Bridging the Semantic-Collaborative Gap

Unified Item Quantization for LLM-based Generative Recommendation

CS5000: Thesis Project

Bohong Lu

Delft University of Technology

TU Delft

Bridging the Semantic-Collaborative Gap

Unified Item Quantization for LLM-based Generative Recommendation

by

Bohong Lu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday May 28, 2026.

Student number: 6205860
Project duration: October 3, 2025 – May 28, 2026
Thesis committee:

Alan Hanjalic	Chair, Full Professor
Stephanie Tan	Core Member, Assistant Professor
Masoud Mansoury	Core Member, Assistant Professor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This thesis, *Bridging the Semantic-Collaborative Gap: Unified Item Quantization for LLM-based Generative Recommendation*, represents the culmination of my Master of Science degree at Delft University of Technology. The research and engineering project detailed within these pages was conducted between October 2025 and May 2026. The project has been an intellectually demanding yet deeply rewarding experience, allowing me to explore the fascinating intersection of large language models and traditional collaborative filtering.

I would like to extend my deepest gratitude to my daily advisor, Dr. Masoud Mansoury, and to Huy Son Nguyen, a PhD student at TU Delft. Their invaluable guidance, continuous encouragement, and sharp technical insights during our weekly meetings were the backbone of this project. Whether we were debugging complex model architectures or brainstorming the best way to frame our methodology, their reliable support was always present.

Their mentorship was particularly instrumental in the later stages of this journey, where we worked closely together to distill the core findings of this comprehensive manuscript into a condensed 10-page full research paper. I am incredibly proud to note that this paper has been submitted to the `35th ACM International Conference on Information and Knowledge Management (CIKM 2026)` and is currently under peer review.

I am equally grateful to the members of my thesis committee, Prof. Dr. Alan Hanjalic and Dr. Stephanie Tan, for their valuable expertise, time, and dedication in evaluating this work and serving on my defense committee. Their constructive feedback has been vital in shaping the final quality of this manuscript.

Finally, I want to express my sincere thanks to my family and friends for their unwavering patience, understanding, and emotional support during the countless hours of research, coding, and writing over the past several months.

Bohong Lu
Delft, May 2026

Abstract

Large Language Model (LLM)-based generative recommendation reformulates item retrieval as an autoregressive sequence generation problem, representing items through discrete semantic identifiers (SIDs) constructed via the vector quantization of item embeddings. However, a critical yet underexplored limitation of existing tokenization methods is the *semantic-collaborative gap*. SIDs derived purely from item content fail to capture latent user preference patterns encoded in historical interaction data, whereas purely collaborative identifiers lack semantic grounding and generalize poorly to sparse or cold-start scenarios.

To bridge this gap, we propose the **Unified Q-Former (UQF)**, a novel pre-quantization fusion framework designed to explicitly integrate semantic and collaborative signals into a unified item representation before discretization. Inspired by the query-based multimodal alignment of BLIP-2, UQF employs a set of learnable queries, parallel cross-attention over pre-trained item text embeddings and graph-based collaborative embeddings (via LightGCN), and adaptive gated fusion to dynamically extract complementary information from both modalities. To ensure robustness and structure preservation, the framework is optimized using a hybrid contrastive learning objective—incorporating both structural and semantic neighbors—coupled with asymmetric modality dropout.

The resulting unified representations are quantized into discrete SIDs via residual vector quantization (RQ-VAE) and utilized as target generation tokens for a downstream LLM recommender. Extensive experiments on two real-world Amazon Review datasets (Office Products and Musical Instruments) demonstrate that UQF consistently improves state-of-the-art LC-Rec and TIGER-style generative recommendation backbones. Our framework outperforms strong traditional, sequential, and recent unified generative baselines, yielding highly interpretable, hierarchical SID structures with significantly improved semantic and collaborative consistency.

Contents

1 Introduction

1.1. Background

Recommender Systems have been ubiquitous in modern digital platforms, generating personalized recommendations for users in online streaming websites, e-commerce giants and social media [12, 8, 3]. Since the early 1990s when collaborative filtering was proposed and formalized [38, 40, 39], traditional statistics-based models dominated the field for a decade. With the development of deep learning, deep neural networks enabled automatic feature interaction from both user and item sides, making these deep models mainstream in the 2010s [1, 9, 11]. These systems operate on a cascade "retrieval, ranking, re-ranking" paradigm that retrieves thousands of candidate items in the retrieval stage, and scores each user-item pair after retrieval to get top-k items, which are then passed through a listwise re-ranker for the final output [3, 27].

However, a paradigm shift is currently underway, driven by the emergence of Large Language Models (LLMs). A transition is being witnessed from discriminative ranking to **LLM-based Generative Recommendation (LLM-GR)**, as shown in Figure 1.1 [14, 49, 25]. In this new paradigm, recommendation is re-framed as a language generation task: instead of scoring a set of candidates, the model directly generates item identifiers as a sequence of tokens [7, 4]. This approach succeeds in leveraging the world knowledge, reasoning abilities, and potential scaling laws of LLMs, making an end-to-end architecture that unifies retrieval, ranking, and re-ranking into a single stage possible [6, 53, 54].

A critical bottleneck in realizing the potential of LLM-GR lies in **Item Representation** [16]. The way each item is represented as a token sequence corresponds to LLM inference effectiveness and item indexing quality. Early attempts utilized Numeric IDs, mapping items to different integers that can be initialized randomly or sequentially. While these guarantee distinctiveness, they lack semantic meaning, such that the similarity between items is not reflected in their numeric IDs. Conversely, representing items via natural language text (e.g., titles or descriptions) captures rich semantics but suffers from variable token length, ambiguous indexing, and hallucination risks, making it unsuitable for item tokenization [25].

To address these limitations, recent research has converged on **Quantized Identifiers (Quantized IDs)**. Methods like TIGER [35] and LC-Rec [52] use vector quantization techniques like Residual-Quantized Variational Autoencoder (RQ-VAE) [21] to map item embeddings into discrete codewords from a trained codebook. These methods generally fall into two categories:

- **Semantic IDs (SIDs):** Constructed from content embeddings (e.g., BERT representations of item titles and descriptions or multi-modal information).
- **Collaborative IDs (CIDs):** Constructed from interaction data (e.g., graph embeddings).

While promising, existing quantization-based item indexing methods still suffer from a fundamental limitation. Most prior studies construct item identifiers primarily from a single modality, especially semantic content embeddings, and therefore do not explicitly address the information loss of collaborative signal.

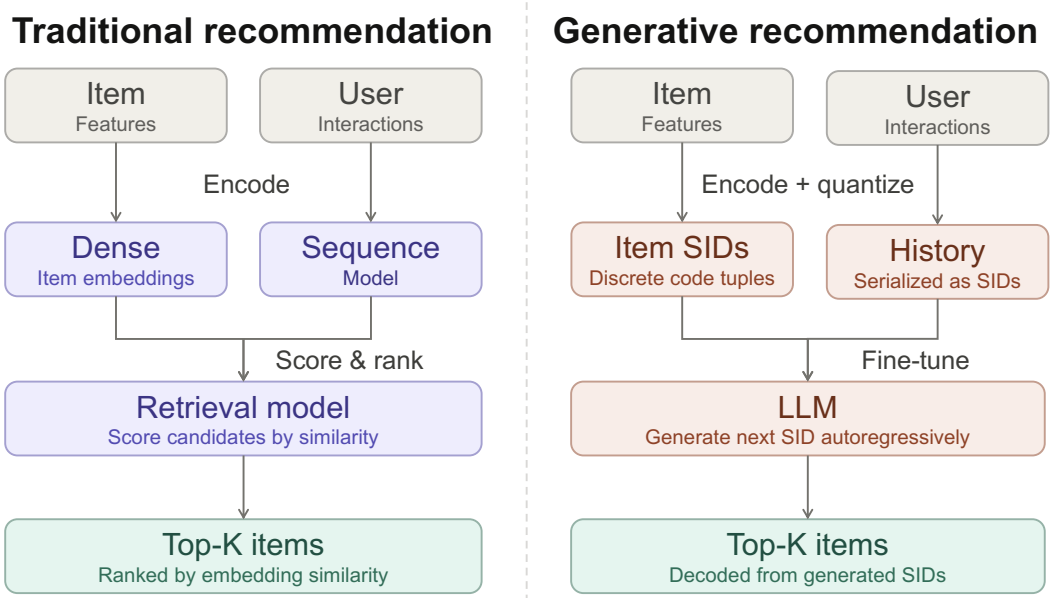


Figure 1.1: The paradigm shift of recommendation

Although a few recent attempts, such as LETTER [45], have begun to incorporate collaborative information into the quantization process, they mainly treat collaborative signals as an auxiliary supervision or regularization objective during quantization, rather than learning a unified item representation.

1.2. Research Gap

As a result, a clear **semantic-collaborative gap** remains in current LLM-based generative recommendation pipelines. SIDs are advantageous in cold-start scenarios because they rely on item semantics rather than historical interactions. However, they fail to capture collaborative patterns that reflect latent user preferences, co-consumption structure, and behavioral proximity between items. In contrast, CIDs constructed from interaction signals are more effective at encoding preference patterns, but they are inherently dependent on historical behavior data and often generalize poorly to cold-start or sparsely interacted items. An illustration of this semantic-collaborative misalignment is shown in Figure 1.2.

More importantly, existing attempts to bridge the two signals remain limited in two aspects. First, they generally lack an explicit **representation-level fusion** mechanism that directly integrates semantic and collaborative information into a unified space before quantization. Consequently, the representation being quantized is still dominated by the semantic modality, while collaborative information only affects learning indirectly. Second, they lack a **fine-grained and adaptive interaction** mechanism to model how semantic cues and collaborative patterns should complement each other during item representation learning. Therefore, the resulting discrete identifiers may still be suboptimal in simultaneously preserving semantic meaning and collaborative relevance.

This reveals an important **research gap**: current studies have not sufficiently explored how to perform unified semantic-collaborative representation learning for item quantization in LLM-based generative recommendation, such that the learned discrete item identifiers can better support downstream recommendation generation.

1.3. Proposed Solution

To address the gap, this paper proposes a novel framework for **Unified Item Quantization** for LLM-based Generative Recommendation. Unlike prior methods that mainly rely on a single source of item signal during token construction, the proposed framework fuses collaborative and semantic information through a **Unified-Q-Former (UQF)**. Inspired by the query-based multimodal alignment paradigm in BLIP-2 [24], where a Q-Former is used to connect visual representations with language space,

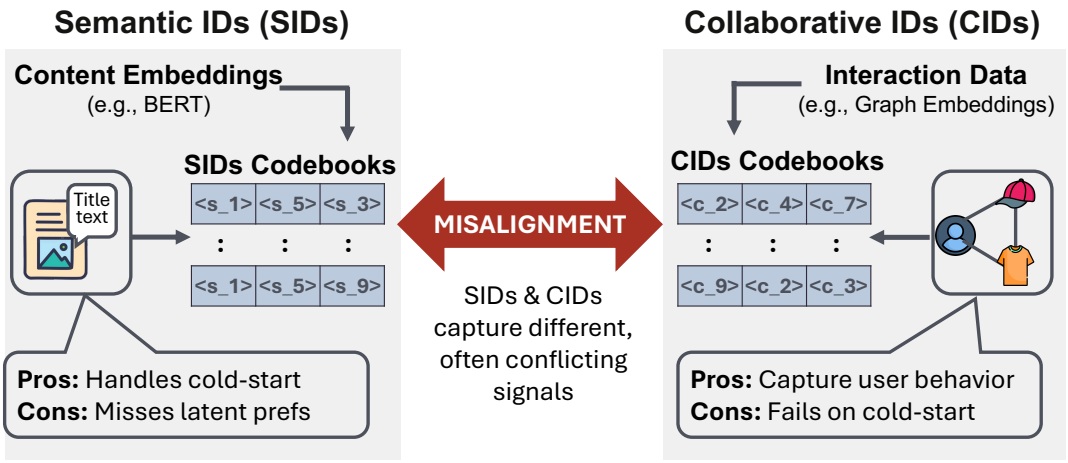


Figure 1.2: Semantic-Collaborative Misalignment between SIDs and CIDs

UQF transfers this design intuition to unify semantic and collaborative item signals in recommendation. Specifically, UQF employs learnable queries, cross-attention, and adaptive gating to model semantic-collaborative interactions. The resulting unified item embeddings are then quantized into discrete Semantic IDs for subsequent generative recommendation tasks. In this way, the learned identifiers are expected to better align item semantics with user preference structure, thereby improving the effectiveness of downstream generative recommendation.

1.4. Research Questions

Based on the proposed UQF-based unified item quantization framework, this work investigates the following research questions:

RQ1 *Does UQF-based item tokenization improve generative recommendation performance compared to existing approaches?* UQF is compared against traditional, sequential, and state-of-the-art generative baselines to validate overall effectiveness.

RQ2 *How should semantic and collaborative information be balanced within UQF, and is the cross-modal fusion mechanism necessary?* Sensitivity analysis is conducted over the contrastive loss weights and ablation studies over input representation strategies to understand the interaction between the two modalities.

RQ3 *How does the framework scale with increasing LLM capacity, and what is the quality of the learned Semantic IDs?* Scaling behavior is examined across different LLM sizes and analyze the semantic and collaborative coherence of items sharing the same SID prefix.

2

Related Work

2.1. General Recommendation Paradigms

2.1.1. Traditional Methods

Early recommender systems were primarily built upon collaborative filtering and latent factor modeling. Memory-based Collaborative Filtering (CF) methods estimate user preferences by aggregating signals from similar users or items [38, 39]. Subsequently, Matrix Factorization (MF) projects users and items into a shared latent space, enabling recommendation through learned low-dimensional representations of global preference patterns [20]. In parallel, feature-based methods such as Factorization Machines (FM) [36] and later Field-aware Factorization Machines (FFM) [17] were proposed to explicitly model second-order feature interactions in sparse recommendation scenarios. These methods laid the foundation for modern recommendation systems by establishing effective paradigms for preference modeling.

2.1.2. Deep Learning Methods

With the rise of deep learning, recommender systems increasingly adopted neural architectures to model complex and non-linear user-item interactions. Neural Collaborative Filtering (NCF) [11] replaces the inner product in MF with multi-layer perceptrons to improve representation capacity. Hybrid architectures such as Wide & Deep [1] and DeepFM [9] further combine the strengths of memorization and generalization, enabling high-order feature interaction learning in large-scale recommendation tasks. Despite their strong empirical performance, these methods are typically deployed within multi-stage recommendation pipelines, where candidate items are first retrieved, then ranked, and finally re-ranked for output. While effective in industrial practice, such pipelines remain systemically complex and fragmented, motivating the search for more unified recommendation paradigms.

2.2. From Sequential to Generative Recommendation

2.2.1. Sequential Recommendation

To better capture the dynamic nature of user interests, sequential recommendation methods were introduced to model dependencies in historical interaction sequences. Early studies primarily relied on recurrent and convolutional architectures. For example, GRU4Rec [13] employs recurrent neural networks for session-based recommendation, while NARM [23] and STAMP [26] further enhance session modeling by using attention mechanisms to capture users' main intents and short-term preferences. In parallel, Caser [44] utilizes convolutional neural networks to model local sequential patterns, and NextItNet [50] extends this line of work with dilated residual convolutions for capturing longer-range dependencies.

Beyond recurrent and convolutional models, graph-based methods were also introduced to capture higher-order connectivity patterns in recommendation. For sequential and session-based recommendation, SR-GNN [47] represents session sequences as directed graphs and leverages graph neural networks for session-based recommendation, while GC-SAN [48] combines graph neural networks

with self-attention to exploit both local and global dependencies. More broadly, graph collaborative filtering methods such as LightGCN [10] demonstrate the effectiveness of simplified graph convolution on the user–item interaction graph, further highlighting the value of graph structure for representation learning in recommendation. Later, Transformer-based architectures further improved sequential modeling capability. SASRec [18] applies self-attention to model long-range dependencies in user behavior sequences, TiSASRec [22] explicitly incorporates temporal interval information into self-attention, and BERT4Rec [43] adopts bidirectional Cloze-style training to capture contextual information from both directions.

Although these methods substantially advance user sequence modeling, they still remain within the paradigm of sequential recommendation rather than generative recommendation. In general, they operate in a latent embedding space, where recommendation is formulated as predicting the next item or its representation from historical interactions. As such, they do not model recommendation as an explicit token generation process, nor are their item representations naturally aligned with the semantic space of large language models (LLMs). Consequently, they cannot directly leverage the language understanding, generation, and reasoning capabilities of LLMs.

2.2.2. Generative Recommendation

The transition toward generative recommendation began with P5 [7], which reformulates recommendation as a language-based prediction task under a unified prompt learning framework. By converting user behaviors and recommendation objectives into textual prompts, P5 demonstrates that recommendation can be cast as a generation problem rather than a conventional scoring problem. For item representation, P5 adopts Numeric IDs and introduces a whole-word embedding strategy to preserve item identity consistency when numeric IDs are tokenized into multiple sub-words.

Building on this line of research, Hua et al. [16] systematically investigate item indexing strategies for generative recommendation and distinguish between Collaborative Indexing and Semantic Indexing. Their study highlights that item representation is a central design choice in LLM-based generative recommendation, and lays the groundwork for subsequent studies on Collaborative IDs, Semantic IDs, and unified item indexing methods.

2.3. Item Representation in Generative Recommendation

Item representation is a central problem in LLM-based generative recommendation, since the model no longer predicts items through conventional scoring over a candidate set, but instead needs to generate item identifiers as discrete token sequences. In this context, the way an item is converted into a code directly affects the quality of sequence modeling, the alignment between item space and language space, and the effectiveness of downstream recommendation. In this work, existing methods are grouped according to the primary information source used to construct item codes, namely **Collaborative ID (CID)-based methods**, **Semantic ID (SID)-based methods**, and **unified methods** that jointly encode semantic and collaborative information.

2.3.1. CID-based Methods

CID-based methods construct item identifiers mainly from user-item interaction signals, with the goal of encoding collaborative filtering knowledge into discrete tokens. GPTRec [31] is an early representative work in this direction. Instead of treating each item as an atomic ID, GPTRec first performs truncated Singular Value Decomposition (SVD) on the user-item interaction matrix to obtain item latent embeddings. It then normalizes each embedding dimension, injects small Gaussian noise, and discretizes each dimension into integer bins, where each quantized value serves as a token. By applying offsets across different dimensions, GPTRec maps each item into a sequence of collaborative latent tokens. Based on these tokenized item representations, it employs a GPT-style decoder to model next-item generation. The significance of GPTRec lies in showing that interaction-based item representations can be discretized into a form suitable for autoregressive generation. However, its item codes are entirely built from collaborative signals and do not explicitly incorporate semantic content.

A more systematic study of indexing strategies is provided by Hua et al. [16], which investigates how to design LLM-compatible item IDs for recommendation foundation models. The paper proposes several indexing schemes, including Sequential Indexing, Collaborative Indexing, Semantic Indexing, and

Hybrid Indexing. In its collaborative indexing design (P5-CID), the authors first construct an item co-occurrence graph from user interaction sequences, and then apply recursive spectral clustering to build a hierarchical clustering tree. An item's collaborative ID is represented by the cluster path from the root to the leaf node, so that items with stronger co-occurrence patterns can share longer code prefixes. Within the P5 framework, such hierarchical collaborative IDs are used as item representations for generative recommendation, and constrained decoding with a prefix tree is further adopted to ensure that generated token sequences correspond to valid item identifiers. This work is important because it explicitly formulates item indexing as a design problem in generative recommendation. Nevertheless, its hybrid strategy mainly concatenates different indexing spaces rather than learning a deeply unified representation.

TokenRec [32] further advances CID-based item representation by introducing a learnable tokenization framework. It first learns user and item collaborative representations through graph-based collaborative filtering, and then performs *masked vector quantization* on these embeddings. Specifically, TokenRec randomly masks the collaborative embeddings, passes them through a K-way encoder, and maps them into multiple sub-codebooks, thereby obtaining several discrete tokens for each item. Unlike spectral clustering-based indexing, TokenRec learns item tokenization in a representation learning manner, and the same design can also be applied to user tokenization. In the downstream recommendation stage, the model does not directly generate an item code sequence as the final output; instead, it produces a generative next-item representation, which is then matched against item representations through similarity-based retrieval. This makes TokenRec more retrieval-oriented than pure autoregressive code generation. Although it improves the learnability and efficiency of collaborative tokenization, the item representation is still derived solely from collaborative information, without semantic content modeling.

2.3.2. SID-based Methods

SID-based methods construct item identifiers mainly from semantic content, such as item titles, descriptions, attributes, or multimodal side information. A seminal work is TIGER [35], which introduces the concept of *Semantic IDs*. TIGER first encodes item textual content into dense semantic embeddings using pre-trained language encoders, and then quantizes these embeddings into ordered code tuples through Residual Quantized Variational AutoEncoder (RQ-VAE) [21]. In this residual quantization process, early codebooks capture coarse semantic structure, while later codebooks progressively quantize the remaining residual information, yielding hierarchical semantic codes for each item. These code tuples are then used as target item identifiers in a sequence-to-sequence generative recommendation framework. In essence, TIGER establishes the standard pipeline of *semantic embedding* $\rightarrow$ *residual quantization* $\rightarrow$ *semantic ID generation*, and demonstrates that semantically meaningful item codes are effective for generative recommendation. However, the learned codes are mainly semantics-oriented and do not explicitly encode collaborative preference information.

LC-Rec [52] extends the SID paradigm by paying more attention to the alignment between learned item indices and the language space of LLMs. Similar to TIGER, LC-Rec first obtains text-based semantic embeddings from item content and then applies tree-structured residual vector quantization to construct hierarchical item indices. A key challenge identified by LC-Rec is that standard vector quantization may cause multiple items to collapse into the same index. To address this issue, it proposes *Uniform Semantic Mapping*, which redistributes collided items more evenly at the leaf level through a Sinkhorn-Knopp [41, 42] based assignment mechanism. Beyond item index construction, LC-Rec further introduces several alignment tuning tasks, including sequential item prediction and recommendation-oriented alignment objectives, so that the LLM can better understand the recommendation semantics carried by the newly introduced item indices. Compared with TIGER, LC-Rec therefore not only studies how to construct semantic IDs, but also how to improve the compatibility between such IDs and LLM-based generation. Nevertheless, the item indices themselves remain primarily semantic in nature.

OneRec [6] pushes SID-based generative recommendation toward a more industrial and end-to-end setting. Unlike prior works that mainly focus on next-item prediction, OneRec aims to unify retrieval and ranking within a single generative framework and further incorporates preference alignment. On the item representation side, it uses multimodal item embeddings that are aligned with real user behavior distributions, and then constructs hierarchical semantic IDs through a multi-level balanced quantization strategy based on residual k-means and balanced k-means. This design explicitly addresses the code

imbalance issue that often arises in large-scale quantization. In the recommendation stage, OneRec generates an entire recommendation session rather than a single item, and then improves generation quality through iterative preference alignment and DPO-style optimization [33]. Although OneRec significantly strengthens the practical scalability of semantic-ID-based generative recommendation, its item token construction still mainly relies on semantic representations, rather than explicit semantic-collaborative fusion before quantization.

2.3.3. Unified Methods

Recognizing the limitations of single-modality item codes, recent studies have started to explore how semantic and collaborative signals can be jointly incorporated into item representation. LETTER [45] is an important representative of this direction. Its core idea can be summarized as learning item codes with a *semantic backbone, collaborative regularization, and distribution balancing*. Concretely, LETTER still adopts an RQ-VAE style hierarchical quantization framework, so that item identifiers naturally preserve coarse-to-fine semantic structure. On top of this semantic tokenizer, it introduces *collaborative regularization* to align quantized item embeddings with collaborative filtering embeddings, thereby encouraging item codes to reflect not only semantic similarity but also collaborative proximity. In addition, LETTER employs *diversity regularization* to alleviate code assignment bias and avoid over-concentration on a small subset of codes. After tokenization, the generated item identifiers are used in downstream generative recommendation with ranking-guided generation loss and constrained decoding. LETTER demonstrates that item codes can be trained to simultaneously carry semantic and collaborative information. However, from the perspective of representation learning, collaborative signals are still mainly injected as auxiliary supervision during semantic quantization, rather than being explicitly fused with semantic embeddings into a unified representation before quantization.

EAGER [46] explores another way of integrating the two modalities through a two-stream generative architecture. Instead of constructing a single unified code, EAGER assigns each item two different code sequences: a behavior code derived from collaborative information and a semantic code derived from content information. Both streams are quantized through hierarchical clustering, and recommendation is performed with a shared encoder and two separate autoregressive decoders. To enhance collaboration between the two streams, EAGER further introduces a global contrastive task for summarizing user preference information and a semantic-guided transfer task that lets semantic information guide behavior code learning. During inference, the outputs from the two decoders are combined through confidence-based fusion. The advantage of EAGER is that it preserves both semantic and behavioral signals explicitly and enables interaction between them during generation. However, its integration is realized through dual-code parallel generation and late fusion, rather than through a truly unified item representation and a single quantization space.

Overall, the evolution of item representation in generative recommendation follows a clear trajectory. Early methods such as GPTRec [31], the collaborative indexing design in [16], and TokenRec [32] mainly study how to transform interaction-derived representations into discrete collaborative codes. Later SID-based methods such as TIGER [35], LC-Rec [52], and OneRec [6] shift the focus to semantic or multimodal content, making item codes more suitable for cold-start scenarios and more compatible with the language space of LLMs. More recent unified methods, including LETTER [45] and EAGER [46], attempt to incorporate both semantic and collaborative signals into item codes. However, existing approaches still differ substantially in how the two modalities are integrated, ranging from auxiliary collaborative regularization, to dual-stream generation, to alignment-based unified embeddings. This motivates further research on more explicit and adaptive semantic-collaborative fusion for item quantization in generative recommendation.

3

Technical Background

This chapter introduces the technical foundations underlying the proposed framework. Since the core contribution of this work lies in the design of the **Unified Q-Former (UQF)** for semantic–collaborative fusion before quantization, several prerequisite components are first reviewed here, including the general paradigm of LLM-based generative recommendation, graph-based collaborative representation learning, residual vector quantization for item tokenization, query-based cross-modal fusion, prompt-based supervised fine-tuning, and constrained decoding for valid item generation.

