

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Action Tokenization for Generative Recommendation

Permalink

<https://escholarship.org/uc/item/1rq7v75v>

ISBN

9798193227014

Author

Hou, Yupeng

Publication Date

2026-10-07

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA SAN DIEGO

Action Tokenization for Generative Recommendation

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Yupeng Hou

Committee in charge:

Professor Julian McAuley, Chair
Professor Zhiting Hu
Professor Lianhui Qin
Professor Jingbo Shang

2026

Copyright

Yupeng Hou, 2026

All rights reserved.

The Dissertation of Yupeng Hou is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

DEDICATION

To my family.

TABLE OF CONTENTS

Dissertation Approval Page	iii
Dedication	iv
Table of Contents	v
List of Figures	xi
List of Tables	xiii
Acknowledgements	xvi
Vita	xx
Abstract of the Dissertation	xxiv
Chapter 1 Introduction	1
1.1 Overall Framework	2
1.2 Dissertation Organization	4
Chapter 2 Bridging Language and Items for Retrieval and Recommendation: Benchmarking LLMs as Semantic Encoders	7
2.1 Introduction	8
2.2 A Large-Scale E-Commerce Dataset: Amazon Reviews 2023	11
2.3 BLAIR Benchmark	12
2.3.1 Task Definitions and Datasets	13
2.3.2 New Subtask: Complex-Query Product Search	15
2.4 Experiments	18
2.4.1 Experimental Setup	18
2.4.2 Main Results	18
2.4.3 Performance w.r.t. Item Metadata	21
2.4.4 Performance w.r.t. Adaptor Design	22
2.5 Benchmarking Toolkit	24
2.6 Dataset Collection Details	24
2.7 Experimental Setup	24
2.7.1 Data Split and Preprocessing	24
2.7.2 Item Metadata Processing	26
2.7.3 Evaluation Metrics	27
2.7.4 Evaluation Pipeline	27
2.8 Detailed Benchmarking Results	28
2.9 Discussion	28
2.10 Case Study: Failure on Complex-Query Product Search	29
2.11 Conclusion	30

Chapter 3	Multi-Behavior Generative Recommendation	31
3.1	Introduction	32
3.2	Methods	34
3.2.1	Problem Formulation	35
3.2.2	Learning Balanced Behavior&Item IDs	36
3.2.3	Generative Multi-Task Multi-Behavior Modeling	38
3.2.4	The Position-and-Behavior-Aware Transformer	39
3.3	Experiment	40
3.3.1	Experiment Setup	40
3.3.2	Overall Performance	45
3.3.3	Ablation Study	48
3.3.4	Further Analysis	50
3.4	Conclusion	52
Chapter 4	ActionPiece: Contextually Tokenizing Action Sequences for Generative Recommendation	54
4.1	Introduction	55
4.2	Method	58
4.2.1	Problem Formulation	58
4.2.2	Contextual Action Sequence Tokenizer	59
4.2.3	Generative Recommendation Models	64
4.2.4	Discussion	65
4.3	Experiments	66
4.3.1	Experimental Setup	66
4.3.2	Overall Performance	67
4.3.3	Ablation Study	69
4.3.4	Further Analysis	70
4.3.5	Case Study	72
4.4	Notations	72
4.5	Algorithmic Details	73
4.5.1	Vocabulary Construction Algorithm	73
4.5.2	Segmentation with Set Permutation Regularization Algorithm	73
4.6	Efficient Vocabulary Construction Implementation	74
4.6.1	Data Structures	75
4.6.2	Time Complexity	77
4.7	Additional Discussions	78
4.7.1	Benefits of Set Permutation Regularization	78
4.7.2	Connections Between ActionPiece and Feature Crossing	79
4.8	Datasets	79
4.9	Baselines	81
4.9.1	ID-Based Sequential Recommendation Methods	81
4.9.2	Feature-Enhanced Sequential Recommendation Methods	81
4.9.3	Generative Recommendation Methods	82
4.10	Implementation Details	83

4.11	Conclusion	85
Chapter 5	Generating Long Semantic IDs in Parallel for Recommendation	87
5.1	Introduction	88
5.2	Method	91
5.2.1	Long Semantic ID-based Item Representation	91
5.2.2	Learning to Generate Semantic IDs in Parallel	93
5.2.3	Next Semantic ID Decoding with Graph Constraints	95
5.2.4	Discussion	97
5.3	Experiments	100
5.3.1	Experimental Setup	100
5.3.2	Overall Performance	102
5.3.3	Inference Efficiency Analysis	103
5.3.4	Ablation Study	103
5.3.5	Further Analysis	105
5.4	Notations	109
5.5	Additional Implementation Details	109
5.6	Additional Discussion	109
5.7	Conclusion	112
Chapter 6	Inductive Generative Recommendation via Retrieval-based Speculation	113
6.1	Introduction	114
6.2	Methodology	116
6.2.1	Problem Setup and Formulation	116
6.2.2	Speculative Generative Recommendation	118
6.2.3	Drafting Strategies	121
6.3	Experiments	124
6.3.1	Experimental Setup	124
6.3.2	Performance Analysis (RQ1)	126
6.3.3	Ablation Study (RQ2)	127
6.3.4	Hyperparameter Analysis (RQ3)	128
6.3.5	Plug-and-Play Framework (RQ4)	129
6.4	Additional Implementation Details	129
6.4.1	SpecGR++ Training Details	129
6.4.2	Embedding Batch Size Ablation	130
6.4.3	Baseline Model Reproductions	130
6.4.4	TIGER _C Implementation	131
6.4.5	SpecGR++ Training Details	131
6.5	Case Study: Guided Redrafting	131
6.6	Ablation: Comparison with Ensemble Methods	132
6.7	Inference Speed Acceleration	133
6.7.1	Drafting and Verification Latency	133
6.7.2	Acceleration w.r.t. Generation Length	134
6.7.3	Theoretical Time Complexity	134

6.7.4	Empirical Acceptance Rates and Speedup	135
6.7.5	Scalability Analysis	136
6.8	Additional Further Analysis	136
6.8.1	Subset Ranking	136
6.8.2	Other New Capabilities	137
6.9	Further discussion on SpecGR++ Architecture	139
6.9.1	Parameter Efficiency and Speed	139
6.9.2	Unified Representation Space	140
6.10	Conclusion	140
Chapter 7	Expressiveness Limits of Autoregressive Semantic ID Generation in Generative Recommendation	142
7.1	Introduction	143
7.2	Preliminaries	145
7.3	Limitations of Autoregressive SID Generation	145
7.3.1	Generation as Tree Traversal	146
7.3.2	Tree Distance vs. Item Generation Probability	146
7.3.3	Limitation on User-Item Preference Modeling	148
7.3.4	Limitation on Item-Item Similarity Modeling	149
7.4	Alleviating Expressive Limits	150
7.4.1	Training: Sampling Latent Tokens	151
7.4.2	Inference: Generating Latent Tokens	151
7.4.3	Improved Expressive Power via Latent Tokens	151
7.5	Experiments	152
7.5.1	Experimental Setup	152
7.5.2	Main Results	153
7.5.3	Tree-Structure Correlation Analysis	153
7.5.4	In-Depth Analysis	154
7.6	Incorporating Inductive Bias into Latent Tokens	156
7.6.1	Latent Tokens as SID Permutation Indicators	156
7.6.2	Experiments and Analysis	158
7.7	Notation	159
7.8	Proofs for Section 7.3.3 (User-Item Preference Limitation)	160
7.8.1	Formal Setup	160
7.8.2	Proof of Rank Reversal Bound	161
7.9	Proofs for Section 7.3.4 (Item-Item Similarity Limitation)	162
7.9.1	Tree Distance as an Ultrametric	163
7.9.2	Correlation and Item Similarity	164
7.9.3	Proof of Limitation on Intransitive Similarity	164
7.9.4	Expressiveness of Latte	166
7.10	Implementation Details	169
7.11	Discussion	171
7.12	Limitations	174
7.13	Conclusion	175

Chapter 8	On the Memorization and Generalization of Generative Recommendation .	176
8.1	Introduction	177
8.2	Defining Memorization and Generalization	179
8.2.1	Task Definition	180
8.2.2	Memorization-Related Data Instance	180
8.2.3	Generalization-Related Data Instance	180
8.2.4	Multi-Hop Generalization	182
8.2.5	Uncategorized Data Instance	185
8.3	Performance Breakdown: Item IDs vs. Semantic IDs	185
8.3.1	Experiment Setup	185
8.3.2	Performance Analysis	186
8.4	Mechanism Analysis: A Token-Level Lens	187
8.4.1	Prefix N-Gram Memorization	188
8.4.2	From Item Generalization to Token Memorization	189
8.4.3	Explaining Performance Trade-off via Token Memorization	191
8.4.4	Mechanism Validation	194
8.5	Memorization-Aware Adaptive Ensemble	195
8.5.1	Adaptive Ensemble Indicators	196
8.5.2	Validation of the Indicator	197
8.5.3	Performance of Adaptive Ensemble	198
8.6	Performance Comparison on Other Item ID-based and GR Models	199
8.7	Adaptive Ensemble Performance Breakdown	200
8.8	Qualitative Analysis	200
8.9	Discussions on Categorization Method	201
8.9.1	Design Rationales	201
8.9.2	Limitations	202
8.10	Conclusion	202
Chapter 9	Related Work	204
9.1	Recommender Systems	204
9.1.1	Sequential Recommendation	204
9.1.2	Generative Recommendation	204
9.1.3	LLMs for Recommendation	205
9.1.4	Multi-Behavior Recommendation	206
9.1.5	Cold-Start and Inductive Recommendation	207
9.1.6	Expressiveness of Recommendation Models	208
9.2	Action Tokenization	209
9.2.1	Semantic Encoders in Recommendation	209
9.2.2	Semantic ID-Based Recommendation	209
9.3	LLMs, Generative Models, and More General Machine Learning Models	210
9.3.1	Benchmarking Embeddings and LLMs for Recommendation	210
9.3.2	Speculative Decoding and Efficient Inference	210
9.3.3	Tokenization for Large Language Models	211
9.3.4	Memorization and Generalization	211

Chapter 10	Conclusion and Future Outlook	213
10.1	Summary of Contributions	213
10.2	Future Directions	214
Bibliography		217

LIST OF FIGURES

Figure 1.1.	Action Tokenization Research Framework	3
Figure 2.1.	BLAIR Benchmark Overview	8
Figure 2.2.	MTEB and BLAIR Performance Correlation	18
Figure 2.3.	Item Metadata Performance Comparison	20
Figure 3.1.	MBGen Framework	35
Figure 3.2.	Baseline Scalability Comparison	50
Figure 3.3.	MBGen Scalability Comparison	50
Figure 3.4.	Beam Search and Behavior-Aware Sampling	51
Figure 4.1.	ActionPiece Tokenization Process	55
Figure 4.2.	Co-Occurring Token Pair Counts	59
Figure 4.3.	Linked-List Token Merge Updates	61
Figure 4.4.	Vocabulary Size Analysis	68
Figure 4.5.	Token Utilization Analysis	69
Figure 4.6.	Ensembled Segment Analysis	71
Figure 4.7.	Efficient Vocabulary Construction Pseudocode	86
Figure 5.1.	RPG Framework	91
Figure 5.2.	Semantic ID Difference and Logit Gap	95
Figure 5.3.	Runtime Memory and Inference Time	104
Figure 5.4.	Semantic ID Length Scalability	105
Figure 5.5.	Cold-Start Analysis	107
Figure 5.6.	Inference Hyperparameter Analysis	108
Figure 6.1.	SpecGR Inductive Recommendation Motivation	114
Figure 6.2.	SpecGR Framework	117

Figure 6.3.	SpecGR Hyperparameter Analysis	127
Figure 6.4.	Semantic ID Digit Speedup	134
Figure 6.5.	Subset Ranking Latency and Acceptance Rate	138
Figure 7.1.	Autoregressive SID Expressive Limitations	144
Figure 7.2.	Tree Distance and Similarity Correlation	146
Figure 7.3.	Latte Framework	150
Figure 7.4.	Latent Token Count Analysis	156
Figure 7.5.	Generated Modality Order Distribution	158
Figure 8.1.	Memorization and Generalization Definitions	177
Figure 8.2.	Multi-Hop Generalization Illustration	181
Figure 8.3.	Item-Level Generalization as Token Memorization	188
Figure 8.4.	Token Memorization Ratios	190
Figure 8.5.	Prefix Transition Counts and Performance	191
Figure 8.6.	TIGER Advantage Breakdown	191
Figure 8.7.	Codebook Training Dynamics	195
Figure 8.8.	MSP Indicator and Model Performance	197

LIST OF TABLES

Table 2.1.	Amazon Reviews Dataset Versions	10
Table 2.2.	Large-Scale Recommendation Datasets	11
Table 2.3.	BLAIR Benchmark Data Statistics	12
Table 2.4.	ESCI and Amazon-C4 Query Comparison	15
Table 2.5.	Reddit-Movie Example	16
Table 2.6.	LLM Semantic Encoder Performance	17
Table 2.7.	MRL and PCA Adaptor Comparison	21
Table 2.8.	Sequential Recommendation Detailed Results	22
Table 2.9.	Collaborative Filtering Detailed Results	23
Table 2.10.	Product Search Detailed Results	23
Table 2.11.	GritLM-7B Failure Case	25
Table 3.1.	Preprocessed Dataset Statistics	40
Table 3.2.	Target Behavior Prediction Performance	42
Table 3.3.	Behavior-Specific Item Prediction Performance	47
Table 3.4.	Behavior-Item Prediction Performance	47
Table 3.5.	MBSR Ablation Analysis	48
Table 3.6.	Item Tokenizer Code Distribution	50
Table 4.1.	Action Tokenization Method Comparison	57
Table 4.2.	BPE and ActionPiece Comparison	64
Table 4.3.	Processed Dataset Statistics	66
Table 4.4.	Amazon Reviews Performance Comparison	66
Table 4.5.	ActionPiece Ablation Analysis	68
Table 4.6.	ActionPiece Notation	73

