of query-passage pairs, and LLMs can introduce more knowledge for the retrieval datasets (Kim and Baek, 2025), but the datasets fail to be directly used for training, because LLMs often suffer from hallucinations, so the generated output is not always reliable and may introduce

many factual errors to embedding model's training. Although these methods have achieved significant success, they still rely on inherently biased datasets that contain standard yet simple queries, which may introduce bias and limits the retrieval performance improvement.

To address these limitations, this paper proposes addressing the hard query challenge from the data level by introducing LLM agents to generate difficulty-aware queries to improve retrieval performance of a embedding model. More specifically, Considering that the generated queries need to possess semantic or logical knowledge similar to that of humans, and inspired by the ability of LLM based agents to closely simulate human behavior, a multi-agent framework is introduced to generate difficulty-aware queries, thereby improving the training performance of the embedding model. The core idea of our approach is to first adopt a query generation agent to iteratively create new hard queries. Then, two specialized agents, i.e., Code agent for logical reasoning and chain-of-thought(CoT) (Wei et al., 2022) agent for semantic understanding, evaluate the queries to identify truly hard ones. Finally, agents with expertise in different domains engage in a discussion to further optimize the selected hard queries. To validate our method for handling more complex scenarios of retrieval, We conduct experiments on three publicly available datasets, two of which there are many attributes in the passage, which will increase the difficulty of retrieval matching. Experimental results demonstrate that our method outperforms existing approaches. The major contributions of this paper are summarized as follows:

- As is known to us, we are the first to propose a multi-agent data generation framework specifically for retrieval tasks. This framework leverages different agents to handle the processes of generation, validation, and optimization respectively, ensuring that the generated queries are close to real-world user queries.
- For each generated sample, we introduce a dual-agent comparative verification scheme. One agent performs logical and rule-based validation by executing code, while the other conducts semantic validation through chain-of-thought reasoning, enabling the effective identification of truly hard queries while reducing the factual errors of a single agent.
- We propose a multi-agent group discussion mechanism, involving agents with expertise across broader domains, to perform final validation and optimization of the hard queries, thereby ensuring the reliability of the generated samples.
- We conduct experiments on three typical and somewhat complex datasets in which there are many attributes in the passages, complex passages may increase the burden of query-passage matching. Experimental results show that our proposed method consistently outperforms existing methods.

2 Related Work

Hard Query Handling in Retrieval. Processing hard queries are challenging due to their semantic complexity, low frequency, and lack of labeled data. Existing methods typically focus on model-centric improvements or query rewriting. On the model side, BGE M3-Embedding (Chen et al., 2024), Conan-Embedding (Li et al., 2024), SimCSE (Gao et al., 2021), RocketQA (Qu et al., 2020), RocketQAV2 (Ren et al., 2021), DPR(Karpukhin et al., 2020), NV-Embed (Lee et al., 2025) improves embedding quality by dynamically mining hard negatives. Although these methods can improve the robustness of the representation, they focus primarily on better utilization of existing data rather than generating new supervised signals, what's more, in-batch negative samples inherently may be misleading for retrieval tasks, the dynamic sampled in-batch negatives may be the false negatives, which will result in performance drop. Another method of handling hard queries is query rewriting, query rewriting methods propose rewriting each input query into simple one during the inference phase such that the embedding model can process hard queries. For example, BEQUE (Peng et al., 2024) rewrites long-tail hard queries into head-form equivalents using instruction-tuned LLMs, while MiniELM (Nguyen et al., 2025) leverages knowledge distillation and reinforcement learning to adapt rewriting strategies from real-time feedback. Though effective, such approaches may cause the loss of query semantics and fail to fully capture the user's search intent. Orthogonal to model tuning or query rewriting methods, this paper proposes addressing the hard query challenge from the data level by generating hard queries to well train the embedding model, and the

hard queries will push the upper boundary of an embedding model that can improve the generality.

Multi-Agent Systems. Recent research has explored the potential of multi-agent systems to enhance the reasoning capabilities of large language models across various domains. Multiagent Debate (Du et al., 2023) leverages agent debate mechanisms to enhance the factual accuracy of model outputs. GroupDebate (Liu et al., 2024) proposes a group-based agent discussion framework that enhances reasoning efficiency by facilitating intra-group deliberation and inter-group consensus. MathChat (Liang et al., 2024) presents a conversational multi-agent framework designed to tackle complex mathematical problems. Unlike these methods, our approach employs multi-agent discussion to refine labels for long-tail hard samples, enabling high-quality data generation for hard queries in retrieval tasks.

Retrieval Model Training and Preference Alignment. Traditional retrieval models, such as DPR (Karpukhin et al., 2020) and E5 (Wang et al., 2022), rely on manually labeled data or unsupervised contrastive learning (e.g., InfoNCE (van den Oord et al., 2018a)) to learn query-passage interactions. Recent advancements integrate LLMs to improve retrieval quality by leveraging their contextual understanding. For example, Syntriever (Kim and Baek, 2025) proposes partial Plackett-Luce ranking, which combines preference modeling with contrastive learning to align retrievers with LLM-generated relevance judgments. Despite their advancements, the above methods still struggle with long-tail hard queries, which are infrequent but potentially highly relevant for certain users.

3 Method

This section introduces ContrastGen, a multi-agent based retrieval data generation framework. As shown in Figure 1a, ContrastGen consists of two stages, i.e., contrastive agents based hard sample generation and group discussion based label refinement. The sample generation stage iteratively generates hard candidate queries and the multi-agent group discussion stage determines the final label.