3.1. LLM-based Generative Recommendation

Traditional recommender systems typically follow a multi-stage pipeline, where candidate items are first retrieved, then ranked, and finally re-ranked before presentation. In contrast, **generative recommendation** reformulates recommendation as a sequence generation problem, where the model directly predicts the identifier of the next item or the next list of items in an autoregressive or sequence-to-sequence manner [7, 35, 52, 6]. This paradigm is particularly compatible with LLMs, since user interaction history can be expressed as a token sequence, and recommendation can be cast as a conditional generation task.

A central issue in generative recommendation is that items must be represented as *discrete symbols or tokens* in LLM's vocabulary. Numeric IDs are straightforward but do not preserve similarity structure, while natural-language titles or descriptions are semantically meaningful but are often ambiguous, long, and difficult to constrain during generation. Therefore, recent studies have increasingly focused on learning structured discrete item identifiers through quantization or indexing [16, 35, 52]. Such identifiers aim to satisfy three desirable properties: (1) they should be *discrete* so that they can be generated token by token, (2) they should preserve useful item relationships, and (3) they should map uniquely back to real items.

A representative pipeline is the **TIGER-style semantic ID pipeline** [35], illustrated in Figure 3.1. It generally consists of two stages. First, item content is encoded into dense embeddings and quantized into multi-level discrete codes, producing a semantic identifier (SID) for each item. Second, user histories are rewritten as sequences of these SIDs, and a generative model is fine-tuned to predict the SID of the next item. Let $\mathcal{I}$ denote the item set, and let $\mathbf{u}_i \in \mathbb{R}^{d_u}$ denote the dense representation of item $i \in \mathcal{I}$ of dimension d_u. RQ-VAE transforms $\mathbf{u}_i$ into a D-level code tuple

$$\mathrm{SID}_i = \left(k_i^{(1)}, k_i^{(2)}, \ldots, k_i^{(D)}\right), \tag{3.1}$$

where each $k_i^{(d)} \in \{0, \ldots, N_c - 1\}$ is the discrete code index selected from the level-d codebook $\mathcal{C}^{(d)} = \{\mathbf{c}_0^{(d)}, \ldots, \mathbf{c}_{N_c-1}^{(d)}\}$. Further details of this vector quantization process using RQ-VAE is illustrated in section 3.3. The recommendation task then becomes:

$$P(\mathrm{SID}_{i_{t+1}} \mid \mathrm{SID}_{i_1}, \mathrm{SID}_{i_2}, \ldots, \mathrm{SID}_{i_t}, \mathbf{x}), \tag{3.2}$$

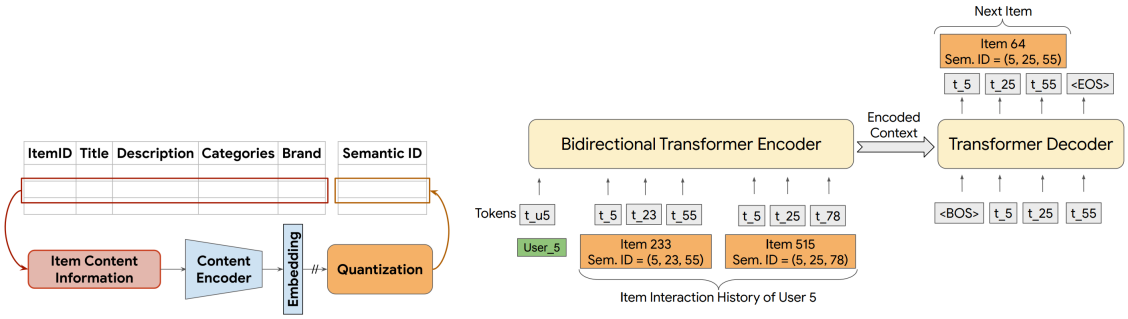


Figure 3.1: TIGER-style semantic ID pipeline. Items are first encoded into semantic embeddings, then quantized into multi-level Semantic IDs, which are used as target sequences for generative recommendation. Reproduced from [35].

where $\mathbf{x}$ denotes the prompt or instruction context. This reformulation bridges item recommendation and text generation.

Later studies such as LC-Rec [52] further show that simply constructing item IDs is not sufficient. The LLM must also learn how these newly introduced item SIDs relate to natural language and to recommendation semantics. This motivates prompt-based alignment tuning, which will be discussed in section 3.5.

3.2. Graph-based Collaborative Representation Learning

Collaborative filtering signals are commonly modeled through the user–item interaction graph, where edges indicate observed interactions such as clicks, purchases, or ratings. A widely used graph-based collaborative filtering model is **LightGCN** [10], which simplifies graph convolutional recommendation by removing nonlinear transformations and feature projection matrices, retaining only neighborhood aggregation.

Let N_u and N_v denote the numbers of users and items, respectively. The user–item bipartite graph can be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{(N_u+N_v)\times(N_u+N_v)}$. Its normalized form is

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \tag{3.3}$$

where $\mathbf{D}$ is the degree matrix. Starting from learnable initial embeddings $\mathbf{E}^{(0)}$, LightGCN performs layer-wise propagation as

$$\mathbf{E}^{(k)} = \tilde{\mathbf{A}}\mathbf{E}^{(k-1)}, \quad k = 1, 2, \ldots, K. \tag{3.4}$$

The final representation is often obtained by averaging or combining embeddings from multiple propagation depths:

$$\mathbf{e}_i = \frac{1}{K+1}\sum_{k=0}^{K} \mathbf{e}_i^{(k)}. \tag{3.5}$$

The model is typically trained with a pairwise ranking objective such as Bayesian Personalized Ranking (BPR) [37]:

$$\mathcal{L}_{\text{BPR}} = -\sum_{(u,i^+,i^-)} \log \sigma\left(\mathbf{e}_u^\top \mathbf{e}_{i^+} - \mathbf{e}_u^\top \mathbf{e}_{i^-}\right) + \lambda\|\Theta\|_2^2, \tag{3.6}$$

where (u, i^+, i^-) denotes a triplet containing a user, a positive interacted item, and a sampled negative item, and Θ denotes the model parameters.

The importance of LightGCN in this work lies in two points. First, it provides item representations that explicitly encode collaborative preference structure. Second, different graph propagation layers naturally capture different levels of structural information, from local identity to higher-order neighborhood context. These graph-derived representations form the collaborative modality in the proposed unified item representation learning framework.

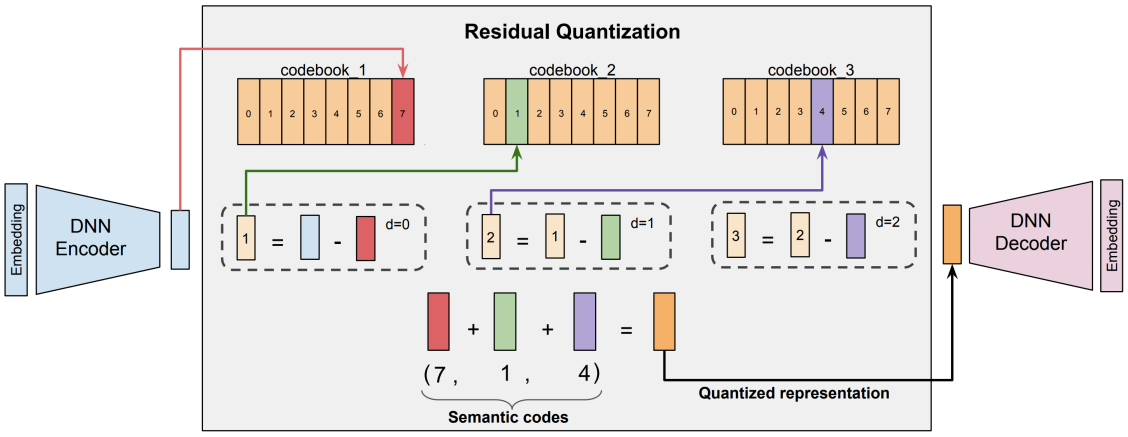


Figure 3.2: Overall architecture of Residual Quantized Variational Autoencoder (RQ-VAE). The latent representation is quantized by multiple codebooks in a residual manner. Reproduced from [35].

3.3. Vector Quantization and Residual Quantized Item IDs

A key requirement in LLM-based generative recommendation is to convert continuous item representations into discrete symbols that can be generated token by token. Let $x_i \in \mathbb{R}^d$ denote the unified embedding of item $i \in \mathcal{I}$. Since large language models operate in a discrete vocabulary space, x_i cannot be used directly as a generation target. This motivates the use of **vector quantization**, which maps a continuous representation into a discrete code index with respect to a learned codebook.

From Single-Step Vector Quantization to Multi-Step Residual Quantization. In standard vector quantization, a continuous vector is approximated by a single codeword selected from a codebook. In the context of item ID construction, the vector to be quantized is typically not the original unified item embedding x_i itself, but its latent representation after encoder projection. Specifically, the encoder maps x_i into

$$\mathbf{z}_i = f_{\text{enc}}(x_i),$$

where $\mathbf{z}_i \in \mathbb{R}^{d_e}$ denotes the latent vector before quantization. Given a learned codebook

$$\mathcal{C} = \{\mathbf{e}_n\}_{n=0}^{N-1},$$

where $\mathbf{e}_n \in \mathbb{R}^{d_e}$ denotes the n-th codeword, the quantizer selects the nearest codeword index:

$$q(\mathbf{z}_i) = \arg\min_{n \in \{0,\ldots,N-1\}} \|\mathbf{z}_i - \mathbf{e}_n\|_2^2 . \tag{3.7}$$

The output of quantization is therefore a discrete index, and the corresponding codeword is used as the quantized approximation of the input vector. This one-step discretization makes continuous representations compatible with autoregressive generation.

However, a single codeword is often not expressive enough to capture the rich semantic structure of an item. In recommendation scenarios, an item may simultaneously contain multiple levels of information, such as coarse category-level semantics, mid-level functional properties, and fine-grained distinguishing details. Compressing all of this information into a single nearest codeword may lead to substantial information loss unless the codebook is made extremely large.

To address this limitation, **residual quantization** extends single-step quantization into a multi-step approximation process. Instead of approximating the latent representation in one shot, the first quantization level captures the most salient semantic component, and each subsequent level quantizes the residual error left by the previous level. In this way, the quantization process proceeds from coarse to fine: earlier levels model dominant semantics, while later levels refine the remaining details. This hierarchical design is especially suitable for item tokenization, since it allows the resulting code sequence to encode multiple semantic levels of an item in a structured way.

RQ-VAE for Residual Quantized Item IDs. Residual Quantized Variational AutoEncoder (RQ-VAE) applies residual quantization within an autoencoder framework. Given the unified item embedding x_i, the encoder first maps it into a latent representation:

$$\mathbf{z}_i = f_{\mathrm{enc}}(x_i), \tag{3.8}$$

where $\mathbf{z}_i \in \mathbb{R}^{d_e}$ is the latent vector before quantization.

Suppose the model uses M quantization levels. For each level $m \in \{1, \ldots, M\}$, there is a dedicated codebook

$$\mathcal{C}^{(m)} = \{\mathbf{e}_n^{(m)}\}_{n=0}^{N-1}, \tag{3.9}$$

where $\mathbf{e}_n^{(m)} \in \mathbb{R}^{d_e}$ denotes the n-th codeword in the level-m codebook. The residual quantization process starts from the initial residual

$$\mathbf{r}_i^{(1)} = \mathbf{z}_i. \tag{3.10}$$

At the first level, the model selects the nearest codeword index from $\mathcal{C}^{(1)}$:

$$c_i^{(1)} = \arg \min_{n \in \{0,\ldots,N-1\}} \left\| \mathbf{r}_i^{(1)} - \mathbf{e}_n^{(1)} \right\|_2^2. \tag{3.11}$$

The selected codeword $\mathbf{e}_{c_i^{(1)}}^{(1)}$ gives the first-stage approximation, and the residual passed to the next level is

$$\mathbf{r}_i^{(2)} = \mathbf{r}_i^{(1)} - \mathbf{e}_{c_i^{(1)}}^{(1)}. \tag{3.12}$$

More generally, at level m, the selected code index is

$$c_i^{(m)} = \arg \min_{n \in \{0,\ldots,N-1\}} \left\| \mathbf{r}_i^{(m)} - \mathbf{e}_n^{(m)} \right\|_2^2, \qquad m = 1, \ldots, M, \tag{3.13}$$

where $c_i^{(m)} \in \{0, \ldots, N-1\}$. The residual is then updated recursively as

$$\mathbf{r}_i^{(m+1)} = \mathbf{r}_i^{(m)} - \mathbf{e}_{c_i^{(m)}}^{(m)}, \qquad m = 1, \ldots, M-1. \tag{3.14}$$

After M levels of residual quantization, the quantized latent representation is reconstructed as the sum of all selected codewords:

$$\hat{\mathbf{z}}_i = \sum_{m=1}^{M} \mathbf{e}_{c_i^{(m)}}^{(m)}. \tag{3.15}$$

The corresponding semantic item ID is the tuple of selected code indices:

$$\mathrm{SID}_i = \left(c_i^{(1)}, c_i^{(2)}, \ldots, c_i^{(M)} \right). \tag{3.16}$$

This formulation makes the semantic meaning of residual quantization explicit. The first code index $c_i^{(1)}$ captures the most dominant semantic pattern of item i, while later indices $c_i^{(2)}, \ldots, c_i^{(M)}$ progressively encode finer-grained information that is not yet explained by earlier levels. Therefore, compared with flat single-code quantization, residual quantization is better able to preserve hierarchical item semantics under a fixed codebook size per level. Figure 3.2 illustrates the overall architecture of RQ-VAE.

Decoding and Reconstruction. Once the quantized latent representation $\hat{\mathbf{z}}_i$ is obtained, it is passed into the decoder to reconstruct the original unified item embedding:

$$\hat{x}_i = f_{\mathrm{dec}}(\hat{\mathbf{z}}_i). \tag{3.17}$$

The encoder, decoder, and codebooks are jointly trained so that the discrete code tuple SID_i preserves as much semantic information from x_i as possible.

Training Objective of RQ-VAE. The training objective of RQ-VAE consists of a reconstruction term and a residual quantization term:

$$\mathcal{L}_{\text{RQ-VAE}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{rq}}. \tag{3.18}$$

The reconstruction loss measures how accurately the decoder recovers the original unified item representation:

$$\mathcal{L}_{\text{recon}} = \|\hat{x}_i - x_i\|_2^2 . \tag{3.19}$$

The residual quantization loss is accumulated over all quantization levels:

$$\mathcal{L}_{\text{rq}} = \sum_{m=1}^{M} \left(\left\| \text{sg}\left[\mathbf{r}_i^{(m)}\right] - \mathbf{e}_{c_i^{(m)}}^{(m)} \right\|_2^2 + \beta \left\| \mathbf{r}_i^{(m)} - \text{sg}\left[\mathbf{e}_{c_i^{(m)}}^{(m)}\right] \right\|_2^2 \right), \tag{3.20}$$

where $\text{sg}[\cdot]$ denotes the stop-gradient operator and β is the commitment coefficient.

The first term

$$\left\| \text{sg}\left[\mathbf{r}_i^{(m)}\right] - \mathbf{e}_{c_i^{(m)}}^{(m)} \right\|_2^2 \tag{3.21}$$

is the **codebook loss**, which updates the selected codeword toward the assigned residual vector. Since the residual is detached, this term does not update the encoder.

The second term

$$\beta \left\| \mathbf{r}_i^{(m)} - \text{sg}\left[\mathbf{e}_{c_i^{(m)}}^{(m)}\right] \right\|_2^2 \tag{3.22}$$

is the **commitment loss**, which updates the encoder so that the latent representation and its intermediate residuals remain close to the discrete code space. Since the codeword is detached here, this term does not update the codebook.

Together, these two terms stabilize training: the codebook loss makes codewords adapt to the data distribution, while the commitment loss encourages the encoder to commit to discrete representations rather than drifting freely in continuous space.

Gradient Propagation and Straight-Through Estimation. A fundamental challenge in vector quantization is that the nearest-neighbor assignment

$$c_i^{(m)} = \arg \min_{n \in \{0,\ldots,N-1\}} \left\| \mathbf{r}_i^{(m)} - \mathbf{e}_n^{(m)} \right\|_2^2 \tag{3.23}$$

is not differentiable. Therefore, gradients from the decoder cannot be propagated directly through the discrete quantization step.

In practice, RQ-VAE uses the **straight-through estimator (STE)** to bypass this problem. The quantized latent is implemented as

$$\tilde{\mathbf{z}}_i = \mathbf{z}_i + \text{sg}[\hat{\mathbf{z}}_i - \mathbf{z}_i] . \tag{3.24}$$

In the forward pass, $\tilde{\mathbf{z}}_i$ is numerically equal to $\hat{\mathbf{z}}_i$, so the decoder receives the quantized latent representation. In the backward pass, however, the gradient of the reconstruction loss is copied directly from $\tilde{\mathbf{z}}_i$ back to $\mathbf{z}_i$, which allows the encoder and decoder to be trained end-to-end despite the non-differentiability of discrete code selection.

Why RQ-VAE is Useful for Semantic Item IDs. For generative recommendation, the main advantage of RQ-VAE is that it converts each item into a structured discrete identifier rather than an arbitrary atomic token. The resulting semantic ID

$$\text{SID}_i = \left(k_i^{(1)}, k_i^{(2)}, \ldots, k_i^{(D)}\right)$$

encodes item information in a hierarchical coarse-to-fine manner. Earlier levels capture broad semantic patterns, and later levels refine the remaining residual semantics. This makes the learned item IDs more semantically meaningful, more compositional, and more suitable for prefix-based generation or constrained decoding than conventional item IDs.

3.4. Query-based Cross-modal Fusion

The proposed UQF module is inspired by the **Q-Former** in BLIP-2 [24]. Q-Former is a lightweight query-based transformer module designed to bridge two modalities, originally visual features and language representations. In BLIP-2, it serves as a trainable bottleneck between a frozen image encoder and a frozen LLM, extracting a fixed number of language-relevant visual features from high-dimensional image representations. Specifically, Q-Former introduces a small set of *learnable query tokens*, which interact with frozen image features through multi-head cross-attention and with text tokens through shared self-attention layers. In this way, the model distills the most relevant information from one modality into a compact latent representation that can be more easily aligned with the other modality.

Formally, let

$$\mathbf{H}^{\mathrm{src}} = \left[\mathbf{h}_1^{\mathrm{src}}, \mathbf{h}_2^{\mathrm{src}}, \ldots, \mathbf{h}_{L_{\mathrm{src}}}^{\mathrm{src}}\right] \in \mathbb{R}^{L_{\mathrm{src}} \times d_{\mathrm{src}}}$$

denote the feature sequence produced by the frozen source-modality encoder, where L_{src} is the source sequence length and d_{src} is the feature dimension. In the original BLIP-2 setting, the source modality is the image modality, and $\mathbf{H}^{\mathrm{src}}$ corresponds to the visual feature sequence extracted by a frozen image encoder.

Q-Former maintains a set of learnable query tokens

$$\mathbf{Q}^{(0)} = \left[\mathbf{q}_1^{(0)}, \mathbf{q}_2^{(0)}, \ldots, \mathbf{q}_M^{(0)}\right] \in \mathbb{R}^{M \times d_q},$$

where M is the number of learnable query tokens and d_q is the hidden dimension of the query representations. These query tokens are independent of the source sequence length L_{src}, which allows Q-Former to compress a variable-length source feature sequence into a fixed-size representation.

At a cross-attention layer, the query states act as attention queries, while the source-modality features act as keys and values. For the ℓ-th layer, this can be written as

$$\mathrm{CrossAttn}\left(\mathbf{Q}^{(\ell)}, \mathbf{H}^{\mathrm{src}}, \mathbf{H}^{\mathrm{src}}\right), \tag{3.25}$$

where $\mathbf{Q}^{(\ell)} \in \mathbb{R}^{M \times d_q}$ denotes the query representation at layer ℓ. Since the query representations and source features may have different dimensions, they are first projected into a shared attention space:

$$\mathbf{A}^{(\ell)} = \mathrm{softmax}\left(\frac{\left(\mathbf{Q}^{(\ell)}\mathbf{W}_Q^{(\ell)}\right)\left(\mathbf{H}^{\mathrm{src}}\mathbf{W}_K^{(\ell)}\right)^{\top}}{\sqrt{d_a}}\right), \tag{3.26}$$

where $\mathbf{W}_Q^{(\ell)} \in \mathbb{R}^{d_q \times d_a}$ and $\mathbf{W}_K^{(\ell)} \in \mathbb{R}^{d_{\mathrm{src}} \times d_a}$ are projection matrices, and d_a is the attention dimension. The cross-attended query representation is then computed as

$$\mathrm{CrossAttn}\left(\mathbf{Q}^{(\ell)}, \mathbf{H}^{\mathrm{src}}, \mathbf{H}^{\mathrm{src}}\right) = \mathbf{A}^{(\ell)}\left(\mathbf{H}^{\mathrm{src}}\mathbf{W}_V^{(\ell)}\right), \tag{3.27}$$

where $\mathbf{W}_V^{(\ell)} \in \mathbb{R}^{d_{\mathrm{src}} \times d_a}$ projects the source features into the value space.

In a multi-layer Q-Former stack, the query tokens are progressively updated through self-attention, cross-attention, and feed-forward transformations. The self-attention layers model interactions among query tokens, while the cross-attention layers allow the queries to selectively read information from the frozen source-modality features. In BLIP-2, cross-attention layers are inserted every other transformer block, so the learnable queries can repeatedly extract useful information from the frozen image features while keeping the trainable interface compact. After the final layer, the query outputs are denoted as

$$\mathbf{Q}^{(*)} = \left[\mathbf{q}_1^{(*)}, \mathbf{q}_2^{(*)}, \ldots, \mathbf{q}_M^{(*)}\right] \in \mathbb{R}^{M \times d_q}. \tag{3.28}$$

This output sequence provides a compressed representation of the source modality, whose length is fixed by the number of query tokens M rather than by the original source sequence length L_{src}.

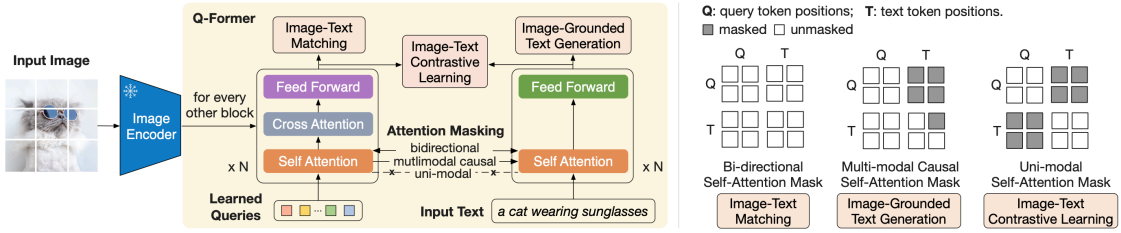


Figure 3.3: Architecture of the Q-Former in BLIP-2, where learnable query tokens interact with modality features through transformer attention to bridge representation spaces. Reproduced from [24].

As shown in Figure 3.3, Q-Former uses learnable query tokens as a compact trainable interface to interact with frozen modality features. This makes it both parameter-efficient and structurally suitable for cross-modal bottleneck learning.

The main relevance of Q-Former to this work is conceptual rather than domain-specific. It demonstrates that a small set of learnable queries can serve as an effective bottleneck for *selective cross-modal information extraction*, and that such query outputs can be pre-trained to align one modality with another before downstream generation. In BLIP-2, the goal is to align image representations with language space; in this work, a related design intuition is adapted to recommendation, where semantic text representations and collaborative graph representations must be fused before quantization. The detailed adaptation is illustrated in the methodology part.

3.5. Prompt-based SFT for Generative Recommendation

After semantic item identifiers are constructed, the next step is to teach an LLM to understand and generate them in recommendation contexts. A representative approach is **prompt-based supervised fine-tuning**, where recommendation data are reformulated into instruction-style input-output pairs [7, 52]. This design allows the model to jointly learn recommendation behavior and the semantics of newly introduced semantic item tokens.

Among existing methods, LC-Rec [52] provides a particularly influential alignment-tuning framework. Instead of using only a single task of predicting the next semantic item ID from historical semantic item IDs, it introduces multiple prompt styles to better connect semantic item identifiers with natural language and recommendation semantics. These tasks can be summarized as follows.

(1) SID Sequential Recommendation. Given a user's historical interaction sequence represented by semantic item IDs, the model predicts the SID of the next item:

$$(\mathrm{SID}_{i_1}, \mathrm{SID}_{i_2}, \ldots, \mathrm{SID}_{i_n}) \rightarrow \mathrm{SID}_{i_{n+1}}. \tag{3.29}$$

This is the most direct formulation of generative recommendation.

(2) SID–Title Grounding. The model learns bidirectional associations between discrete semantic item IDs and their corresponding natural-language titles or descriptions:

$$\mathrm{SID}_i \leftrightarrow \mathrm{Title}(i). \tag{3.30}$$

This grounding task helps the LLM attach interpretable semantics to newly introduced semantic item tokens.

(3) Fusion Sequential Recommendation. The model receives historical semantic item IDs as input but is asked to generate the target item's title or description rather than its SID:

$$(\mathrm{SID}_{i_1}, \ldots, \mathrm{SID}_{i_n}) \rightarrow \mathrm{Title}(i_{n+1}) \text{ or } \mathrm{Description}(i_{n+1}). \tag{3.31}$$

This encourages the model to connect symbolic SID sequences with natural-language semantics.

(4) Title Sequential Recommendation. The historical interaction sequence is represented directly by item titles, and the model predicts the next item's title:

$$(\mathrm{Title}(i_1), \ldots, \mathrm{Title}(i_n)) \ \rightarrow \ \mathrm{Title}(i_{n+1}). \tag{3.32}$$

This serves as a pure natural-language recommendation task.