Table 4.7.	ActionPiece Hyperparameters	83
Table 5.1.	Processed Dataset Statistics	98
Table 5.2.	RPG Performance Comparison	99
Table 5.3.	RPG Ablation Analysis	104
Table 5.4.	Short and Long Semantic ID Expressiveness	107
Table 5.5.	RPG Notation	110
Table 5.6.	RPG Hyperparameters	110
Table 5.7.	TIGER Semantic ID Digit Analysis	111
Table 5.8.	Memory Consumption Breakdown	111
Table 6.1.	SpecGR Performance Comparison	123
Table 6.2.	In-Sample and Unseen Performance Breakdown	123
Table 6.3.	SpecGR++ Ablation Study	125
Table 6.4.	Backbone and Drafter Configurations	128
Table 6.5.	Embedding Batch Size Analysis	130
Table 6.6.	SpecGR Dataset Statistics	131
Table 6.7.	Guided Re-Drafting Acceptance Rate	132
Table 6.8.	Guided Re-Drafting Performance	132
Table 6.9.	SpecGR-Aux Ensemble Comparison	133
Table 6.10.	Drafting and Verification Latency	133
Table 6.11.	Acceptance Rate and Expected Speedup	135
Table 6.12.	Parameter Efficiency and Training Time	138
Table 6.13.	SpecGR Capability Comparison	139
Table 7.1.	Latte Performance Comparison	153
Table 7.2.	Tree Distance Correlation	154

Table 7.3.	Tokenization Method Comparison	155
Table 7.4.	Aggregation Method Comparison	155
Table 7.5.	MPD Modality Order Performance	157
Table 7.6.	Latte Notation	159
Table 7.7.	Latte Dataset Statistics	160
Table 7.8.	PSID Hyperparameters	165
Table 7.9.	Latte Hyperparameters	165
Table 7.10.	Latent-Token Generation Probabilities	169
Table 7.11.	Structurally Coupled Item Statistics	170
Table 7.12.	Inference Time on Instruments	173
Table 7.13.	Noise-Based Augmentation Comparison	173
Table 7.14.	Noise-Based Tree Distance Correlation	174
Table 8.1.	Dataset Statistics	182
Table 8.2.	Memorization and Generalization Performance Breakdown	184
Table 8.3.	SID Configuration Memorization Ratios	193
Table 8.4.	Ensemble Strategy Performance	195
Table 8.5.	Beauty Case Studies	199
Table 8.6.	Additional Model Subset Performance	199
Table 8.7.	Fixed and Adaptive Ensemble Comparison	199

ACKNOWLEDGEMENTS

I am deeply grateful and truly fortunate to have worked with Prof. Julian McAuley, spending some of the most formative years of my life under his guidance while exploring the frontier of recommendation research. I greatly appreciate the many discussions with Julian, as well as the financial and emotional support he has generously provided. His trust, freedom, and support gave me the courage to pursue risky but intriguing research directions and problems. His professionalism in doing research, thinking through problems, and engaging with the world, together with his passion for the work he loves, has taught me a great deal. I will continue striving to follow this example in my own career and whole life.

My sincere thanks also go to my dissertation committee members, Prof. Zhiting Hu, Prof. Lianhui Qin, and Prof. Jingbo Shang, for their time, guidance, and valuable feedback throughout my Ph.D. journey.

I am deeply grateful to my parents and family members, whose selfless support and love have always given me the courage and confidence to study abroad and pursue my dreams. Special thanks go to my girlfriend, Xiwei Cheng, for her companionship, encouragement, support, love, and for standing by me through both the toughest moments and the most joyful milestones.

I am grateful for all the friendships I have been fortunate to build throughout this journey. Huge thanks to my peers and close friends Feng Yao, Xintong Li, and Yuwei Zhang. We met on day one and have worked together toward a shared goal ever since. They have brought so much joy to my Ph.D. life. I am also thankful to my roommates, Sijie Luo and Zixue Li, and to my friends from Renmin University of China, including Kun Zhou, Zikai Wang, Junkai Liang, Ruofan Wu, as well as many other friends, both in and outside the United States, with whom I have stayed closely connected online.

I feel lucky to have had such brilliant and friendly labmates, including Zhankui He, Jiacheng Li, Zhouhang Xie, Noveen Sachdeva, Canwen Xu, Jianmo Ni, Yu Wang, Wang-Cheng Kang, Lei Zhang, An Yan, Hyunsik Jeon, Petros Karypis, Sheldon Yu, Yueqi Wang, Johannes Kruse, Zachary Novack, Haven Kim, Phillip Long, Ruining He, Yu Xia, Zexue He, Ahmad

Bin Rabiah, Eduardo Escoto, Sean O'Brien, Jingyue Huang, Xin Xu, Junda Wu, Tianyang Liu, Zihan Huang, Nafis Sadeq, Jessica Echterhoff, Se-eun Yoon, Hao-Wen Dong, Erik Delanois, Bodhisattwa Prasad Majumder, and many others. Working and spending time with you have been among the most enjoyable parts of this journey.

My special thanks also go to our neighboring labs, Shang Data Lab, including Letian Peng, Chengyu Dong, Yufan Zhuang, Zilong Wang, Jiannan Xiang, Wentao Chen, Jinya Jiang, Zihan Wang, Zi Lin, Shuheng Li, Jiayun Zhang, Dheeraj Mekala, Xiyuan Zhang, Yiming Huang, Nan Huang, Ziche Liu, Junxia Cui, Shang Zhou, Longfei Yun, Bill Hogan, Weitang Liu, and all other members. My thanks also go to Shibo Hao, Xiaohan Fu, Shihan Ran, Zhen Wang, Xiaodi Yuan, Ke Chen, Chenyang An, Minghua Liu, Sein Kim, Ashley Shin, Jianli Zhao, and all the other friends I met at UCSD. I truly enjoyed the time we spent hanging out and having fun together, which filled my Ph.D. journey with laughter.

I am honored to have had the chance to collaborate with many talented individuals, including Derek Zhiyuan Cheng, Neil Shah, Leheng Sheng, An Zhang, Xiang Wang, Clark Mingxuan Ju, Shuai Shao, Junliang Yu, Xiushi Chen, Jiancan Wu, Tat-Seng Chua, Jianing Wang, Jinsung Jeon, Ed H. Chi, Kaveh Hassani, Zhengyi Yang, Tong Zhao, Wei Shao, Tianxin Wei, Pengyue Jia, Xiaobei Wang, Xuying Ning, Ning Yao, Zongwei Wang, Min Gao, and many others I have had the privilege to work with. I'm also grateful for friends we met during internships and collaborations, including Gabriele Berton, Ruoxi Wang, Ben Coleman, Nathan Yan, Jianling Wang, Jonathan Valverde, Luning Yang, Yin Zhang, Lukasz Heldt, Weilong Yang, Su-Lin Wu, Li Wei, Xin Liu, Hejie Cui, Changlong Yu, Zheng Li, and Juyoung Wy.

To the collaborators who met me in their early career, thank you for your trust, and for sharing your time and passion in pursuing exciting research topics together. This includes Yijie Ding, Zihan Liu, Zitian Guo, Tingchen Fu, Xiangjun Fu, Qiyong Zhong, Ruohan Zhang, Amit Namburi, Abhishek Santhanam, Saathvik Dirisala, and many others.

I gratefully acknowledge the financial support I received throughout my Ph.D. studies from NSF grant IIS-2432486, Google DeepMind, Snap Research, Meta, YouTube, Samsung,

UCSD CSE, and UCSD JCHI. This support provided me with the resources and opportunities to pursue my research, collaborate with outstanding researchers, and grow as a scholar.

Finally, I am grateful to my co-authors for their kind approval of the publications and material included in my dissertation.

Chapter 2, in part, is a reprint of the material as it appears in “Bridging Language and Items for Retrieval and Recommendation: Benchmarking LLMs as Semantic Encoders.” by Yupeng Hou*, Jiacheng Li*, Xiangjun Fu*, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley, accepted for publication at the 64th Annual Meeting of the Association for Computational Linguistics in 2026, referenced as [95]. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the material as it appears in “Multi-Behavior Generative Recommendation.” by Zihan Liu*, Yupeng Hou*, and Julian McAuley, in Proceedings of the 33rd ACM International Conference on Information and Knowledge Management in 2024, referenced as [144]. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in “ActionPiece: Contextually Tokenizing Action Sequences for Generative Recommendation.” by Yupeng Hou, Jianmo Ni, Zhankui He, Naveen Sachdeva, Wang-Cheng Kang, Ed H. Chi, Julian McAuley, and Derek Zhiyuan Cheng, in International Conference on Machine Learning in 2025, referenced as [91]. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in “Generating Long Semantic IDs in Parallel for Recommendation.” by Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, and Julian McAuley, in Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2 in 2025, referenced as [90]. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, is a reprint of the material as it appears in “Inductive Generative Recommendation via Retrieval-Based Speculation.” by Yijie Ding, Jiacheng Li, Julian McAuley,

and Yupeng Hou, in Proceedings of the AAAI Conference on Artificial Intelligence in 2026, referenced as [46]. The dissertation author was the primary investigator and author of this paper.

Chapter 7, in part, is a reprint of the material as it appears in “Expressiveness Limits of Autoregressive Semantic ID Generation in Generative Recommendation.” by Yupeng Hou, Haven Kim, Clark Mingxuan Ju, Eduardo Escoto, Neil Shah, and Julian McAuley, referenced as [94]. The dissertation author was the primary investigator and author of this manuscript.

Chapter 8, in part, is a reprint of the material as it appears in “On the Memorization and Generalization of Generative Recommendation.” by Yijie Ding, Zitian Guo, Jiacheng Li, Letian Peng, Shuai Shao, Wei Shao, Xiaoqiang Luo, Luke Simon, Jingbo Shang, Julian McAuley, and Yupeng Hou, accepted for publication at the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining in 2026, referenced as [45]. The dissertation author was the primary investigator and author of this paper.

VITA

2016–2020 B.Eng., Renmin University of China
2020–2023 M.Eng., Renmin University of China
2023–2026 Ph.D., University of California San Diego

PUBLICATIONS

Yupeng Hou, Haven Kim, Clark Mingxuan Ju, Eduardo Escoto, Neil Shah, Julian McAuley. Expressiveness Limits of Autoregressive Semantic ID Generation in Generative Recommendation. arXiv preprint, 2026.

Yijie Ding, Zitian Guo, Jiacheng Li, Letian Peng, Shuai Shao, Wei Shao, Xiaoqiang Luo, Luke Simon, Jingbo Shang, Julian McAuley, **Yupeng Hou**. On the Memorization and Generalization of Generative Recommendation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2026.

Yupeng Hou, Jiacheng Li, Xiangjun Fu, Zhankui He, An Yan, Xiusi Chen, Julian McAuley. Bridging Language and Items for Retrieval and Recommendation: Benchmarking LLMs as Semantic Encoders. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Letian Peng, Kun Zhou, Longfei Yun, **Yupeng Hou**, Jingbo Shang. Deriving Character Logic from Storyline as Codified Decision Trees. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Letian Peng, **Yupeng Hou**, Kun Zhou, Jingbo Shang. Codified Finite-state Machines for Role-playing. In *Proceedings of the Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Jingyue Huang, Zachary Novack, Phillip Long, **Yupeng Hou**, Ke Chen, Taylor Berg-Kirkpatrick, Julian McAuley. MuseTok: Symbolic Music Tokenization for Generation and Semantic Understanding. In *Proceedings of 2026 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2026.

Yijie Ding, Jiacheng Li, Julian McAuley, **Yupeng Hou**. Inductive Generative Recommendation via Retrieval-based Speculation. In *Proceedings of the 40th Annual AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, **Yupeng Hou**, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, Ji-Rong Wen. A Survey of Large Language Models. In *Frontiers of Computer Science (FCS)*, Volume 20, article number 2012627, (2026).

Yupeng Hou, An Zhang, Leheng Sheng, Jiancan Wu, Xiang Wang, Tat-Seng Chua, Julian McAuley. Towards Large Generative Recommendation: A Tokenization Perspective. Tutorial at *The 34th ACM International Conference on Information and Knowledge Management (CIKM)*, 2025.

Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, Julian McAuley. Generating Long Semantic IDs in Parallel for Recommendation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2025.

Yupeng Hou, Jianmo Ni, Zhankui He, Noveen Sachdeva, Wang-Cheng Kang, Ed H. Chi, Julian McAuley, Derek Zhiyuan Cheng. ActionPiece: Contextually Tokenizing Action Sequences for Generative Recommendation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Junjie Zhang, Wenqi Sun, **Yupeng Hou**, Wayne Xin Zhao, Ji-Rong Wen. Review-Enhanced Universal Sequence Representation Learning for Recommender Systems. In *ACM Transactions on Information Systems (TOIS)* 43, no. 3 (2025): 1-31.

Tingchen Fu, **Yupeng Hou**, Julian McAuley, Rui Yan. Unlocking Decoding-time Controllability: Gradient-Free Multi-Objective Alignment with Contrastive Prompts. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

Yupeng Hou, An Zhang, Leheng Sheng, Zhengyi Yang, Xiang Wang, Tat-Seng Chua, Julian McAuley. Generative Recommendation Models: Progress and Directions. Tutorial at *The ACM Web Conference 2025 (WWW)*, 2025.

Junjie Zhang, Ruobing Xie, **Yupeng Hou**, Wayne Xin Zhao, Leyu Lin, Ji-Rong Wen. Recommendation as Instruction Following: A Large Language Model Empowered Recommendation Approach. In *ACM Transactions on Information Systems (TOIS)* 43, no. 5 (2026): 1-37.

Gaowei Zhang, **Yupeng Hou**, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ji-Rong Wen. Scaling Law of Large Sequential Recommendation Models. In *Proceedings of the 18th ACM Conference on Recommender Systems (RecSys)*, 2024.

Zihan Liu, **Yupeng Hou**, Julian McAuley. Multi-Behavior Generative Recommendation. In *Proceedings of the 33rd ACM International Conference on Information & Knowledge Management (CIKM)*, 2024.

Chen Yang, Sunhao Dai, **Yupeng Hou**, Wayne Xin Zhao, Jun Xu, Yang Song, Hengshu Zhu. Revisiting Reciprocal Recommender Systems: Metrics, Formulation, and Method. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2024.

Junda Wu, Cheng-Chun Chang, Tong Yu, Zhankui He, Jianing Wang, **Yupeng Hou**, Julian McAuley. CoRAL: Collaborative Retrieval-Augmented Large Language Models Improve Long-tail Recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2024.

Jianing Wang, Junda Wu, **Yupeng Hou**, Yao Liu, Ming Gao, Julian McAuley. InstructGraph: Boosting Large Language Models via Graph-centric Instruction Tuning and Preference Alignment. In *Findings of the Association for Computational Linguistics (ACL)*, 2024.

Bowen Zheng, **Yupeng Hou**, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ji-Rong Wen. Adapting Large Language Models by Integrating Collaborative Semantics for Recommendation. In *IEEE 40th International Conference on Data Engineering (ICDE)*, 2024.