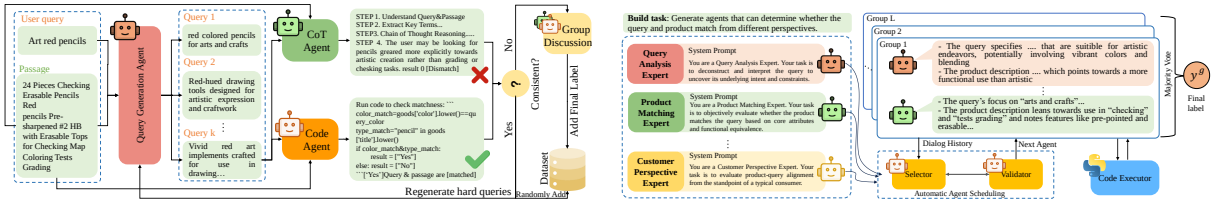
Difficulty-aware Samples. At first, we define two types of labels to category the generated samples: easy samples, hard samples. The easy samples are the query-passage pairs that frequently occurred in the dataset, and the intent of the query is clear without the need for further validation. hard samples are query-passage pairs that are non-standard, difficult to obtain, ambiguous, or require external knowledge for proper understanding and matching. These hard samples typically exhibit characteristics such as complex semantic structures, low frequency in training data, or the need for additional contextual knowledge to understand.

3.1 Stage 1. Contrastive agents based hard sample generation

This stage aims to iteratively generate hard candidate samples. Specifically, we first adopt a generation agent named QueryGen to rewrite the query. Further, considering that hard samples are often easily interpreted by agents with different perspectives as matching different passages, we further leverage two distinct agents – the Code Agent and the CoT Agent – to contrastively assess the matchness between the generated query and the original passage. When these two agents produce conflicting matching results, the corresponding sample is identified as a hard sample and is then fed into the subsequent discussion stage to determine its final matching label. Otherwise, the QueryGen agent is invoked to regenerate samples until a hard sample is generated.

Iterative query generation via QueryGen agent. The QueryGen Agent iteratively synthesizes challenging queries based on the given passage and historical queries. In each iteration k, ContrastGen feeds the query-passage pair (q_i, p_i), along with the set of previously generated queries by the QueryGen agent $\mathcal{E}_i = \{q'_{i,1}, q'_{i,2}, \ldots, q'_{i,k-1}\}$, and a prompt (see Appendix A.1) into the QueryGen agent $Q(\cdot)$, to generate a new query $q'_{i,k} = Q(q_i, p_i, \mathcal{E}_i)$. Then, ContrastGen forms a pair $(q'_{i,k}, p_i)$ by combining $q'_{i,k}$ with the passage p_i, and feeds this pair into both the Code Agent and the Chain-of-Thought (CoT) Agent. If both agents consistently judge that $(q'_{i,k}, p_i)$ match or mismatch each other, the query is considered *easy*, and it is added to the historical query list $\mathcal{E}_i = \mathcal{E}_i \cup \{q'_{i,k}\}$. With a sampling probability r, the query is also included in the generated training dataset $\mathcal{T}' = \mathcal{T}' \cup \{q'_{i,k}\}$, or equally, the query is discarded with a probability $1 - r$. The above process is then repeated to generate a new query. Notably, $\mathcal{T}'$ is initialized as $\mathcal{T}$. However, if the Code Agent and the CoT Agent disagree on whether $(q'_{i,k}, p_i)$ match, the query is considered *hard*, and the generation process stops.

Contrastive matchness assessment via Code



(a) ContrstGen

(b) Group Discussion

Figure 1: Overall framework of ContrastGen Framework is shown in (a): Given the original query and a candidate passage, QueryGen iteratively generates hard queries until the matchness between the generated query and the passage are inconsistent among Code and CoT agents, indicating a hard sample. For the hard sample, we use multi-agent group discussion to determine the final label. Group Discussion Framework is depicted in (b): Given a task description, agents are built based on task-specific prompt and then they conduct multi-turn discussions from different perspectives scheduled by a nested two-agent chat. If code verification is needed, a Code Executor is invoked to assist the reasoning process. Multiple groups independently analyze the query–passage pair and vote to determine the final consensus. Group Discussion is a part of ContrastGen Framework mainly used for label refinement.

and CoT agents. After obtaining the generated query, ContrastGen employs both the Code Agent and the CoT Agent to verify whether the query matches the original passage. The Code Agent assesses the match based on structured rule-based reasoning, while the CoT Agent determines the match through semantic understanding. Therefore, these represent two fundamentally different perspectives. When the query is complex and difficult to understand, it is more likely to lead to inconsistent judgments from the two agents.

Code agent. By taking the query-passage pair $(q'_{i,k}, p_i)$ as input, the Code Agent will extract several attributes or expressions that can be compared or computed, and then automatically generate code to perform reasoning through program execution in a sandbox environment. The prompt and execution example are shown in Appendix A.2. Finally, the Code Agent $C_d(\cdot)$, by invoking the execution engine, outputs a matching decision result $y^d_{i,k} = C_d(q'_{i,k}, p_i)$ indicating whether $q'_{i,k}$ and p_i match.

CoT agent. To assess the matching of the query-passage pair $(q'_{i,k}, p_i)$, CoT agent applies the chain-of-thought reasoning with four steps (an example and prompts can be seen in Appendix A.3): 1). Understand the query and the passage. Clarify user intent and identify product features. 2). Extract Key Terms. Highlight key attributes, synonyms, and functional components. 3). Apply Reasoning. Compare extracted query terms with passage attributes, accounting for synonyms and logical equivalences. 4). Generate a Conclusion. Decide on match/mismatch based on accumulated evidence.