(5) Title-to-SID Sequential Recommendation. The model takes title-based history as input and outputs the next item's discrete semantic identifier:

$$(\mathrm{Title}(i_1), \ldots, \mathrm{Title}(i_n)) \ \rightarrow \ \mathrm{SID}_{i_{n+1}}. \tag{3.33}$$

This task acts as a bridge between language-space input and SID-space output.

These prompt styles can be unified under the standard language modeling objective:

$$\mathcal{L}_{\mathrm{LLM}} = -\sum_{\tau=1}^{|\mathbf{y}|} \log P_\theta(y_\tau \mid y_{<\tau}, \mathbf{x}), \tag{3.34}$$

where $\mathbf{x}$ is the instruction-formatted input, $\mathbf{y}$ is the target output sequence, and y_τ is the target output token at decoding step τ. The importance of such multi-task prompt tuning is that it goes beyond merely training the model to output a valid semantic item code; it also helps the model understand the semantic and recommendation-related meaning behind the code.

3.6. Constrained Decoding for Valid Item Generation

Once semantic item IDs are introduced as generation targets, inference must ensure that the generated SID corresponds to a valid item. Unconstrained autoregressive decoding may produce invalid combinations of SID tokens that do not map to any real item. To address this problem, many generative recommendation methods adopt **constrained decoding** based on a prefix tree (Trie) or other lexical constraints [16, 45, 52].

Recall that each item $i \in \mathcal{I}$ is represented by a semantic item ID produced by residual quantization:

$$\mathrm{SID}_i = \left(k_i^{(1)}, k_i^{(2)}, \ldots, k_i^{(D)}\right),$$

where D is the number of residual quantization levels and $k_i^{(d)} \in \{0, \ldots, N_c - 1\}$ is the selected code index at level d.

In practice, these code indices are serialized into level-specific special tokens before being used as LLM generation targets. Let

$$s_i^{(d)} = \phi_d(k_i^{(d)})$$

denote the special token corresponding to the level-d code index of item i, where $\phi_d(\cdot)$ maps a code index to its token form. For example, if $D = 3$, an item may have

$$\mathrm{SID}_0 = (251, 253, 195),$$

which is serialized as

$$\mathbf{y}_0^{\mathrm{SID}} = (\langle a_{251}\rangle, \langle b_{253}\rangle, \langle c_{195}\rangle).$$

Here, the prefixes $\langle a.\rangle$, $\langle b.\rangle$, and $\langle c.\rangle$ indicate different quantization levels.

Therefore, the valid SID token sequence set is defined as

$$\mathcal{Y}_{\mathrm{valid}} = \left\{ \mathbf{y}_i^{\mathrm{SID}} = \left(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(D)}\right) \mid i \in \mathcal{I} \right\}. \tag{3.35}$$

In the next-item SID prediction task, the LLM receives an instruction-formatted prompt $\mathbf{x}$, which contains the user's historical interaction sequence and task instruction. The model then autoregressively generates the target SID token by token. Let

$$\mathbf{y} = (y_1, y_2, \ldots, y_D)$$

denote the generated SID token sequence, where y_d is the generated SID token at the d-th quantization level. Ideally, for the target item i_{n+1}, we have

$$y_d = s_{i_{n+1}}^{(d)}, \qquad d = 1, \ldots, D.$$

Given the prompt $\mathbf{x}$ and the previously generated SID prefix

$$y_{<d} = (y_1, \ldots, y_{d-1}),$$

the LLM computes a logit vector

$$\mathbf{o}_d = g_\theta(\mathbf{x}, y_{<d}), \tag{3.36}$$

where $\mathbf{o}_d \in \mathbb{R}^{|\mathcal{V}|}$ and $\mathcal{V}$ is the generation vocabulary. The next-token probability is obtained by applying softmax over the logits:

$$P_\theta(y_d = v \mid \mathbf{x}, y_{<d}) = \frac{\exp(\mathbf{o}_d[v])}{\sum_{v' \in \mathcal{V}} \exp(\mathbf{o}_d[v'])}, \qquad v \in \mathcal{V}. \tag{3.37}$$

However, not every candidate token can lead to a valid SID. Since $\mathcal{Y}_{\text{valid}}$ is known in advance from the item index file, a Trie can be built over all valid SID token sequences. At decoding step d, the set of possible next tokens is determined by the current prefix:

$$\mathcal{A}_d(y_{<d}) = \left\{ s_i^{(d)} \mid i \in \mathcal{I}, \left(s_i^{(1)}, \ldots, s_i^{(d-1)}\right) = y_{<d} \right\}. \tag{3.38}$$

For example, suppose the index file contains the following valid SID token sequences:

$$\begin{aligned} &0 : (\langle a_{251} \rangle, \langle b_{253} \rangle, \langle c_{195} \rangle), \\ &1 : (\langle a_{24} \rangle, \langle b_{21} \rangle, \langle c_{13} \rangle), \\ &2 : (\langle a_{24} \rangle, \langle b_{21} \rangle, \langle c_0 \rangle). \end{aligned}$$

At the first decoding step, the prefix is empty, so the possible first tokens are

$$\mathcal{A}_1(\emptyset) = \{\langle a_{251} \rangle, \langle a_{24} \rangle\}.$$

If the model has already generated

$$y_1 = \langle a_{24} \rangle,$$

then the possible second token set becomes

$$\mathcal{A}_2(\langle a_{24} \rangle) = \{\langle b_{21} \rangle\}.$$

After the prefix

$$(y_1, y_2) = (\langle a_{24} \rangle, \langle b_{21} \rangle),$$

the possible third token set is

$$\mathcal{A}_3(\langle a_{24} \rangle, \langle b_{21} \rangle) = \{\langle c_{13} \rangle, \langle c_0 \rangle\}.$$

Thus, the model may choose either $\langle c_{13} \rangle$ or $\langle c_0 \rangle$ according to its learned probability distribution, but tokens such as $\langle c_{195} \rangle$ are masked out under this prefix because they would not form a valid item SID.

The constrained decoding mask is defined as

$$m_d(v) = \begin{cases} 0, & v \in \mathcal{A}_d(y_{<d}), \\ -\infty, & \text{otherwise.} \end{cases} \tag{3.39}$$

The constrained decoding score for a candidate token v is then

$$\text{score}(v, d) = \log P_\theta(y_d = v \mid \mathbf{x}, y_{<d}) + m_d(v). \tag{3.40}$$

Equivalently, constrained decoding can be viewed as re-normalizing the next-token distribution over only the possible tokens:

$$P_\theta^{\mathrm{con}}(y_d = v \mid \mathbf{x}, y_{<d}) = \frac{\exp(\mathbf{o}_d[v]) \cdot \mathbb{I}\left[v \in \mathcal{A}_d(y_{<d})\right]}{\sum_{v' \in \mathcal{A}_d(y_{<d})} \exp(\mathbf{o}_d[v'])}. \tag{3.41}$$

Therefore, P_θ still computes the next-token probability from the prompt and the generated SID prefix, while the Trie-based constraint ensures that only tokens leading to valid item SIDs can be selected.

After D decoding steps, the generated sequence

$$\hat{\mathbf{y}}^{\mathrm{SID}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_D)$$

is guaranteed to belong to $\mathcal{Y}_{\mathrm{valid}}$, and can therefore be mapped back to a valid item.

4

Methodology

This chapter presents the proposed framework for **unified item quantization** in LLM-based generative recommendation. The central idea is to inject collaborative filtering signals into semantic item representations *before* vector quantization, so that the resulting discrete item identifiers are both semantically meaningful and collaboratively informative. To this end, we propose a **Unified Q-Former (UQF)** that explicitly fuses sequence-level text embeddings and layer-level graph embeddings into a unified item representation, which is then quantized into discrete Semantic IDs (SIDs) for downstream LLM-based recommendation.

4.1. Problem Formulation

Let $\mathcal{U} = \{u_1, u_2, \ldots, u_{N_u}\}$ denote the user set. For each user $u \in \mathcal{U}$, the historical interaction sequence is denoted by

$$\mathcal{H}_u = (i_{u,1}, i_{u,2}, \ldots, i_{u,T_u}), \tag{4.1}$$

where $i_{u,t} \in \mathcal{I}$ is the item interacted with by user u at time step t, and T_u is the sequence length.

The task considered in this work is **next-item generative recommendation**. Given a user history $\mathcal{H}_u$, the model aims to predict the next most likely interacted item

$$i_u^+ = i_{u,T_u+1}, \tag{4.2}$$

and return the top-K candidate items during inference. In conventional item-level recommendation, this objective can be written as

$$P(i_u^+ \mid \mathcal{H}_u). \tag{4.3}$$

Under the generative recommendation setting, each item is represented by a semantic item ID. Following the notation in section 3.3, the SID of item i is defined as

$$\mathrm{SID}_i = \left(k_i^{(1)}, k_i^{(2)}, \ldots, k_i^{(D)}\right), \tag{4.4}$$

where D is the number of residual quantization levels and $k_i^{(d)} \in \{0, \ldots, N_c - 1\}$ is the selected code index at level d.

In practice, the code-index tuple SID_i is serialized into a sequence of level-specific SID tokens before being used as the generation target of the LLM. Let

$$\mathbf{y}_i^{\mathrm{SID}} = \left(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(D)}\right) \tag{4.5}$$

denote the serialized SID token sequence of item i, where

$$s_i^{(d)} = \phi_d(k_i^{(d)}) \tag{4.6}$$

is the special token corresponding to the level-d code index. For example, if

$$\mathrm{SID}_0 = (251, 253, 195),$$

then its serialized form may be

$$\mathbf{y}_0^{\mathrm{SID}} = (\langle a_{251}\rangle, \langle b_{253}\rangle, \langle c_{195}\rangle).$$

Given the user's historical sequence, the instruction-formatted prompt is denoted as

$$\mathbf{x}_u = \mathrm{Prompt}\left(\mathbf{y}_{i_{u,1}}^{\mathrm{SID}}, \mathbf{y}_{i_{u,2}}^{\mathrm{SID}}, \ldots, \mathbf{y}_{i_{u,T_u}}^{\mathrm{SID}}\right), \tag{4.7}$$

where $\mathbf{x}_u$ contains both the task instruction and the user's historical interaction information.

The **next-item recommendation task** is then reformulated as generating the SID token sequence of the target item:

$$P_\theta\left(\mathbf{y}_{i_u^+}^{\mathrm{SID}} \mid \mathbf{x}_u\right). \tag{4.8}$$

The methodological focus of this work is therefore to learn a unified item representation function

$$f_{\mathrm{uni}} : \mathcal{I} \to \mathbb{R}^{d_u}, \qquad \mathbf{u}_i = f_{\mathrm{uni}}(i), \tag{4.9}$$

such that $\mathbf{u}_i$ preserves both semantic content and collaborative preference structure. The learned representation is then encoded and quantized into a residual semantic item ID:

$$\mathrm{SID}_i = Q_{\mathrm{RQ}}\left(f_{\mathrm{enc}}(\mathbf{u}_i)\right), \tag{4.10}$$

where $Q_{\mathrm{RQ}}(\cdot)$ denotes the residual quantization process that produces the code-index tuple

$$\left(k_i^{(1)}, k_i^{(2)}, \ldots, k_i^{(D)}\right).$$

4.2. Framework Overview

The overall pipeline of the proposed framework is illustrated in Figure 4.1.

The framework consists of four stages:

1. **Semantic and collaborative input construction:** each item is represented by a text embedding sequence and a graph embedding sequence.
2. **Unified item representation learning:** the proposed UQF explicitly fuses both modalities into a unified embedding before quantization.
3. **Item quantization:** the unified embeddings are quantized into multi-level discrete SIDs through RQ-VAE.
4. **Downstream generative recommendation:** the SID tokens are added into the vocabulary of a pretrained LLM and used in multi-task supervised fine-tuning.

Among these stages, Stages 1, 3, and 4 follow standard or adapted practices from prior generative recommendation pipelines, while the main methodological contribution of this work lies in Stage 2, namely the proposed **Unified Q-Former** for pre-quantization semantic–collaborative fusion.

4.3. Item Tokenization

Item tokenization consists of stages 1, 2 and 3, which generates the final discrete representation of items containing both semantic and collaborative information.

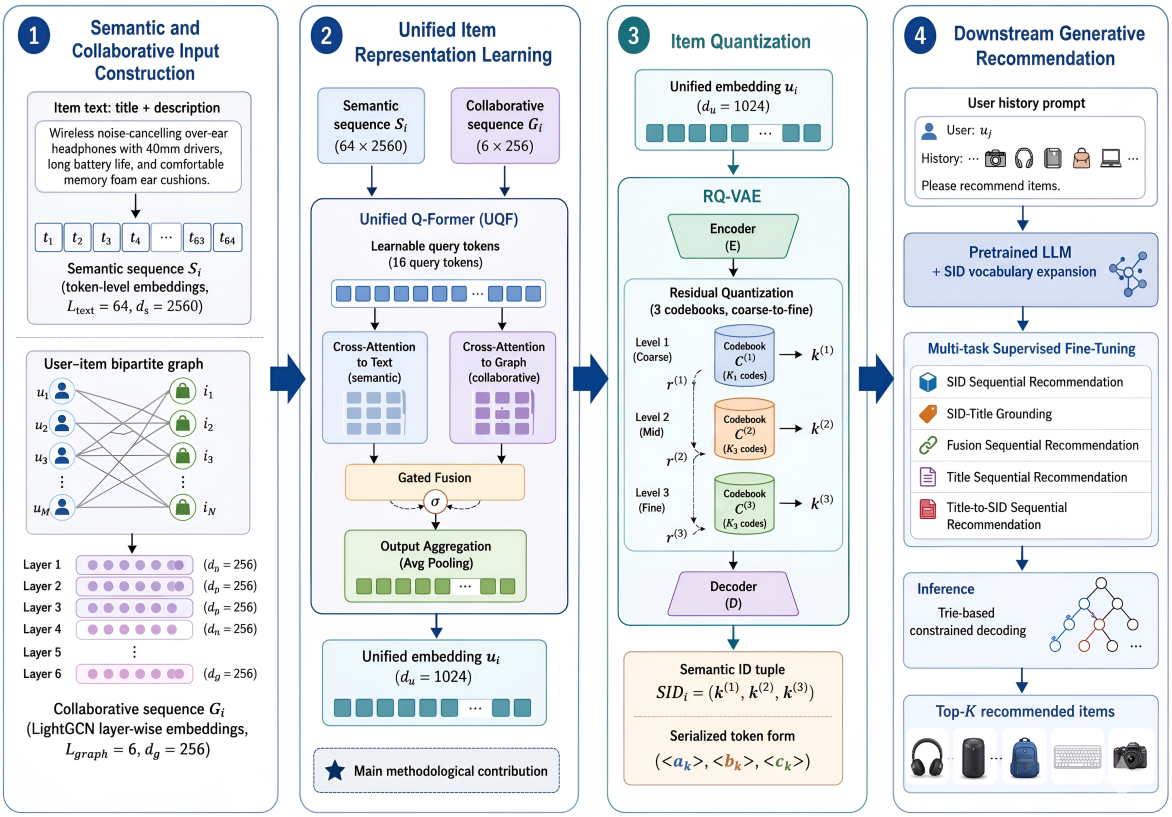


Figure 4.1: The 4-stage overall pipeline of the proposed framework.

4.3.1. Semantic and Collaborative Input Construction

Semantic Sequence Construction For each item $i \in \mathcal{I}$, we concatenate its textual metadata, including title and description, into a text string t_i. We employ a pretrained language model (PLM), specifically `Qwen3-Embedding-4B`, to encode item text. Instead of using a single pooled representation, we retain the full token-level hidden states in sequence mode:

$$\mathbf{S}_i = \mathrm{PLM}(t_i) \in \mathbb{R}^{L_{\text{text}} \times d_s}, \tag{4.11}$$

where L_{text} is the maximum text length after padding or truncation, and d_s is the semantic embedding dimension used in `Qwen3-Embedding-4B`.

This sequence-level representation serves as the semantic input to UQF.

Collaborative Sequence Construction To encode collaborative filtering signals, we train a LightGCN model on the global user–item interaction graph and retain the item embeddings from all propagation layers. For each item i, the collaborative representation is

$$\mathbf{G}_i = \left[\mathbf{q}_i^{(0)}; \mathbf{q}_i^{(1)}; \ldots; \mathbf{q}_i^{(J)}\right] \in \mathbb{R}^{L_{\text{graph}} \times d_g}, \tag{4.12}$$

where J is the number of propagation layers, $L_{\text{graph}} = J + 1$, and d_g is the embedding dimension.

Instead of averaging the graph embeddings across layers, we keep the full layer-wise sequence so that UQF can dynamically attend to collaborative information from different propagation depths.

4.3.2. Unified Item Representation Learning

Unified Q-Former We propose the **Unified Q-Former (UQF)**, a cross-modal fusion architecture inspired by BLIP-2's Q-Former [24], to learn a unified item representation before quantization. Unlike previous unified tokenization methods that mainly rely on auxiliary regularization, dual-code design,

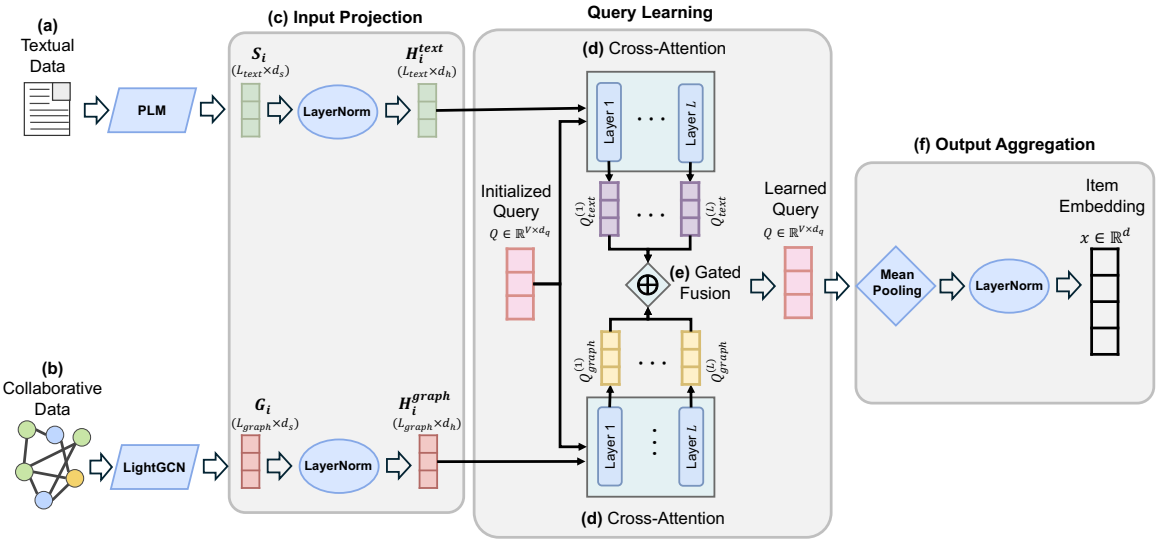


Figure 4.2: Overall architecture of UQF

or alignment-based fusion, UQF explicitly performs semantic–collaborative at the representation level. The overall architecture of UQF is shown in Figure 4.2.

The design of UQF is guided by three key ideas:

- **Sequence-level cross-modal fusion:** both semantic and collaborative inputs are retained as sequences to avoid information loss in pooled-vector fusion;
- **Parallel cross-attention with adaptive gating:** the two modalities are fused symmetrically without sequential modality bias;
- **Hybrid supervision with robustness regularization:** the unified embedding is trained to preserve both structural and semantic neighborhood information while remaining robust to missing modality cues.

A key design motivation arises from the *cross-attention degeneration problem*. If the input to cross-attention has sequence length L, namely a single pooled vector, the attention weight collapses to a constant:

$$\mathrm{softmax}([s]) = 1.0, \tag{4.13}$$

which reduces cross-attention to a fixed linear mapping regardless of the query. Therefore, meaningful attention-based fusion requires sequence-level inputs from both modalities. In our setting, the text input is the token-level semantic sequence $\mathbf{S}_i \in \mathbb{R}^{L_{\text{text}} \times d_s}$, while the graph input is the layer-wise collaborative sequence $\mathbf{G}_i \in \mathbb{R}^{L_{\text{graph}} \times d_g}$.

Input Projection and Normalization. The two modality sequences lie in different feature spaces and scales. We first project each modality independently into a shared hidden space and normalize each token or layer representation:

$$\mathbf{H}_i^{\text{text}} = \mathrm{LayerNorm}\big(\mathbf{S}_i \mathbf{W}^{\text{text}}\big) \in \mathbb{R}^{L_{\text{text}} \times d_h}, \tag{4.14}$$

$$\mathbf{H}_i^{\text{graph}} = \mathrm{LayerNorm}\big(\mathbf{G}_i \mathbf{W}^{\text{graph}}\big) \in \mathbb{R}^{L_{\text{graph}} \times d_h}, \tag{4.15}$$

where $\mathbf{W}^{\text{text}} \in \mathbb{R}^{d_s \times d_h}$, $\mathbf{W}^{\text{graph}} \in \mathbb{R}^{d_g \times d_h}$, and d_h is the shared hidden dimension.

Learnable Query Tokens. A set of V learnable query tokens

$$\mathbf{Q} \in \mathbb{R}^{V \times d_q}, \tag{4.16}$$

with d_q, serves as the information bottleneck of UQF. These query tokens are shared across all items and initialized from standard distribution $\mathcal{N}(0, \sigma^2)$. For a mini-batch $\mathcal{B}$ with size $B = |\mathcal{B}|$, they are

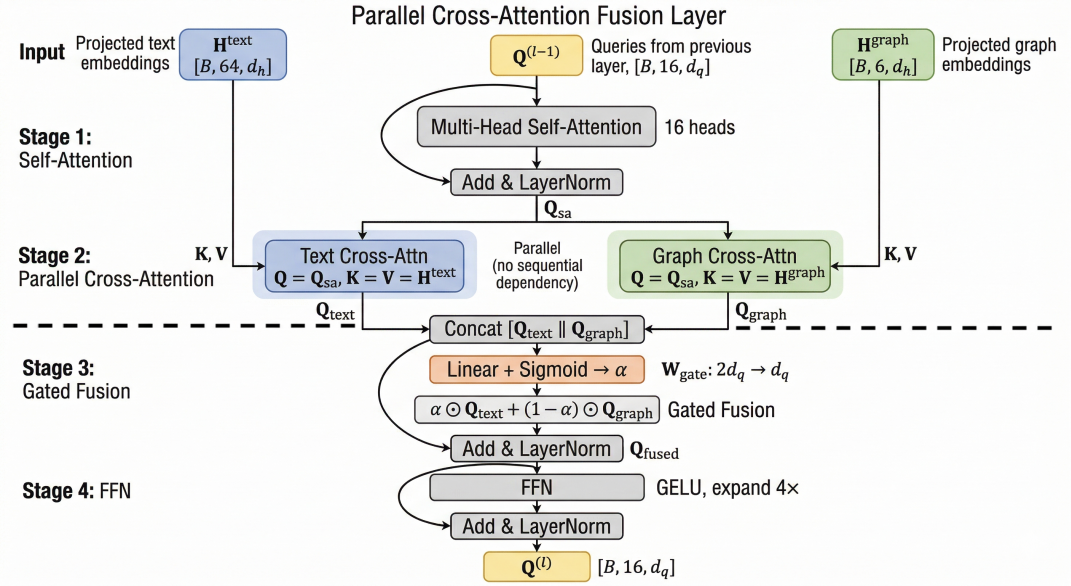


Figure 4.3: Illustration of parallel cross-attention fusion layer in UQF. Upper part shows parallel cross-attention; lower part shows gated fusion. The figure is divided by a horizontal line.

expanded to

$$\mathbf{Q}^{(0)} \in \mathbb{R}^{B \times V \times d_q}. \tag{4.17}$$

The learnable queries act as latent probes that selectively extract and aggregate information from semantic and collaborative sequences in the following parallel cross-attention fusion layer.

Parallel Cross-Attention Fusion Layer. UQF contains L stacked fusion layers. Each layer consists of four stages. The detailed structure of parallel cross-attention fusion layer is illustrated in Figure 4.3.

(1) Self-Attention on Queries. First, self-attention is performed among the query tokens:

$$\mathbf{Q}^{(l)} = \mathrm{LN}\big(\mathbf{Q}^{(l-1)} + \mathrm{Dropout}\big(\mathrm{MHA}_{\text{self}}(\mathbf{Q}^{(l-1)}, \mathbf{Q}^{(l-1)}, \mathbf{Q}^{(l-1)})\big)\big), \tag{4.18}$$

where $\mathrm{MHA}_{\text{self}}$ is multi-head self-attention with H heads, and $\mathrm{LN}(\cdot)$ denotes Layer Normalization.

(2) Parallel Cross-Attention. The updated queries attend to text and graph sequences *in parallel* through two independent cross-attention modules:

$$\mathbf{Q}^{(l)}_{\text{text}} = \mathrm{MHA}_{\text{text}}\big(\mathbf{Q}^{(l)}, \mathbf{H}^{\text{text}}, \mathbf{H}^{\text{text}}\big) \in \mathbb{R}^{B \times V \times d_q}, \tag{4.19}$$

$$\mathbf{Q}^{(l)}_{\text{graph}} = \mathrm{MHA}_{\text{graph}}\big(\mathbf{Q}^{(l)}, \mathbf{H}^{\text{graph}}, \mathbf{H}^{\text{graph}}\big) \in \mathbb{R}^{B \times V \times d_q}. \tag{4.20}$$

For each head h, scaled dot-product attention is computed as

$$\mathrm{head}_h = \mathrm{softmax}\Big(\frac{\mathbf{Q}^{(l)}\mathbf{W}_{Q,h}\big(\mathbf{H}^{(\cdot)}\mathbf{W}_{K,h}\big)^{\top}}{\sqrt{d_k}}\Big)\mathbf{H}^{(\cdot)}\mathbf{W}_{V,h}, \tag{4.21}$$

where $d_k = d_h/H$.

Crucially, the two cross-attention modules operate on the *same* query state rather than sequentially. This avoids the bias that would occur if one modality were attended first and its information then altered the query state before attending to the other modality.