Junjie Zhang, **Yupeng Hou**, Ruobing Xie, Wenqi Sun, Julian McAuley, Wayne Xin Zhao, Leyu Lin, Ji-Rong Wen. AgentCF: Collaborative Learning with Autonomous Language Agents for Recommender Systems. In *Proceedings of the ACM Web Conference 2024 (WWW)*, 2024.

Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, Wayne Xin Zhao. Large Language Models are Zero-Shot Rankers for Recommender Systems. In *European Conference on Information Retrieval (ECIR)*, 2024.

Chenzhan Shang, **Yupeng Hou**, Wayne Xin Zhao, Yaliang Li, Jing Zhang. Multi-grained Hypergraph Interest Modeling for Conversational Recommendation. In *AI Open, Volume 4*, 2023, Pages 154-164.

Bowen Zheng, **Yupeng Hou**, Wayne Xin Zhao, Yang Song, Hengshu Zhu. Reciprocal Sequential Recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems (RecSys)*, 2023.

Lanling Xu, Zhen Tian, Gaowei Zhang, Junjie Zhang, Lei Wang, Bowen Zheng, Yifan Li, Jiakai Tang, Zeyu Zhang, **Yupeng Hou**, Xingyu Pan, Wayne Xin Zhao, Xu Chen, Ji-Rong Wen. Towards a More User-Friendly and Easy-to-Use Benchmark Library for Recommender Systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2023.

Yupeng Hou, Zhankui He, Julian McAuley, Wayne Xin Zhao. Learning Vector-Quantized Item Representation for Transferable Sequential Recommenders. In *Proceedings of the ACM Web Conference 2023 (WWW)*, 2023.

Yupeng Hou, Wayne Xin Zhao, Yaliang Li, Ji-Rong Wen. Privacy-Preserved Neural Graph Similarity Learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, 2022.

Shanlei Mu, **Yupeng Hou**, Wayne Xin Zhao, Yaliang Li, Bolin Ding. ID-Agnostic User Behavior Pre-training for Sequential Recommendation. In *China Conference on Information Retrieval (CCIR)*, 2022.

Wayne Xin Zhao, **Yupeng Hou**, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, Yushuo Chen, Lanling Xu, Gaowei Zhang, Zhen Tian, Changxin Tian, Shanlei Mu, Xinyan Fan, Xu Chen, Ji-Rong Wen. RecBole 2.0: Towards a More Up-to-Date Recommendation Library. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM)*, Resource Track, 2022.

Chen Yang, **Yupeng Hou**, Yang Song, Tao Zhang, Ji-Rong Wen, Wayne Xin Zhao. Modeling Two-Way Selection Preference for Person-Job Fit. In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys)*, 2022.

Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, Ji-Rong Wen. Towards Universal Sequence Representation Learning for Recommender Systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2022.

Yupeng Hou, Binbin Hu, Zhiqiang Zhang, Wayne Xin Zhao. CORE: Simple and Effective Session-based Recommendation within Consistent Representation Space. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval (SIGIR)*, Short Paper, 2022.

Yupeng Hou, Xingyu Pan, Wayne Xin Zhao, Shuqing Bian, Yang Song, Tao Zhang, Ji-Rong Wen. Leveraging Search History for Improving Person-Job Fit. In *International Conference on Database Systems for Advanced Applications (DASFAA)*, 2022.

Zihan Lin, Changxin Tian, **Yupeng Hou**, Wayne Xin Zhao. Improving Graph Collaborative Filtering with Neighborhood-enriched Contrastive Learning. In *Proceedings of the ACM Web Conference 2022 (WWW)*, 2022.

Yupeng Hou, Binbin Hu, Wayne Xin Zhao, Zhiqiang Zhang, Jun Zhou, Ji-Rong Wen. Neural Graph Matching for Pre-training Graph Neural Networks. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, 2022.

Wayne Xin Zhao, Shanlei Mu, **Yupeng Hou**, Zihan Lin, Kaiyuan Li, Yushuo Chen, Yujie Lu, Hui Wang, Changxin Tian, Xingyu Pan, Yingqian Min, Zhichao Feng, Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, Ji-Rong Wen. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, Resource Track, 2021.

Shuqing Bian, Xu Chen, Wayne Xin Zhao, Kun Zhou, **Yupeng Hou**, Yang Song, Tao Zhang, Ji-Rong Wen. Learning to Match Jobs with Resumes from Sparse Interaction Data using Multi-View Co-Teaching Network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*, 2020.

Wayne Xin Zhao, **Yupeng Hou**, Junhua Chen, Jonathan Zhu, Jing Yin, Hanting Su, Ji-Rong Wen. Learning Semantic Representations from Directed Social Links to Tag Microblog Users at Scale. *ACM Transactions on Information Systems (TOIS)* 38, no. 2 (2020): 1-30.

ABSTRACT OF THE DISSERTATION

Action Tokenization for Generative Recommendation

by

Yupeng Hou

Doctor of Philosophy in Computer Science

University of California San Diego, 2026

Professor Julian McAuley, Chair

The rapid development of large generative models has motivated a shift in recommender systems toward deep generative architectures with compact vocabularies. A fundamental operation in this paradigm is action tokenization: the process of converting human-perceivable actions, such as user-item interactions, into machine-readable token sequences, often referred to as semantic IDs.

In this dissertation, we investigate the role of action tokenization in generative recommendation, focusing on how human actions should be tokenized and leveraged. Our research follows three main directions. First, we develop stronger action tokenization methods that make semantic IDs more expressive and effective, including scaling semantic IDs from short to long

sequences and moving from context-independent to context-aware tokenization. Second, we study how to better leverage semantic IDs during training and inference so that generative models can use them more effectively, including improving new-item recommendation and addressing the personalization limits introduced by autoregressive token generation. Third, we show that generative recommendation models can excel at generalization, and empirically demonstrate that this capability is closely tied to their action tokenization mechanisms.

Together, these findings advance both the methodology and understanding of action tokenization in generative recommendation, providing a foundation for developing future recommender systems powered by large generative models.

Chapter 1

Introduction

Recent advances in language modeling, as well as in vision, audio, and other modalities, have increasingly converged toward large generative model architectures. By tokenizing diverse forms of data into token sequences, large-scale generative models can be trained in a sequence-to-sequence manner. Since recommendation is often formulated as predicting users’ future actions from sequences of historical actions, this has motivated growing interest in extending the generative modeling paradigm to recommendation. The broader goal is to enable recommender systems to benefit from the favorable scaling behavior of generative models, while moving toward unified cross-modal models that can ultimately support personalized assistants capable of understanding and responding to user needs across different types of input.

However, recommendation has not yet fully embraced this trend. Most existing recommender systems still follow a discriminative paradigm, relying on large, sparse embedding tables and relatively shallow encoding layers. We identify a key challenge that limits the adoption of generative models in recommendation: how user action data is represented. In traditional recommender systems, each user action is typically represented as a collection of features, such as the unique identifier (ID), title, price, color, and other attributes of the interacted item. The heterogeneous nature of these action features often requires models to maintain many sparse embedding tables. On the one hand, it is non-trivial to map these diverse features into token sequences, as required by generative models. On the other hand, some important features have

very large cardinality. For example, item IDs are usually proportional to the number of items in the system, whereas the vocabularies used in generative models for other modalities are often orders of magnitude smaller, such as the roughly 200K-token vocabularies used in modern large language models. This discrepancy creates optimization challenges and makes it difficult to scale recommendation models by increasing depth (*i.e.*, by adding more layers), which is widely regarded as a key factor behind the success of generative models.

In this dissertation, we explore alternative approaches to representing user action data, centered on the idea of *action tokenization*: converting human-perceivable actions into machine-readable token sequences that can be effectively processed by large generative models. Starting from the idea of using dedicated tokenizer models to encode raw features into semantics-preserving tokens, we primarily investigate *semantic IDs*: short sequences of discrete tokens drawn from a shared, compact vocabulary that jointly index an item. We propose improved tokenization methods and more effective ways to leverage semantic IDs, while also deepening our understanding of why this action tokenization approach is beneficial for recommendation tasks.

1.1 Overall Framework

Motivated by the need to develop action tokenization methods and build generative recommendation models, this dissertation organizes our research according to the data format used to represent tokenized actions, including both semantic representations and semantic IDs, as illustrated in Figure 1.1.

Semantic representations represent each user action as a dense vector in a semantic space. They typically leverage pre-trained semantic encoders, such as sentence embedding models, to encode raw action features into dense vectors that capture the semantic meaning of the associated features. Methods in this category still require large, sparse embedding tables to store dense semantic vectors. However, they provide an important foundation for developing semantic ID-

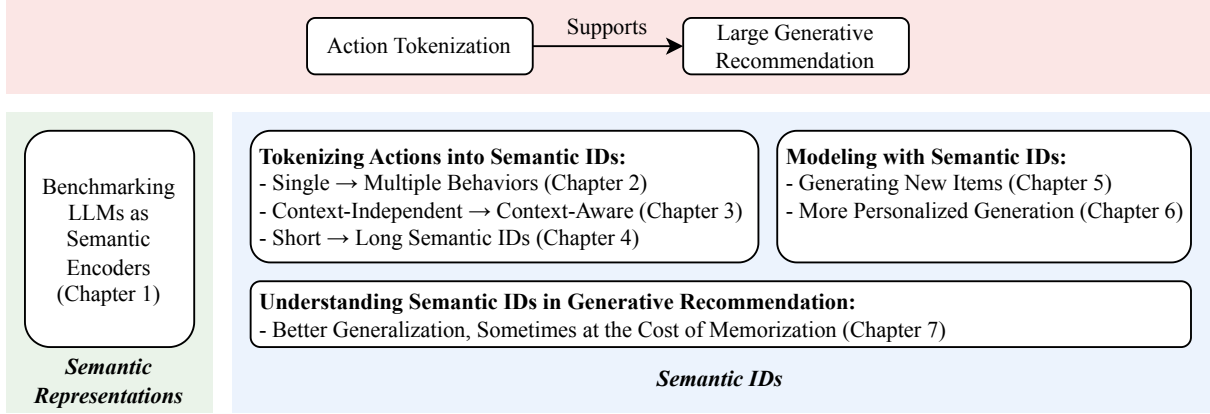


Figure 1.1. Our research framework for action tokenization in generative recommendation.

based models, which trade off some information richness for improved scalability and efficiency in large generative recommendation models.

Semantic IDs are short sequences of discrete tokens drawn from a shared, compact vocabulary that jointly index an item. The first step in constructing semantic IDs is usually similar to that of semantic representations: raw features are encoded into dense semantic vectors. These dense vectors are then quantized into sequences of discrete tokens, which serve as semantic indices for the corresponding actions. This allows semantically similar actions to be represented by similar token sequences, while greatly reducing the vocabulary size needed in the indexing system, since each token is drawn from one of the compact vocabularies shared across all items. This is the main focus of this dissertation, where we study semantic IDs along three main thrusts:

- *Tokenizing actions into semantic IDs:* This direction develops methods for creating more expressive semantic IDs. As the basic units of generative recommendation, semantic IDs directly affect the capabilities of recommendation models trained on these token sequences. Our research identifies key bottlenecks in existing approaches to semantic ID construction and tokenization, and proposes methods that either better leverage information from raw features or make the resulting token sequences more informative.
- *Modeling with semantic IDs:* This direction studies how generative models should be

designed and trained to effectively prefill and generate semantic ID sequences. As a new modeling paradigm, generative recommendation models may behave differently from traditional discriminative models. We identify properties that generative models often miss but are important for effective recommendation, and propose modifications to the modeling process that improve how generative models use semantic IDs for recommendation.

- *Understanding semantic IDs in generative recommendation:* This direction analyzes the benefits and unique properties introduced by action tokenization and the generative recommendation paradigm. Understanding how semantic IDs support recommendation provides valuable insights and can inspire new directions for developing better semantic ID tokenization methods, as well as better approaches to modeling semantic IDs in generative recommendation.

1.2 Dissertation Organization

In Chapter 1, we present the background, motivation, and overall framework of this dissertation. We first discuss why the recent success of large generative models motivates a shift in recommender systems toward action tokenization and compact token vocabularies. We then organize the dissertation around two forms of tokenized action representations, semantic representations and semantic IDs, with a primary focus on tokenizing actions into semantic IDs, modeling with semantic IDs, and understanding their role in generative recommendation.

In Chapter 2, we study semantic representations as an early form of action representation for recommendation. We introduce BLAIR, a benchmark for evaluating large language models as semantic encoders in recommendation scenarios. This chapter contributes a large-scale Amazon Reviews 2023 dataset, a unified benchmark covering multiple recommendation tasks, and empirical analyses showing that general-purpose embedding benchmarks do not reliably reflect semantic encoding quality for recommendation.

In Chapter 3, we study how tokenization can be incorporated into multi-behavior sequential recommendation. We propose MBGen, a generative recommendation framework that jointly

models behavior tokens and item tokens in a unified autoregressive sequence. This chapter shows that predicting the next behavior type before predicting the next item can better capture user intent, while allowing the model to leverage more training data even when these new data has different behavior types from the target behavior.

In Chapter 4, we study how to tokenize actions into more expressive semantic IDs by making tokenization context-aware. We propose ActionPiece, which constructs a vocabulary by merging frequent feature patterns both within actions and across adjacent actions. In this way, the same action can be tokenized into different tokens depending on the neighboring actions.

In Chapter 5, we study how to scale semantic IDs beyond short token sequences. We argue that existing approaches typically tokenize each action into a short sequence, such as 4 tokens per action, which limits the expressiveness of the resulting semantic IDs. We propose RPG, a framework that uses unordered, long semantic IDs with up to 64 tokens and predicts these tokens in parallel through a multi-token prediction objective.

In Chapter 6, we study how generative recommendation models can recommend new items in an inductive setting. We empirically show existing generative recommendation models struggle with recommending items that were not seen during training. We propose SpecGR, a plug-and-play framework in which an inductive drafter proposes candidate items and a generative recommendation model verifies them using its learned ranking capability.

In Chapter 7, we show that token-by-token autoregressive generation induces a structured decoding tree, causing items that are close in the tree to receive highly correlated probabilities across users, therefore leading to less personalized recommendation. To mitigate this limitation, we propose Latte, which introduces latent tokens before semantic IDs to create multiple latent-token-conditioned decoding trees and relax tree-induced probability coupling.

In Chapter 8, we investigate the generalization of generative recommendation models in comparison to conventional item ID-based models. We categorize test instances according to whether they require memorization of observed item transitions or generalization to unseen transitions, and show that the two modeling paradigms excel in different regimes. This chapter

further explains the advantage of generative recommendation from a token-level perspective and proposes a memorization-aware indicator for adaptively combining generative and item ID-based models.