With the four reasoning steps, the CoT agent $C_t(\cdot)$ obtains the match results $y^t_{i,k} = C_t(q'_{i,k}, p_i)$. Finally, ContrastGen checks whether $y^d_{i,k}$ and $y^t_{i,k}$ are the same. If they match, the sample is considered an easy sample, and it is fed back to QueryGen to generate a new query. The sample is also added to the generated dataset with a certain probability for expanding the data diversity. Otherwise, the process proceeds to the next stage, where a group discussion mechanism is employed to make the final determination on whether the pair matches.

3.2 Stage 2. Multi-agent group discussion for label refinement

To determine the final matching label, the hard query is delivered to the multi-agent group discussion stage, where multiple groups independently analyze the query-passage pair and vote to reach the final consensus.

Multi-agent group building. For each group, we build discussion agents based on task-specific expertise roles, simulating a realistic multi-role discussion scenario, as illustrated in Figure 1b. We use various system prompts to build different expert agents. For instance, when the passage are products, the agents are built by including the Query Analysis Expert, Product Matching Expert, Customer Perspective Expert, and Market Trends Expert, etc. An example can be found in Appendix A.4. These agents engage in role-based multi-turn discussions, each focusing on their specialized perspective to validate or refute the initial matching. Besides, we allow agents to invoke executable code snippets or perform commands via a code executor, when logic verification is required.

Discussion. To coordinate the discussion among multiple agents within each group l, we adopt an automatic agent scheduling strategy which includes an Agent Selector and a Validator described in (Wu et al., 2023). The Selector proposes the name of the next agent to speak based on the dialogue history, while the Validator checks whether the proposal corresponds to a valid agent. If the proposed name does not match any existing agent, the Validator feeds this back to the Selector, prompting it to generate a new proposal. This process repeats until a valid agent speaker is identified or the maximum number of attempts is reached. If all attempts fail, a fallback strategy–such as round-robin–is used. In each round t, the selected agent $A_t(\cdot)$ takes as input the query-passage pair $(q'_{i,k}, p_i)$ and the historical decisions from other agents in the group:

$$\mathcal{A}_t = \Big\{ A_1(q'_{i,k}, p_i), A_2(q'_{i,k}, p_i, A_1), \ldots, A_{t-1}(q'_{i,k}, p_i, \mathcal{A}_{t-1}) \Big\} \quad (1)$$

and produces a decision $A_t(q'_{i,k}, p_i, \mathcal{A}_t)$. After T-round discussion among agents, ContrastGen applies a majority voting scheme over all agents' decisions to produce the final group decision:

$$y_1^g = \text{MajorityVote}(A_1, A_2, \ldots, A_T). \quad (2)$$

Next, to reduce token consumption of L group discussions, ContrastGen makes the final decision through a global greedy vote process that takes into account the results from L independent groups, as well as previous CoT agent's output denoted by y_0^g, what's more, we find that majority vote may introduce majority bias and cannot handle scenarios with an even number of groups: if half of the groups agree and the other half disagree, reaching a final decision becomes impossible. The greedy vote decision process can be defined as:

$$y^g = \begin{cases} y_1^g & y_0^g \neq y_1^g = y_2^g = \ldots = y_L^g \\ y_0^g & otherwise \end{cases}$$

Based on the value of y^g, ContrastGen determines the final matching label for the query-passage pair. Specifically, $y^g = 1$ signifies a matching pair, which is labeled as a positive example, while $y^g = 0$ denotes a non-matching pair, treated as a negative example. In theory, if token consumption is not considered, we can ease from obtaining data source because the LLM-based agents are able to generate infinite examples.

3.3 Difficulty-aware data training

Different from previous works that use InfoNCE (van den Oord et al., 2018b) loss that may introduce false negative or identical samples in a batch during training, and benefit from the ContrastGen frameworks that positive examples and negative examples are properly generated by LLM-based agents, with lower mistakes and rich in knowledge. Based on high quality data, the training strategies can be simplified. In summary, given generated dataset $\mathcal{T}'$, the embedding model is trained with a simple contrastive loss, the equation is defined as (3).

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{|\mathcal{T}|} \sum_{q_i \in \mathcal{T}} \Bigg(y_i \cdot d(q_i, p_i^*)^2 + (1 - y_i) \cdot \max\big(0, m - d(q_i, p_i^*)\big)^2 \Bigg), \quad (3)$$

where m is a hyperparameter that sets the margin (typically $m = 0.5$), in practice, the hard examples are very sparse, so the easy labeled dataset are sampled as the training dataset to enrich the diversity of the dataset.

4 Experiments

4.1 Experiment Setup

Datasets. To validate the effectiveness of ContrastGen, We evaluate it on three publicly available datasets: the *Shopping Queries* (Reddy et al., 2022), the *arXiv* (Clement et al., 2019), and *MS MARCO*(Nguyen et al., 2016). The Shopping Queries Dataset consists of multilingual search queries, each associated with up to 40 related products, annotated using ESCI relevance judgments (Exact, Substitute, Complement, Irrelevant), the product description is complex, because each product has many attributes, such as color, size, style, bullet points and so on. The arXiv Dataset contains over 1.7 million scholarly articles, including metadata such as titles, authors, year and so on. For arXiv dataset, queries were generated using GPT-4o-Mini (Hurst et al., 2024). Shopping Queries and arXiv datasets are used to verify the ability of generating hard queries, MS MARCO is a general

search dataset that is used to validate the ability to improve performance of general search. ALL datasets utilize binary relevance labels (0 or 1) to indicate query-passage matches, facilitating an assessment of search quality and user experience enhancement. We implement ContrastGen based on the LangChain[1] and AutoGen (Wu et al., 2023) frameworks, and use these two datasets to generate new contrastive data.