(3) Gated Fusion. The text-attended and graph-attended query outputs are fused through a learned

element-wise gate:

$$\boldsymbol{\alpha}^{(l)} = \sigma\Big(\mathbf{W}_\alpha \big[\mathbf{Q}_{\text{text}}^{(l)} \| \mathbf{Q}_{\text{graph}}^{(l)}\big] + \mathbf{b}_\alpha\Big) \in (0,1)^{B\times V\times d_q}, \tag{4.22}$$

$$\mathbf{Q}_{\text{fused}}^{(l)} = \boldsymbol{\alpha}^{(l)} \odot \mathbf{Q}_{\text{text}}^{(l)} + \big(1-\boldsymbol{\alpha}^{(l)}\big) \odot \mathbf{Q}_{\text{graph}}^{(l)}, \tag{4.23}$$

$$\mathbf{Q}^{(l)} = \mathrm{LN}\big(\mathbf{Q}^{(l)} + \mathrm{Dropout}(\mathbf{Q}_{\text{fused}}^{(l)})\big), \tag{4.24}$$

where $\mathbf{W}_\alpha \in \mathbb{R}^{2d_q\times d_q}$, σ is the sigmoid function, $\|$ denotes concatenation, and $\odot$ denotes element-wise multiplication.

The gate is computed per sample, per query, and per dimension, enabling fine-grained item-adaptive modality weighting.

(4) Feed-Forward Network. Finally, a feed-forward network with GELU activation and expansion ratio is applied:

$$\mathbf{Q}^{(l)} = \mathrm{LN}\big(\mathbf{Q}^{(l)} + \mathrm{FFN}(\mathbf{Q}^{(l)})\big), \tag{4.25}$$

where

$$\mathrm{FFN}(\mathbf{x}) = \mathrm{Dropout}\big(\mathbf{W}_2\,\mathrm{GELU}\big(\mathrm{Dropout}(\mathbf{W}_1\mathbf{x})\big)\big), \tag{4.26}$$

with $\mathbf{W}_1 \in \mathbb{R}^{d_q\times 4d_h}$ and $\mathbf{W}_2 \in \mathbb{R}^{4d_h\times d_q}$.

Output Aggregation. After the final UQF layer, the V query tokens are aggregated through mean pooling and projected to the final unified embedding:

$$x_i = \mathrm{LayerNorm}\Big(\mathbf{W}_{\text{out}} \cdot \frac{1}{V}\sum_{v=1}^{V} \mathbf{Q}_v^{(L)}\Big) \in \mathbb{R}^d, \tag{4.27}$$

where $\mathbf{W}_{\text{out}} \in \mathbb{R}^{d_q\times d}$ and d is the unified embedding dimension.

Modality Dropout. To prevent single-modality shortcut learning, we apply modality dropout *only to anchor samples* during training. For each sample, we draw $r \sim \mathrm{Uniform}(0,1)$ and define

$$(\tilde{\mathbf{S}}_i, \tilde{\mathbf{G}}_i) = \begin{cases} (\mathbf{0}, \mathbf{G}_i), & \text{if } r < p_{\text{mod}}/2, \\ (\mathbf{S}_i, \mathbf{0}), & \text{if } p_{\text{mod}}/2 \le r < p_{\text{mod}}, \\ (\mathbf{S}_i, \mathbf{G}_i), & \text{if } r \ge p_{\text{mod}}, \end{cases} \tag{4.28}$$

with p_{mod}.

Positive samples used in contrastive training always receive full, unmasked inputs.

Hybrid Contrastive Training. To preserve both collaborative and semantic neighborhood structure, we adopt hybrid contrastive learning with two types of positives.

Before training, we pre-compute two neighbor tables for each item:

- **Structure neighbors:** based on item co-occurrence frequencies in the interaction data, we retain the top-K_s neighbors for each item, denoted as $\mathcal{N}_{\text{struc}}(i)$;
- **Semantic neighbors:** based on cosine similarity between mean-pooled text embeddings, we retain the top-K_t neighbors for each item, denoted as $\mathcal{N}_{\text{sem}}(i)$.

For each anchor item i in a mini-batch $\mathcal{B}$, we sample one structure positive $i_s^+ \in \mathcal{N}_{\text{struc}}(i)$ and one semantic positive $i_t^+ \in \mathcal{N}_{\text{sem}}(i)$. The anchor is encoded with modality dropout, while the positives are encoded without dropout:

$$x_i = \mathrm{UQF}(\tilde{\mathbf{S}}_i, \tilde{\mathbf{G}}_i; \texttt{dropout=True}), \tag{4.29}$$

$$x_{i_s^+} = \mathrm{UQF}(\mathbf{S}_{i_s^+}, \mathbf{G}_{i_s^+}; \texttt{dropout=False}), \tag{4.30}$$

$$x_{i_t^+} = \mathrm{UQF}(\mathbf{S}_{i_t^+}, \mathbf{G}_{i_t^+}; \texttt{dropout=False}). \tag{4.31}$$

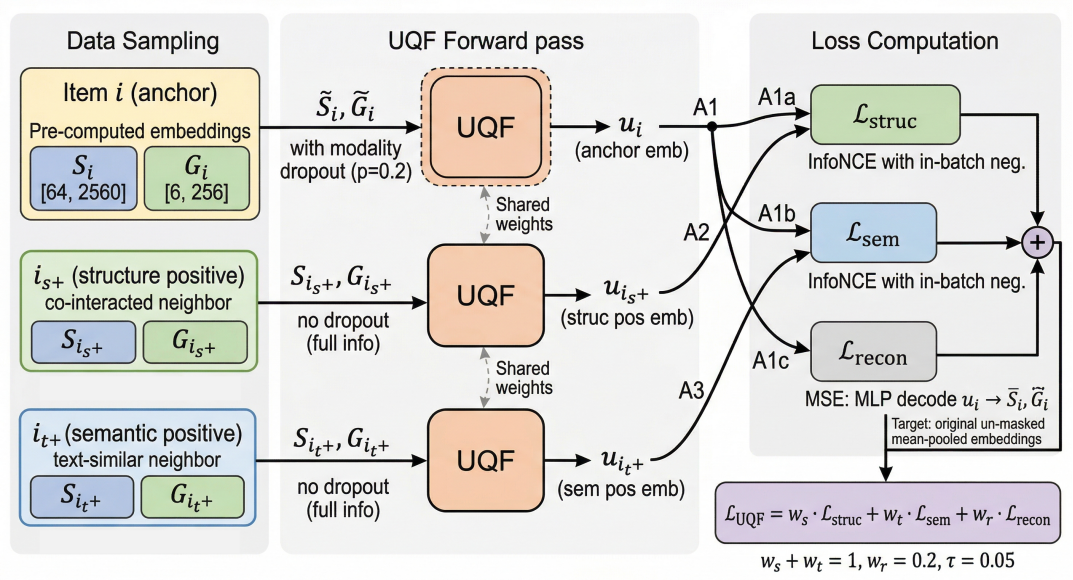


Figure 4.4: Overall training pipeline of UQF

All embeddings are ℓ_2-normalized before similarity computation. The structure and semantic contrastive losses both adopt InfoNCE [29]:

$$\mathcal{L}_{\text{struc}} = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log \frac{\exp\big(\text{sim}(x_i, x_{i_s^+})/\tau\big)}{\exp\big(\text{sim}(x_i, x_{i_s^+})/\tau\big) + \sum_{j \in \mathcal{B} \setminus \{i\}} \exp\big(\text{sim}(x_i, x_{j_s^+})/\tau\big)}, \tag{4.32}$$

$$\mathcal{L}_{\text{sem}} = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log \frac{\exp\big(\text{sim}(x_i, x_{i_t^+})/\tau\big)}{\exp\big(\text{sim}(x_i, x_{i_t^+})/\tau\big) + \sum_{j \in \mathcal{B} \setminus \{i\}} \exp\big(\text{sim}(x_i, x_{j_t^+})/\tau\big)}, \tag{4.33}$$

where $\mathcal{B}$ denotes the mini-batch, j_s^+ and j_t^+ denote the sampled structure and semantic positives of another item $j \in \mathcal{B} \setminus \{i\}$, respectively, and they are used as in-batch negatives for anchor item i. The similarity function is cosine similarity:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}, \tag{4.34}$$

and τ is the temperature.

Optional Reconstruction Loss. To preserve sufficient information from both modalities in the unified embedding, we optionally add a reconstruction objective. Two lightweight MLP decoders reconstruct the original mean-pooled semantic and graph embeddings:

$$\hat{\mathbf{s}}_i = \text{MLP}_{\text{text}}(x_i) = \mathbf{W}_4 \, \text{GELU}(\mathbf{W}_3 x_i) \in \mathbb{R}^{d_s}, \tag{4.35}$$

$$\hat{\mathbf{g}}_i = \text{MLP}_{\text{graph}}(x_i) = \mathbf{W}_6 \, \text{GELU}(\mathbf{W}_5 x_i) \in \mathbb{R}^{d_g}. \tag{4.36}$$

The reconstruction loss is defined as

$$\mathcal{L}_{\text{recon}} = \text{MSE}(\hat{\mathbf{s}}_i, \bar{\mathbf{s}}_i) + \text{MSE}(\hat{\mathbf{g}}_i, \bar{\mathbf{g}}_i), \tag{4.37}$$

where $\bar{\mathbf{s}}_i$ and $\bar{\mathbf{g}}_i$ are the original unmasked mean-pooled semantic and graph embeddings.

Total UQF Objective. The final training objective of UQF is

$$\mathcal{L}_{\text{UQF}} = w_s \cdot \mathcal{L}_{\text{struc}} + w_t \cdot \mathcal{L}_{\text{sem}} + w_r \cdot \mathcal{L}_{\text{recon}}. \tag{4.38}$$

The training pipeline from initial data sampling to final loss computation is shown in Figure 4.4.

In the default setting, w_r, while w_s and w_t are treated as tunable hyperparameters satisfying $w_s + w_t = 1$, and are explored over a range of values.

4.3.3. Residual Quantization of Unified Embeddings

After unified representation learning, all item embeddings $\{x_i\}_{i\in\mathcal{I}}$ are quantized into discrete multi-level identifiers using an RQ-VAE. Since the general principle of residual vector quantization has been introduced in section 3.3, we only describe the specific instantiation used in this work.

Encoder and Decoder Each unified embedding $x_i \in \mathbb{R}^d$ is first compressed into a low-dimensional latent vector:

$$\mathbf{z}_i = f_{\text{enc}}(x_i) \in \mathbb{R}^{d_e}, \tag{4.39}$$

where d_e is the latent code dimension. The encoder is an MLP with layer with ReLU activations. The decoder mirrors the encoder and reconstructs the unified embedding from the quantized latent:

$$\hat{x}_i = f_{\text{dec}}(\hat{\mathbf{z}}_i). \tag{4.40}$$

Residual Quantization We adopt M residual quantization levels, and each level contains a codebook with N entries:

$$\mathcal{C}^{(m)} = \{\mathbf{e}_n^{(m)}\}_{n=0}^{N-1}, \qquad m = 1, \ldots, M, \tag{4.41}$$

where $\mathbf{e}_n^{(m)} \in \mathbb{R}^{d_e}$ denotes the n-th codeword in the level-m codebook.

For each latent vector $\mathbf{z}_i$, residual quantization selects one code index from each quantization level. The selected code index at level m is denoted as $c_i^{(m)} \in \{0, \ldots, N-1\}$. The quantized latent representation is then obtained by summing the selected codewords:

$$\hat{\mathbf{z}}_i = \sum_{m=1}^{M} \mathbf{e}_{c_i^{(m)}}^{(m)}. \tag{4.42}$$

The semantic item ID of item i is defined as the tuple of selected code indices:

$$\text{SID}_i = \left(c_i^{(1)}, c_i^{(2)}, c_i^{(3)}\right). \tag{4.43}$$

Before being used as LLM generation targets, the code-index tuple is serialized into level-specific SID tokens. Let

$$s_i^{(m)} = \phi_m(c_i^{(m)}) \tag{4.44}$$

denote the SID token corresponding to the level-m code index, where $\phi_m(\cdot)$ maps a code index to its level-specific token form. In our implementation,

$$\phi_1(n) = \langle a_n \rangle, \qquad \phi_2(n) = \langle b_n \rangle, \qquad \phi_3(n) = \langle c_n \rangle.$$

Therefore, the serialized SID token sequence of item i is

$$\mathbf{y}_i^{\text{SID}} = \left(s_i^{(1)}, s_i^{(2)}, s_i^{(3)}\right) = \left(\langle a_{c_i^{(1)}} \rangle, \langle b_{c_i^{(2)}} \rangle, \langle c_{c_i^{(3)}} \rangle\right). \tag{4.45}$$

For example, item “0” is mapped to the SID token sequence

$$\mathbf{y}_0^{\text{SID}} = [\texttt{<a\_205>}, \texttt{<b\_104>}, \texttt{<c\_207>}],$$

which corresponds to the code-index tuple

$$\text{SID}_0 = (205, 104, 207).$$

Similarly, item “1” is represented as

$$\mathbf{y}_1^{\text{SID}} = [\texttt{<a\_79>}, \texttt{<b\_82>}, \texttt{<c\_146>}],$$

corresponding to

$$\text{SID}_1 = (79, 82, 146).$$

Balanced Assignment and Collision Resolution To improve code utilization, the codebooks are initialized with K-Means over the latent vectors. During training, optional Sinkhorn–Knopp balanced assignment is applied to encourage more uniform code usage [52]. After quantization, if multiple items share the same SID tuple, collisions are resolved by iterative reassignment on the final quantization level until each item obtains a unique identifier.

Training Objective The RQ-VAE is trained with reconstruction and residual quantization losses:

$$\mathcal{L}_{\text{RQ-VAE}} = \|\hat{x}_i - x_i\|_2^2 + w_q \cdot \mathcal{L}_{\text{rq}}, \tag{4.46}$$

where $\mathcal{L}_{\text{rq}}$ consists of the standard codebook loss and commitment loss over residual quantization levels, and the commitment coefficient is set to β.

4.4. Downstream Generative Recommendation

4.4.1. SID Vocabulary Expansion

After assigning each item a semantic item ID, we add the corresponding SID tokens to the tokenizer vocabulary of the downstream LLM. Let $\mathcal{V}_{\text{LLM}}$ denote the original LLM vocabulary, and let $\mathcal{V}_{\text{SID}}$ denote the newly introduced SID token set:

$$\mathcal{V}_{\text{SID}} = \{\phi_d(k) \mid d \in \{1, \ldots, D\}, k \in \{0, \ldots, N_c - 1\}\}, \tag{4.47}$$

where $\phi_d(k)$ maps the level-d code index k to its corresponding special token. For example, when $D = 3$, the SID tuple $(251, 253, 195)$ may be serialized as

$$(\langle a_{251}\rangle, \langle b_{253}\rangle, \langle c_{195}\rangle).$$

The embedding matrix of the downstream LLM is expanded accordingly:

$$\mathbf{E}_{\text{new}} \in \mathbb{R}^{(|\mathcal{V}_{\text{LLM}}|+|\mathcal{V}_{\text{SID}}|)\times d_{\text{LLM}}}, \tag{4.48}$$

where $|\mathcal{V}_{\text{SID}}| \leq D \times N_c = 768$ in our setting. The newly added SID token embeddings are randomly initialized and jointly optimized during supervised fine-tuning.

4.4.2. Supervised Fine-Tuning

We instantiate downstream generative recommendation with a pretrained LLM backbone and adopt a five-task supervised fine-tuning setup following the LC-Rec-style prompt paradigm [52]. The training corpus is formed by concatenating five task-specific datasets:

1. SID Sequential Recommendation ($\mathcal{D}_1$): history SID token sequences $\rightarrow$ next-item SID token sequence;
2. SID–Title Grounding ($\mathcal{D}_2$): SID token sequence $\leftrightarrow$ item title;
3. Fusion Sequential Recommendation ($\mathcal{D}_3$): history SID token sequences $\rightarrow$ target title or description;
4. Title Sequential Recommendation ($\mathcal{D}_4$): history titles $\rightarrow$ next-item title;
5. Title-to-SID Sequential Recommendation ($\mathcal{D}_5$): history titles $\rightarrow$ next-item SID token sequence.

The supervised fine-tuning objective is the standard causal language modeling loss:

$$\mathcal{L}_{\text{SFT}} = -\sum_{\tau=1}^{|\mathbf{y}|} \log P_\theta\left(y_\tau \mid y_{<\tau}, \mathbf{x}\right), \tag{4.49}$$

where $\mathbf{x}$ is the instruction-formatted input, $\mathbf{y}$ is the target output sequence, y_τ is the target token at decoding step τ, and θ denotes the trainable parameters of the LLM.

4.4.3. Inference

During inference, we focus on the next-item SID prediction task. Given an instruction-formatted prompt $\mathbf{x}$ constructed from the user's historical interactions, the fine-tuned LLM autoregressively generates a SID token sequence:

$$\hat{\mathbf{y}}^{\mathrm{SID}} = (\hat{s}^{(1)}, \hat{s}^{(2)}, \ldots, \hat{s}^{(D)}).$$

Each generated token $\hat{s}^{(d)}$ corresponds to the predicted code index at the d-th quantization level.

To ensure that the generated SID corresponds to a real item, we apply the constrained decoding strategy introduced in section 3.6. Specifically, we construct the valid SID token sequence set from the item index:

$$\mathcal{Y}_{\mathrm{valid}} = \left\{ \mathbf{y}_i^{\mathrm{SID}} = \left(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(D)} \right) \mid i \in \mathcal{I} \right\}, \tag{4.50}$$

where $s_i^{(d)} = \phi_d(k_i^{(d)})$ is the level-specific SID token of item i. A Trie is built over $\mathcal{Y}_{\mathrm{valid}}$, and constrained beam search is used to restrict each decoding step to valid next SID tokens under the current prefix.

The top-K highest-scoring complete SID token sequences are then mapped back to their corresponding item IDs and returned as the recommendation results.

5

Experiments

This chapter evaluates the proposed unified item quantization framework on public recommendation benchmarks. We first introduce the datasets, preprocessing pipeline, baselines, and implementation details, and then describe the evaluation protocol used in all experiments.

5.1. Datasets and Preprocessing

We evaluate our framework on the **Amazon Review 2018** dataset [28], a widely used benchmark for sequential and generative recommendation. The raw data is obtained from the public repository,[1] consisting of product review records (user IDs, item ASINs, ratings, timestamps, review texts) and item metadata (titles, descriptions, brands, categories). We select two sub-domains of varying sparsity and scale: **Office Products** (OP) and **Musical Instruments** (MI).

The preprocessing pipeline follows the standard protocol established by prior work [15] and consists of the following steps. First, item titles are cleaned by removing HTML tags and entities; items with missing titles, titles containing `<span>` elements, or titles exceeding 20 words are discarded. Second, we perform iterative K-core filtering with $K{=}5$: in each iteration, users and items with fewer than K interactions within the specified time window are removed, and the process repeats until convergence. Each domain is assigned an initial time window $[t_{\text{start}}, t_{\text{end}}]$ with t_{end} fixed at November 2018; if the resulting item count falls below 3,000 after convergence, t_{start} is recursively shifted one year earlier and filtering is re-run, ensuring a minimum dataset scale. Third, for each user, all interactions are sorted chronologically, and a sliding window of maximum length 10 is applied: for the i-th interaction, the history is defined as items at positions $[\max(i{-}10,\,0),\,i)$, and the target is the item at position i. Finally, all interaction sequences are pooled and sorted globally by target timestamp, with the first 80% forming the training set, the next 10% the validation set, and the final 10% the test set. This *global temporal split* ensures that no future information leaks into training.

The resulting dataset statistics are summarized in Table 5.1.

5.2. Compared Method

We compare UQF with conventional, sequential, and generative recommendation methods.

Traditional Collaborative Filtering.

- **Pop**: Recommends items by global popularity.
- **ItemKNN** [**sarwar2001item**]: Item-based collaborative filtering using cosine similarity on the co-interaction matrix.
- **BPR** [37]: Matrix factorization optimized with Bayesian Personalized Ranking pairwise loss.

[1] `https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/`

Table 5.1: Dataset statistics after preprocessing (5-core filtering).

Statistic	Office Products	Musical Instruments
Time window	Oct 2016 – Nov 2018	Oct 1996 – Nov 2018
#Users	8,328	13,601
#Items	3,459	5,555
Total sequences	48,656	94,654
Train sequences	38,924	75,723
Valid sequences	4,866	9,465
Test sequences	4,866	9,466

Graph-based Collaborative Filtering.

- **LightGCN** [10]: Graph-based collaborative filtering with simplified graph convolution.

Sequential Recommendation Models.

- **GRU4Rec** [13]: GRU-based sequential recommendation model.
- **Caser** [44]: CNN-based sequential recommendation model.
- **BERT4Rec** [43]: Bidirectional Transformer encoder with masked item prediction.

All conventional baselines are implemented using the RecBole framework [51]. They are trained for 100 epochs with batch size 2,048, learning rate 10^{-3}, and early stopping based on validation NDCG@10. Traditional and graph-based models use pre-split benchmark files with uniform negative sampling (1 negative per positive), while sequential models are trained with cross-entropy loss without explicit negative sampling. Evaluation is conducted in full-ranking mode with the entire item catalog as candidate set.

Generative Recommendation Baselines. We further compare with representative generative recommendation baselines under two complementary settings: *controlled same-backbone tokenization comparison* and *additional architecture-level unified baseline comparison*.

(1) Controlled same-backbone GR comparisons. To isolate the effect of item tokenization quality, we adopt two downstream GR paradigms:

- **TIGER-style** [35]: semantic-ID-based generative recommendation;
- **LC-Rec-style** [52]: language-aligned multi-task generative recommendation.

For each downstream GR paradigm, we compare three different item tokenization strategies:

- **Semantic-only**: item quantization based only on text embeddings from Qwen3-Embedding-4B;
- **LETTER-based** [45]: prior unified item tokenization that injects collaborative signals through auxiliary regularization during token learning;
- **UQF-based (Ours)**: the proposed Unified Q-Former for explicit semantic–collaborative fusion before quantization.

Accordingly, the controlled GR comparisons include **TIGER**, **LETTER-TIGER**, **UQF-TIGER**, and **LC-Rec**, **LETTER-LC-Rec**, **UQF-LC-Rec**. To ensure fairness, all these variants share the same text encoder, the same RQ-VAE configuration (3-level SID with 256 codes per level), the same downstream LLM backbone (`Qwen2.5-3B-Instruct`), and the same constrained decoding strategy. Therefore, within this comparison group, the primary factor under evaluation is the quality of the learned item tokenization.

(2) Additional unified GR baseline: EAGER. We additionally include **EAGER** [46], a recent strong unified generative recommendation baseline. Unlike the above same-backbone comparisons, EAGER

is an architecture-level unified framework rather than a plug-in tokenization variant. It adopts a two-stream generation architecture with a shared encoder, dual codes for behavior and semantic information, two separate autoregressive decoders, a global contrastive task with a summary token, and a semantic-guided transfer task, followed by confidence-based ranking to merge predictions from the two streams.

For fair comparison with our main GR backbones, we preserve EAGER's original two-stream design but scale its generation backbone to the same model family and capacity level used in our experiments. Specifically, we replace the original shallow decoder setting in EAGER with a 36-layer decoder stack to match the scale of `Qwen/Qwen2.5-3B-Instruct` backbone family in our LC-Rec- and TIGER-based pipelines. In this way, EAGER is evaluated as a strong current unified GR baseline under comparable backbone capacity, while still retaining its original dual-stream behavior–semantic generation paradigm.

5.3. Implementation Details

The implementation and codes are available at `https://github.com/github-bowen/Unified-Q-Former-LLM4REC`.

For clarity and reproducibility, the main hyperparameters used in each stage of the proposed framework are summarized in Tables 5.2–5.5.

Hardware and Environment. Unless otherwise specified, all generative recommendation experiments are conducted on a single NVIDIA H100 GPU with 80GB memory. The supervised fine-tuning stage is launched with the `accelerate` framework and. The random seed is fixed to 42 in all experiments.

Semantic and Collaborative Representation Learning. For semantic input construction, each item title and description are concatenated and encoded by `Qwen3-Embedding-4B`. Token-level hidden states are retained in sequence mode with maximum length $L_{\text{text}} = 64$ and semantic dimension $d_s = 2560$. For collaborative input construction, a LightGCN model with $K = 5$ propagation layers is trained on the user–item interaction graph, and all layer-wise item embeddings are retained, resulting in a graph sequence of length $L_{\text{graph}} = 6$ with dimension $d_g = 256$.

Table 5.2: Hyperparameters for semantic and collaborative representation construction.

Stage	Hyperparameter	Value
Semantic embedding	Backbone model	Qwen3-Embedding-4B
Semantic embedding	Maximum text length L_{text}	64
Semantic embedding	Semantic embedding dimension d_s	2560
Semantic embedding	Input fields	title + description
Collaborative embedding	Backbone model	LightGCN
Collaborative embedding	Number of propagation layers K	5
Collaborative embedding	Graph sequence length L_{graph}	6
Collaborative embedding	Graph embedding dimension d_g	256
Collaborative embedding	Training objective	BPR loss

UQF Training. The proposed UQF uses $M = 16$ learnable query tokens, $L = 4$ fusion layers, and shared hidden dimension $d_h = d_q = d_u = 1024$. Modality dropout is applied only to anchor samples with probability $p_{\text{mod}} = 0.2$. For hybrid contrastive learning, both structure and semantic neighbor tables retain the top-50 neighbors for each item. The temperature in InfoNCE is set to $\tau = 0.05$, and the reconstruction loss weight is fixed at $w_r = 0.2$. The structure and semantic contrastive weights satisfy $w_s + w_t = 1$ and are tuned over $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. UQF is optimized with AdamW using learning rate 10^{-4}, weight decay 10^{-5}, cosine annealing schedule, batch size 256, training for up to 1000 epochs, and early stopping with patience 40 based on validation loss.

Table 5.3: Hyperparameters for Unified Q-Former (UQF).