In Chapter 9, we present related work of recommender systems, generative models, and action tokenization. In Chapter 10, we summarize the main contributions of this dissertation and discuss future directions for research in generative recommendation.

Chapter 2

Bridging Language and Items for Retrieval and Recommendation: Benchmarking LLMs as Semantic Encoders

Following the dissertation overview in the previous chapter, this chapter begins with semantic representations as an early form of semantic action tokenization method for recommendation. It establishes how well language models can encode item semantics before the later chapters discretize such representations into memory-efficient and scalable semantic IDs.

Feature engineering has long been central to recommender systems, yet effectively leveraging textual item features remains challenging. Recent advances in large language models (LLMs) have enabled their use as semantic encoders for recommendation, but their roles and behaviors in this setting are still not well understood. Prior studies often rely on general-purpose embedding benchmarks (*e.g.*, MTEB) when selecting LLMs, overlooking the unique characteristics of recommendation tasks. To address this gap, we introduce BLAIR, a comprehensive benchmark for evaluating LLMs as semantic encoders in recommendation scenarios. We contribute (1) a new large-scale Amazon Reviews 2023 dataset with over 570 million reviews and 48 million items, (2) a unified benchmark covering sequential recommendation, collaborative filtering, and product search, and (3) a new complex-query product search task featuring both semi-synthetic and real-world evaluation datasets. Experiments with 11 leading LLMs show that their rankings on BLAIR show little correlation with MTEB, highlighting the unique challenges

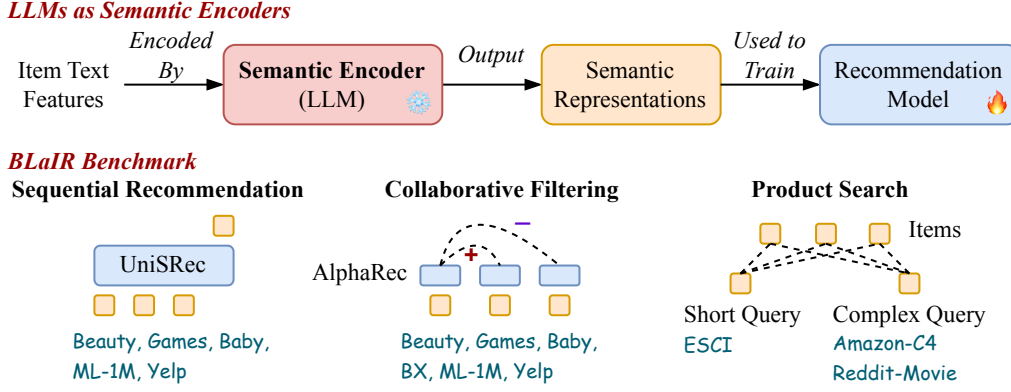


Figure 2.1. Overview of BLAIR benchmark. We evaluate LLMs as semantic encoders to convert item text features into semantic representations, which then train downstream recommendation models. The benchmark covers three scenarios: (i) sequential recommendation (*e.g.*, UniSRec [87]); (ii) collaborative filtering (*e.g.*, AlphaRec [189]); and (iii) product search with short queries and complex queries.

of semantic encoding in recommendation.¹

2.1 Introduction

Feature engineering has long been central to modern recommender systems [181, 25, 28]. However, effectively leveraging textual information (*e.g.*, item descriptions) has remained a challenge. While text features are rich in semantics, they are often noisy, unstructured, and difficult to integrate directly into recommendation models. Early approaches relied on keyword tagging [152, 210, 203], which were largely heuristic and discarded substantial semantic information. Moreover, textual features were typically modeled using shallow statistical techniques, lacking the broader common-sense knowledge inherent in natural language.

With the rapid advancement of large language models (LLMs) [83, 172, 30, 206, 288], researchers have begun to employ them as semantic encoders for recommendation. Pretrained on large-scale corpora, LLMs capture extensive world knowledge and demonstrate strong capabilities in understanding and representing natural language. Consequently, an increasing number of studies has explored using LLM-encoded text representations as inputs to train recommendation

¹Our code is available at <https://github.com/hyp1231/BLAIR-Bench>.

models, a direction referred to as modality-based recommendation [87, 266]. This paradigm has shown promising results across various tasks, including sequential recommendation [87, 266], collaborative filtering [180, 189], and product search [124, 177].

Despite this progress, the role of LLMs as semantic encoders in recommendation remains not well understood. Existing studies typically adopt popular or recently released LLMs, or rely on general-purpose sentence embedding benchmarks such as MTEB [154, 55] for model selection. However, the requirements of recommendation differ substantially from those of generic text embedding tasks. (1) In recommendation, encoded representations are usually used as input features to train downstream models such as Transformer decoders for sequential recommendation, whereas in text embedding tasks, representations are more often consumed directly for similarity search or simple classifiers. (2) In recommendation, text inputs are often short and noisy (*e.g.*, item titles), requiring semantic encoders to provide strong disambiguation based on world knowledge, while text embedding benchmarks typically evaluate well-formed sentences or paragraphs.

To address this gap, we present BLAIR, which stands for **B**ridging **L**anguage and **I**tems for **R**etrieval and **R**ecommendation, a comprehensive benchmark designed to evaluate LLMs as semantic encoders in recommendation scenarios (Figure 2.1). Our contributions are as follows:

- **A new large-scale e-commerce dataset: Amazon Reviews 2023.** Previous Amazon Reviews datasets [147, 75, 156] are widely used for benchmarking but were released several years ago (last released in 2018) and contain noisy metadata. We collect a new dataset with cleaned metadata, containing over 570 million reviews and 48 million items, with a cutoff date of September 2023.
- **A comprehensive benchmark: BLAIR.** We evaluate three representative recommendation scenarios: sequential recommendation, collaborative filtering, and product search. For each, we implement strong baselines and conduct extensive experiments with 11 top-performing LLMs on MTEB.

Table 2.1. Statistics of different versions of Amazon Reviews datasets. #Tokens denotes the total number of text tokens obtained using the `cl100k_base` tokenizer.

Version	Amazon’13	Amazon’14	Amazon’18	Amazon’23
#Categories	28	24	29	33
<i>Customer Reviews</i>				
#Reviews	34,686,771	82,456,877	233,055,327	571,544,897
#Users	6,643,669	21,128,805	43,531,850	54,514,264
#Items	2,441,053	9,857,241	15,167,257	48,185,153
#Tokens	5.9B	9.1B	15.7B	30.1B
Min Time	Jun 1995	Jun 1996	Jun 1996	Jun 1996
Max Time	Mar 2013	July 2014	Oct 2018	Sep 2023
<i>Item Metadata</i>				
#Tokens	-	4.1B	7.9B	30.7B
# Meta	-	9,430,088	14,741,571	35,393,189

- **A new subtask: complex-query product search.** Existing product search datasets typically involve short queries. To align with emerging applications that require understanding long, complex, and ambiguous queries, we introduce a new subtask, complex-query product search. We construct two datasets: one semi-synthetic dataset derived from real user reviews and another curated from real-world forum posts.

Our results yield several notable insights: (1) LLM rankings on BLAIR are not positively correlated with their rankings on MTEB, highlighting that recommendation poses distinct challenges compared to general embedding benchmarks. (2) Even when downstream models have the same number of parameters, performance still scales with the size of the semantic encoder, though the effect weakens as task complexity grows. (3) We observe a strong correlation between results on semi-synthetic and real-world datasets for complex query product search tasks, implying that semi-synthetic data can be a cost-effective proxy for evaluation.

Table 2.2. Statistics of representative large-scale recommendation datasets.

Dataset	#Items	#Users	#Interactions
Netflix [8]	17,770	480,189	100,480,507
Amazon’18 [156]	15,167,257	43,531,850	233,055,327
Google Local [254]	4,963,111	113,643,107	666,324,103
Tenrec [265]	3,753,436	5,022,750	142,321,193
MicroLens [159]	1,142,528	34,492,051	1,006,528,709
Amazon Reviews 2023	48,185,153	54,514,264	571,544,897

2.2 A Large-Scale E-Commerce Dataset: Amazon Reviews 2023

The Amazon Reviews datasets are among the most representative publicly available resources for evaluating recommendation models [147, 75, 156], offering rich item features. Given the wide use of the Amazon Reviews datasets, we aim to include them in our benchmark. However, existing versions of the Amazon Reviews datasets are outdated. The most recent one was last updated in 2018.

To enhance the quality of our proposed benchmark, we collect a new version of the dataset, named *Amazon Reviews 2023*². Table 2.1 presents key statistics comparing Amazon Reviews 2023 with earlier versions, while Table 2.2 compares Amazon Reviews 2023 with other representative large-scale datasets. The new dataset offers several notable improvements:

- *Larger scale*: Amazon Reviews 2023 is substantially larger than previous versions across all dimensions. Specifically, it contains $3.18\times$ more items and $2.58\times$ more text tokens in reviews and item metadata than the 2018 version.
- *More recent interactions*: The dataset contains more recent reviews from Amazon, extending the previous version with new data ranging from Oct 2018 to Sep 2023.
- *Richer and cleaner metadata*: We re-parsed the original HTML product pages into structured JSON format, resulting in metadata with more descriptive fields (*e.g.*, item descriptions and features) and multi-modal information (*e.g.*, product videos and images at multiple resolutions).

²<https://amazon-reviews-2023.github.io/>

Table 2.3. Data statistics of the BLAIR benchmark. Each row is a dataset. Cells that do not apply are marked *n/a*. “Real” indicates real-world data, while “Sync” indicates semi-synthetic data. More details about data split strategies (by timestamp, by ratio, or leave-last-out) can be found in Section 2.7.1.

Dataset	Data Source / Subtask	Type	#Items	#Interactions			Split	Avg. #Items Per User	Avg. Item Metadata	Avg. Query
				Train	Val	Test			#Chars	#Chars
Sequential Recommendation										
Beauty Games Baby	Amazon'23	Real	43,978	82,800	14,268	7,698	By Timestamp (08/11/2021, 07/16/2022)	2.45	118.83	n/a
			115,813	2,127,563	189,268	215,809		3.41	85.40	
			179,128	2,930,822	304,617	347,884		3.53	107.26	
ML-1M Yelp	ML-1M Yelp		3,043	983,412	6,040	6,040	Leave-Last-Out	164.82	23.78	
			20,033	255,492	30,431	30,431		10.01	604.60	
Collaborative Filtering										
Beauty Games Baby	Amazon'23	Real	43,978	41,906	31,429	31,431	By Ratio (4:3:3)	2.45	118.83	n/a
			115,813	1,013,056	759,792	759,792		3.41	85.40	
			179,128	1,433,329	1,074,996	1,074,998		3.53	107.26	
BX	BX		5,335	101,183	75,804	75,862		40.31	27.57	
ML-1M	ML-1M		3,043	398,184	298,631	298,677		164.82	23.78	
Yelp	Yelp		20,033	126,542	94,906	94,906		10.01	604.60	
Product Search										
ESCI	Short	Real	1,367,729			27,643			96.57	22.46
Amazon-C4	Complex	Sync	1,058,417	n/a	n/a	21,223	n/a	n/a	99.81	229.89
R-Movies	Complex	Real	51,203			1,627			19.63	435.85

- *Finer-grained timestamps*: Previous datasets provided timestamps only at the day level, which can introduce inaccuracies for time-sensitive tasks such as sequential recommendation. In contrast, Amazon Reviews 2023 offers timestamps with millisecond precision.

2.3 BLAIR Benchmark

In this section, we present the structure of the proposed benchmark, BLAIR. The benchmark comprises three recommendation-related scenarios: sequential recommendation, collaborative filtering, and product search. We first introduce the definitions and used datasets of each task in Section 2.3.1. Then, we describe in detail a newly introduced subtask under the product search scenario, namely complex-query product search, which contains long and ambiguous user queries (Section 2.3.2).

2.3.1 Task Definitions and Datasets

Given an item v with textual features x_v (such as title), the first step in using LLMs as semantic encoders is to transform these text features into dense representations: $\mathbf{e}_v = \text{LLM}(x_v)$. Here, $\text{LLM}(\cdot)$ denotes a pretrained and frozen LLM, and $\mathbf{e}_v \in \mathbb{R}^d$ is the resulting semantic representation for item v . The encoded representations $\{\mathbf{e}_v\}$ can then be used as inputs to train downstream recommendation models or directly for retrieval tasks. We consider three representative scenarios, each with its own task definition and dataset. The statistics of all datasets used in BLAIR are summarized in Table 2.3.

Sequential Recommendation.

In the sequential recommendation task, a user’s interaction history is represented as a time-ordered sequence of items $S_{t-1} = [v_1, v_2, \dots, v_{t-1}]$. The goal is to predict the next item v_t that the user will interact with. Following common practice [87, 266, 189], we first add adapter layers to align the semantic embedding space with the recommendation space. Specifically, each item embedding is projected as $\mathbf{e}'_v = \text{Proj}_{\text{seq}}(\mathbf{e}_v) \in \mathbb{R}^{d'}$, where $\text{Proj}_{\text{seq}}(\cdot)$ denotes a learnable projection layer. This design ensures that the downstream recommendation model maintains a consistent number of parameters, regardless of the dimensionality of the underlying LLM. We then adopt the UniSRec [87] architecture to implement the sequential recommendation model. Formally, the next-item probability is defined as:

$$P(v_t | S_{t-1}) \propto \text{Trm}(\mathbf{e}'_{v_1}, \mathbf{e}'_{v_2}, \dots, \mathbf{e}'_{v_{t-1}}) \cdot \mathbf{e}'_{v_t},$$

where $\text{Trm}(\cdot)$ is a Transformer decoder [208]. The model is trained by maximizing the likelihood of the next items using a cross-entropy loss.

For sequential recommendation tasks, we select three categories from the Amazon Reviews 2023 dataset: All Beauty (*Beauty*), Video Games (*Games*), and Baby Products (*Baby*). These categories differ in both scale (0.7M, 4.5M, and 6.0M interactions, respectively) and

domain (fashion, entertainment, and childcare). In addition, we include two widely used public datasets, Movielens-1M (*ML-1M*) [73] and *Yelp* [263].

Collaborative Filtering.