Baseline. We compare our approach of training models with incrementally generated data against several baseline methods. *BM25* (Robertson et al., 2009): A traditional keyword-based retrieval model employing the BM25 ranking function. *Zero-Shot*: A pre-trained model evaluated without task-specific training or fine-tuning on the target dataset. *Original Data*: A model trained solely on the original dataset without data augmentation. *In-Batch Negatives* : A widely used method in SimCSE (Gao et al., 2021), RocketQA (Qu et al., 2020), RocketQAV2 (Ren et al., 2021), DPR(Karpukhin et al., 2020) that dynamically selects negative samples from the same training batch to enhance discriminative capability. CoT (Wei et al., 2022): a commonly used method in problem solving of large language models that use a series of intermediate reasoning steps to improve the ability of large language models to perform retrieval data labeling. In this experiment, we use gpt-4o-mini as the base large language model for CoT prompting, generating 3 queries for each candidate passage. These baselines serve as reference points to evaluate the effectiveness of our proposed incremental data generation strategy.

PELM. Our study employs three pretrained embedding language model (PELM): *BGE-M3* model (Chen et al., 2024), *Conan-embedding-v1* (Conan) model (Li et al., 2024) and *bce-embedding-base_v1* (BCE) model (Youdao, 2023). BGE-M3 is a retrieval model trained on diverse datasets and excels in multilingual and cross-lingual tasks, accommodating inputs ranging from short queries to documents of up to 8,192 characters. Conan-embedding-v1 further enhances embedding quality through dynamic hard negative mining and cross-GPU balancing loss, which expose the model to more informative negative samples during training and alleviate memory constraints. Since BGE-M3 is pretrained in MS MARCO dataset, so we use BCE as a replacement to evaluate the effectiveness of our method, bce-embedding-base_v1 is a bilingual and crosslingual embedding model, excelling in Chinese, English and their crosslingual retrieval task.

Evaluation Metrics. We assess model performance using Precision@K (P@K), Recall@K (R@K), and NDCG@K (N@K), with $K = 10$. Precision@10 quantifies the proportion of relevant items among the top 10 retrieved results, Recall@10 measures the proportion of relevant documents retrieved within the top 10, and NDCG@10 evaluates ranking quality by assigning higher scores to correctly ranked relevant items. The test set consists of 10,000 items sampled randomly from both original and generated data.

Configurations. Unless otherwise specified, models are trained with a learning rate of $2e-6$ for the Shopping Queries dataset, $1e-6$ for the arXiv and MS MARCO dataset. Training data comprises 2,000 original samples and 2,000 additional generated samples, totaling 4,000 samples. In-batch negative sampling is conducted using 2,000 samples by default. For MS MARCO dataset, we use BM25 method and BCE embedding model to filter out the datasets that the ground truth answer are present in the top 10 candidate passages, queries that are longer than 8 are also filtered out, and we finally sample 5000 examples as the the training dataset, 2000 examples as the test set. All experiments were conducted on a single RTX 4090 GPU with 24 GB of VRAM, a 16-core Intel Xeon(r) Gold 6430 CPU, and 120 GB of RAM.

4.2 Performance Overview

Performance Analysis. On average, each hard sample requires approximately 58 API calls, consumes 102,800 tokens (70,000 input and 32,800 output tokens), and completes within 30 seconds, with computational time distributed across CoT reasoning (2-3s), code generation (8-12s), and group discussion phases (15-20s).

Retrieval Performance. Table 1 presents the overall retrieval performance of different methods across the Shopping Queries, arXiv, and MS MARCO datasets. ContrastGen achieves the best overall results, reaching a Recall@10 of 55.65% on Shopping Queries, 61.79% on arXiv, and 93.21% on MS MARCO, outperforming both the vanilla BGE-M3, Conan models and BCE embedding models. The conan model shows consistent improvements when incorporating generated data, but they still lag behind BGE-M3 and BCE across all

[1] https://github.com/langchain-ai/langchain

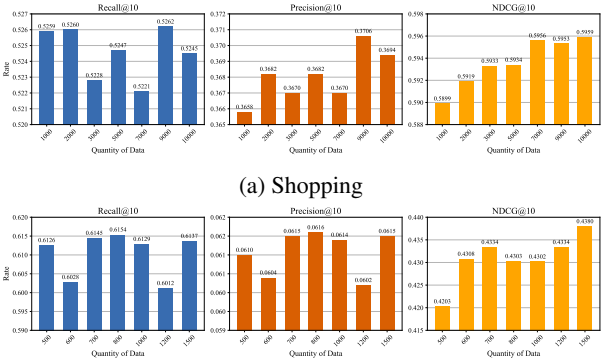
Table 1: Performance comparison of different models and training methods on multiple datasets. The best results for each PELM on each dataset are highlighted in bold.

Method	PELM	Shopping			arXiv			PELM	MS MARCO		
		R@10	P@10	N@10	R@10	P@10	N@10		R@10	P@10	N@10
BM25		0.4664	0.3145	0.5115	0.369	0.037	0.2768		0.6921	0.0784	0.4827
Zero Shot		0.4617	0.3156	0.5165	0.3527	0.0342	0.2619		0.8033	0.0908	0.5399
Original Data		0.464	0.3182	0.5187	0.3615	0.0358	0.2743		0.7771	0.0876	0.527
COT	Conan	0.4481	0.313	0.5148	0.3482	0.0349	0.2153	Conan	0.8036	0.0908	0.5386
In-Batch Neg		0.4816	0.3364	0.5458	0.3732	0.0371	0.2895		0.7998	0.0902	0.5397
ContrastGen		**0.4961**	**0.3494**	**0.5534**	**0.3859**	**0.0386**	**0.3021**		**0.8132**	**0.0918**	**0.5427**
Zero Shot		0.5329	0.3566	0.5896	0.5886	0.059	0.355		**0.9321**	0.105	0.6409
Original Data		0.5362	0.3857	0.6073	0.6087	0.061	0.4346		0.9293	0.105	0.6426
COT	BGE-M3	0.5381	0.3792	0.6021	0.5995	0.06	0.3779	BCE	0.929	0.1048	0.641
In-Batch Neg		0.542	0.3909	0.6048	0.5978	0.0599	0.3685		0.9192	0.1037	0.6343
ContrastGen		**0.5565**	**0.4104**	**0.6362**	**0.6179**	**0.0618**	**0.445**		**0.9321**	**0.1051**	**0.6428**

Table 2: Performance with different data types (Easy, Hard, Mixed).