Module	Hyperparameter	Value
UQF architecture	Number of query tokens M	16
UQF architecture	Number of fusion layers L	4
UQF architecture	Shared hidden dimension d_h	1024
UQF architecture	Query dimension d_q	1024
UQF architecture	Unified embedding dimension d_u	1024
UQF architecture	Number of attention heads H	16
Regularization	Modality dropout probability p_{mod}	0.2
Contrastive learning	Structure neighbors K_s	50
Contrastive learning	Semantic neighbors K_t	50
Contrastive learning	Temperature τ	0.05
Loss weights	Reconstruction weight w_r	0.2
Loss weights	Structure / semantic weights w_s, w_t	$w_s + w_t = 1$
Optimization	Optimizer	AdamW
Optimization	Learning rate	10^{-4}
Optimization	Weight decay	10^{-5}
Optimization	Learning rate schedule	Cosine annealing
Optimization	Batch size	256
Optimization	Maximum epochs	1000
Optimization	Early stopping patience	40

RQ-VAE Quantization. The unified embeddings are quantized using a 3-level RQ-VAE with latent dimension $d_e = 32$ and 256 codewords per level. The encoder is an MLP with layer dimensions

$$[1024, 2048, 1024, 512, 256, 128, 64, 32],$$

and the decoder mirrors this structure. Codebooks are initialized by K-Means on the latent embeddings. Optional Sinkhorn–Knopp balancing is applied during training to encourage uniform code utilization. The commitment coefficient is set to $\beta = 0.25$. After training, collisions are resolved by iterative reassignment on the final quantization level until all items have unique SID tuples.

Table 5.4: Hyperparameters for RQ-VAE quantization.

Module	Hyperparameter	Value
Encoder–decoder	Input dimension d_u	1024
Encoder–decoder	Latent dimension d_e	32
Encoder–decoder	Encoder architecture	[1024, 2048, 1024, 512, 256, 128, 64, 32]
Encoder–decoder	Decoder architecture	Mirror of encoder
Residual quantization	Number of quantization levels D	3
Residual quantization	Codewords per level N_c	256
Residual quantization	SID format	3-level code tuple
Training	Codebook initialization	K-Means
Training	Commitment coefficient β	0.25
Training	Balanced assignment	Sinkhorn–Knopp (optional)
Post-processing	Collision resolution	Reassignment on final level

Supervised Fine-tuning for GR. For all GR variants, we use `Qwen/Qwen2.5-3B-Instruct` as the downstream backbone model. The full training corpus is formed by concatenating five task-specific datasets: SID sequential recommendation, SID–title grounding, fusion sequential recommendation, title sequential recommendation, and title-to-SID sequential recommendation. Fine-tuning is performed with `accelerate`. The global batch size is 1024, the micro-batch size is 16, and the model is trained for

10 epochs. All LLM parameters are updated (`freeze_LLM=False`), gradient checkpointing is disabled, and the best checkpoint is selected according to validation performance.

Table 5.5: Hyperparameters for downstream generative recommendation.

Stage	Hyperparameter	Value
SFT backbone	LLM	Qwen/Qwen2.5-3B-Instruct
SFT backbone	Training framework	Accelerate + [DeepSpeed ZeRO Stage 1]
SFT backbone	Hardware	1 $\times$ NVIDIA H100 80GB
SFT backbone	Random seed	42
SFT training	Global batch size	1024
SFT training	Micro batch size	16
SFT training	Number of epochs	10
SFT training	Freeze LLM	False
SFT training	Gradient checkpointing	False
SFT training	Model selection	Best checkpoint on validation set
Inference	Decoding method	Constrained beam search
Inference	Beam size B	50
Inference	Length penalty	0.0
Inference	Max new tokens	256
Inference	Constraint structure	Prefix tree / trie over valid SID sequences

5.4. Evaluation Protocol

For conventional, graph-based, and sequential baselines, we rank candidate items according to the scores produced by each model.

Constrained Beam Search. For generative recommendation models, the fine-tuned LLM generates SID token sequences under prefix-tree constrained beam search [5]. Before evaluation, all valid SID strings are tokenized and inserted into a trie-structured hash map. During decoding, a custom `ConstrainedLogitsProcessor` retrieves the valid continuations for the current prefix and masks all invalid next tokens in log-softmax space. We use beam size $B = 50$, `length_penalty`$=0.0$, and `max_new_tokens`$=256$.

Metrics. We report Hit Ratio (**HR@**K) and Normalized Discounted Cumulative Gain (**NDCG@**K) at cut-offs $K \in \{5, 10, 20\}$. For each test instance, let r_i denote the rank position of the ground-truth target item in the predicted recommendation list. If the ground-truth item does not appear in the top-K results, its contribution is defined as zero.

The Hit Ratio at cut-off K is defined as

$$\mathrm{HR@}K = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(r_i \leq K), \tag{5.1}$$

where N is the number of test instances and $\mathbf{1}(\cdot)$ is the indicator function. HR@K measures whether the ground-truth item appears in the top-K recommendation list.

The Normalized Discounted Cumulative Gain at cut-off K is defined as

$$\mathrm{NDCG@}K = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathbf{1}(r_i \leq K)}{\log_2(r_i + 1)}. \tag{5.2}$$

NDCG@K further considers the rank position of the correct item: a hit at a higher rank receives a larger score than a hit at a lower rank.

In our setting, each test instance has exactly one ground-truth target item. Therefore, a prediction is counted as correct if the generated complete SID maps back to the ground-truth target item.

The corresponding results are presented in the following sections.

6

Results

6.1. Overall Performance (RQ1)

To answer RQ1, Table 6.1 compares the proposed UQF-based item tokenization and generative recommendation framework with traditional, graph-based, sequential, and state-of-the-art generative recommendation baselines on *Office Products* and *Musical Instruments*.

Table 6.1 presents the full performance comparison across all model families on both datasets.

From the results, we make the following observations.

Generative recommendation consistently outperforms conventional baselines. All GR models outperform the traditional, graph-based, and sequential baselines on both datasets. For example, on *Office Products*, UQF-LC-Rec achieves HR@10 of 0.1504, exceeding the strongest sequential baseline GRU4Rec (0.1387) by 8.4%. On *Musical Instruments*, the gap becomes larger: UQF-LC-Rec reaches HR@10 of 0.1107, compared with GRU4Rec's 0.0932 (+18.8%). This confirms the effectiveness of the generative recommendation paradigm over conventional recommenders.

UQF achieves the best overall performance across both datasets and both controlled backbone families. Within both the LC-Rec and TIGER families, the UQF-based variants consistently achieve the best results on all reported metrics. On *Office Products*, UQF-LC-Rec improves over LC-Rec from 0.1441 to 0.1504 in HR@10 (+4.4%) and from 0.1170 to 0.1234 in NDCG@20 (+5.5%). On *Musical Instruments*, the gains become larger, improving over LC-Rec by +7.3% in HR@10 and +13.3% in NDCG@20. The same pattern is observed for TIGER, where UQF-TIGER improves over TIGER by +5.2% / +6.6% on *Office Products* and +7.2% / +13.6% on *Musical Instruments* in HR@10 / NDCG@20, respectively. These results directly answer **RQ1**: improving item tokenization through UQF consistently enhances downstream generative recommendation.

UQF also surpasses EAGER, a strong current unified GR baseline. Beyond the controlled same-backbone comparisons, UQF further outperforms the scaled EAGER baseline on both datasets. On *Office Products*, UQF-LC-Rec improves over EAGER from 0.1438 to 0.1504 in HR@10 (+4.6%) and from 0.1152 to 0.1234 in NDCG@20 (+7.1%). On *Musical Instruments*, the gains are even larger, improving from 0.1047 to 0.1107 in HR@10 (+5.7%) and from 0.0800 to 0.0894 in NDCG@20 (+11.8%). UQF-TIGER shows a similar advantage over EAGER. This indicates that explicit pre-quantization semantic–collaborative fusion can be more effective than dual-stream generation with late confidence-based ranking.

UQF outperforms prior unified baselines from two different design paradigms. Compared with LETTER, which injects collaborative signals mainly through auxiliary regularization during token learning, UQF achieves superior results across the board. Compared with EAGER, which uses a dual-code and dual-decoder architecture, UQF still obtains better performance with a single unified code space.

Table 6.1: Overall performance comparison on *Office Products* and *Musical Instruments*. H@n = HR@n; N@n = NDCG@n. Within each LC-Rec/TIGER backbone family, the best result is in **bold** and the runner-up is underlined. EAGER is reported as an additional strong current unified generative baseline.

Category	Model	Office_Products						Musical_Instruments					
		H@5	H@10	H@20	N@5	N@10	N@20	H@5	H@10	H@20	N@5	N@10	N@20
Traditional	Pop	0.0331	0.0603	0.0925	0.0095	0.0133	0.0207	0.0274	0.0457	0.0551	0.0103	0.0164	0.0248
	ItemKNN	0.0198	0.0445	0.0633	0.0101	0.0147	0.0174	0.0439	0.0542	0.0698	0.0112	0.0124	0.0154
	BPR	0.0351	0.0554	0.0826	0.0175	0.0213	0.0261	0.0268	0.0430	0.0724	0.0104	0.0133	0.0176
Graph-based	LightGCN	0.0855	0.1097	0.1413	0.0831	0.0988	0.1059	0.0698	0.0833	0.0945	0.0603	0.0733	0.0776
Sequential	GRU4Rec	0.1172	0.1387	0.1635	0.0963	0.1032	0.1095	0.0772	0.0932	0.1143	0.0652	0.0703	0.0757
	Caser	0.0716	0.0847	0.1043	0.0563	0.0606	0.0655	0.0487	0.0590	0.0736	0.0412	0.0446	0.0482
	BERT4Rec	0.1078	0.1239	0.1426	0.0885	0.0938	0.0985	0.0691	0.0818	0.1006	0.0596	0.0637	0.0684
GR	EAGER	0.1249	0.1438	0.1678	0.1031	0.1092	0.1152	0.0884	0.1047	0.1256	0.0696	0.0748	0.0800
	LC-Rec	0.1280	0.1441	0.1636	0.1069	0.1121	0.1170	0.0842	0.1032	0.1243	0.0675	0.0736	0.0789
	LETTER-LC-Rec	0.1256	0.1447	0.1689	0.1037	0.1099	0.1159	0.0900	0.1059	0.1269	0.0705	0.0756	0.0809
	UQF-LC-Rec	**0.1309**	**0.1504**	**0.1774**	**0.1103**	**0.1166**	**0.1234**	**0.0936**	**0.1107**	**0.1331**	**0.0782**	**0.0837**	**0.0894**
	TIGER	0.1274	0.1436	0.1631	0.1061	0.1115	0.1163	0.0836	0.1025	0.1235	0.0668	0.0729	0.0781
	LETTER-TIGER	0.1263	0.1455	0.1696	0.1041	0.1104	0.1165	0.0893	0.1052	0.1261	0.0700	0.0751	0.0803
	UQF-TIGER	**0.1315**	**0.1510**	**0.1781**	**0.1108**	**0.1171**	**0.1240**	**0.0928**	**0.1099**	**0.1325**	**0.0776**	**0.0831**	**0.0887**

This suggests that the quality of item representation before quantization is more critical than merely introducing auxiliary fusion objectives or multi-stream generation after token construction.

The advantage is backbone-agnostic and primarily comes from better item tokenization. UQF provides consistent improvements on both LC-Rec and TIGER with comparable relative gains, confirming that its benefit lies upstream in the item tokenization stage rather than in a specific downstream decoder design. In other words, UQF acts as a plug-and-play enhancement for generative recommendation pipelines by improving the quality of the learned discrete item IDs.

The larger gains on Musical Instruments indicate stronger benefits from semantic–collaborative fusion in more behavior-sensitive domains. Compared with *Office Products*, the relative improvements of UQF over both semantic-only and unified baselines are consistently larger on *Musical Instruments*, especially on NDCG metrics. A likely reason is that item semantics alone are less sufficient to resolve fine-grained user preference patterns in this domain, making collaborative information particularly valuable for disambiguation. By fusing text and graph representations before quantization, UQF produces more behavior-aware and semantically coherent item IDs, which leads to stronger downstream gains.

6.1.1. Statistical Significance

To further verify that the observed gains are not due to random variation, we conduct **one-sided paired t-tests** on the controlled same-backbone comparisons, namely UQF-LC-Rec vs. LC-Rec and UQF-LC-Rec vs. LETTER-LC-Rec, under the directional hypothesis that UQF-LC-Rec performs better.

Following the evaluation protocol, the tests are conducted on three metric groups: **HR** ({1, 3, 5, 10, 20, 50}), **NDCG** ({1, 3, 5, 10, 20, 50}), and **ALL** (the union of both groups). In addition to p-values, we also report the mean absolute improvement, relative improvement percentage, paired Cohen's d, and the 95% confidence interval of the paired difference.

Table 6.2 shows that the improvements of UQF-LC-Rec are statistically reliable across both datasets. As a complementary visualization, Figure 6.1 presents a forest plot of the mean improvements and their 95% confidence intervals for UQF-LC-Rec against the two baselines across different metric groups.

All grouped comparisons are significant under the directional hypothesis. Across the two datasets, all one-sided paired t-tests yield $p < 0.05$, indicating that UQF-LC-Rec consistently outperforms both LC-Rec and LETTER-LC-Rec at the grouped-metric level. Most comparisons are substantially stronger,

Table 6.2: Statistical significance analysis of UQF-LC-Rec against LC-Rec and LETTER-LC-Rec using one-sided paired t-tests over grouped evaluation metrics. Mean Δ denotes the average absolute improvement of UQF-LC-Rec over the baseline within each metric group.

Dataset	Baseline	Group	Mean Δ	Rel. Δ (%)	p (one-sided)	Cohen's d	95% CI
Office_Products	LC-Rec	HR	0.0084	6.11	0.0271	1.02	[-0.0002, 0.0170]
	LC-Rec	NDCG	0.0048	4.53	0.0011	2.38	[0.0027, 0.0070]
	LC-Rec	ALL	0.0066	5.42	0.0014	1.11	[0.0028, 0.0104]
	LETTER-LC-Rec	HR	0.0058	4.16	0.0006	2.72	[0.0036, 0.0081]
	LETTER-LC-Rec	NDCG	0.0069	6.54	$< 10^{-6}$	16.25	[0.0064, 0.0073]
	LETTER-LC-Rec	ALL	0.0063	5.18	$< 10^{-6}$	4.04	[0.0053, 0.0073]
Musical_Instruments	LC-Rec	HR	0.0099	9.92	1.55×10^{-5}	5.80	[0.0081, 0.0117]
	LC-Rec	NDCG	0.0107	15.32	5.17×10^{-8}	18.33	[0.0101, 0.0113]
	LC-Rec	ALL	0.0103	12.15	7.76×10^{-12}	8.01	[0.0095, 0.0111]
	LETTER-LC-Rec	HR	0.0085	8.40	0.0049	1.65	[0.0031, 0.0139]
	LETTER-LC-Rec	NDCG	0.0095	13.35	4.43×10^{-5}	4.68	[0.0074, 0.0116]
	LETTER-LC-Rec	ALL	0.0090	10.44	2.35×10^{-6}	2.39	[0.0066, 0.0114]

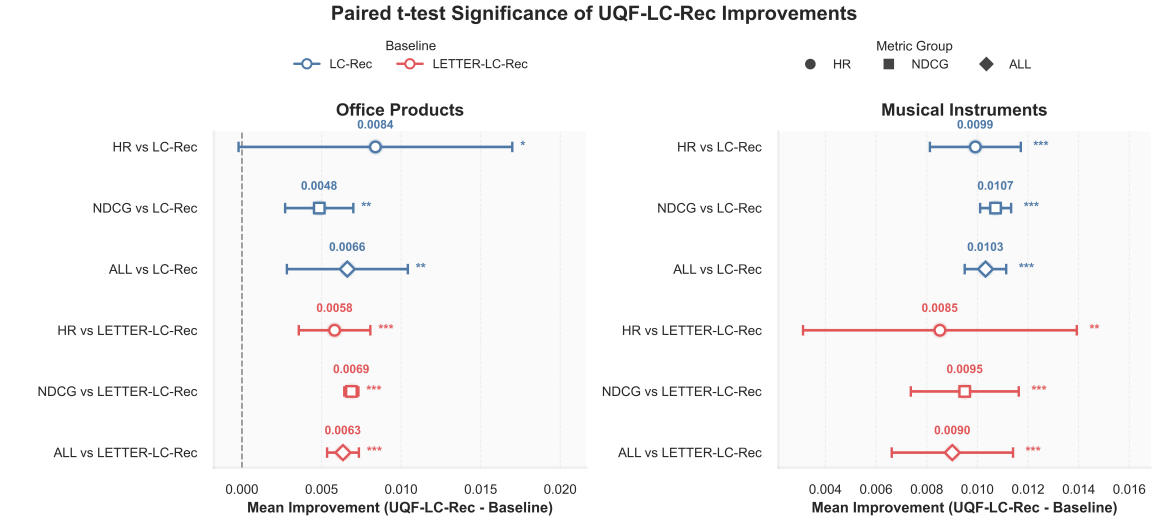


Figure 6.1: Forest plot of the paired t-test significance analysis for UQF-LC-Rec against LC-Rec and LETTER-LC-Rec on *Office Products* and *Musical Instruments*. Each point denotes the mean improvement of UQF-LC-Rec over the corresponding baseline within a metric group (HR, NDCG, or ALL), and the horizontal bars indicate the 95% confidence intervals.

reaching $p < 0.01$ or even $p < 0.001$. In particular, on *Musical Instruments*, all comparisons against LC-Rec and all but one comparison against LETTER-LC-Rec reach the strongest significance level.

The improvements are not only statistically significant but also practically meaningful. The paired Cohen's d values are consistently larger than 1.0, and in many cases much larger, indicating large effect sizes. For instance, against LC-Rec on *Musical Instruments*, UQF-LC-Rec achieves Cohen's $d = 5.80$ on HR, 18.33 on NDCG, and 8.01 on ALL metrics. Even on the comparatively milder *Office Products* dataset, the effect sizes remain substantial, ranging from $d = 1.02$ to $d = 16.25$. This shows that the observed gains are not only statistically detectable but also meaningful in magnitude.

The strongest evidence appears on Musical Instruments, consistent with the main-table trend. Against LC-Rec on *Musical Instruments*, the mean relative improvements reach 9.92% for HR, 15.32% for NDCG, and 12.15% overall. Against LETTER-LC-Rec, the corresponding gains remain strong at 8.40%, 13.35%, and 10.44%. These results match the overall-performance observation that UQF is particularly beneficial on the more challenging *Musical Instruments* dataset.

A small caveat appears only for the HR group on Office Products against LC-Rec. For this single comparison, the one-sided test is still significant ($p = 0.0271$), but the corresponding two-sided 95% confidence interval marginally crosses zero, i.e., $[-0.0002, 0.0170]$. This is likely due to the small number of paired observations in the HR group ($n = 6$ cutoffs). Nevertheless, the NDCG group and the combined ALL group on the same dataset are both clearly significant, with positive confidence intervals and large effect sizes.

Although EAGER is not included in the paired significance table, the main-results comparison in Table 6.1 still shows that both UQF-LC-Rec and UQF-TIGER outperform the scaled EAGER baseline on both datasets, providing further evidence that the proposed pre-quantization fusion strategy is competitive against current state-of-the-art unified generative recommenders.

Overall, the significance analysis corroborates the answer to RQ1: the performance gains of UQF are not accidental, but statistically and practically reliable across datasets and baselines.

6.2. Sensitivity Analysis (RQ2)

To answer the first part of RQ2, we investigate how the balance between structural and semantic supervision, i.e. collaborative and semantic weights affects downstream recommendation performance. Specifically, we vary the contrastive loss weights w_s (structure) and w_t (semantic) subject to $w_s + w_t = 1$, exploring $w_s \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ while fixing the reconstruction weight at $w_r = 0.2$. All other hyperparameters and downstream settings remain unchanged.

Figure 6.2 plots HR@{5,10} and NDCG@{5,10} as a function of w_s on both datasets.

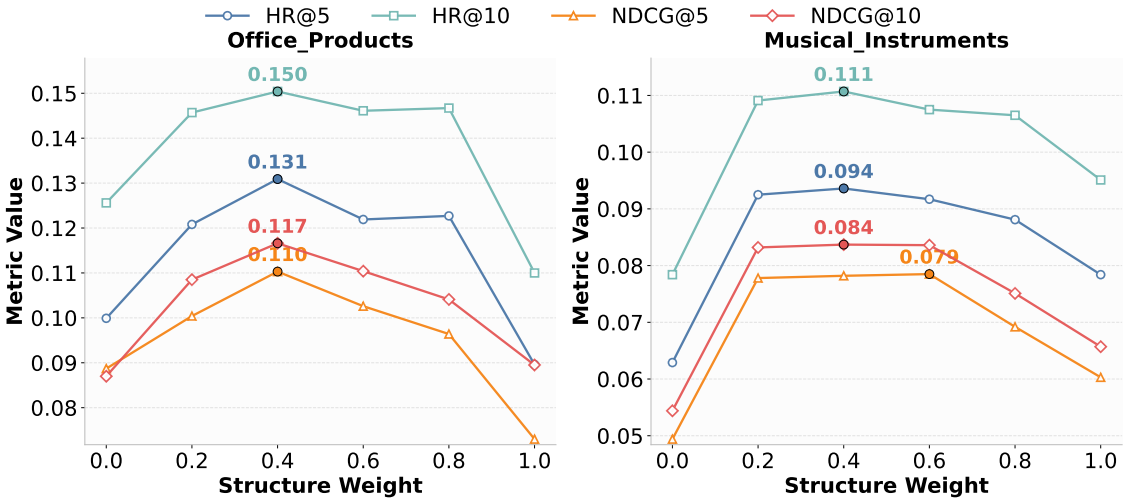


Figure 6.2: Sensitivity of UQF-LC-Rec to the structure–semantic weight ratio $w_s : w_t$ ($w_s + w_t = 1$). The x-axis shows w_s (structure weight); the y-axis shows metric values. HR@{5,10} are in blue tones; NDCG@{5,10} are in orange/red tones. Both datasets exhibit an inverted-U shape peaking near $w_s = 0.4$.

From the results, we make the following observations.

A mixed semantic–collaborative objective is clearly superior to either extreme. On both datasets, the best performance is achieved by intermediate weight settings rather than by either endpoint. For *Office Products*, the best configuration w_s=0.4, w_t=0.6 reaches HR@10 of 0.1504, compared with 0.1256 at the semantic-only contrastive extreme (w_s=0.0) and 0.1100 at the structure-only contrastive extreme (w_s=1.0). For *Musical Instruments*, the same best configuration reaches HR@10 of 0.1107, while the two extremes obtain only 0.0784 and 0.0951, respectively. This shows that neither semantic supervision nor structural supervision alone is sufficient to produce the best item codes; the gains of UQF come precisely from learning a unified representation under *joint* semantic and collaborative constraints.

Variant	Model	Office_Products						Musical_Instruments					
		H@5	H@10	H@20	N@5	N@10	N@20	H@5	H@10	H@20	N@5	N@10	N@20
Text	LC-Rec	0.1280	0.1441	0.1636	0.1069	0.1121	0.1170	0.0842	0.1032	0.1243	0.0675	0.0736	0.0789
Graph	LightGCN-LC-Rec	0.1198	0.1391	0.1728	0.0988	0.1050	0.1135	0.0839	0.1022	0.1242	0.0701	0.0760	0.0815
Concat	Concat-LC-Rec	0.1054	0.1280	0.1521	0.0859	0.0932	0.0993	0.0754	0.0905	0.1141	0.0608	0.0656	0.0716
Unified	**UQF-LC-Rec**	**0.1309**	**0.1504**	**0.1774**	**0.1103**	**0.1166**	**0.1234**	**0.0936**	**0.1107**	**0.1331**	**0.0782**	**0.0837**	**0.0894**
Text	TIGER	0.1274	0.1436	0.1631	0.1061	0.1115	0.1163	0.0836	0.1025	0.1235	0.0668	0.0729	0.0781
Graph	LightGCN-TIGER	0.1207	0.1400	0.1732	0.0994	0.1056	0.1140	0.0832	0.1016	0.1234	0.0695	0.0754	0.0808
Concat	Concat-TIGER	0.1060	0.1286	0.1529	0.0864	0.0938	0.1000	0.0748	0.0898	0.1133	0.0603	0.0650	0.0710
Unified	**UQF-TIGER**	**0.1315**	**0.1510**	**0.1781**	**0.1108**	**0.1171**	**0.1240**	**0.0928**	**0.1099**	**0.1325**	**0.0776**	**0.0831**	**0.0887**

Table 6.3: Ablation study of input representation strategies. Best in **bold**, runner-up underlined within each backbone family.

The optimal balance is consistent across datasets and slightly favors semantic supervision. Across both datasets, the best overall configuration is consistently observed at $w_s{=}0.4$, $w_t{=}0.6$. At this point, *Office Products* achieves the best values on all reported HR@{5,10} and NDCG@{5,10} metrics, and the same pattern holds for *Musical Instruments*. This indicates that the unified embedding should remain slightly more anchored to semantic structure, while still being substantially regularized by collaborative signals. A likely reason is that the final discrete IDs must ultimately serve as LLM-compatible symbolic representations, so preserving semantically coherent structure remains essential; collaborative information is most effective when injected as complementary behavioral guidance rather than as the sole optimization target.

Performance is robust within a broad mid-range of weight settings. Although $w_s{=}0.4$ is the global optimum, neighboring configurations also perform strongly. On *Office Products*, HR@10 varies only from 0.1457 to 0.1504 across $w_s \in \{0.2, 0.4, 0.6, 0.8\}$, corresponding to a relative spread of about 3.1%. On *Musical Instruments*, the corresponding HR@10 range is 0.1065 to 0.1107, a spread of about 3.8%. This robustness suggests that UQF does not rely on a narrowly tuned supervision ratio. Instead, once both semantic and collaborative objectives are simultaneously present, the gated fusion mechanism can adaptively redistribute their influence at the representation level.