Given historical user-item interactions, the goal of collaborative filtering is to predict which item a user u is most likely to interact with next. Similar to the sequential recommendation setup, we add an adapter layer to obtain projected item representations $\mathbf{e}'_v = \text{Proj}_{\text{cf}}(\mathbf{e}_v) \in \mathbb{R}^{d'}$. Following AlphaRec [189], the user representation $\mathbf{e}'_u$ is computed as the average of the projected representations of items that the user has interacted with. The prediction probability is then defined as:

$$P(v|u) \propto \cos(\mathbf{W}\mathbf{e}'_u, \mathbf{W}\mathbf{e}'_v)/\tau,$$

where $\mathbf{W} \in \mathbb{R}^{d'' \times d'}$ is a linear layer, $\cos(\cdot, \cdot)$ denotes the cosine similarity function, and τ is a temperature hyperparameter. The model is trained using the InfoNCE loss with in-batch negative sampling [27].

For collaborative filtering tasks, we use the same five datasets as in the sequential recommendation scenario: *Beauty*, *Games*, and *Baby* from Amazon Reviews 2023, along with *ML-1M* and *Yelp*. Additionally, we include the widely used Book-Crossing dataset (*BX*) [307]. We exclude Book-Crossing from the sequential recommendation scenario because it does not provide timestamps to order user interactions.

Product Search.

In the product search task, given a user query q , the goal is to retrieve the most relevant items from a large corpus. The query is encoded using the same LLM as for items: $\mathbf{e}_q = \text{LLM}(q) \in \mathbb{R}^d$. For this task, we do not apply any adapter layers and directly use the raw LLM-encoded representations for retrieval. The relevance score between a query q and an item v is computed as $\mathbf{e}_q \cdot \mathbf{e}_v$ in a zero-shot manner.

We consider two subtasks under the product search scenario: (1) *short-query product*

Table 2.4. Comparison of user queries for the same item in the ESCI and Amazon-C4 datasets.

Item ASIN/ID	B07RJZNY5C
Query (ESCI)	salt gun
Query (Amazon-C4)	I want a gun that I can use while garden- ing to get rid of stink bugs, ants, flies, and spiders in my house. It needs to be amazing and help me feel less scared.
Metadata	BUG-A-SALT 3.0 Black Fly Edition.

search, the conventional setting where user queries are short and specific, typically containing concise keywords; and (2) *complex-query product search*, where user queries are longer, more descriptive, and often ambiguous. For short-query product search, we use the *ESCI* dataset [177]. The latter subtask is newly introduced in this work, with detailed task definitions and evaluation datasets presented in Section 2.3.2.

2.3.2 New Subtask: Complex-Query Product Search

As the natural language understanding capabilities of large language models continue to grow, users are no longer limited to issuing short, keyword-based queries when searching for items. Instead, they can describe their needs in more natural and expressive ways, which can be long, ambiguous, or require reasoning to identify relevant items (*e.g.*, Buy it in ChatGPT³ and Amazon Rufus⁴). To reflect this emerging trend, we introduce this subtask into our benchmark.

However, there are few public datasets that support evaluation in this setting. Collecting real paired complex user queries and their corresponding items is infeasible, as user chat histories are and should remain private. Therefore, we construct two datasets for this task: a semi-synthetic dataset derived from real user reviews in the Amazon Reviews 2023 (collected in Section 2.2), and a real-world dataset curated from Reddit forum posts.

³<https://openai.com/index/buy-it-in-chatgpt/>

⁴<https://www.amazon.com/rufus/>

Table 2.5. An example from the Reddit-Movie dataset.

IMDb ID	tt2582802
Query (Reddit-Movie)	Movies with a character that is very determined and goes through a vigorous training to be very good/the best at something? It doesn't necessarily have to be a training, it could be the character learning by trial and error, experience and doing the thing himself. Pretty much any genre is okay, action, sci fi, thriller, horror, ...
Movie	Whiplash (2014)

Semi-Synthetic Complex Queries: Amazon-C4.

The first dataset we introduce is *Amazon-C4*⁵, short for **Complex Contexts Created by ChatGPT**. The key idea is that user reviews often contain detailed intentions and rich contextual information, which can be repurposed into complex user queries. However, directly using reviews as queries is problematic. Reviews are usually written in a reflective tone, not in the style of a user query. In addition, reviews may include information about the target item that causes data leakage. To address this, we use large language models to rephrase user reviews into natural, query-like forms, creating the semi-synthetic dataset.

Prompt for complex query synthesizing in Amazon-C4.

Given a review from Amazon, can you rephrase it with a first-person tone as if you are the customer looking for a product? Give me the rephrased output without saying anything else. Note that the name of the product must not show in the output since you are looking for it. You should ignore irrelevant information that doesn't help with the rewriting.

Specifically, we uniformly sample more than 20,000 5-star reviews from the Amazon Reviews 2023 dataset with at least 100 characters per review to ensure sufficient detail. We use ChatGPT to rephrase each review into a first-person query expressing the intent to find a suitable item (see Prompt 1 for the prompt). We then filter out cases where ChatGPT fails to perform the

⁵<https://huggingface.co/datasets/McAuley-Lab/Amazon-C4>

Table 2.6. Performance comparison of LLMs as semantic encoders across different recommendation scenarios. “Seq. Rec.”, “Col. Fil.”, “Short”, and “Complex” stand for sequential recommendation (5 datasets), collaborative filtering (6 datasets), short-query product search (1 dataset), and complex-query product search (2 datasets), respectively. The best results in each column are highlighted in **bold**.

Model	Rank (Borda)*	Avg. (Overall)	Avg. (Task)	Seq. Rec.	Col. Fil.	Product Search	
						Short	Complex
Open-Source Models (< 1B parameters)							
FacebookAI/roberta-large	11 (15.0)	0.0263	0.0190	0.0393	0.0269	0.0096	0.0001
Qwen/Qwen3-Embedding-0.6B	10 (35.5)	0.0507	0.0829	0.0415	0.0274	0.1876	0.0750
princeton-nlp/sup-simcse-roberta-large	9 (37.0)	0.0397	0.0536	0.0426	0.0265	0.1063	0.0389
sentence-transformers/sentence-t5-large	8 (42.5)	0.0513	0.0801	0.0418	0.0304	0.1691	0.0790
Open-Source Models (≥ 1B parameters)							
Qwen/Qwen3-Embedding-4B	7 (69.5)	0.0620	0.1036	0.0416	0.0350	0.2258	0.1120
Qwen/Qwen3-Embedding-8B	6 (54.0)	0.0637	0.1069	0.0415	0.0362	0.2328	0.1172
Salesforce/SFR-Embedding-Mistral	4 (98)	0.0679	0.1160	0.0433	0.0372	0.2560	0.1273
intfloat/e5-mistral-7b-instruct	3 (101)	0.0666	0.1120	0.0434	0.0377	0.2437	0.1232
GritLM/GritLM-7B	2 (105)	0.0685	0.1161	0.0434	0.0385	0.2537	0.1290
Proprietary Models							
gemini-embedding-001 (Google)	5 (96.5)	0.0629	0.1040	0.0434	0.0355	0.2233	0.1140
text-embedding-3-large (OpenAI)	1 (116)	0.0665	0.1112	0.0440	0.0366	0.2366	0.1278

* Rank ($\downarrow$) is determined by the Borda Count score. For the Borda score and all other metrics, higher values are better ($\uparrow$).

conversion. A comparison between a user query in Amazon-C4 and ESCI for the same target item is shown in Table 2.4.

Real-World Complex Queries: Reddit-Movies.

The second dataset, Reddit-Movies (*R-Movies*), is curated from real forum discussions. As Amazon-C4 is semi-synthetic, one may question whether the generated queries are realistic enough for model evaluation. To complement it, we build Reddit-Movies based on the `reddit_movie_large_v1` dataset [79], which collects real Reddit discussions about movies and links each movie to its IMDb ID. Specifically, we keep only posts where users explicitly ask for movie suggestions (with the label `is_seeker=True`) that have at least one valid response with more than 20 upvotes, which indicates a believable answer. For item metadata, we use the movie title and release year. The final dataset contains more than 20,000 query-item pairs, split into training, validation, and test sets following the original data split [79]. Note that only the test

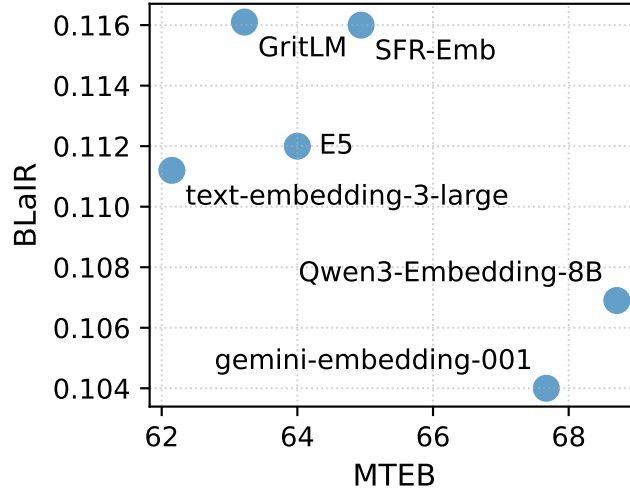


Figure 2.2. Performance correlation between MTEB (eng, v2) and BLAIR (avg. per task).

split is used for evaluation in our benchmark.

2.4 Experiments

2.4.1 Experimental Setup

We evaluate 11 top-performing LLMs from the widely used general text embedding benchmark MTEB [154, 55]: (1) open-source LLMs with $< 1B$ parameters (RoBERTa [142], SimCSE [60], Sentence-T5 [157]); (2) open-source LLMs with $\geq 1B$ parameters (E5 [215], SFT-Embedding [148], GritLM [155]); and (3) proprietary text embedding models from Google (gemini-embedding-001) and OpenAI (text-embedding-3-large). Among these, we further benchmark Qwen3-Embedding models of varying sizes (0.6B, 4B, and 8B) [284] to better understand the scaling behavior of LLM-based semantic encoders. Please refer to Section 2.7 for more implementation details.

2.4.2 Main Results

We present the benchmarking results in Table 2.6, with detailed per-dataset results in Section 2.8. Several key observations emerge:

(1) The ranking on BLAIR shows little correlation with MTEB. By analyzing the Spearman correlation between the per-task average scores on BLAIR and MTEB (eng, v2) [55], we observe a correlation coefficient of -0.476 with $p = 0.233$ (see Figure 2.2). This indicates that the performance of LLMs on general text embedding benchmarks does not necessarily reflect their effectiveness as semantic encoders for recommendation tasks. We attribute this to three possible factors. First, MTEB includes tasks such as clustering and summarization that are not directly aligned with the objectives of recommendation systems. Second, representations in recommendation tasks are expected to be highly discriminative [87, 189], a property not emphasized in MTEB. Third, as competition on MTEB intensifies, models may become overfitted to its specific tasks, resulting in reduced generalization to other domains.

(2) The scaling behavior of semantic encoders weakens when downstream tasks become complex. According to scaling laws, larger models typically achieve better performance [109], yet this trend is not consistently observed in our benchmark. We ensure a fair comparison by controlling the downstream model parameters across different LLM-based semantic encoders, using adaptors before feeding the representations into the downstream models. In tasks with simple downstream architectures, such as collaborative filtering (a single linear layer; see Section 2.3.1), larger LLMs generally yield higher performance. However, in more complex settings like sequential recommendation, where the downstream model involves a Transformer decoder, the scaling trend becomes less pronounced. For instance, models in the Qwen3-Embedding family show comparable performance across different sizes on the sequential recommendation task. A discussion on this counterintuitive observation is provided in Section 2.9.

(3) The semi-synthetic dataset Amazon-C4 exhibits a strong linear correlation with the real-world dataset Reddit-Movie on the complex-query product search task. By computing the Pearson correlation coefficient between the NDCG@100 scores of all evaluated models on Amazon-C4 and Reddit-Movie (Table 2.10), we find a strong linearly positive correlation of 0.94 ($p < 0.01$). This indicates that Amazon-C4, despite being semi-synthetic

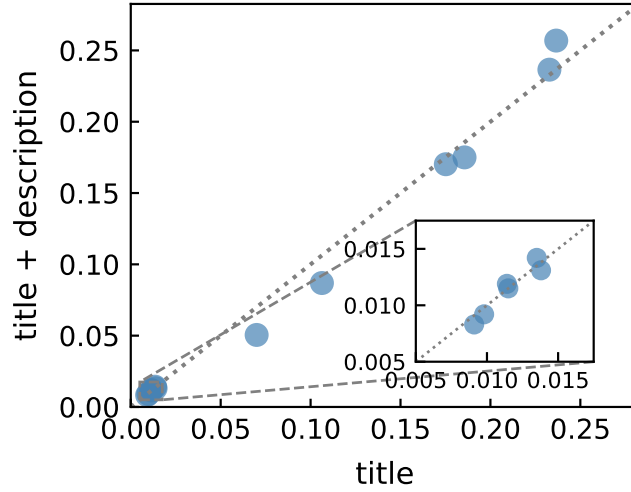


Figure 2.3. Comparison of performance when using different item metadata. Each point represents a model on a specific dataset.

and covering domains different from that in real-world complex queries, effectively captures essential characteristics of such queries. These results demonstrate that Amazon-C4 serves as a reliable proxy for evaluating (and potentially training) LLMs on complex-query product search tasks.

(4) `text-emb-3-large` **demonstrates unexpectedly strong generalization on BLAIR.** Developed by a leading company, `text-emb-3-large` is one of the most widely used text embedding models. Despite ranking only 42nd (as of Oct 6, AoE) on MTEB (English, v2), which is relatively low compared to other models evaluated in this chapter, raising doubts about its overall capability, it exhibits remarkably good performance across diverse datasets and tasks in BLAIR. Although its average performance is not the highest, its Borda Count ranking reveals consistently strong results across different scenarios. These observations suggest that `text-emb-3-large` possesses notable generalization ability and further indicate that evaluating LLM generalization remains an open and non-trivial challenge.

Table 2.7. Comparison between adaptors using matryoshka representation learning (MRL) and principal component analysis (PCA).

Model	Adaptor	Seq. Rec.	Col. Fil.
Qwen3-Embedding-8B	PCA	0.0415	0.0362
	MRL	0.0359	0.0392
gemini-embedding-001	PCA	0.0434	0.0355
	MRL	0.0384	0.0313
text-embedding-3-large	PCA	0.0440	0.0366
	MRL	0.0383	0.0379

2.4.3 Performance w.r.t. Item Metadata

Although numerous item metadata attributes are available for use as item features in the collected Amazon Reviews 2023 dataset, in BLAIR we include only the item title, as it is the most universally available attribute, providing high information density in a concise and easily identifiable form. In this section, we investigate whether incorporating additional metadata can further enhance the performance of LLM-based semantic encoders. Specifically, we conduct experiments using both the item title and description as input features, and compare the results with those obtained using the title alone. We evaluate three representative models (text-emb-3-large, Qwen3-Emb-8B, and SimCSE) across four datasets: Games (sequential recommendation), Games (collaborative filtering), Amazon-C4, and ESCI.