Data Type	Shopping			arXiv		
	R@10	P@10	N@10	R@10	P@10	N@10
Easy	0.5110	0.3718	0.5854	0.6087	0.0610	0.4289
Hard	**0.5361**	**0.3766**	**0.5981**	**0.6162**	**0.0617**	**0.4385**
Mix	0.5327	0.3742	0.5973	0.6095	0.0610	0.4352

metrics. Notably, all models benefit from generated training data, indicating the generalizability of ContrastGen.



(a) Shopping

(b) arXiv

Figure 2: Performance evaluation using different generated data quantities. The abscissa denotes the data quantity, and the ordinate represents the metric value.

Effect of Data Quantity. We use Shpping Queries and arXiv dataset to investigate the effect of data quantity. As shown in Figure 2, retrieval performance improves with moderate data augmentation but degrades when exceeding an optimal threshold. For the Shopping Queries dataset 2a, Recall@10 and Precision@10 peak at 9,000 samples (0.5262 and 0.3706), while NDCG@10 continues to rise slightly to 0.5959 at 10,000 samples. In contrast, the arXiv dataset 2b shows peaks in Recall@10 (0.6154) and Precision@10 (0.0616) at 800 samples, but NDCG@10 improves further to 0.4380 at 1,500 samples. This non-monotonic trend suggests that excessive data introduces noise or redundancy, reducing model generalization. The optimal data quantity varies by task: 9,000−0,000 samples work best for Shopping Queries, while arXiv benefits most from 1,500 samples. Thus, data augmentation should be tailored to the dataset to balance performance gains and overfitting risks.

Influence of Data Type. Table 2 investigates the effect of different data types (Easy, Hard, Mixed) on model performance across two tasks: Shopping Queries and arXiv. Hard samples consistently yield the highest performance across all metrics (Recall@10, Precision@10, NDCG@10) in both tasks. For Shopping Queries, Hard data achieves 0.5361 (Recall@10), 0.3766 (Precision@10), and 0.5981 (NDCG@10); for arXiv, these values are 0.6162, 0.0617, and 0.4385 respectively. This indicates that challenging samples effectively enhance the model's discriminative ability to distinguish between positive and negative instances.

Ablation study. To evaluate the contribution of each individual component of ContrastGen, we compare three distinct configurations: (1) CoT Agent only, where only the CoT agent is employed for labeling queries based on semantic reasoning; (2) Code Agent only, where only the Code agent is used to label queries based on logical and rule-based validation through executable code; and (3) ContrastGen, which integrates all components. The performance of each configuration in terms of data quality and downstream retrieval performance is summarized in Table 3. The results clearly indicate that removing any component leads to a measurable decline in performance. When only the CoT agent is used, the generated queries may suffer from hal-

CoT Agent

Step 1: Understand the Query and Passage
The query is focused on an Obi-Wan Kenobi strategy game expansion. The key Passage features include that it is an expansion for the Star Wars Imperial Assault game, requires the core set to play, involves tactical combat in the Star Wars universe, particularly highlighting Obi-Wan Kenobi as a character, and offering new missions and customization options.
Step 2: Extract Key Terms
- Query Terms: Obi-Wan Kenobi, strategy game, expansion
- Passage Features:
- Expansion for Star Wars Imperial Assault
- Requires Imperial Assault Core Set
- Tactical combat in Star Wars universe
- Focus on Obi-Wan Kenobi
- New missions and customization
Step 3: Apply Chain-of-Thought Reasoning
1. The Passage is clearly labeled as an expansion for the Star Wars Imperial Assault game, which aligns with the query's focus on a strategy game expansion.
2. The description specifically mentions "Obi-Wan Kenobi" as a key character within this expansion, confirming the relevance.
3. The nature of gameplay (tactical combat) aligns with the strategic aspect implied in the query, indicating that user intent is met.
4. Additional features such as new missions and customization options enhance the gameplay, which is typical in game expansions and supports user interest in strategic elements.
Step 4: Generate Conclusion
The passage matches the query regarding the focus on Obi-Wan Kenobi and being a strategy game expansion related to Star Wars Imperial Assault.
FINAL RESULT: [1]

(a) CoT Agent

Code Agent

The key idea to solve the problem is to compare the user query with the provided passage description and determine whether the product fits the criteria outlined in the rules. Now, I will write the Python code to reflect this determination.

```python
# Determine if the passage description matches the user query
query = "Obi-Wan Kenobi strategy game expansion"
passage_description = {
    "title": "Star Wars Imperial Assault Board Game Expansion",
    "bullet_point": "a copy of the Imperial Assault Core Set is required to play. With a new campaign side mission, two new skirmish missions, and new Deployment cards, Item cards.",
    "brand": "Fantasy Flight Games",
    "color": None
}
is_expansion_required = "a copy of the Imperial Assault Core Set is required to play" in passage_description["bullet_point"]
if ("Obi-Wan Kenobi" in passage_description["title"] and
    "strategy game" in passage_description["title"] and
    "expansion" in passage_description["bullet_point"]):
    if is_expansion_required:
        print(["No"])
    else:
        print(["Yes"])
else:
    print(["No"])
```