The degradation pattern is dataset-dependent, revealing different roles of semantic and collaborative information. On *Office Products*, performance drops much more severely when structure dominates. For example, NDCG@5 decreases from 0.1103 at $w_s{=}0.4$ to 0.0730 at $w_s{=}1.0$, but only to 0.0887 at $w_s{=}0.0$. This suggests that semantic content provides a relatively reliable foundation for this dataset, while collaborative supervision mainly serves as complementary refinement. In contrast, on *Musical Instruments*, the semantic-only extreme is the weakest setting overall: HR@10 falls to 0.0784 at $w_s{=}0.0$, compared with 0.0951 at $w_s{=}1.0$. This implies that for musically related products, semantic metadata alone may be insufficient to distinguish fine-grained user preference patterns, making collaborative co-occurrence information particularly important for disambiguation. Taken together, these results indicate that the *optimal* balance is slightly semantic-leaning, but the *necessity* of collaborative supervision becomes more pronounced on harder and more behavior-dependent domains.

6.3. Ablation Study (RQ2)

To answer the second part of RQ2, we conduct an ablation study on the item representation strategy before quantization. Specifically, we compare four variants under both LC-Rec and TIGER backbones:

- **Text**: semantic embeddings generated from pretrained `Qwen3-Embedding-4B` only;
- **Graph**: graph embeddings generated from LightGCN only;
- **Concat**: direct concatenation of text and graph embeddings without explicit fusion;
- **Unified** (Ours): UQF-based semantic–collaborative fusion.

From Table 6.3, we draw the following conclusions.

Naïve concatenation is not sufficient and can even be harmful. Across both backbones and both datasets, the Concat variant consistently underperforms the Unified variant, and in most cases also underperforms the better single-modality baseline. For example, on *Office Products*, Concat-LC-Rec obtains HR@10 of only 0.1280, compared with 0.1441 for Text and 0.1391 for Graph. On *Musical Instruments*, Concat-LC-Rec reaches HR@10 of 0.0905, substantially below both Text (0.1032) and Graph (0.1022). A similar pattern holds for TIGER. This shows that simply merging heterogeneous embeddings at the feature level does not automatically yield a better tokenization space. Instead, unaligned semantic and collaborative signals can distort the geometry of the embedding manifold, making downstream quantization more difficult and reducing recommendation quality.

UQF fusion is the key factor that turns two modalities into a performance gain. In contrast to direct concatenation, the Unified variant consistently achieves the best results in every backbone family. On *Office Products*, UQF-LC-Rec improves over the strongest single-modality baseline (Text) from 0.1441 to 0.1504 in HR@10 (+4.4%), while UQF-TIGER improves over TIGER from 0.1436 to 0.1510 (+5.2%). On *Musical Instruments*, the gains are even larger: UQF-LC-Rec improves over the best single-modality baseline from 0.1032 to 0.1107 in HR@10 (+7.3%), and UQF-TIGER improves from 0.1025 to 0.1099 (+7.2%). These results demonstrate that the advantage does not come merely from using more input information, but from *how* the two modalities are fused. The parallel cross-attention and gated fusion design in UQF provides a better aligned representation space for subsequent RQ-VAE quantization.

Semantic and collaborative inputs exhibit clear complementarity. The single-modality variants reveal different strengths. On *Office Products*, the Text variant generally performs better at top-ranked metrics such as HR@10 and NDCG@10, whereas the Graph variant is competitive or superior at deeper cutoffs such as HR@20. On *Musical Instruments*, the Graph variant is particularly strong on NDCG metrics, e.g., LightGCN-LC-Rec achieves NDCG@10 of 0.0760 versus 0.0736 for Text. This indicates that semantic embeddings provide strong item understanding and top-rank discrimination, while collaborative embeddings provide broader behavioral coverage and additional preference structure. UQF is effective precisely because it exploits this complementarity rather than forcing one modality to dominate the other.

The ablation pattern is highly consistent across backbones. For both LC-Rec and TIGER, the same ordering is observed:

$$\text{Unified} > \text{Text / Graph} > \text{Concat}.$$

This consistency confirms that the benefit of UQF lies at the item tokenization stage rather than in any backbone-specific interaction. In other words, UQF improves the quality of the discrete item IDs themselves, and this benefit transfers reliably across different generative recommendation architectures.

6.4. LLM Scalability Analysis (RQ3)

To answer the first part of RQ3, we examine how the proposed framework scales with increasing LLM capacity. Using the same UQF-generated SIDs ($w_s{=}0.4$, $w_t{=}0.6$), we train UQF-LC-Rec with three Qwen2.5-Instruct backbones: **0.5B**, **1.5B**, and **3B** parameters. The quantitative results are reported in Table 6.4, while Figure 6.3 and Figure 6.4 visualize the scaling trend from complementary perspectives.

From Table 6.4, Figure 6.3, and Figure 6.4, we make the following observations.

Increasing LLM size generally improves recommendation performance. On *Office Products*, the trend is strictly monotonic across all reported metrics. For example, HR@10 increases from 0.1422 (0.5B) to 0.1482 (1.5B) and further to 0.1504 (3B), while HR@20 rises from 0.1603 to 0.1708 and then to 0.1774. The same positive scaling tendency is also observed on *Musical Instruments*, especially for top-ranked metrics such as HR@5, HR@10, NDCG@5, and NDCG@10. These results indicate that once high-quality UQF-based item IDs are available, larger language models can exploit them more effectively for downstream generative recommendation, which potentially inherit the established scaling laws in NLP [19].

Table 6.4: UQF-LC-Rec performance with different LLM backbone sizes.

	Office Products			Musical Instruments		
Size	**H@5**	**H@10**	**H@20**	**H@5**	**H@10**	**H@20**
0.5B	0.1231	0.1422	0.1603	0.0874	0.1063	0.1232
1.5B	0.1258	0.1482	0.1708	0.0912	0.1070	0.1334
3B	0.1309	0.1504	0.1774	0.0936	0.1107	0.1331
	N@5	**N@10**	**N@20**	**N@5**	**N@10**	**N@20**
0.5B	0.1052	0.1113	0.1159	0.0719	0.0780	0.0848
1.5B	0.1067	0.1139	0.1196	0.0732	0.0809	0.0896
3B	0.1103	0.1166	0.1234	0.0782	0.0837	0.0894

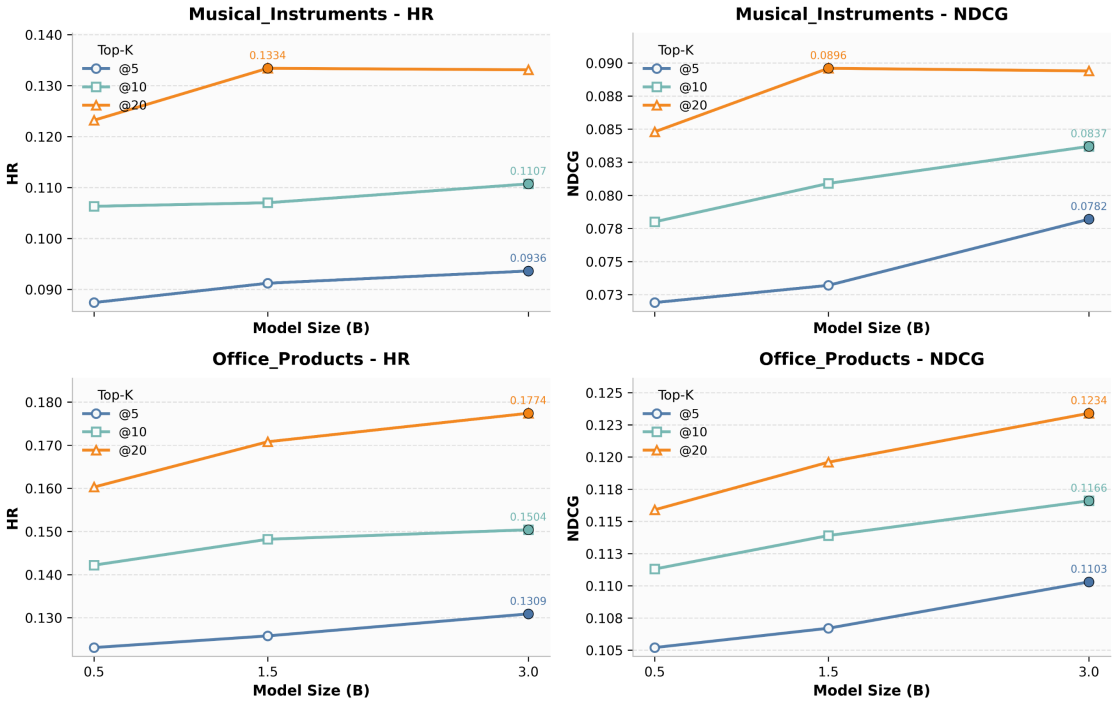


Figure 6.3: Scaling behavior of UQF-LC-Rec: HR@{5,10,20} and NDCG@{5,10,20} versus LLM backbone size (0.5B, 1.5B, 3B) on both datasets.

The overall scaling trend is positive, although some deeper-cutoff metrics show signs of saturation. While the global tendency remains upward, the gains are not perfectly uniform across all metrics. On *Musical Instruments*, HR@20 reaches 0.1334 at 1.5B and remains nearly unchanged at 0.1331 for 3B, and NDCG@20 changes only marginally from 0.0896 to 0.0894. This suggests that once the backbone reaches a certain size, performance at deeper ranking cutoffs may become increasingly constrained by dataset scale, supervision quality, or the information ceiling of the learned SID space, rather than by model capacity alone. Therefore, larger LLMs continue to help, but the marginal return is metric-dependent.

Scaling mainly strengthens the model's ability to exploit SID semantics and user history at top and medium cutoffs. The clearest gains from 1.5B to 3B appear on top-ranked recommendation quality rather than uniformly at all depths. For example, on *Musical Instruments*, moving from 1.5B to 3B improves HR@10 from 0.1070 to 0.1107 and NDCG@10 from 0.0809 to 0.0837, whereas HR@20 remains nearly flat. A similar but smoother pattern is observed on *Office Products*. This indicates that larger LLMs are particularly effective at refining the ranking of the most plausible next items, suggesting that better language modeling capacity helps the model make more precise use of the learned discrete

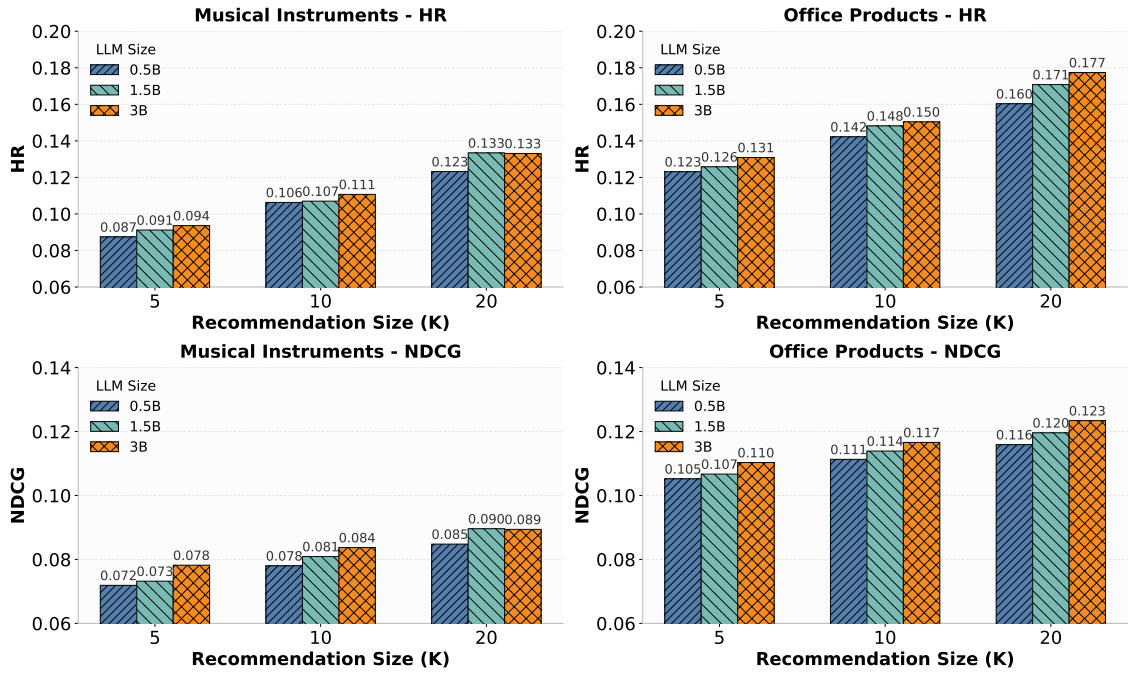


Figure 6.4: Performance comparison of UQF-LC-Rec across different LLM backbone sizes on *Office Products* and *Musical Instruments*. The bar chart complements Figure 6.3 by showing the absolute metric values at each model scale more directly.

item semantics.

Even compact LLMs remain competitive under the proposed tokenization framework. Despite being much smaller, the 0.5B variant already surpasses all traditional and sequential baselines on both datasets. For instance, on *Office Products*, the 0.5B model achieves HR@10 of 0.1422, higher than GRU4Rec's 0.1387; on *Musical Instruments*, it reaches 0.1063, substantially above GRU4Rec's 0.0932. This shows that the benefit of UQF is not limited to the large-model regime. By improving the quality of item tokenization itself, UQF provides a strong representational foundation that remains effective even when the downstream LLM is relatively compact.

6.5. SID Interpretability Analysis (RQ3)

To further answer RQ3, we analyze whether the semantic IDs (SIDs) learned by UQF preserve meaningful item structures after quantization. Since UQF fuses semantic and collaborative signals before constructing discrete item IDs, a desirable SID space should satisfy two properties. First, items sharing similar SID prefixes should be semantically related. Second, they should also exhibit similar user interaction patterns. Accordingly, we evaluate SID interpretability from two complementary perspectives: semantic consistency and collaborative consistency.

From Figure 6.5 and Figure 6.6, we make the following observations.

The learned SIDs preserve meaningful semantic category structure. Figure 6.5 shows that SID prefixes are clearly associated with real product categories on both datasets. On *Musical Instruments*, first-level SID tokens tend to group coherent product types such as ukulele-related items, guitar picks, guitar strings, and guitar straps. The second-level tokens further divide these coarse groups into more fine-grained subclusters, suggesting that the SID hierarchy captures category structure beyond flat item assignment.

A similar pattern is observed on *Office Products*. Items such as printer consumables, writing instruments, and drawing supplies are assigned to semantically meaningful regions in the SID space. Compared with *Musical Instruments*, the category boundaries are less visually separated, which is reasonable because many office-supply categories are naturally close in function and textual description. For

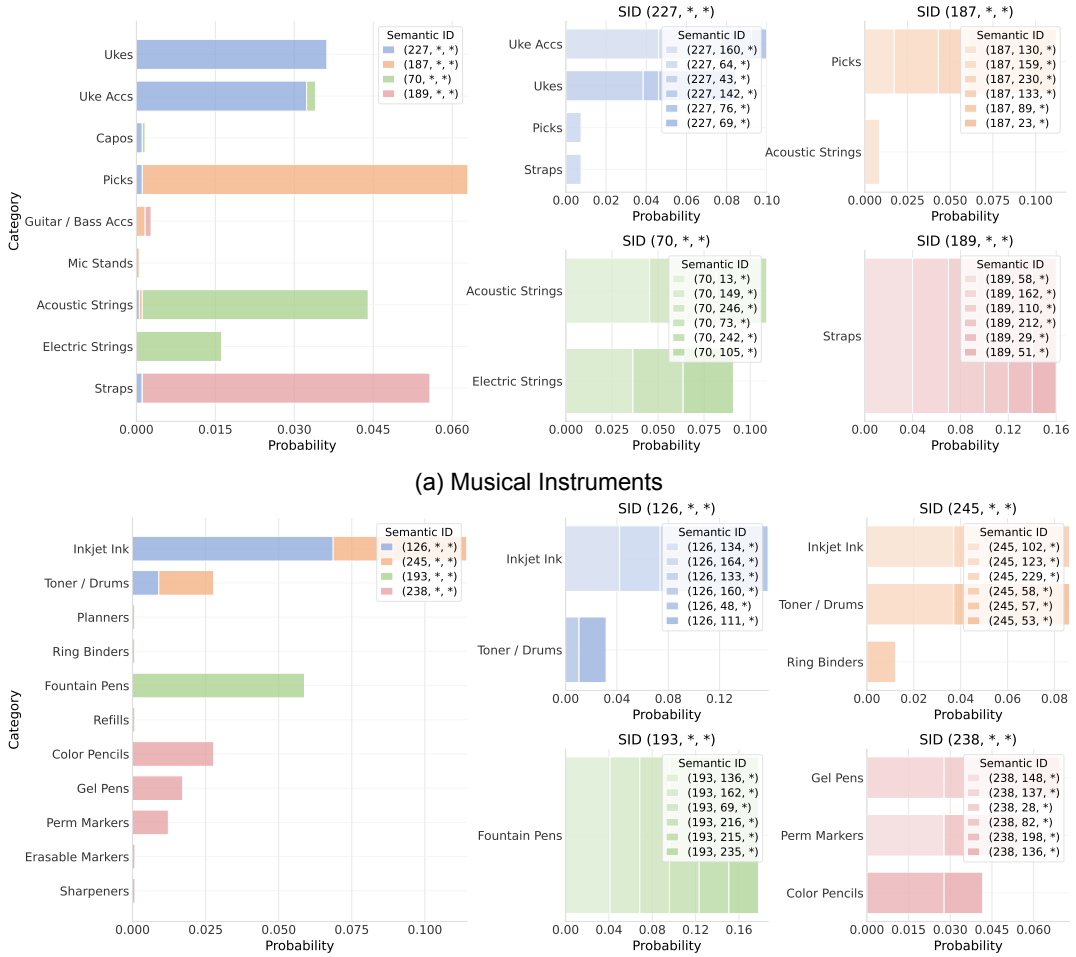


Figure 6.5: Semantic interpretability analysis of learned SIDs. The left part of each subfigure shows the correspondence between first-level SID token c_1 and item categories, while the right part shows how second-level token c_2 further partitions items under selected c_1 prefixes.

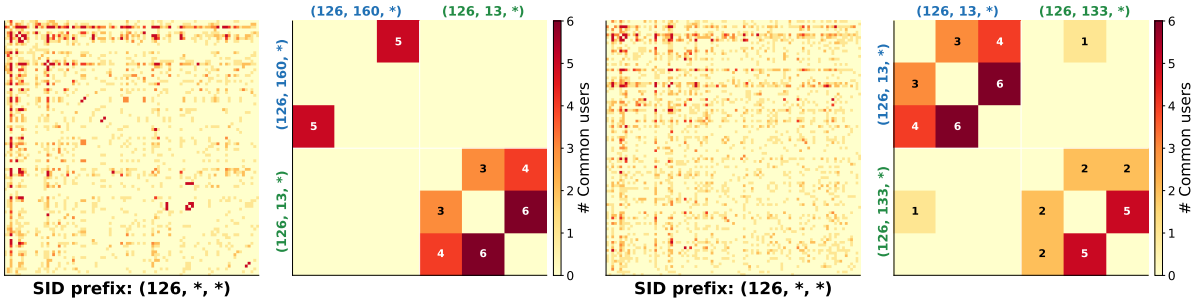
example, ink, toner, pens, and markers may share similar usage contexts, leading to partially overlapping SID regions. Nevertheless, the overall structure remains semantically coherent.

The SID hierarchy exhibits a coarse-to-fine organization. The first-level token c_1 captures relatively coarse item groups, while the second-level token c_2 provides additional refinement within each group. This hierarchical behavior is particularly visible in *Musical Instruments*, where category boundaries are more distinct. In *Office Products*, some second-level partitions appear less separated by the available category labels, but this does not necessarily indicate weak interpretability. Instead, the lower-level SID tokens may capture finer item attributes, such as brand, function, style, or usage scenario, which are not fully reflected by the coarse metadata categories. Therefore, the learned SID space preserves semantic structure at multiple granularities.

Items sharing SID prefixes also show collaborative consistency. To examine whether SIDs encode behavioral information, we visualize item–item user-overlap matrices in Figure 6.6. For two items i and j, the user overlap is defined as:

$$\text{Overlap}(i, j) = |U_i \cap U_j|, \tag{6.1}$$

where U_i and U_j denote the sets of users who interacted with the two items. A larger value indicates stronger collaborative similarity.



(a) Musical Instruments (b) Office Products

Figure 6.6: Collaborative interpretability analysis of learned SIDs based on item–item user overlap. For each dataset, the left panel shows within-prefix user overlap among items sharing the same first-level SID prefix (e.g., $(126, *, *)$). The right panel compares within-prefix and cross-prefix user overlap for sibling prefixes under the same first-level prefix (e.g., $(126, 160, *)$ and $(126, 13, *)$).

On *Musical Instruments*, the L1-prefix matrix contains many non-zero overlap values, indicating that items assigned to the same first-level SID group are often consumed by overlapping user groups. The L2 sibling-prefix matrix further shows stronger within-prefix overlap than cross-prefix overlap, suggesting that second-level SID tokens form finer collaborative subclusters under the same coarse SID prefix.

A similar pattern is observed on *Office Products*. Items sharing the same first-level SID prefix also exhibit visible user-overlap signals, and the second-level sibling-prefix comparison shows that within-prefix regions are generally stronger than cross-prefix regions. This indicates that the learned SID hierarchy preserves collaborative structure across both datasets. Meanwhile, the existence of weak cross-prefix overlap is also reasonable, because sibling second-level prefixes still share the same first-level token and therefore remain related at a coarser level.

UQF produces item IDs that are both language-compatible and behavior-aware. The semantic and collaborative visualizations jointly show that the learned SIDs are not arbitrary discrete codes. From the semantic perspective, SID prefixes correspond to meaningful product categories and form a coarse-to-fine hierarchy. From the collaborative perspective, items sharing SID prefixes also tend to have overlapping user groups, indicating that the SID hierarchy reflects user preference structure.

These results complement the quantitative findings in RQ1 and the ablation results in RQ2. They suggest that UQF improves recommendation performance by learning better item tokenization rather than merely relying on downstream model capacity. By fusing semantic and collaborative signals before quantization, UQF constructs discrete item IDs that are semantically coherent, behavior-aware, and suitable for generative recommendation.

7

Discussion

This chapter discusses the main findings of this work from the perspective of item representation learning for LLM-based generative recommendation. Rather than merely restating the experimental results, we focus on what these results imply about semantic–collaborative item tokenization, why the proposed UQF is effective, and what limitations and future directions remain.

7.1. Main Findings

The experimental results provide clear answers to the three research questions raised in this work.

For **RQ1**, the results show that UQF-based item tokenization consistently improves downstream generative recommendation performance over semantic-only baselines, collaborative-only variants, and strong unified baselines such as LETTER and EAGER. These improvements are observed on both datasets and under both LC-Rec and TIGER backbones, indicating that the proposed method is not tied to a specific generative architecture.

For **RQ2**, the sensitivity analysis and ablation study together show that semantic and collaborative information are both indispensable. The best performance is achieved not by either single-modality extreme, but by a balanced fusion setting that slightly favors semantic supervision. Moreover, naive concatenation of text and graph embeddings fails to deliver gains, while explicit cross-modal fusion through UQF consistently improves performance. This demonstrates that the key issue is not whether both modalities are used, but how they are fused.

For **RQ3**, the scalability analysis shows that the proposed framework continues to benefit from increasing LLM capacity, while remaining competitive even with compact backbones. This suggests that improving item tokenization quality is not only beneficial in the large-model regime, but also valuable under more practical deployment constraints. Together, these findings support the central claim of this work: *the quality of item representation before quantization is a critical determinant of downstream generative recommendation performance*.

7.2. Why UQF Works

A key reason for the effectiveness of UQF is that it performs **explicit semantic–collaborative fusion before quantization**. This differs fundamentally from approaches that quantize semantic embeddings first and only later inject collaborative information through auxiliary supervision or alignment objectives. Since the final discrete IDs are produced by RQ-VAE, the geometry of the embedding space entering the quantizer directly determines the quality of the learned code tuples. By learning a unified embedding before quantization, UQF allows the resulting SIDs to encode both semantic meaning and behavioral relevance at the same time.

Another important factor is the use of **sequence-level representations** from both modalities. On the semantic side, token-level text embeddings preserve fine-grained textual cues such as category terms, attributes, and domain-specific descriptors. On the collaborative side, retaining all LightGCN layer

outputs preserves structural information at different propagation depths, ranging from local identity to higher-order neighborhood patterns. This design enables the learnable queries in UQF to attend selectively to informative text tokens and graph layers, rather than being forced to operate on already-collapsed single vectors.

The **parallel cross-attention with gated fusion** mechanism is also critical. The ablation results show that simply concatenating heterogeneous embeddings can even reduce performance, suggesting that raw multi-source information may introduce representational conflict rather than complementarity. UQF addresses this issue by aligning the two modalities through query-based interaction and then adaptively controlling their relative contribution with a fine-grained gate. As a result, the model does not treat semantic and collaborative information as equally important for all items. Instead, it learns an item-dependent balance between the two sources.

Finally, the **hybrid contrastive objective** further strengthens the unified space. By simultaneously pulling the anchor representation toward semantic neighbors and structural neighbors, UQF is encouraged to preserve both content similarity and collaborative proximity. This provides a more direct supervision signal for representation unification than relying on a single regularization term. The sensitivity analysis further confirms that the best performance emerges when both types of supervision are present.

7.3. Implications for Item Representation in Generative Recommendation

The findings of this work suggest that item representation in generative recommendation should not be viewed as a purely semantic indexing problem or a purely collaborative indexing problem. Instead, it should be understood as a **dual-objective representation problem**, where the learned discrete item identifiers must satisfy two requirements simultaneously.

First, they need to remain compatible with the language modeling process. This requires item IDs to preserve meaningful semantic organization, so that the LLM can learn stable and generalizable generation patterns over them. Second, they also need to reflect user preference structure. This requires item IDs to capture collaborative regularities that are often not explicitly available from item metadata alone.

The results imply that high-quality generative recommendation depends not only on the capacity of the downstream LLM, but also on whether the item code space itself is well structured. From this perspective, UQF can be viewed as an upstream representation learning module that improves the quality of the discrete symbol system on which downstream generation operates. This also explains why the gains of UQF are relatively backbone-agnostic: once the item tokenization space is improved, different generative backbones can all benefit.