The results in Figure 2.3 show that incorporating both the title and description does not consistently improve performance relative to using only the title. We attribute this to two factors. First, longer text inputs may introduce additional noise, making it harder for the model to extract relevant information. Second, LLM world knowledge may already capture some of the information provided in item descriptions. These observations suggest that effectively leveraging rich, heterogeneous, and noisy item metadata in semantic encoders remains a challenging open problem.

2.4.4 Performance w.r.t. Adaptor Design

In our experiments, to ensure fair comparisons across different LLM-based semantic encoders, we employ adaptors to align their representation dimensions. We use principal component analysis (PCA) with whitening as the default adaptor due to its simplicity and efficiency [97, 196]. Meanwhile, several LLMs are capable of directly producing low-dimensional representations through matryoshka representation learning (MRL) [117]. In this section, we compare the performance of PCA and MRL on three representative models (Qwen3-Embedding-8B, gemini-embedding-001, and text-embedding-3-large) across all recommendation datasets that require an adaptor.

As shown in Table 2.7, for tasks involving complex downstream models such as sequential recommendation, PCA outperforms MRL, likely because whitening encourages more distinguishable representations. In contrast, for tasks with simpler architectures like collaborative filtering, MRL tends to achieve better results, possibly due to its ability to preserve task-relevant information in low-dimensional spaces. These findings suggest that the choice of adaptor should be guided by the complexity of the downstream task and model architecture.

Table 2.8. Sequential Recommendation (NDCG@10) detailed results. Best per column in **bold**, second best underlined.

Model / Datasets	All_Beauty	Video_Games	Baby_Products	ML1M	Yelp
<i>Open-Source Models (< 1B parameters)</i>					
FacebookAI/roberta-large	0.0177	0.0113	0.0070	0.1215	0.0391
Qwen/Qwen3-Embedding-0.6B	0.0224	0.0126	0.0072	0.1266	0.0385
princeton-nlp/sup-simcse-roberta-large	0.0232	0.0114	0.0069	0.1312	0.0404
sentence-transformers/sentence-t5-large	0.0212	0.0125	0.0075	0.1275	0.0405
<i>Open-Source Models ($\geq$ 1B parameters)</i>					
Qwen/Qwen3-Embedding-4B	0.0212	0.0127	0.0075	0.1276	0.0392
Qwen/Qwen3-Embedding-8B	0.0216	0.0138	0.0072	0.1248	0.0400
Salesforce/SFR-Embedding-Mistral	0.0231	0.0131	<u>0.0076</u>	0.1311	0.0416
intfloat/e5-mistral-7b-instruct	<u>0.0238</u>	<u>0.0136</u>	0.0072	<u>0.1324</u>	0.0398
GritLM/GritLM-7B	0.0216	0.0133	0.0074	0.1321	0.0426
<i>Proprietary Models</i>					
gemini-embedding-001 (Google)	0.0241	0.0136	0.0073	0.1297	0.0421
text-embedding-3-large (OpenAI)	0.0237	0.0135	0.0078	0.1330	0.0418

Table 2.9. Collaborative Filtering (NDCG@20) detailed results. Best per column in **bold**, second best underlined.

Model / Datasets	All_Beauty	Video_Games	Baby_Products	Book_Crossing	ML1M	Yelp
<i>Open-Source Models (< 1B parameters)</i>						
FacebookAI/roberta-large	0.0183	0.0087	0.0037	0.0338	0.0658	0.0310
Qwen/Qwen3-Embedding-0.6B	0.0220	0.0090	0.0047	0.0296	0.0713	0.0280
princeton-nlp/sup-simcse-roberta-large	0.0205	0.0091	0.0040	0.0285	0.0676	0.0295
sentence-transformers/sentence-t5-large	0.0202	0.0101	0.0043	0.0329	0.0869	0.0280
<i>Open-Source Models ($\geq$ 1B parameters)</i>						
Qwen/Qwen3-Embedding-4B	0.0219	0.0099	0.0041	0.0406	0.1047	0.0285
Qwen/Qwen3-Embedding-8B	0.0226	0.0098	0.0042	0.0427	0.1097	0.0284
Salesforce/SFR-Embedding-Mistral	0.0236	0.0107	0.0049	0.0416	0.1142	0.0284
intfloat/e5-mistral-7b-instruct	0.0232	<u>0.0110</u>	0.0049	0.0421	<u>0.1162</u>	0.0287
GritLM/GritLM-7B	0.0226	<u>0.0110</u>	0.0046	0.0425	0.1216	0.0287
<i>Proprietary Models</i>						
gemini-embedding-001 (Google)	0.0242	0.0108	0.0051	0.0371	0.1054	<u>0.0304</u>
text-embedding-3-large (OpenAI)	<u>0.0238</u>	0.0115	<u>0.0050</u>	0.0436	0.1071	0.0286

Table 2.10. Product Search (NDCG@100) detailed results. Best per column in **bold**, second best underlined.

Model / Datasets	Amazon-C4	ESCI	Reddit-Movies
<i>Open-Source Models (< 1B parameters)</i>			
FacebookAI/roberta-large	0.0001	0.0096	0.0000
Qwen/Qwen3-Embedding-0.6B	0.1288	0.1876	0.0211
princeton-nlp/sup-simcse-roberta-large	0.0700	0.1063	0.0078
sentence-transformers/sentence-t5-large	0.1219	0.1691	0.0360
<i>Open-Source Models ($\geq$ 1B parameters)</i>			
Qwen/Qwen3-Embedding-4B	0.1663	0.2258	0.0576
Qwen/Qwen3-Embedding-8B	0.1752	0.2328	0.0591
Salesforce/SFR-Embedding-Mistral	0.1836	0.2560	<u>0.0710</u>
intfloat/e5-mistral-7b-instruct	0.1759	0.2437	0.0705
GritLM/GritLM-7B	<u>0.1845</u>	<u>0.2537</u>	0.0734
<i>Proprietary Models</i>			
gemini-embedding-001 (Google)	0.1701	0.2233	0.0578
text-embedding-3-large (OpenAI)	0.1856	0.2366	0.0699

2.5 Benchmarking Toolkit

To facilitate reproducibility and simplify the integration of new models and datasets, we have developed a benchmarking toolkit available at <https://github.com/hyp1231/BLaIR-Bench>. The toolkit includes implementations of all datasets, tasks, and evaluation metrics used in this chapter, along with scripts for model training and evaluation. We welcome community contributions to further extend the benchmark.

2.6 Dataset Collection Details

The Amazon Reviews 2023 dataset is collected using a large-scale, user-centric pipeline. We first sample user identifiers from public Amazon pages and then iteratively collect all review pages associated with each user. This user-level design ensures completeness of individual review histories, which is particularly beneficial for recommendation and user modeling research.

However, the review coverage for a given item may be incomplete if some users are not included in our user pool. Additionally, due to the temporal span of the collecting process, recently posted reviews might be missing even before the dataset cut-off date.

2.7 Experimental Setup

2.7.1 Data Split and Preprocessing

Sequential Recommendation.

We group all user-item interactions by user ID and sort them by timestamp. For each user, we truncate their interaction history to the most recent 50 items (200 for ML-1M). Dataset-specific preprocessing steps are as follows:

- **Amazon Reviews 2023.** To preserve the natural distribution, we do not filter out any users or items based on interaction counts. Since sequential recommendation is time-sensitive, we split the dataset by time. To be specific, we determined two cut-off timestamps

Table 2.11. A failure case of the top-performing semantic encoder GritLM-7B on complex-query product search in the Reddit-Movies dataset.

User query	“Over the Top Bonkers Action Movies?. I recently saw the new <i>Suicide Squad</i> movie and loved it. It had all the qualities I like in an action movie: violence, shock factor, comedy, and overall bizarreness. What are some other action movies with these qualities? <i>Deadpool</i> , <i>Kick-Ass</i> , <i>Turbo Kid</i> , <i>Tropic Thunder</i> , and <i>Drive Angry</i> are some of my other favorites.”
Relevant items	<i>Nobody</i> (2021), <i>Hardcore Henry</i> (2015), <i>Slow Moe</i> (2010), <i>Dredd</i> (2012)
Top-3 retrieved by GritLM-7B	<i>Ninjas vs. Zombies</i> (2008), <i>Cowboys vs. Zombies</i> (2014), <i>Cowboys vs. Vampires</i> (2010)
Analysis	The retrieved movies appear to match the query only at a superficial level. For example, they contain unusual or fantastical entities such as zombies, ninjas, and vampires, which may loosely correlate with “bizarreness.” However, they fail to capture the core intent of the query, namely over-the-top action movies that combine stylized violence, dark humor, and mainstream appeal. This example suggests that current semantic encoders still struggle to model complex intent beyond shallow lexical or thematic similarity.

1628643414042 and 1658002729837 that split all the reviews in Amazon Reviews 2023 in a ratio of 8 : 1 : 1. These timestamps are shared across the three categories: All Beauty, Video Games, and Baby Products.

- **Movielens-1M.** We iteratively remove users and items with fewer than 20 interactions until convergence, then apply the leave-last-out strategy: the last two interactions per user are used for validation and testing, while all previous interactions are used for training.
- **Yelp.** We iteratively remove users and items with fewer than 5 interactions until convergence and apply the same leave-last-out strategy as ML-1M. Using the 2020 competition dataset, we keep only interactions from 2019.

Collaborative Filtering.

Following Sheng et al. [189], we discard timestamps and split user-item interactions into training, validation, and test sets with a 4:3:3 ratio. For ML-1M and Book-Crossing, users and items with fewer than 20 interactions are iteratively removed. For Yelp, the threshold is 5. No filtering is applied to the three Amazon Reviews 2023 categories.

Short-Query Product Search (ESCI).

We use the ESCI dataset [177] and retain only $jquery, item_i$ pairs with the label “Exact”. We adopt the small version due to its higher-quality queries, filter out non-English queries, and keep only those whose timestamps fall within our predefined test period (after 1658002729837). Each item ID (ASIN) is linked to its metadata in Amazon Reviews 2023. We construct a large multi-category candidate pool by sampling 50 in-domain items (from the same category) for each ground-truth pair. Ranking is then performed over all multi-domain candidates for each query.

Complex-Query Product Search.

- **Amazon-C4.** The data synthesizing process is described in Section 2.3.2. After that, we follow the same multi-category candidate sampling strategy as ESCI, sampling 50 in-domain items for each ground-truth pair. Ranking is computed over both in-domain and cross-domain candidates.
- **Reddit-Movie.** We use the `reddit_movie_large_v1` dataset [79], which contains multi-turn conversations where users seek movie recommendations. We retain only conversations with the label `is_seeker=True` (indicating the user is seeking movie recommendations) and apply several filtering steps: (1) contexts must include at least one valid seeker turn; (2) recommendations must have at least 20 upvotes; (3) duplicate entries are removed using MD5 hashes of concatenated text and context; (4) samples must contain valid movie IDs matched with metadata. The dataset is split following the original partition, and each $jquery, item_i$ pair represents a complex query and a recommended movie.

2.7.2 Item Metadata Processing

- **Amazon Reviews 2023.** For all Amazon-related datasets (Beauty, Games, Baby, Amazon-C4, and ESCI), we use product titles as item metadata. Although we conducted a small-scale experiment using concatenated titles and descriptions (Section 2.4.3), the inclusion

of descriptions did not consistently improve performance. Therefore, we use only titles for simplicity and consistency.

- **Movielens-1M.** Movie titles and release years are used as metadata.
- **Yelp.** We concatenate each business’s ‘name’, ‘categories’, ‘stars’, ‘review_count’, ‘city’, ‘state’, ‘attributes’, and ‘hours’ fields.
- **Book-Crossing.** We use book titles as metadata.
- **Reddit-Movie.** We use movie titles and release years as metadata.

2.7.3 Evaluation Metrics

For each dataset, we use the normalized discounted cumulative gain (NDCG) as the primary metric, with task-specific cutoffs: NDCG@10 for sequential recommendation, NDCG@20 for collaborative filtering, and NDCG@100 for product search.

To summarize results across datasets, we report three aggregated metrics in Table 2.6. First, to alleviate the bias in the number of datasets per task and the differences in metric scales, we follow MMTEB [55] and adopt the Borda Count [32] as the primary ranking criterion. Second, for intuitive comparison, we also report the average across all datasets (Avg. Overall) and the average after first averaging within each (sub)task (Avg. per Task).

2.7.4 Evaluation Pipeline

For tasks requiring model training (sequential recommendation and collaborative filtering), we search over three learning rates ($\{1 \times 10^{-3}, 3 \times 10^{-4}, 1 \times 10^{-4}\}$) and select the checkpoint with the best validation performance for final testing.

For product search, evaluation is conducted in a zero-shot manner without training, following each LLM’s recommended encoding procedure; for models that suggest a prompt when encoding queries, we use “*Given a user query describing a need, retrieve relevant entities (such as products, services, or media items) that satisfy the intent and constraints.*”.

2.8 Detailed Benchmarking Results

The results for each task are summarized in Table 2.6. Below, we provide detailed results for all evaluated LLMs across individual datasets. The results for sequential recommendation, collaborative filtering, and product search are presented in Table 2.8, Table 2.9, and Table 2.10, respectively.

2.9 Discussion

Why is the correlation between BLAIR and MTEB results weak?

We hypothesize that building a strong semantic encoder requires three key capabilities that may not be simultaneously emphasized in existing embedding benchmarks:

- The ability to accurately model semantic similarity, which is essential for tasks such as collaborative filtering and short-query product search.
- The ability to produce distinguishable and informative feature representations, which is important for tasks like sequential recommendation.
- Strong instruction-following and reasoning abilities, which are crucial for handling complex-query product search.

Why scaling semantic encoders yields limited gains on complex tasks?

As observed in Section 2.4.2, in tasks with complex downstream architectures, such as sequential recommendation (a Transformer decoder), the scaling trend becomes less pronounced than tasks with simple downstream model architectures like collaborative filtering (a simple linear layer). Our current hypotheses are:

- *Differences in the nature of the tasks:* The representational capabilities required differ between tasks. Collaborative filtering is primarily about encoding similarity. A stronger semantic encoder can represent item similarities in a richer and more accurate manner, which directly

translates into performance gains in collaborative filtering tasks as the encoder scales up. For example, Sheng et al. [189] found that stronger semantic encoders yield representations whose distribution is more closely aligned with those learned from pure interaction data. Sequential recommendation is essentially a sequence modeling problem that requires memorizing and reproducing behavior patterns. To achieve this, item representations may require strong discriminative capability (akin to functioning as unique indices). Representations produced by even a smaller semantic encoder might already be sufficiently distinguishable for a complex downstream Transformer model. Consequently, scaling the semantic encoder does not significantly enhance discriminability from the perspective of the downstream model, leading to smaller gains compared to collaborative filtering. A similar observation was reported by Hou et al. [87], who showed that treatments such as whitening enhance the discriminative power of item representations and improve sequential recommendation performance.