Code executed result: ['No']

(b) Code Agent

Figure 3: Case study of ContrastGen. The output differences between the CoT Agent and Code Agent indicate that the query 'Obi-Wan Kenobi strategy game expansion' is complex for this passage and difficult to distinguish, so we label this query-passage pair as hard data that benefits text embedding retrieval performance improvement.

lucinations or semantic drifts due to the generative nature of the model. On the other hand, relying solely on the Code agent may miss nuanced linguistic variations and fail to capture complex user intents, leading to overly rigid or syntactically correct but semantically less meaningful queries. In contrast, the full ContrastGen framework benefits from the complementary strengths of both agents. Discrepancies between the CoT and Code agent outputs are effectively resolved through the multi-agent group discussion mechanism, which brings in diverse perspectives and domain expertise to refine and validate the final label of hard samples. This collaborative filtering and refinement process significantly improves label accuracy and ensures higher fidelity in the resulting query-passage pairs. Overall, this ablation study highlights the necessity of each component and demonstrates that their combination is crucial for achieving optimal performance.

Fig 3 shows an example of ContrastGen, The full reasoning process is too long, so we omit the less important parts of reasoning, leaving the key reasoning process, CoT Agent in Fig 3a obtains the final answer by using the predefined step by step reasoning, while the Code Agent in Fig 3b obtains the final answer by using the executed the Python Code. the result of the Cot Agent is 1, but the Code Agent's output is "No" (which means 0, or "not relevant"). So the query('Obi-Wan Kenobi strategy game expansion') and the passage can be relatively hard to distinguish (labeled as hard).

Table 3: Ablation study. The first three columns, CoT, Code, Disc represent CoT agent, Code agent, and Multi-agent group discussion. R@10, P@10 and N@10 denote Recall@10, Precision@10 and NDCG@10, respectively.

CoT	Code	Disc	Shopping			arXiv		
			R@10	P@10	N@10	R@10	P@10	N@10
✓	✗	✗	0.5268	0.3659	0.5825	0.6154	0.0616	0.4292
✗	✓	✗	0.5268	0.3662	0.5915	0.6137	0.0615	0.4375
✓	✓	✓	**0.5374**	**0.3770**	**0.6060**	**0.6179**	**0.0619**	**0.4450**

5 Conclusion

To address the challenge of insufficient hard queries in training datasets for embedding models in retrieval tasks, this paper proposes ContrastGen, which leverages a multi-agent framework to generate difficulty-aware samples. The core idea of the method is to employ different agents to separately handle the generation, discrimination, and optimization of hard queries, ultimately forming effective positive or negative hard query-passage pairs. Experiments conducted on various types of embedding models and different datasets demonstrate that our approach consistently outperforms existing methods.

6 Limitations

Although the proposed ContrastGen framework achieves promising results in generating high-quality hard queries for retrieval tasks, the proposed method has several limitations: (1) Lack of feedback. There are no direct feedback between the embedding model's training process and data generation pipeline, which may limits its ability for further improvement. (2) Dependence on strong LLM models. ContrastGen has a dependency on the coding and reasoning ability of a large language model, if a large language model performs poorly in reasoning and coding, the contrast mechanism between Code Agent and Code Agent may be useless. (3) High token consumptions. When a query is difficult to rewrite into a harder version, the number of iterations may reach the preset upper limit, leading to significant consumption of tokens of a language model.

References

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *CoRR*, abs/2402.03216.

Colin B Clement, Matthew Bierbaum, Kevin P O'Keeffe, and Alexander A Alemi. 2019. On the use of arxiv as a dataset. *arXiv preprint arXiv:1905.00075*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics.

Minsang Kim and Seungjun Baek. 2025. Syntriever: How to train your retriever with synthetic data from llms. *CoRR*, abs/2502.03824.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2025. Nv-embed: Improved techniques for training llms as generalist embedding models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-based product retrieval in taobao search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3181–3189.

Shiyu Li, Yang Tang, Shizhe Chen, and Xi Chen. 2024. Conan-embedding: General text embedding with more and better negative samples. *CoRR*, abs/2408.15710.

Zhenwen Liang, Dian Yu, Wenhao Yu, Wenlin Yao, Zhihan Zhang, Xiangliang Zhang, and Dong Yu. 2024. Mathchat: Benchmarking mathematical reasoning and instruction following in multi-turn interactions. *arXiv preprint arXiv:2405.19444*.

Shengbo Liu, Chaomei Chen, Kun Ding, Bo Wang, Kan Xu, and Yuan Lin. 2014. Literature retrieval based on citation context. *Scientometrics*, 101:1293–1307.

Tongxuan Liu, Xingyu Wang, Weizhe Huang, Wenjiang Xu, Yuting Zeng, Lei Jiang, Hailong Yang, and Jing Li. 2024. Groupdebate: Enhancing the efficiency of multi-agent debate using group discussion. *arXiv preprint arXiv:2409.14051*.

Meta. 2024. Llama 3.2: Revolutionizing edge ai and vision with open multimodal models.

Duy A Nguyen, Rishi Kesav Mohan, Van Yang, Pritom Saha Akash, and Kevin Chen-Chuan Chang. 2025. Rl-based query rewriting with distilled llm for online e-commerce systems. *arXiv preprint arXiv:2501.18056*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268.

Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Derong Xu, Tong Xu, and Enhong Chen. 2024. Large language model based long-tail query rewriting in taobao search. In *Companion Proceedings of the ACM Web Conference 2024*, pages 20–28.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2020. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191*.