The stronger gains observed on *Musical Instruments* are also informative. Compared with *Office Products*, this dataset likely contains more subtle preference patterns and finer-grained item distinctions that cannot be fully resolved by text semantics alone. In such cases, collaborative information becomes especially useful for disambiguation. This indicates that unified tokenization may be particularly valuable in domains where semantic metadata is informative but insufficient for capturing user behavior.

7.4. Practical Implications

From a practical perspective, the proposed framework has two attractive properties.

First, UQF is **modular**. It improves the item representation and quantization stages without requiring fundamental changes to the downstream generative recommender. This is supported by the consistent improvements observed under both LC-Rec and TIGER backbones. Therefore, UQF can be regarded as a plug-and-play enhancement for existing SID-based generative recommendation pipelines.

Second, the framework remains useful even with relatively small LLM backbones. The scalability analysis shows that although larger LLMs achieve stronger performance, the 0.5B model already surpasses conventional baselines when equipped with UQF-generated SIDs. This suggests that better tokeniza-

tion can partially compensate for limited model size, which is important for practical systems with deployment and inference constraints.

7.5. Limitations

Despite its effectiveness, this work still has several limitations.

First, the experiments are conducted on two Amazon benchmark datasets, which, although widely used, do not cover the full diversity of recommendation domains. The generality of the findings should therefore be further validated on larger-scale and more heterogeneous datasets.

Second, the current semantic modality is primarily text-based. Although the framework is conceptually compatible with richer multimodal inputs, such as images or audio, the present study only investigates text plus graph fusion. As a result, the proposed “unified” representation is still limited to a two-modality setting.

Third, the fusion stage and the quantization stage are trained sequentially rather than jointly in a fully end-to-end manner. While this improves modularity and engineering simplicity, it may prevent the quantizer and the fusion encoder from reaching a globally optimal coordination.

Fourth, although the SID analysis provides some evidence of structural coherence, the interpretability of the learned code space has not yet been studied in a fully systematic manner. In particular, more quantitative analyses of codebook utilization, prefix purity, semantic cohesion, and collaborative cohesion would further strengthen the understanding of the learned discrete representations.

7.6. Future Work

Several directions may extend this work in the future.

First, the current framework can be generalized from text–graph fusion to richer multimodal item tokenization. In the present study, the semantic modality is primarily text-based, while the collaborative modality is derived from the interaction graph. In many real-world recommendation scenarios, however, items are associated with more diverse side information, such as product images, short video content, audio signals, or structured attributes. Extending UQF from text–image–graph fusion to more general multimodal fusion may further improve the quality of unified item representations, especially in domains where textual metadata alone is insufficient to capture item semantics.

Second, the fusion stage and the quantization stage may be optimized in a more tightly coupled or end-to-end manner. In the current framework, UQF is first trained to produce unified embeddings, which are then quantized by a subsequent RQ-VAE module. Although this modular design is simple and practical, it may limit the extent to which representation learning and discrete code assignment can adapt to each other. A promising future direction is to investigate joint training strategies that allow the fusion encoder and the quantizer to be optimized together, so that the learned unified space is more directly aligned with the downstream discrete item ID space.

Third, the interpretability of the learned Semantic IDs deserves more systematic investigation. While the current analysis provides preliminary evidence that UQF-generated SIDs capture meaningful structure, a deeper study of the code space remains necessary. Future work may introduce more explicit quantitative measures, such as codebook utilization balance, prefix purity, semantic cohesion, collaborative cohesion, and cross-prefix behavioral consistency. Such analyses would help better understand how semantic and collaborative knowledge are distributed across different code levels and prefixes.

Fourth, the current experiments are conducted on relatively small public datasets containing only tens of thousands of interactions, which remain far from the scale and richness of real industrial recommendation systems. In practical applications, recommendation platforms usually involve vastly larger interaction logs and much richer user feedback signals, such as clicks, add-to-cart actions, purchases, watch time, dwell time, skips, likes, and other implicit or explicit behavioral feedback. Therefore, an important future direction is to validate the proposed framework under industrial-scale settings and explore how unified item tokenization behaves when exposed to much denser and more diverse supervision signals.

Relatedly, future work may combine the proposed framework with post-training techniques inspired by RLHF and preference alignment in industrial scenarios, where real user feedback signals—such as clicks and purchases in e-commerce, or watch duration and engagement signals in video recommendation—can be treated as preference supervision for post-training and policy refinement [2, 30, 34]. Under such a setting, UQF-generated item IDs would provide a structured discrete action space, while preference tuning could further align the downstream generative recommender with actual user utility. This direction is also consistent with recent generative recommendation work such as OneRec, which incorporates iterative preference alignment and DPO-style optimization in an industrial setting [6].

Finally, it would also be valuable to study the interaction between unified item tokenization and stronger downstream alignment frameworks. Beyond standard supervised fine-tuning, future work may investigate how the proposed item representation can be integrated with direct preference optimization, reinforcement learning, or other recommendation-oriented alignment strategies, so that both the item code space and the generative policy can be optimized toward long-term user satisfaction.

Overall, these directions suggest that unified semantic–collaborative item quantization is not only a standalone research problem, but also a promising foundation for larger-scale, multimodal, and preference-aligned generative recommendation systems.

8

Conclusion

This thesis studies the problem of item representation in LLM-based generative recommendation, with a particular focus on how semantic and collaborative information should be integrated before vector quantization. Existing approaches largely construct item identifiers from either semantic content or collaborative interactions alone, or incorporate the second modality only in a limited auxiliary manner. Motivated by this gap, this work proposes a unified item quantization framework centered on the **Unified Q-Former (UQF)**.

The proposed UQF learns a unified item embedding from sequence-level text embeddings and graph embeddings through query-based parallel cross-attention, adaptive gated fusion, modality dropout, and hybrid contrastive learning. The unified embeddings are then quantized into discrete Semantic IDs through RQ-VAE and used for downstream generative recommendation under LC-Rec and TIGER style pipelines. In this way, semantic and collaborative information are fused explicitly at the representation level before item tokenization.

Extensive experiments show that the proposed method consistently improves recommendation performance across datasets and generative backbones, and outperforms strong unified generative baselines including LETTER and the scaled EAGER framework. The sensitivity analysis demonstrates that the best results are achieved when semantic and collaborative supervision are balanced rather than used in isolation. The ablation study confirms that explicit fusion is necessary, and that naive concatenation is insufficient. The scalability analysis further shows that the proposed framework benefits from larger LLM backbones while remaining competitive even with compact models. Taken together, these results provide strong evidence that improving pre-quantization item representation is an effective way to enhance generative recommendation.

The main conclusion of this work is that the success of LLM-based generative recommendation depends not only on the downstream language model, but also on the quality of the discrete item representation space on which generation is performed. By explicitly fusing semantic meaning and collaborative preference structure before quantization, UQF provides a practical and effective path toward better item tokenization for generative recommendation.

References

[1] Heng-Tze Cheng et al. “Wide & deep learning for recommender systems”. In: *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016, pp. 7–10.

[2] Paul F Christiano et al. “Deep reinforcement learning from human preferences”. In: *Advances in neural information processing systems* 30 (2017).

[3] Paul Covington, Jay Adams, and Emre Sargin. “Deep neural networks for youtube recommendations”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 191–198.

[4] Zeyu Cui et al. “M6-rec: Generative pretrained language models are open-ended recommender systems”. In: *arXiv preprint arXiv:2205.08084* (2022).

[5] Nicola De Cao et al. “Autoregressive entity retrieval”. In: *arXiv preprint arXiv:2010.00904* (2020).

[6] Jiaxin Deng et al. “Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment”. In: *arXiv preprint arXiv:2502.18965* (2025).

[7] Shijie Geng et al. “Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5)”. In: *Proceedings of the 16th ACM conference on recommender systems*. 2022, pp. 299–315.

[8] Carlos A Gomez-Uribe and Neil Hunt. “The netflix recommender system: Algorithms, business value, and innovation”. In: *ACM Transactions on Management Information Systems (TMIS)* 6.4 (2015), pp. 1–19.

[9] Huifeng Guo et al. “DeepFM: a factorization-machine based neural network for CTR prediction”. In: *arXiv preprint arXiv:1703.04247* (2017).

[10] Xiangnan He et al. “Lightgcn: Simplifying and powering graph convolution network for recommendation”. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 639–648.

[11] Xiangnan He et al. “Neural collaborative filtering”. In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[12] Xinran He et al. “Practical lessons from predicting clicks on ads at facebook”. In: *Proceedings of the eighth international workshop on data mining for online advertising*. 2014, pp. 1–9.

[13] Balázs Hidasi et al. “Session-based recommendations with recurrent neural networks”. In: *arXiv preprint arXiv:1511.06939* (2015).

[14] Min Hou et al. “A survey on generative recommendation: Data, model, and tasks”. In: *arXiv preprint arXiv:2510.27157* (2025).

[15] Yupeng Hou et al. “Bridging language and items for retrieval and recommendation”. In: *arXiv preprint arXiv:2403.03952* (2024).

[16] Wenyue Hua et al. “How to index item ids for recommendation foundation models”. In: *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*. 2023, pp. 195–204.

[17] Yuchin Juan et al. “Field-aware factorization machines for CTR prediction”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 43–50.

[18] Wang-Cheng Kang and Julian McAuley. “Self-attentive sequential recommendation”. In: *2018 IEEE international conference on data mining (ICDM)*. IEEE. 2018, pp. 197–206.

[19] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009), pp. 30–37.

[21] Doyup Lee et al. “Autoregressive image generation using residual quantization”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 11523–11532.

[22] Jiacheng Li, Yujie Wang, and Julian McAuley. “Time interval aware self-attention for sequential recommendation”. In: *Proceedings of the 13th international conference on web search and data mining*. 2020, pp. 322–330.

[23] Jing Li et al. “Neural attentive session-based recommendation”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 1419–1428.

[24] Junnan Li et al. “Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models”. In: *International conference on machine learning*. PMLR. 2023, pp. 19730–19742.

[25] Yongqi Li et al. “A survey of generative search and recommendation in the era of large language models”. In: *arXiv preprint arXiv:2404.16924* (2024).

[26] Qiao Liu et al. “STAMP: short-term attention/memory priority model for session-based recommendation”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 1831–1839.

[27] Weiwen Liu et al. “Neural re-ranking in multi-stage recommender systems: A review”. In: *arXiv preprint arXiv:2202.06602* (2022).

[28] Jianmo Ni, Jiacheng Li, and Julian McAuley. “Justifying recommendations using distantly-labeled reviews and fine-grained aspects”. In: *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 2019, pp. 188–197.

[29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: *arXiv preprint arXiv:1807.03748* (2018).

[30] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[31] Aleksandr V Petrov and Craig Macdonald. “Generative sequential recommendation with gptrec”. In: *arXiv preprint arXiv:2306.11114* (2023).

[32] Haohao Qu et al. “Tokenrec: Learning to tokenize id for llm-based generative recommendations”. In: *IEEE Transactions on Knowledge and Data Engineering* (2025).

[33] Rafael Rafailov et al. “Direct preference optimization: Your language model is secretly a reward model”. In: *Advances in neural information processing systems* 36 (2023), pp. 53728–53741.

[34] Rafael Rafailov et al. “Direct preference optimization: Your language model is secretly a reward model”. In: *Advances in neural information processing systems* 36 (2023), pp. 53728–53741.

[35] Shashank Rajput et al. “Recommender systems with generative retrieval”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 10299–10315.

[36] Steffen Rendle. “Factorization machines”. In: *2010 IEEE International conference on data mining*. IEEE. 2010, pp. 995–1000.

[37] Steffen Rendle et al. “BPR: Bayesian personalized ranking from implicit feedback”. In: *arXiv preprint arXiv:1205.2618* (2012).

[38] Paul Resnick et al. “Grouplens: An open architecture for collaborative filtering of netnews”. In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. 1994, pp. 175–186.

[39] Badrul Sarwar et al. “Item-based collaborative filtering recommendation algorithms”. In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pp. 285–295.

[40] Upendra Shardanand and Pattie Maes. “Social information filtering: Algorithms for automating ”word of mouth””. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1995, pp. 210–217.

[41] Richard Sinkhorn. “Diagonal equivalence to matrices with prescribed row and column sums”. In: *The American Mathematical Monthly* 74.4 (1967), pp. 402–405.

[42] Richard Sinkhorn and Paul Knopp. “Concerning nonnegative matrices and doubly stochastic matrices”. In: *Pacific Journal of Mathematics* 21.2 (1967), pp. 343–348.

[43] Fei Sun et al. “BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 1441–1450.

[44] Jiaxi Tang and Ke Wang. “Personalized top-n sequential recommendation via convolutional sequence embedding”. In: *Proceedings of the eleventh ACM international conference on web search and data mining*. 2018, pp. 565–573.

[45] Wenjie Wang et al. “Learnable item tokenization for generative recommendation”. In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 2024, pp. 2400–2409.

[46] Ye Wang et al. “Eager: Two-stream generative recommender with behavior-semantic collaboration”. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024, pp. 3245–3254.

[47] Shu Wu et al. “Session-based recommendation with graph neural networks”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 346–353.

[48] Chengfeng Xu et al. “Graph contextualized self-attention network for session-based recommendation.” In: *Ijcai*. Vol. 19. 2019. 2019, pp. 3940–3946.

[49] Zhen Yang, Haitao Lin, Ziji Zhang, et al. “Gr-llms: Recent advances in generative recommendation based on large language models”. In: *arXiv preprint arXiv:2507.06507* (2025).

[50] Fajie Yuan et al. “A simple convolutional generative network for next item recommendation”. In: *Proceedings of the twelfth ACM international conference on web search and data mining*. 2019, pp. 582–590.

[51] Wayne Xin Zhao et al. “Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms”. In: *proceedings of the 30th acm international conference on information & knowledge management*. 2021, pp. 4653–4664.

[52] Bowen Zheng et al. “Adapting large language models by integrating collaborative semantics for recommendation”. In: *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE. 2024, pp. 1435–1448.

[53] Guorui Zhou et al. “Onerec technical report”. In: *arXiv preprint arXiv:2506.13695* (2025).

[54] Guorui Zhou et al. “Onerec-v2 technical report”. In: *arXiv preprint arXiv:2508.20900* (2025).

A

Declaration of Generative AI Usage

Following the university's academic honesty and transparency guidelines, this chapter lists the generative AI tools used during this project. As the author, I keep full creative and intellectual responsibility for all the text, code, and final results in this thesis.

A.1. Coding and Data Analysis Support

AI tools were used as assistants to speed up coding tasks and help with data processing without replacing the core research contribution.

- **Tools Used**: GitHub Copilot, Google Antigravity, and OpenAI Codex.
- **Scope of Work**: These tools helped write boilerplate code, fix bugs, and make the code easier to read. They were also used to create scripts for data analysis and charts for data visualization. All generated code was manually tested and verified before use.

A.2. Writing and Language Support

AI assistants were used to improve the clarity and quality of the writing while keeping my original ideas intact.

- **Tools Used**: Google Gemini and OpenAI ChatGPT.
- **Scope of Work**: These tools were used to check grammar, improve academic style, and make text easier to read. They also helped reword complex sentences for better clarity. All final arguments and text structures remain my own work.

A.3. Data Privacy and Verification Statement

Because generative AI tools can sometimes create incorrect information or false references, I manually checked all facts, math formulas, and literature citations to ensure accuracy. Additionally, no private, sensitive, or confidential datasets were uploaded to any external AI tools during this project.

B

Source Code

This appendix presents the core implementation of the proposed Unified Q-Former (UQF) framework. The implementation of the UQF model and its training pipeline are originally developed for this work, while the residual quantization (RQ-VAE) and generative recommendation modules are adapted from existing open-source implementations of prior works. The code is organized according to the following pipeline stages: (1) collaborative embedding extraction via LightGCN, (2) semantic embedding extraction via a pre-trained language model, (3) Q-Former fusion model, (4) UQF training with hybrid contrastive learning, (5) residual quantization (RQ-VAE), (6) SID generation and collision resolution, and (7) generative recommendation via LLM supervised fine-tuning. Full codes are available at `https://github.com/github-bowen/Unified-Q-Former-LLM4REC`.

B.1. LightGCN for Collaborative Embedding Extraction

The `LightGCN` module implements graph-based collaborative filtering on the user–item interaction graph. It performs multi-layer graph convolution and optionally returns all layer embeddings $\mathbf{G}_i = [\mathbf{q}_i^{(0)}; \ldots; \mathbf{q}_i^{(J)}]$.

Listing B.1: LightGCN model (`rq/unified-emb/lightgcn.py`).

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import scipy.sparse as sp
import numpy as np


class LightGCN(nn.Module):
    def __init__(self, num_users, num_items, embedding_dim=64,
                 num_layers=3, dropout=0.0):
        super(LightGCN, self).__init__()
        self.num_users = num_users
        self.num_items = num_items
        self.embedding_dim = embedding_dim
        self.num_layers = num_layers
        self.dropout = dropout

        self.user_embedding = nn.Embedding(num_users, embedding_dim)
        self.item_embedding = nn.Embedding(num_items, embedding_dim)
        self._init_weights()
        self.adj_matrix = None

    def _init_weights(self):
        nn.init.xavier_uniform_(self.user_embedding.weight)
        nn.init.xavier_uniform_(self.item_embedding.weight)

    def set_adj_matrix(self, adj_matrix):
        self.adj_matrix = adj_matrix

```

```
    def forward(self, return_all_layers=False):
        all_embeddings = torch.cat([
            self.user_embedding.weight,
            self.item_embedding.weight
        ], dim=0)

        embeddings_list = [all_embeddings]
        adj_matrix = self.adj_matrix

        for layer in range(self.num_layers):
            all_embeddings = torch.sparse.mm(adj_matrix, all_embeddings)
            embeddings_list.append(all_embeddings)

        if return_all_layers:
            all_embeddings_stacked = torch.stack(embeddings_list, dim=0)
            user_embeddings = all_embeddings_stacked[:, :self.num_users, :]
            item_embeddings = all_embeddings_stacked[:, self.num_users:, :]
            return user_embeddings, item_embeddings
        else:
            all_embeddings = torch.stack(embeddings_list, dim=0)
            all_embeddings = torch.mean(all_embeddings, dim=0)
            user_embeddings = all_embeddings[:self.num_users]
            item_embeddings = all_embeddings[self.num_users:]
            return user_embeddings, item_embeddings

    def bpr_loss(self, user_ids, pos_item_ids, neg_item_ids):
        user_embeddings, item_embeddings = self.forward()
        user_emb = user_embeddings[user_ids]
        pos_item_emb = item_embeddings[pos_item_ids]
        neg_item_emb = item_embeddings[neg_item_ids]

        pos_scores = torch.sum(user_emb * pos_item_emb, dim=1)
        neg_scores = torch.sum(user_emb * neg_item_emb, dim=1)
        loss = -torch.mean(F.logsigmoid(pos_scores - neg_scores))

        user_emb_init = self.user_embedding(user_ids)
        pos_item_emb_init = self.item_embedding(pos_item_ids)
        neg_item_emb_init = self.item_embedding(neg_item_ids)
        reg_loss = (
            torch.norm(user_emb_init, 2).pow(2) +
            torch.norm(pos_item_emb_init, 2).pow(2) +
            torch.norm(neg_item_emb_init, 2).pow(2)
        ) / (2 * len(user_ids))

        return loss, reg_loss


def create_adj_matrix(num_users, num_items, user_item_pairs, device='cpu'):
    n_nodes = num_users + num_items
    users = [pair[0] for pair in user_item_pairs]
    items = [pair[1] + num_users for pair in user_item_pairs]

    row = users + items
    col = items + users
    data = np.ones(len(row))
    adj = sp.coo_matrix((data, (row, col)), shape=(n_nodes, n_nodes))

    rowsum = np.array(adj.sum(axis=1)).flatten()
    d_inv_sqrt = np.power(rowsum, -0.5)
    d_inv_sqrt[np.isinf(d_inv_sqrt)] = 0.
    d_mat_inv_sqrt = sp.diags(d_inv_sqrt)
    norm_adj = d_mat_inv_sqrt.dot(adj).dot(d_mat_inv_sqrt).tocoo()

    indices = torch.LongTensor(np.vstack([norm_adj.row, norm_adj.col]))
    values = torch.FloatTensor(norm_adj.data)
    shape = torch.Size(norm_adj.shape)
    adj_tensor = torch.sparse_coo_tensor(indices, values, shape).to(device)
    return adj_tensor
```

B.2. Semantic Embedding Extraction

The semantic embedding extraction module encodes item textual metadata (title and description) into dense representations using a pre-trained language model. It supports both mean-pooled and full-sequence output modes, and uses multi-process acceleration for large-scale inference.

Listing B.2: Semantic embedding extraction (`rq/text2emb/amazon_text2emb.py`).

```python
import argparse
import json
import os
import random
import torch
from tqdm import tqdm
import numpy as np
from transformers import AutoTokenizer, AutoModel
from accelerate import Accelerator
from accelerate.utils import gather_object


def generate_text(item2feature, features):
    """Concatenate item metadata fields into text."""
    item_text_list = []
    for item in item2feature:
        data = item2feature[item]
        text = []
        for meta_key in features:
            if meta_key in data:
                meta_value = clean_text(data[meta_key])
                cleaned = meta_value.strip()
                if cleaned != "":
                    text.append(cleaned)
        if len(text) == 0:
            text = ["unknown item"]
        try:
            item_id = int(item)
        except:
            item_id = item
        item_text_list.append((item_id, " ".join(text)))
    return item_text_list


def generate_item_embedding(
        args, item_text_list, tokenizer, model,
        accelerator, word_drop_ratio=-1):
    """Generate embeddings for all items using a PLM."""
    all_ids, all_texts = zip(*item_text_list)
    total_items = len(all_texts)

    # Distribute items across processes
    num_processes = accelerator.num_processes
    process_index = accelerator.process_index
    chunk_size = int(np.ceil(total_items / num_processes))
    start_idx = process_index * chunk_size
    end_idx = min(start_idx + chunk_size, total_items)
    local_ids = all_ids[start_idx:end_idx]
    local_texts = all_texts[start_idx:end_idx]

    local_results = []
    batch_size = args.batch_size
    tokenizer.padding_side = "right"
    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token

    with torch.no_grad():
        for i in range(0, len(local_texts), batch_size):
            batch_texts = list(
                local_texts[i : i + batch_size])
            batch_ids = local_ids[i : i + batch_size]

            # Optional word dropout for augmentation
```

```
            if word_drop_ratio > 0:
                processed_batch = []
                for text in batch_texts:
                    sent = text.split('␣')
                    new_sent = [
                        wd for wd in sent
                        if random.random() > word_drop_ratio]
                    processed_batch.append(
                        '␣'.join(new_sent))
                batch_texts = processed_batch

            encoded = tokenizer(
                batch_texts,
                max_length=args.max_sent_len,
                truncation=True,
                return_tensors='pt',
                padding=True
            ).to(accelerator.device)

            outputs = model(
                input_ids=encoded.input_ids,
                attention_mask=encoded.attention_mask)
            last_hidden = outputs.last_hidden_state

            if args.emb_type == 'pooled':
                # Mean pooling over non-padding tokens
                mask_expanded = (
                    encoded.attention_mask.unsqueeze(-1)
                    .expand(last_hidden.size()).float())
                sum_emb = torch.sum(
                    last_hidden * mask_expanded, dim=1)
                sum_mask = torch.clamp(
                    mask_expanded.sum(dim=1), min=1e-9)
                mean_output = (sum_emb / sum_mask
                               ).cpu().numpy()
                for idx, emb in zip(batch_ids, mean_output):
                    local_results.append((idx, emb))

            elif args.emb_type == 'sequence':
                # Save full token-level sequences
                bs, seq_len, dim = last_hidden.shape
                if seq_len > args.max_seq_len_saved:
                    seq_out = last_hidden[
                        :, :args.max_seq_len_saved, :]
                else:
                    pad_len = args.max_seq_len_saved - seq_len
                    seq_out = torch.nn.functional.pad(
                        last_hidden, (0, 0, 0, pad_len),
                        value=0.0)
                seq_out = seq_out.cpu().numpy()
                for idx, emb in zip(batch_ids, seq_out):
                    local_results.append((idx, emb))

    accelerator.wait_for_everyone()
    all_results = gather_object(local_results)

    if accelerator.is_main_process:
        all_results.sort(key=lambda x: x[0])
        final_emb = np.stack(
            [x[1] for x in all_results], axis=0)
        file_path = os.path.join(
            args.root,
            f"{args.dataset}.emb-{args.plm_name}-td.npy")
        np.save(file_path, final_emb)
```

B.3. Unified Q-Former (UQF) Fusion Model

The following code implements the core UQF architecture: parallel cross-attention fusion layers with adaptive gating, learnable query tokens, modality dropout, and output aggregation.

Listing B.3: Multi-Head Cross-Attention module (`rq/unified-emb/qformer_fusion.py`).

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import math


class MultiHeadCrossAttention(nn.Module):
    def __init__(self, query_dim, kv_dim, hidden_dim,
                 num_heads=8, dropout=0.1):
        super().__init__()
        self.num_heads = num_heads
        self.hidden_dim = hidden_dim
        self.head_dim = hidden_dim // num_heads

        self.q_proj = nn.Linear(query_dim, hidden_dim)
        self.k_proj = nn.Linear(kv_dim, hidden_dim)
        self.v_proj = nn.Linear(kv_dim, hidden_dim)
        self.out_proj = nn.Linear(hidden_dim, query_dim)
        self.dropout = nn.Dropout(dropout)
        self.scale = math.sqrt(self.head_dim)

    def forward(self, query, kv, attention_mask=None):
        batch_size, num_queries, _ = query.shape
        seq_len = kv.shape[1]

        Q = self.q_proj(query)
        K = self.k_proj(kv)
        V = self.v_proj(kv)

        Q = Q.view(batch_size, num_queries,
                   self.num_heads, self.head_dim).transpose(1, 2)
        K = K.view(batch_size, seq_len,
                   self.num_heads, self.head_dim).transpose(1, 2)
        V = V.view(batch_size, seq_len,
                   self.num_heads, self.head_dim).transpose(1, 2)

        attn_weights = torch.matmul(Q, K.transpose(-2, -1)) / self.scale
        if attention_mask is not None:
            attn_weights = attn_weights.masked_fill(
                attention_mask == 0, float('-inf'))
        attn_weights = F.softmax(attn_weights, dim=-1)
        attn_weights = self.dropout(attn_weights)

        attn_output = torch.matmul(attn_weights, V)
        attn_output = attn_output.transpose(1, 2).contiguous().view(
            batch_size, num_queries, self.hidden_dim)
        return self.out_proj(attn_output)
```