- *Sequentially dependent neural network systems and scaling behaviors:* Our setup functions as a two-stage system: a first-stage semantic encoder whose output is consumed by a second-stage model (a linear layer for collaborative filtering, or a Transformer decoder for sequential recommendation). We hypothesize that in such multi-stage systems, scaling the later-stage modules may diminish the impact of scaling earlier-stage modules. While there is limited systematic work on scaling laws in sequentially dependent systems, our findings may suggest that where we allocate additional parameters may matter as much as how many parameters we add.

2.10 Case Study: Failure on Complex-Query Product Search

Table 2.11 presents a failure case from the Reddit-Movies dataset, which requires understanding a complex user query with multiple implicit preferences. This example highlights headroom for improvement in complex-query product search. In particular, even the best-performing semantic encoder on this dataset, GritLM-7B, only achieves an NDCG@100 of

0.0734 over a candidate set of 50k movies, suggesting that surface-level semantic matching is far from sufficient. Future improvements may require stronger intent modeling, explicit reasoning, or the incorporation of additional ranking signals such as popularity.

2.11 Conclusion

In this chapter, we introduced BLAIR, a comprehensive benchmark for evaluating LLMs as semantic encoders in recommendation scenarios. We contributed (1) a new large-scale Amazon Reviews 2023 dataset, (2) a unified benchmark spanning sequential recommendation, collaborative filtering, and product search, and (3) a novel complex-query product search task with both semi-synthetic and real-world evaluation datasets. Our experiments with 11 leading LLMs revealed several key insights: (1) LLM rankings on BLAIR show little correlation with general text embedding benchmarks like MTEB, indicating that the role of semantic encoders poses unique challenges. (2) Scaling laws apply to semantic encoders even when downstream models are parameter-controlled, though the effect weakens with task complexity increases. (3) Our semi-synthetic Amazon-C4 dataset shows strong correlation with real-world complex queries, indicating that it can serve as a reliable evaluation proxy and may also benefit the training of LLMs for complex-query tasks.

Chapter 2, in part, is a reprint of the material as it appears in “Bridging Language and Items for Retrieval and Recommendation: Benchmarking LLMs as Semantic Encoders.” by Yupeng Hou*, Jiacheng Li*, Xiangjun Fu*, Zhankui He, An Yan, Xiushi Chen, and Julian McAuley, accepted for publication at the 64th Annual Meeting of the Association for Computational Linguistics in 2026, referenced as [95]. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Multi-Behavior Generative Recommendation

Building on the discussion of semantic representations in Chapter 2, this chapter begins the main part of the dissertation: tokenizing actions into semantic IDs. In the next three chapters, we present methods for developing more expressive semantic IDs. This chapter serves as a first step by showing how actions with different behavior types can be tokenized together with item tokens for semantic ID-based generative recommendation.

Multi-behavior sequential recommendation (MBSR) aims to incorporate behavior types of interactions for better recommendations. Existing approaches focus on the next-item prediction objective, neglecting the value of integrating the target behavior type into the learning objective. In this chapter, we propose **MBGen**, a novel Multi-Behavior sequential Generative recommendation framework. We formulate the MBSR task into a consecutive two-step process: (1) given item sequences, MBGen first predicts the next behavior type to frame the user intention, (2) given item sequences and a target behavior type, MBGen then predicts the next items. To model such a two-step process, we tokenize both behaviors and items into tokens and construct one single token sequence with both behaviors and items placed interleaved. Furthermore, MBGen learns to autoregressively generate the next behavior and item tokens in a unified generative recommendation paradigm, naturally enabling a multi-task capability. Additionally, we exploit the heterogeneous nature of token sequences in the generative recommendation and propose

a position-routed sparse architecture to efficiently and effectively scale up models. Extensive experiments on public datasets demonstrate that MBGen significantly outperforms existing MBSR models across multiple tasks.¹

3.1 Introduction

Multi-behavior sequential recommendation (MBSR) has gained attention for its ability to incorporate various behavior types (*e.g.*, click, purchase, add-to-cart) in addition to conventional sequential recommendation [69, 70, 264, 195, 258]. In the MBSR problem, the model is given an additional sequence of behaviors alongside the item sequence to predict the next interacted item. There are two lines of existing work. The first uses a two-stage aggregation paradigm that groups item sequences into subsequences based on the corresponding behavior types, then the subsequences are encoded separately and then aggregated for prediction [70, 69]. Another paradigm directly models the complete item sequence, using the behavior types as auxiliary sequential input [264, 195, 258].

Despite their effectiveness, existing models are primarily trained to predict the next items, neglecting the value of integrating *behavior types* into the training objective. Most previous methods can only predict the next items given a specific target behavior [258, 103, 244, 245]. Some methods extend the model’s capability to predict the next items under various behavior types [264, 195]. However, none of these models can jointly predict the next items and behaviors. The merits of learning to predict the next behavior types are twofold. First, it helps identify the user’s intention, which further benefits in predicting the next items. For instance, after users add an item to their cart, they may proceed to checkout, add another complementary item, or continue exploring. Based on these varying intentions, the recommendations should be adjusted accordingly. Second, behavior-type prediction enables platforms to introduce new functions to improve user experience. Depending on the user’s intention, we could redirect the user to the checkout page, display a pop-up window with complementary items, or guide the user back to

¹Our code is available at <https://github.com/anananan116/MBGen>.

the home page for further exploration.

Considering the above discussions, we argue that the MBSR task should be broken down into two consecutive steps. Given the item and behavior sequence, the first step is determining the next behavior. The second step is to predict the next item to interact with, based on both the predicted behavior and the input sequences. However, it’s non-trivial to model such a two-step process. (1) *Two-step dependency*. An intuitive solution is to follow the multi-task learning paradigm and design an auxiliary behavior-type prediction objective. Although such a multi-task learning approach can simultaneously predict the next items and behavior types, it fails to model the two-step dependency. In the proposed two-step scheme, the item prediction should also take the predicted behavior type as input. (2) *Varying magnitude of solution spaces*. Furthermore, the magnitude of the solution space of these two tasks significantly varies. There are usually no more than ten behavior types to predict [264, 195]. In contrast, the items to predict are usually millions of scale [271, 34], resulting in a much larger solution space. Learning two objectives with varying solution spaces may result in unbalanced learning and lead to sub-optimal performance [277].

(3) *Model capabilities*. In addition, the complex two-step modeling scheme challenges the model’s capabilities. Most current MBSR models follow the settings of conventional sequential recommendation and are quite shallow (typically with 2 Transformer layers) [264, 195, 258]. It could be difficult for models with limited capabilities to capture the sequential patterns of both behavior types and items. An effective way to improve model capability is to scale up the model parameters [109]. However, existing literature shows that it is difficult to scale up the model parameters of sequential recommendation models [275]. We also observe that existing MBSR models face similar difficulties in scaling up. (See Figure 3.2)

To this end, we introduce **MBGen**, the first Multi-Behavior sequential recommendation model under the Generative recommendation framework. To address the imbalanced solution space issue, we tokenize each item into a tuple of “item tokens”, where item tokens share a much smaller solution space. We propose modeling the two-step process using a data-centric approach, where behavior tokens and item tokens are flattened and placed interleaved into a

single token sequence (Figure 3.1). This tokenized sequence allows the attention mechanism to capture token-level fine-grained $\text{behavior} \rightarrow \text{item}$, $\text{item} \rightarrow \text{item}$, and $\text{behavior} \rightarrow \text{behavior}$ dependencies. Furthermore, we can train a sequence-to-sequence generative model with a unified next-token prediction objective. MBGen naturally learns to predict the next behavior types and items in an auto-regressive manner. Such a data-driven approach allows us to model the two-step process in a unified framework. Note that the proposed framework inherently has multi-task capability. We can provide any target behavior type as the prompt input to condition the item token generation. MBGen can also jointly predict the next behavior types and items as usual if the target behavior type is not given. Lastly, the constructed token sequence is heterogeneous by nature. We propose a position-routed sparse network to route each input to different expert networks, which efficiently scales up the model and improves performance.

The main contributions of our work are as follows:

- We frame the multi-behavior sequential recommendation task as a two-step process and integrate target behavior types into the learning objective.
- We present the first data-centric, token-level generative model for the multi-behavior sequential recommendation tasks with a unified next-token prediction objective.
- Extensive experiments on public datasets show that our model significantly outperforms previous MBSR models by 31% $\sim$ 70%.

3.2 Methods

In this section, we present MBGen, a framework designed to seamlessly integrate various behavior-related recommendation tasks into a single framework. In Section 3.2.1, we first formally define the three multi-behavior recommendation tasks. Then, in Section 3.2.2, we introduce our tokenizer that maps each raw item ID to multiple item tokens to achieve token-level fine-grained pattern modeling. We then elaborate on our model’s capability on multiple multi-behavior recommendation tasks in Section 3.2.3. We finally present the proposed position-

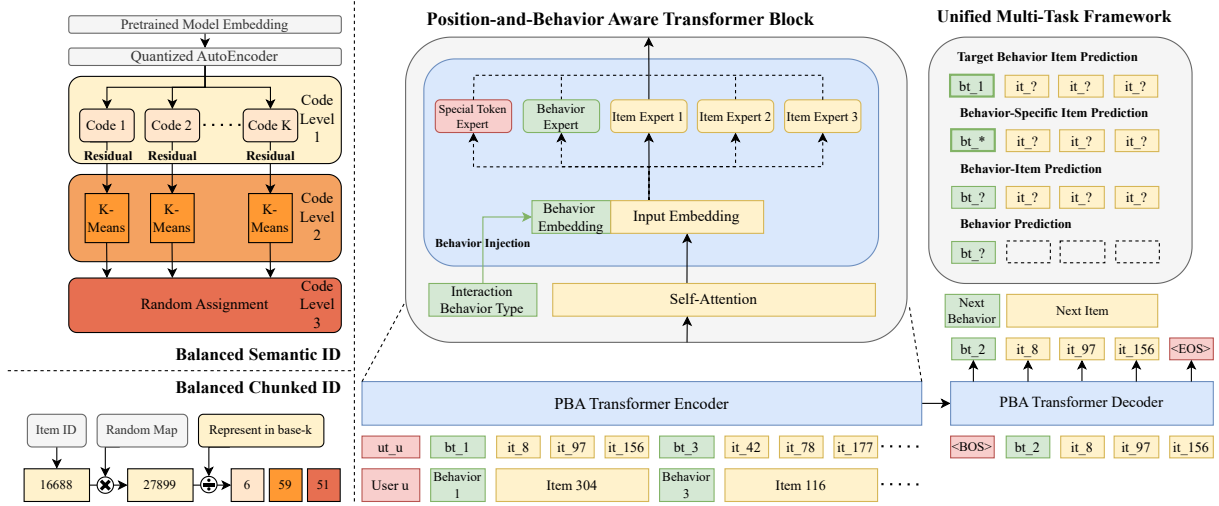


Figure 3.1. The overall framework of the proposed multi-behavior generative recommendation framework MBGen. We use “ut” to denote the user tokens, “bt” to denote the behavior tokens, and “it” to denote the item tokens.

and-behavior-aware Transformer block in Section 3.2.4 to better model the heterogeneous token sequences. The overall framework of MBGen is illustrated in Figure 3.1.

3.2.1 Problem Formulation

Multi-behavior sequential recommendation models the next item to interact with, given the historical interaction sequence (in other words, item sequence) and the corresponding behavior types. Formally, we denote $\mathcal{U} = \{u_1, u_2, \dots, u_{|\mathcal{U}|}\}$ as the set of users and $\mathcal{V} = \{v_1, v_2, \dots, v_{|\mathcal{V}|}\}$ as the set of items. At the same time, we define a set of behaviors that are associated with each interaction: $\mathcal{B} = \{b_1, b_2, \dots, b_{|\mathcal{B}|}\}$, among which the most important behavior is marked to be the target behavior, *e.g.*, “purchase” is the typical target behavior in e-commerce platforms in comparison to other behaviors like “view”, “add-to-cart”, and “add-to-favorite”. For each user u , we define the behavior-aware interaction history sequence to be $S_u = [\langle v_1^u, b_1^u \rangle, \dots, \langle v_{n-1}^u, b_{n-1}^u \rangle]$. The user’s last interaction $\langle v_n^u, b_n^u \rangle$ is the prediction target. As discussed in Section 3.1, we propose to formulate the MBSR problem as a two-step process. First, the interaction history S_u determines the behavior type of the next interaction b_n^u , which helps frame the user intention. Then, S_u and b_n^u jointly determine the next interacted item v_n^u . Thus, we can model the probability

distribution of the next behaviors and items as follows:

$$P(b_n, v_n | S_u) = P(b_n | S_u) \cdot P(v_n | S_u, b_n).$$

Then, we describe the three behavior-related tasks in the MBSR problem, *i.e.*, target behavior item prediction, behavior-specific item prediction, and behavior-item prediction. In the target behavior item prediction task, we model $P(v_n | S_u, b_n)$ given $b_n = b_1$, where b_1 is the target behavior for simplicity. Similarly, in the behavior-specific task, the model predict $P(v_n | S_u, b_n)$ given arbitrary behavior $b_n \in \mathcal{B}$. In contrast, the behavior-item prediction task gives no information about the next behavior. The model predicts both $P(b_n | S_u)$ and $P(v_n | S_u, b_n)$ given S_u .

3.2.2 Learning Balanced Behavior&Item IDs

In the presented generative recommendation framework, to balance the solution space of the next behaviors (usually less than ten) and items (millions), we propose to tokenize each item into a tuple of tokens. Existing works [174, 132] use RQ-VAE [268] as the item tokenizer to encode dense item features into tokens. However, we observe that this practice is suboptimal due to the extremely unbalanced code distribution (Table 3.6), which necessitates a larger pool of tokens and an additional digit to prevent collisions. In this section, we propose to tokenize items into distribution-balanced tokens. More balanced token distribution leads to smaller token pools and reduced solution spaces, resulting in better performance [166, 88]. We refer to the item tokens that come from associate item features as Balanced Semantic ID (SID). We also propose a balanced item tokenization method that takes only item IDs as inputs, referred to as the Balanced Chunked ID (CID).