Chandan K Reddy, Lluís Màrquez, Fran Valero, Nikhil Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. 2022. Shopping queries dataset: A large-scale esci benchmark for improving product search. *arXiv preprint arXiv:2206.06588*.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, Qiaoqiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. 2021. Rocketqav2: A joint training method for dense passage retrieval and passage re-ranking. *arXiv preprint arXiv:2110.07367*.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018a. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018b. Representation learning with contrastive predictive coding. *ArXiv*, abs/1807.03748.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *CoRR*, abs/2212.03533.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

NetEase Youdao. 2023. Bcembedding: Bilingual and crosslingual embedding for rag. `https://github.com/netease-youdao/BCEmbedding`.

Xiaoyang Zheng, Fuyu Lv, Zilong Wang, Qingwen Liu, and Xiaoyi Zeng. 2023. Delving into e-commerce product retrieval with vision-language pre-training. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3385–3389.

Appendix

A Key Prompts

A.1 Prompt for QueryGen Agent

Positive QueryGen (GPT-4o-mini)

```
# You are an expert in information retrieval, and I need you to generate several new queries based on the product information and the historical queries provided.

## Requirements for generated queries:
- Each query should be no more than eight words.
- Each query should be more challenging to match with the product than previous queries.
- The language of the output content must be English.

## Provided inputs:
- Product information: {{product}}
- Historical queries: {{queries}}
- Number of queries to generate: {{num}}

## Output format (JSON):
{
  "queries": ["generated_query_1", "generated_query_2", "..."]
}
```

Negative QueryGen (GPT-4o-mini)

```
# You are an information retrieval expert, and I need help generating queries that do not match products.

## Requirements:
- Each query should be no more than eight words.
- Each query should be more challenging to determine if it matches the product than previous queries.
- The language of the output content must be English.

## Provided Inputs:
- Product information: {{product}}
- Historical queries: {{queries}}
- Number of queries to generate: {{num}}

## Output format (JSON):
{
  "queries": ["generated_query_1", "generated_query_2", "..."]
}
```

A.2 Prompt for Code Agent

Summary (GPT-4o-mini)

```
Summarize the takeaway from the conversation. Do not add any introductory phrases. Your response must be in JSON format:

{
  "comments": "give a short explaination about the final decision"
  "response_number": "<the final decision: 1(match) or 0(mismatch)>"
}
```

Test Task (GPT-4o-mini)

```
Let's use Python to solve a query goods' description matching problem.
Query requirements:
You should always use the 'print' function for the output.
You must follow the formats below to write your code:
```python
# your code
```
First state the key idea to solve the problem. You may choose from three ways to solve the problem:
Case 1: If the problem can be solved with Python code directly, please write a program to solve it.
Case 2: If the problem is mostly reasoning, you can solve it by yourself directly.
Case 3: If the problem cannot be handled in the above two ways, please follow this process:
1. Solve the problem step by step (do not over-divide the steps).
2. Take out any queries that can be asked through Python (for example, any calculations that can be calculated).
3. Wait for me to give the results.
4. Continue if you think the result is correct. If the result is invalid or unexpected, please correct your query or reasoning.
After you get the answer, put the answer in [] and reply TERMINATE.
```

Problem (GPT-4o-mini)

```
problem: given a user query and a product retrieved for this query, the goal of this task is to classify each product as match or mismatch for the query.
user query: {{query}}
goods description: {{description}}
rules:
- match: the goods is relevant for the query, and satisfies all the query specifications (e.g., water bottle matching all attributes of a query "plastic water bottle 24oz", such as material and size)
- mismatch:
1. the goods is somewhat relevant: it fails to fulfill some aspects of the query but the item can be used as a functional substitute (e.g., fleece for a "sweater" query)
2. the goods does not fulfill the query, but could be used in combination with an exact item (e.g., track pants for a "running shoe" query)
3. the goods is irrelevant, or it fails to fulfill a central aspect of the query.

please return ["Yes"] if the goods description match the query, and ["No"] if they do not match.
```

A.3 Prompt for CoT Agent

System (GPT-4o-mini)

```
You are a CoT reasoner who can solve problems through a step-by-step thinking approach.Return 'TERMINATE' when the task is done.
```

User (GPT-4o-mini)

```
Query: {{query}}
Product: {{passage}}
Task: Determine if the query matches the product using reasoning steps.
Return 1 for match, 0 for mismatch.
Step 1: Understand the Query and Product
Identify the user's intent and key product features.
Step 2: Extract Key Terms
Extract relevant keywords, synonyms, and domain-specific terms.
Step 3: Apply Chain-of-Thought Reasoning
Analyze relationships between query terms and product attributes.
Step 4: Generate Conclusion
FINAL RESULT: [ ]
```

A.4 Prompt for Discussion Agent

Discussion Task (GPT-4o)

```
TASK: Determine if the query adequately describes the product's core features.
QUERY: {{query}}
PRODUCT: {{passage}}
INSTRUCTIONS:
1. Analyze the relationship between query and product
2. Discuss from different perspectives and allow semantic variations (e.g., synonyms, different phrasing)
3. Consider logical equivalences (e.g., 'waterproof' vs 'can be used underwater')
4. Reach a consensus considering comprehensive coverage
5. Check scope inclusion: if the product's features fully cover the query requirements, it is match
6. Provide a binary output:
- Return 1 if query matches product
- Return 0 if query does not match product
The final generated content can only be 0 or 1 and do not generate anything else.
```

B Additional Experiments

Impact of Positive-Negative Ratios. Table 4 examines the influence of different positive-to-negative sample ratios on model performance. Optimal performance is observed when the ratio is balanced within a specific range, whereas an excessive number of negative samples adversely affects model effectiveness. Notably, a 10:9 ratio yields the highest Recall@10 (0.5353) on Shopping Queries, whereas an imbalanced 1:2 ratio leads to reduced NDCG scores.