Listing B.4: Parallel cross-attention fusion layer with adaptive gating (`rq/unified-emb/qformer_fusion.py`).

```
class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim, dropout=0.1):
        super().__init__()
        self.fc1 = nn.Linear(dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, dim)
        self.dropout = nn.Dropout(dropout)
        self.activation = nn.GELU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.activation(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.dropout(x)
        return x


class ParallelCrossAttentionFusionLayer(nn.Module):
    def __init__(self, query_dim, text_dim, graph_dim,
                 hidden_dim, num_heads=8, dropout=0.1):
```

```
        super().__init__()
        # Self-attention for queries
        self.self_attn = nn.MultiheadAttention(
            embed_dim=query_dim, num_heads=num_heads,
            dropout=dropout, batch_first=True)
        self.self_attn_norm = nn.LayerNorm(query_dim)

        # Parallel cross-attention to text and graph
        self.text_cross_attn = MultiHeadCrossAttention(
            query_dim, text_dim, hidden_dim, num_heads, dropout)
        self.graph_cross_attn = MultiHeadCrossAttention(
            query_dim, graph_dim, hidden_dim, num_heads, dropout)

        # Gated fusion
        self.fusion_gate = nn.Sequential(
            nn.Linear(query_dim * 2, query_dim),
            nn.Sigmoid())
        self.cross_attn_norm = nn.LayerNorm(query_dim)

        # Feed-forward
        self.ffn = FeedForward(query_dim, hidden_dim * 4, dropout)
        self.ffn_norm = nn.LayerNorm(query_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, queries, text_emb, graph_emb):
        # 1. Self-attention on queries
        residual = queries
        queries_sa, _ = self.self_attn(queries, queries, queries)
        queries = self.self_attn_norm(
            residual + self.dropout(queries_sa))

        # 2. Parallel cross-attention
        residual = queries
        queries_text = self.text_cross_attn(queries, text_emb)
        queries_graph = self.graph_cross_attn(queries, graph_emb)

        # 3. Gated fusion
        combined = torch.cat(
            [queries_text, queries_graph], dim=-1)
        gate = self.fusion_gate(combined)
        queries_fused = (gate * queries_text
                         + (1 - gate) * queries_graph)
        queries = self.cross_attn_norm(
            residual + self.dropout(queries_fused))

        # 4. Feed-forward
        residual = queries
        queries_ffn = self.ffn(queries)
        queries = self.ffn_norm(residual + queries_ffn)

        return queries
```

Listing B.5: UQF main model with learnable queries and modality dropout (`rq/unified-emb/qformer_fusion.py`).

```
class QFormerFusionV2(nn.Module):
    def __init__(self, text_dim=1024, graph_dim=64, output_dim=256,
                 num_queries=8, query_dim=256, hidden_dim=512,
                 num_layers=4, num_heads=8, dropout=0.1,
                 pooling='mean', modality_dropout=0.3):
        super().__init__()
        self.text_dim = text_dim
        self.graph_dim = graph_dim
        self.output_dim = output_dim
        self.num_queries = num_queries
        self.query_dim = query_dim
        self.pooling = pooling
        self.modality_dropout = modality_dropout

        # Learnable query tokens
        self.query_tokens = nn.Parameter(
            torch.zeros(1, num_queries, query_dim))
```

```
        nn.init.normal_(self.query_tokens, std=0.02)

        # Input projections with normalization
        self.text_proj = nn.Linear(text_dim, hidden_dim)
        self.graph_proj = nn.Linear(graph_dim, hidden_dim)
        self.text_norm = nn.LayerNorm(hidden_dim)
        self.graph_norm = nn.LayerNorm(hidden_dim)

        # Parallel cross-attention fusion layers
        self.layers = nn.ModuleList([
            ParallelCrossAttentionFusionLayer(
                query_dim, hidden_dim, hidden_dim,
                hidden_dim, num_heads, dropout)
            for _ in range(num_layers)])

        # Output projection
        self.output_proj = nn.Linear(query_dim, output_dim)
        self.output_norm = nn.LayerNorm(output_dim)

    def _apply_modality_dropout(self, text_emb, graph_emb):
        if not self.training or self.modality_dropout <= 0:
            return text_emb, graph_emb

        batch_size = text_emb.shape[0]
        device = text_emb.device
        rand_vals = torch.rand(batch_size, device=device)

        if text_emb.dim() == 2:
            mask_text = (rand_vals < self.modality_dropout / 2
                         ).float().unsqueeze(-1)
        else:
            mask_text = (rand_vals < self.modality_dropout / 2
                         ).float().view(batch_size, 1, 1)

        if graph_emb.dim() == 2:
            mask_graph = (
                (rand_vals >= self.modality_dropout / 2) &
                (rand_vals < self.modality_dropout)
            ).float().unsqueeze(-1)
        else:
            mask_graph = (
                (rand_vals >= self.modality_dropout / 2) &
                (rand_vals < self.modality_dropout)
            ).float().view(batch_size, 1, 1)

        text_emb = text_emb * (1 - mask_text)
        graph_emb = graph_emb * (1 - mask_graph)
        return text_emb, graph_emb

    def forward(self, text_emb, graph_emb,
                apply_modality_dropout=True):
        batch_size = text_emb.shape[0]

        if apply_modality_dropout:
            text_emb, graph_emb = self._apply_modality_dropout(
                text_emb, graph_emb)

        if text_emb.dim() == 2:
            text_emb = text_emb.unsqueeze(1)
        if graph_emb.dim() == 2:
            graph_emb = graph_emb.unsqueeze(1)

        batch_size, text_seq_len, _ = text_emb.shape
        _, graph_seq_len, _ = graph_emb.shape

        text_hidden = self.text_proj(
            text_emb.view(-1, self.text_dim))
        text_hidden = text_hidden.view(
            batch_size, text_seq_len, -1)
        graph_hidden = self.graph_proj(
            graph_emb.view(-1, self.graph_dim))
```

```
        graph_hidden = graph_hidden.view(
            batch_size, graph_seq_len, -1)

        text_hidden = self.text_norm(
            text_hidden.view(-1, text_hidden.shape[-1])
        ).view(batch_size, text_seq_len, -1)
        graph_hidden = self.graph_norm(
            graph_hidden.view(-1, graph_hidden.shape[-1])
        ).view(batch_size, graph_seq_len, -1)

        queries = self.query_tokens.expand(batch_size, -1, -1)

        for layer in self.layers:
            queries = layer(queries, text_hidden, graph_hidden)

        # Output aggregation: mean pooling
        pooled = queries.mean(dim=1)
        unified_emb = self.output_proj(pooled)
        unified_emb = self.output_norm(unified_emb)
        return unified_emb
```

B.4. UQF Training Objectives

The hybrid contrastive loss and reconstruction loss used to train UQF (Section **??**).

Listing B.6: Hybrid contrastive loss (`rq/unified-emb/qformer_fusion.py`).

```
class HybridContrastiveLoss(nn.Module):
    def __init__(self, temperature=0.07,
                 structure_weight=0.5, semantic_weight=0.5):
        super().__init__()
        self.temperature = temperature
        self.structure_weight = structure_weight
        self.semantic_weight = semantic_weight

    def forward(self, anchor_emb, struc_pos_emb, sem_pos_emb):
        anchor_emb = F.normalize(anchor_emb, dim=-1)
        struc_pos_emb = F.normalize(struc_pos_emb, dim=-1)
        sem_pos_emb = F.normalize(sem_pos_emb, dim=-1)
        batch_size = anchor_emb.shape[0]

        # Structure contrastive loss (InfoNCE)
        struc_pos_sim = torch.sum(
            anchor_emb * struc_pos_emb, dim=-1, keepdim=True)
        struc_neg_sim = torch.matmul(
            anchor_emb, struc_pos_emb.t())
        mask = torch.eye(batch_size,
                         device=anchor_emb.device).bool()
        struc_neg_sim = struc_neg_sim.masked_fill(
            mask, float('-inf'))
        struc_logits = torch.cat(
            [struc_pos_sim, struc_neg_sim], dim=1
        ) / self.temperature
        struc_labels = torch.zeros(
            batch_size, dtype=torch.long,
            device=anchor_emb.device)
        struc_loss = F.cross_entropy(
            struc_logits, struc_labels)

        # Semantic contrastive loss (InfoNCE)
        sem_pos_sim = torch.sum(
            anchor_emb * sem_pos_emb, dim=-1, keepdim=True)
        sem_neg_sim = torch.matmul(
            anchor_emb, sem_pos_emb.t())
        sem_neg_sim = sem_neg_sim.masked_fill(
            mask, float('-inf'))
        sem_logits = torch.cat(
            [sem_pos_sim, sem_neg_sim], dim=1
        ) / self.temperature
        sem_labels = torch.zeros(
```

```
            batch_size, dtype=torch.long,
            device=anchor_emb.device)
        sem_loss = F.cross_entropy(sem_logits, sem_labels)

        total_loss = (self.structure_weight * struc_loss
                      + self.semantic_weight * sem_loss)
        return total_loss, struc_loss, sem_loss
```

Listing B.7: Reconstruction loss (`rq/unified-emb/qformer_fusion.py`).

```
class ReconstructionLoss(nn.Module):
    def __init__(self, unified_dim, text_dim, graph_dim):
        super().__init__()
        self.text_decoder = nn.Sequential(
            nn.Linear(unified_dim, unified_dim * 2),
            nn.GELU(),
            nn.Linear(unified_dim * 2, text_dim))
        self.graph_decoder = nn.Sequential(
            nn.Linear(unified_dim, unified_dim // 2),
            nn.GELU(),
            nn.Linear(unified_dim // 2, graph_dim))

    def forward(self, unified_emb,
                text_emb_original, graph_emb_original):
        if text_emb_original.dim() == 3:
            text_emb_target = text_emb_original.mean(dim=1)
        else:
            text_emb_target = text_emb_original
        if graph_emb_original.dim() == 3:
            graph_emb_target = graph_emb_original.mean(dim=1)
        else:
            graph_emb_target = graph_emb_original

        text_recon = self.text_decoder(unified_emb)
        graph_recon = self.graph_decoder(unified_emb)

        text_loss = F.mse_loss(text_recon, text_emb_target)
        graph_loss = F.mse_loss(graph_recon, graph_emb_target)
        return text_loss + graph_loss, text_loss, graph_loss
```

B.5. UQF Training Loop

The training procedure for UQF with hybrid contrastive learning and asymmetric modality dropout.

Listing B.8: UQF training epoch with hybrid contrastive learning (`rq/unified-emb/amazon_unified_emb.py`).

```
def train_epoch_hybrid(model, dataloader, optimizer, accelerator,
                       contrastive_loss_fn, recon_loss_fn=None,
                       recon_weight=0.1):
    model.train()
    total_loss = 0
    total_struc_loss = 0
    total_sem_loss = 0
    total_recon_loss = 0

    for batch in dataloader:
        anchor_text = batch['anchor_text']
        anchor_graph = batch['anchor_graph']
        struc_pos_text = batch['struc_pos_text']
        struc_pos_graph = batch['struc_pos_graph']
        sem_pos_text = batch['sem_pos_text']
        sem_pos_graph = batch['sem_pos_graph']

        optimizer.zero_grad()

        # Anchor WITH modality dropout
        anchor_unified = model(
            anchor_text, anchor_graph,
            apply_modality_dropout=True)
        # Positives WITHOUT modality dropout
```

```
        struc_pos_unified = model(
            struc_pos_text, struc_pos_graph,
            apply_modality_dropout=False)
        sem_pos_unified = model(
            sem_pos_text, sem_pos_graph,
            apply_modality_dropout=False)

        contrast_loss, struc_loss, sem_loss = \
            contrastive_loss_fn(
                anchor_unified, struc_pos_unified,
                sem_pos_unified)

        loss = contrast_loss
        recon_loss_val = 0.0
        if recon_loss_fn is not None:
            recon_loss, _, _ = recon_loss_fn(
                anchor_unified, anchor_text, anchor_graph)
            loss = loss + recon_weight * recon_loss
            recon_loss_val = recon_loss.item()

        accelerator.backward(loss)
        optimizer.step()

        total_loss += loss.item()
        total_struc_loss += struc_loss.item()
        total_sem_loss += sem_loss.item()
        total_recon_loss += recon_loss_val

    n = len(dataloader)
    return (total_loss / n, total_struc_loss / n,
            total_sem_loss / n, total_recon_loss / n)
```

B.6. Residual Quantization (RQ-VAE)

The vector quantizer, residual quantizer, and RQ-VAE autoencoder.

Listing B.9: Vector quantizer with Sinkhorn balancing (`rq/models/vq.py`).

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class VectorQuantizer(nn.Module):
    def __init__(self, n_e, e_dim, beta=0.25,
                 kmeans_init=False, kmeans_iters=10,
                 sk_epsilon=0.003, sk_iters=100):
        super().__init__()
        self.n_e = n_e
        self.e_dim = e_dim
        self.beta = beta
        self.sk_epsilon = sk_epsilon
        self.sk_iters = sk_iters
        self.embedding = nn.Embedding(self.n_e, self.e_dim)
        if not kmeans_init:
            self.initted = True
            self.embedding.weight.data.uniform_(
                -1.0 / self.n_e, 1.0 / self.n_e)
        else:
            self.initted = False
            self.embedding.weight.data.zero_()

    def forward(self, x, use_sk=True):
        latent = x.view(-1, self.e_dim)
        if not self.initted and self.training:
            self.init_emb(latent)

        # L2 distance to codebook
        d = (torch.sum(latent**2, dim=1, keepdim=True)
             + torch.sum(self.embedding.weight**2,
```

```
                          dim=1, keepdim=True).t()
             - 2 * torch.matmul(
                 latent, self.embedding.weight.t()))

        if not use_sk or self.sk_epsilon <= 0:
            indices = torch.argmin(d, dim=-1)
        else:
            d = self.center_distance_for_constraint(d)
            d = d.double()
            Q = sinkhorn_algorithm(
                d, self.sk_epsilon, self.sk_iters)
            indices = torch.argmax(Q, dim=-1)

        x_q = self.embedding(indices).view(x.shape)

        # Quantization losses
        commitment_loss = F.mse_loss(x_q.detach(), x)
        codebook_loss = F.mse_loss(x_q, x.detach())
        loss = codebook_loss + self.beta * commitment_loss

        # Straight-through estimator
        x_q = x + (x_q - x).detach()
        indices = indices.view(x.shape[:-1])
        return x_q, loss, indices
```

Listing B.10: Residual vector quantizer (`rq/models/rq.py`).

```
class ResidualVectorQuantizer(nn.Module):
    def __init__(self, n_e_list, e_dim, sk_epsilons,
                 beta=0.25, kmeans_init=False,
                 kmeans_iters=100, sk_iters=100):
        super().__init__()
        self.n_e_list = n_e_list
        self.e_dim = e_dim
        self.num_quantizers = len(n_e_list)
        self.vq_layers = nn.ModuleList([
            VectorQuantizer(
                n_e, e_dim, beta=beta,
                kmeans_init=kmeans_init,
                kmeans_iters=kmeans_iters,
                sk_epsilon=sk_epsilon,
                sk_iters=sk_iters)
            for n_e, sk_epsilon
            in zip(n_e_list, sk_epsilons)])

    def forward(self, x, use_sk=True):
        all_losses = []
        all_indices = []
        x_q = 0
        residual = x

        for quantizer in self.vq_layers:
            x_res, loss, indices = quantizer(
                residual, use_sk=use_sk)
            residual = residual - x_res
            x_q = x_q + x_res
            all_losses.append(loss)
            all_indices.append(indices)

        mean_losses = torch.stack(all_losses).mean()
        all_indices = torch.stack(all_indices, dim=-1)
        return x_q, mean_losses, all_indices
```

Listing B.11: RQ-VAE autoencoder (`rq/models/rqvae.py`).

```
class RQVAE(nn.Module):
    def __init__(self, in_dim=768, num_emb_list=None,
                 e_dim=64, layers=None, dropout_prob=0.0,
                 bn=False, loss_type="mse",
                 quant_loss_weight=1.0, beta=0.25,
                 kmeans_init=False, kmeans_iters=100,
```

```
                  sk_epsilons=None, sk_iters=100):
        super(RQVAE, self).__init__()
        self.in_dim = in_dim
        self.e_dim = e_dim
        self.quant_loss_weight = quant_loss_weight

        self.encode_layer_dims = (
            [in_dim] + layers + [e_dim])
        self.encoder = MLPLayers(
            layers=self.encode_layer_dims,
            dropout=dropout_prob, bn=bn)

        self.rq = ResidualVectorQuantizer(
            num_emb_list, e_dim, beta=beta,
            kmeans_init=kmeans_init,
            kmeans_iters=kmeans_iters,
            sk_epsilons=sk_epsilons,
            sk_iters=sk_iters)

        self.decode_layer_dims = (
            self.encode_layer_dims[::-1])
        self.decoder = MLPLayers(
            layers=self.decode_layer_dims,
            dropout=dropout_prob, bn=bn)

    def forward(self, x, use_sk=True):
        x = self.encoder(x)
        x_q, rq_loss, indices = self.rq(x, use_sk=use_sk)
        out = self.decoder(x_q)
        return out, rq_loss, indices

    @torch.no_grad()
    def get_indices(self, xs, use_sk=False):
        x_e = self.encoder(xs)
        _, _, indices = self.rq(x_e, use_sk=use_sk)
        return indices

    def compute_loss(self, out, quant_loss, xs=None):
        loss_recon = F.mse_loss(out, xs, reduction='mean')
        loss_total = (loss_recon
                      + self.quant_loss_weight * quant_loss)
        return loss_total, loss_recon
```

B.7. SID Generation and Collision Resolution

After training the RQ-VAE, each item is assigned a discrete Semantic Item Identifier (SID) by encoding its embedding and extracting the quantization indices. The collision resolution procedure iteratively reassigns colliding items using Sinkhorn-balanced quantization.

Listing B.12: SID generation with collision resolution (`rq/generate_indices.py`).

```
import collections
import json
import numpy as np
import torch
from torch.utils.data import DataLoader

# Level-specific token prefixes for LLM vocabulary
prefix = ["<a_{}>","<b_{}>","<c_{}>","<d_{}>","<e_{}>"]

# --- Generate initial indices ---
all_indices = []
all_indices_str = []

for d in data_loader:
    d = d.to(device)
    indices = model.get_indices(d, use_sk=False)
    indices = indices.view(-1, indices.shape[-1]).cpu().numpy()
    for index in indices:
        code = []
```

```
        for i, ind in enumerate(index):
            code.append(prefix[i].format(int(ind)))
        all_indices.append(code)
        all_indices_str.append(str(code))

all_indices = np.array(all_indices)
all_indices_str = np.array(all_indices_str)

# --- Collision resolution via Sinkhorn reassignment ---
for vq in model.rq.vq_layers[:-1]:
    vq.sk_epsilon = 0.0
if model.rq.vq_layers[-1].sk_epsilon == 0.0:
    model.rq.vq_layers[-1].sk_epsilon = 0.003

tt = 0
while True:
    if tt >= 20 or check_collision(all_indices_str):
        break

    collision_item_groups = get_collision_item(
        all_indices_str)
    for collision_items in collision_item_groups:
        d = data[collision_items].to(device)
        indices = model.get_indices(d, use_sk=True)
        indices = indices.view(
            -1, indices.shape[-1]).cpu().numpy()
        for item, index in zip(collision_items, indices):
            code = []
            for i, ind in enumerate(index):
                code.append(prefix[i].format(int(ind)))
            all_indices[item] = code
            all_indices_str[item] = str(code)
    tt += 1

# Save final SID assignments
all_indices_dict = {}
for item, indices in enumerate(all_indices.tolist()):
    all_indices_dict[item] = list(indices)

with open(output_file, 'w') as fp:
    json.dump(all_indices_dict, fp)
```

B.8. Generative Recommendation via LLM Fine-Tuning

The following code shows the five SFT dataset classes used for multi-task alignment training. Each dataset corresponds to one of the five prompt styles in the LC-Rec framework.

Listing B.13: SID sequential recommendation dataset: $\mathrm{SID}_{i_1}, \ldots, \mathrm{SID}_{i_{T_u}} \rightarrow \mathrm{SID}_{i_{T_u+1}}$ (`data.py`).

```
class SidSFTDataset(Dataset):
    def get_history(self, row):
        row['history_item_sid'] = eval(
            row['history_item_sid'])
        L = len(row['history_item_sid'])
        history = ""
        for i in range(L):
            if i == 0:
                history += row['history_item_sid'][i]
            else:
                history += ", " + row['history_item_sid'][i]
        target_item = str(row['item_sid'])
        return {
            "input": (
                f"The user has interacted with items "
                f"{history} in chronological order. "
                f"Can you predict the next possible item "
                f"that the user may expect?"),
            "output": target_item + "\n"}
```

Listing B.14: SID–title grounding dataset: $\mathrm{SID}_i \leftrightarrow \mathrm{Title}(i)$ (`data.py`).

```
class SidItemFeatDataset(Dataset):
    def generate_prompt(self, data_point):
        if data_point['task'] == 'title2sid':
            prompt = (f"Which␣item␣has␣the␣title:␣"
                      f"{data_point['input']}?")
            response = data_point['output']
        else:  # sid2title
            prompt = (f'What␣is␣the␣title␣of␣item␣'
                      f'"{data_point["input"]}"?')
            response = data_point['output']
        return f"""###␣User␣Input:
{prompt}

###␣Response:\n{response}"""
```

Listing B.15: Fusion sequential recommendation dataset: $\mathrm{SID}_{i_1}, \ldots, \mathrm{SID}_{i_{T_u}} \rightarrow \mathrm{Title}(i_{T_u+1})$ (`data.py`).

```
class FusionSeqRecDataset(Dataset):
    def generate_prompt_title(self, history):
        return (
            f"The␣user␣has␣sequentially␣interacted␣"
            f"with␣items␣{history}.␣Can␣you␣recommend␣"
            f"the␣next␣item␣for␣him?␣"
            f"Tell␣me␣the␣title␣of␣the␣item")

    def get_history(self, row):
        history_item_sid = eval(row['history_item_sid'])
        history_str = ",␣".join(history_item_sid)
        target_sid = row['item_sid']
        target_title = self.sid2title.get(
            target_sid, target_sid)
        return {
            "history_str": history_str,
            "target_title": target_title}
```

Listing B.16: Title sequential recommendation dataset: $\mathrm{Title}(i_1), \ldots, \mathrm{Title}(i_{T_u}) \rightarrow \mathrm{Title}(i_{T_u+1})$ (`data.py`).

```
class SFTData(Dataset):
    def get_history(self, row):
        row['history_item_title'] = eval(
            row['history_item_title'])
        L = len(row['history_item_title'])
        history = ""
        for i in range(L):
            if i == 0:
                history += ('"'
                    + row['history_item_title'][i] + '"')
            else:
                history += (',\t"'
                    + row['history_item_title'][i] + '"')
        target_item = str(row['item_title'])
        target_item = '"' + target_item + '"\n'
        return {
            "input": (
                f"The␣user␣has␣played␣the␣following␣"
                f"{self.category}s␣before:␣{history}"),
            "output": target_item}
```

Listing B.17: Title-to-SID sequential recommendation dataset: $\mathrm{Title}(i_1), \ldots, \mathrm{Title}(i_{T_u}) \rightarrow \mathrm{SID}_{i_{T_u+1}}$ (`data.py`).

```
class TitleHistory2SidSFTDataset(Dataset):
    def get_history(self, row):
        history_item_title = eval(
            row['history_item_title'])
        history_titles = ",␣".join(
            [f'"{title}"' for title in history_item_title])
        target_item_id = str(row['item_id'])
        target_sid = self.id2sid.get(
            target_item_id, target_item_id)
```

```
        return {
            "input": (
                f"The␣user␣has␣interacted␣with␣the␣"
                f"following␣{self.category}␣items␣in␣"
                f"chronological␣order:␣{history_titles}.␣"
                f"Can␣you␣predict␣the␣next␣item␣the␣user␣"
                f"may␣expect?"),
            "output": target_sid + "\n"}
```

Listing B.18: LLM SFT training with multi-task dataset concatenation (`sft.py`).

```
def train(base_model, train_file, eval_file, output_dir,
          sid_index_path, item_meta_path, category,
          data_format="lc-rec", ...):
    model = AutoModelForCausalLM.from_pretrained(
        base_model, torch_dtype=torch.bfloat16)
    tokenizer = AutoTokenizer.from_pretrained(base_model)

    # Add SID tokens to vocabulary
    token_extender = TokenExtender(
        data_path=...,
        index_file_full_path=sid_index_path)
    new_tokens = token_extender.get_new_tokens()
    tokenizer.add_tokens(new_tokens)
    model.resize_token_embeddings(len(tokenizer))

    # Multi-task dataset (5 prompt styles)
    train_datasets = []
    train_datasets.append(SidSFTDataset(...))
    train_datasets.append(SidItemFeatDataset(...))
    train_datasets.append(FusionSeqRecDataset(...))
    train_datasets.append(SFTData(...))
    train_datasets.append(TitleHistory2SidSFTDataset(...))
    train_data = ConcatDataset(train_datasets)

    trainer = transformers.Trainer(
        model=model,
        train_dataset=train_data,
        args=transformers.TrainingArguments(
            num_train_epochs=num_epochs,
            learning_rate=learning_rate,
            ...),
        callbacks=[
            EarlyStoppingCallback(
                early_stopping_patience=3)])
    trainer.train()
```