Table 4: Impact of positive-negative sample ratios on performance.

Ratio	Shopping			arXiv		
	R@10	P@10	N@10	R@10	P@10	N@10
0	0.5302	0.3894	0.5937	0.6104	0.0611	0.4321
10:2	0.5242	0.3882	0.5961	0.6237	0.0625	0.4193
10:3	0.5255	0.3870	0.5920	0.6171	0.0618	0.4153
10:4	0.5232	0.3870	0.5906	0.6171	0.0618	0.4192
10:5	0.5272	0.3882	0.5927	0.6179	0.0619	0.4205
10:9	**0.5353**	**0.3966**	**0.5957**	**0.6254**	**0.0626**	0.4271
1:1	0.5213	0.3894	0.5911	0.6212	0.0622	**0.4335**
1:2	0.5265	0.3869	0.5942	0.6112	0.0612	0.4193

Effectiveness of Generated Data under In-Batch Negatives Training. Figure 4compares BGE-M3 performance trained with varying quantities of original and generated data using the in-batch negatives strategy. Across all scales, models trained on generated data consistently outperform those using original data. For instance, with 500 samples, generated data achieves a Recall@10 of 0.5555 compared to 0.5365 for original data. This advantage persists even as data volume increases, indicating greater robustness and generalization. Additionally, performance with original data slightly declines as quantity increases, suggesting redundancy or domain saturation. These results highlight the effectiveness of high-quality synthetic data in retrieval tasks under contrastive training settings.

Effectiveness of Greedy vote. table 5 compares majority vote with the greedy vote strategy on the Shopping Queries dataset, the default groups are three, the first row is a raw majority vote strategy, and the second row is the greedy vote strategy, the greedy vote strategy shows higher performance than the majority vote with the same configuration setting, because each result is based on the group discussion, if one agent insist on its opinion, other agents may follow this agent's opinion, which will lead to error propagation and generate the wrong label.

Effectiveness of base LLMs as shown in the table 6, we have tested the results from three open-source models (different model versions): Llama 3.2 3B (Meta, 2024), Llama 3.1 8B, Llama 3.3 70B (Grattafiori et al., 2024). Llama 3.2 3B model has poor instruction following ability; the Code Agent's code writing contains many syntax errors, which cause ContrastGen to crash. Llama 3.3 70B has an advantage in recall compared to Llama 3.1 8B, but is slightly worse in NDCG. Therefore, ContrastGen requires models with good instruction following, coding integrated reasoning, and multi role chatting, otherwise, the performance will drop rapidly. what's more, ContrastGen performs well on GPT-4o-mini, because GPT performs good in instruction following and coding. In total, ContrastGen performs well on LLMs with relative good code integrated reasoning and CoT reasoning ability.

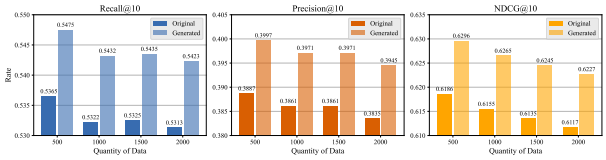


Figure 4: Performance comparison of the BGE-M3 model with original and generated data of different scales under In-Batch Negatives training.

PELM	LLM	With Cot	R@10	P@10	N@10
BGE-M3	GPT-4o-mini	no	0.5336	0.3792	0.5986
BGE-M3	GPT-4o-mini	yes	**0.5391**	**0.3857**	**0.6089**

Table 5: Comparison results for majority vote and greedy vote

PELM	query gen	discussion	R@10	P@10	N@10
BGE-M3	Llama3.2 3B	GPT-4o-mini	0.5303	0.3792	0.5995
BGE-M3	Llama3.1 8B	GPT-4o-mini	0.5338	0.3844	0.6048
BGE-M3	Llama3.3 70B	GPT-4o-mini	0.5349	0.3831	0.6010
BGE-M3	GPT-4o-mini	GPT-4o-mini	**0.5565**	**0.4104**	**0.6362**

Table 6: Comparison results for different model versions

Table 7: Instance of ContrastGen-enhanced data

Passage	Type	Query	Label
Distressed Baseball Cap - Mom Life (Black) Vintage style; Washed & distressed; Low profile crown. Unconstructed style gives off a "dad hat" vibe. Suitable for all seasons. Features: adjustable closure, unstructured crown, all-day comfort.	Original	black baseball cap	1
	Easy	vintage unstructured dad hat for moms	1
	Hard	unstructured relaxed fit black cap for all seasons	1
Mellanni Queen Sheet Set - Hotel Luxury 1800 Bedding. Extra soft cooling sheets & pillowcases; deep pocket (up to 16 inch mattress). Wrinkle, fade, stain resistant (4 Piece, Queen, White).	Original	6 quart crockpot	0
	Easy	cool beds for boys	0
	Hard	soft queen blankets for winter warmth	0

Case study. Table 7 presents two examples of the ContrastGen-enhanced data, including the original query, the generated easy and hard queries, and their corresponding matching labels with the passage.The first example illustrates the process for generating a hard positive sample: the original positive query "black baseball cap" is gradually elab-

orated through ContrastGen's iterative generation, resulting in the hard positive query "unstructured relaxed fit black cap for all seasons". While the second example shows the process for generating a hard negative sample: the original negative query "6 quart crockpot" is similarly refined through iterative generation, producing the hard negative query "soft queen blankets for winter warmth". This table provides an intuitive illustration of the diversity and complexity of the generated data, further validating the effectiveness of ContrastGen in producing high-quality hard samples.