Balanced Semantic ID. For the widely-used RQ-VAE tokenization [174], we observe that token imbalances stem from the high correlations between residual vectors and the quantized first-digit token. The second and subsequent digits of tokens derived from the residuals should ideally

be uniformly distributed, regardless of the first digit of the token. However, existing methods use a global codebook to quantize residual vectors, regardless of which first-digit token the item is assigned. Thus, residuals with the same first-digit token are likely to be quantized to have the same second-digit (also subsequent) token. With this in mind, we propose individually quantizing the residuals of each first-digit token rather than maintaining a single codebook for all residuals.

(1) *Code level 1.* When deriving the first digit, we fit a quantized auto-encoder with the item features as input. This process can also be seen as an RQ-VAE [268] with only one quantization layer. We first feed the item feature $\mathbf{x}$ into an encoder $E(\cdot)$ (a simple feed-forward network in our case) to learn the latent representation $\mathbf{z} := E(\mathbf{x}) \in \mathbb{R}^d$, where d is the dimension size. Meanwhile, we maintain a codebook $\mathcal{C} := \mathbf{E} \in \mathbb{R}^{K \times d}$, where K is the codebook size. We then find the closest codebook entry and use the corresponding entry index r as the first-digit token, *i.e.*, $r = \operatorname{argmin}_j \|\mathbf{z} - \mathbf{e}_j\|$. We then use another neural network as a decoder to reconstruct the input item feature: $\hat{\mathbf{x}} := D(\mathbf{e}_r)$. This network is trained with two losses: (1) the reconstruction loss $\mathcal{L}_{rec} = \|\hat{\mathbf{x}} - \mathbf{x}\|^2$ and, (2) the quantization loss $\mathcal{L}_{quan} = \|\mathbf{z} - \operatorname{sg}[\mathbf{e}_r]\|^2$, where $\operatorname{sg}[\cdot]$ denotes the stop gradient operation. The final loss is given by $\mathcal{L} = \mathcal{L}_{rec} + \beta \mathcal{L}_{quan}$, where β is the hyperparameter to balance the two losses. The codebook is maintained by running average as proposed in Razavi *et al.* [176]. The residual $\mathbf{z} - \mathbf{e}_r$ will be used for deriving the second-digit token.

(1) *Code level 2 and 3.* When deriving the second digit, for every item that shares the first digit, we fit an individual k-means model with K as the number of clusters. Then the index of each cluster is used as the second-digit token. By fitting individual k-means models for residuals that share the same first-digit token, we can generate more balanced second-digit tokens. Such balanced distribution allows us to tokenize the same number of items with fewer codebook layers and smaller codebook sizes. In this way, the proposed balanced semantic ID (SID) tokenization method reduces the model’s decision space and computational consumption due to the shorter input sequences [166]. Eventually, we randomly assign the third-digit token for every item with

the same first and second digits to resolve the collision issue and ensure every SID uniquely identifies an item.

Balanced Chunked ID. We also propose a tokenizer that does not require any associated item features. For a given item ID, we first remap it into one random integer i . To reduce the space size $|\mathcal{V}|$ of these integers, we represent the integer in base k . This process ensures that each item is represented by a tuple of digits, where each digit serves as a token. Although this approach lacks semantic information from item features, it still generates effective item tokens by ensuring a balanced token distribution.

behavior, item Tokenizer. We use a separate behavior token to encode each type of behavior and add the behavior token before the item tokens to encode behavior information in the multi-behavior interaction sequence. Overall, we will tokenize each item $v \in \mathcal{V}$ into a tuple of tokens $C^v := [c_1^v, c_2^v, \dots, c_m^v]$ that uniquely identifies each item v . Combining the behavior token b , each interaction can be represented as the tuple $[b, c_1^v, c_2^v, c_3^v]$. In this way, the above-mentioned MBSR problem has been formalized as a *next-token prediction* task. Given the user interaction history $S'_u = [b^{v_1}, c_1^{v_1}, c_2^{v_1}, c_3^{v_1}, \dots, b^{v_{n-1}}, c_1^{v_{n-1}}, c_2^{v_{n-1}}, c_3^{v_{n-1}}]$, the task is to predict the next interaction $[b^{v_n}, c_1^{v_n}, c_2^{v_n}, c_3^{v_n}]$ token-by-token.

3.2.3 Generative Multi-Task Multi-Behavior Modeling

Sequence-to-Sequence Generative Framework. To solve the above-mentioned two-step MBSR problem, we first tokenize the original item and behavior sequence into one single token sequence using the item and behavior tokenizer proposed in Section 3.2.2. Then, the token sequences are fed into an encoder-decoder Transformer [208] module to generate the next tokens autoregressively. When the decoder is trained to give the next token prediction, it naturally learns the two-step prediction objective in the MBSR problem, *i.e.*, first predicts the next behavior to frame user intention, then predicts the next item given the next behavior type. Additionally, to capture the personalized sequential and behavior interaction pattern, we follow Rajput et al.

[174] and map the raw user IDs to U tokens using the hashing trick [233]. The mapped user ID is prepended to the input token sequence to help capture the personalized interaction features on the encoder side.

Multi-Task Capability. Learning from the proposed token sequences naturally gives MBGen multi-tasking capabilities. As mentioned, we formulate the MBSR problem into three tasks: target behavior item prediction, behavior-specific item prediction, and behavior-item prediction. For both target behavior item prediction and behavior-specific item prediction tasks, we prompt the sequence model with $[\langle \text{BOS} \rangle, b]$, where b is the target behavior token in target behavior item prediction task, and can also be an arbitrary behavior token in behavior-specific item prediction task. In this way, we effectively condition the generation of the following item tokens using the provided behavior token. For the behavior-item prediction task, the model is only provided with the $\langle \text{BOS} \rangle$ token and is responsible for generating both the behavior token and the item tokens. To the best of our knowledge, MBGen is the first model that can make predictions on the next behavior and the behavior-specific next-item prediction simultaneously.

3.2.4 The Position-and-Behavior-Aware Transformer

Position Routed Mixture of Experts. Note that the input token sequences described in Section 3.2.2 are heterogeneous by nature. For example, the first token in each behavior-item tuple always encodes the behavior information. Inspired by the Mixture of Expert (MoE) architecture [56, 100], we propose to set up multiple expert networks to model the sequence heterogeneity. The input of each digit from a behavior-item tuple $[b, c_1, c_2, c_3]$ will be routed to different expert networks. Existing MoE models typically use a trainable gating network to route the input to each expert network. It’s known that such a gating technique usually suffers from problems like representation collapse [29]. Thus, we propose to route the inputs purely by the relative position of each token. In detail, we duplicate the Feed Forward Network (FFN) in the Transformer blocks. The behavior token and three item tokens are routed to four different FFN

Table 3.1. Statistics of the preprocessed datasets. “Avg. n ” denotes the average length of item sequences. “Avg. w ” denotes the average number of words in the item text.

Datasets	#Users	#Items	#Inters.	Avg. n	Avg. w
Retail	147,894	97,842	7,651,493	78.20	51.74
IJCAI	423,119	351,221	32,685,371	93.06	77.25

experts, respectively. All other tokens (*e.g.*, $\langle EOS \rangle$, $\langle BOS \rangle$, user token) are routed to another FFN expert. Another merit of the proposed position routed module is that it efficiently scales up the model parameters without additional computational budget.

Behavior Injected Feed Forward Layer. To further enhance the model’s capability of understanding the fine-grained behavior-item interaction, we design an additional behavior embedding table in the FFN layers. Each FFN layer accepts an additional input of the current behavior type. The behavior types for the item tokens are decided by the associated behavior token, and the behavior types for the behavior tokens and the special tokens are set to be the padding token. The corresponding behavior embedding is then concatenated with the input of the FFN layer to facilitate a better mix of item and behavior information.

3.3 Experiment

3.3.1 Experiment Setup

Datasets

We evaluate our proposed MBGen model with the following two public datasets[245]:

Retail. The retail dataset is collected from Taobao, one of the world’s largest e-commerce platforms. There are four types of behaviors: purchase, add-to-cart, add-to-favorite, and click. The purchase behavior is seen as the target behavior in this dataset. Some baselines also refer to this dataset as “Taobao”/“Tmall”.

IJCAI. This dataset was used for the IJCAI competition and contains real-world data collected from e-commerce platforms. Similar to the retail dataset, there are four types of

behaviors: purchase, add-to-cart, add-to-favorite, and click. The purchase behavior is seen as the target behavior in this dataset.

We filter all the datasets to exclude the items that appear less than 5 times in the training sequence. The statistics of the filtered datasets are shown in Table 3.1. We do not use the Yelp dataset used as in [264, 195]. In the Yelp dataset, the user rating field (on a scale of 1 (worst) to 5 (best)) is split into three types of behaviors: dislike (rating ≤ 2), neutral ($2 < \text{rating} < 4$), and like (rating ≥ 4). However, this doesn't align with the real-life distribution of behaviors.

Table 3.2. Performance comparison of different models on the target behavior prediction task. The best performance is denoted in bold font. We use the underlined font to denote the best performance in baseline models if one variant of our proposed MBGen achieves the best performance. “*” indicates that our method statistically significantly (*i.e.*, $p\text{-value} \leq 0.05$ in the paired t-test) outperform the best baseline.

Model Type	Model	Retail				IJCAI			
		HR@5	NDCG@5	HR@10	NDCG@10	HR@5	NDCG@5	HR@10	NDCG@10
Multi-Behavior Recommendation	MB-GCN	0.0485	0.0316	0.0732	0.0396	N/A	N/A	N/A	N/A
	MB-GMN	0.0012	0.0008	0.0022	0.0011	N/A	N/A	N/A	N/A
Sequential Recommendation	GRU4Rec	0.2869	0.206	0.3554	0.2282	0.2117	0.1539	0.272	0.1734
	BERT4Rec _M	0.0807	0.0670	0.0934	0.0711	0.0617	0.0457	0.0799	0.0516
	SASRec _M	0.3041	0.2243	0.3678	0.2449	0.2294	0.1785	0.2818	0.1954
	TIGER	0.2887	0.2049	0.3523	0.2263	0.3728	0.2595	0.4310	0.2788
Multi-Behavior Sequential Recommendation	BERT4Rec _B	0.005	0.0032	0.0076	0.0040	0.0298	0.0207	0.0417	0.0245
	SASRec _B	0.1369	0.0945	0.1788	0.1081	0.1295	0.0950	0.1662	0.1069
	PBAT	0.1112	0.075	0.1545	0.089	0.1497	0.1048	0.2011	0.1214
	MBHT	0.2984	0.2183	0.3608	0.2385	0.1652	0.1249	0.2069	0.1384
	MB-STR	0.2814	0.2066	0.3500	0.2289	0.1874	0.1386	0.236	0.1544
Multi-Behavior Generative Recommendation	MBGen (SID)	0.4859*	0.3800*	0.5329*	0.3954*	0.4895*	0.3498*	0.6285*	0.3949*
	MBGen (CID)	0.4886*	0.3815*	0.5245*	0.3934*	0.4866*	0.3454*	0.6281*	0.3913*
	# Improve	+60.67%	+70.08%	+44.89%	+61.45%	+31.30%	+34.45%	+45.82%	+41.64%

Baselines

We compare the proposed approach with the following baseline methods:

(1) Sequential Recommendation Models. For all sequential recommendation models, we treat the multi-behavior item sequence as a regular single-behavior item sequence to enhance the sequential baseline models.

- **GRU4Rec** [201] uses a Gated Recurring Unit (GRU) to enhance long-term memory in recommendation tasks.

- **SASRec_M** [108] utilizes a self-attentive model to capture item correlations. We use the **M** subscript to denote the usage of multi-behavior sequence, aligning with the notation in previous works [264].

- **BERT4Rec_M** [197] adopts a bi-directional self-attentive model with a cloze objective for sequence modeling. We use the **M** subscript to denote the usage of multi-behavior sequence, aligning with the notation in previous works [264].

- **TIGER** [174] proposes a novel learned Semantic ID representation for each item and achieved State of the Art (SOTA) performance in sequential recommendation tasks.

(2) Multi-behavior Recommendation Models.

- **MB-GCN** [103] leverages a unified graph structure to represent multiple types of user-item interactions.

- **MB-GMN** [245] incorporates a meta-learning paradigm to capture interaction diversity and behavior heterogeneity.

(3) Multi-behavior Sequential Recommendation Models.

- **HSTU** [271] uses actions to unify retrieval and ranking in the same model. In our reproduction, we include behaviors as “actions”. In detail, we place the item and behavior tokens interleaved as $\langle \text{item}, \text{behavior} \rangle_i$ for each interaction.

- **MBHT** [258] utilizes hypergraph learning to uncover item-wise multi-behavior correlations.

- **PBAT** [195] takes the user’s personalized behavior interaction pattern into account using learned user-associated features.

- **MB-STR** [264] adopts a sparse Mixture of Experts (MoE) architecture and sequential pattern generator to handle the semantic gap between different behaviors.

- **SASRec_B** [108] and **BERT4Rec_B** [197] are enhanced from the original sequential recommendation methods by seeing each item with a different behavior type as a new item, thus building a new set of items of size $|\mathcal{V}| \times |\mathcal{B}|$. The enhanced models gain the ability to perform both behavior-specific and behavior-item prediction tasks. We denote the enhanced version of SASRec as **SASRec_B**, and the enhanced version of BERT4Rec as **BERT4Rec_B**.

Evaluation Settings

Following previous works [264, 195, 258], we adopt two widely used ranking metrics, Recall@K and NDCG@K, where $K \in \{5, 10\}$. For dataset splitting, we apply the leave-one-out strategy, *i.e.* the latest interacted item as test data, the item before the last one as validation data. Different from previous works [264, 195, 258], the ground-truth item of each sequence is ranked among *all the other items* instead of some sampled negative items for reliable evaluation [113]. We conduct a beam search with 50 beams to rank the top 10 candidates in all items to evaluate the Recall@K and NDCG@K metrics for the proposed MBGen model. All item sequences are truncated to a length of 50 for training and evaluation of our proposed model and the baselines.

Implementation Details

We reproduce most of the baselines (apart from MB-GMN) in the RecBole [285, 286] framework and use the same truncated item sequence to train all the models.

Item Tokenizer. We use a vocabulary size of $(K \times m)$ where $m = 3$ for all datasets. $K = 96$ for the IJCAI dataset, $K = 64$ for the retail dataset. Unlike previous works [174, 104, 132] that require associated item features as additional input, we use the embedding table of a pretrained sequential recommendation model (MB-STR [264] in our case) as the item feature

and input of semantic ID generation. In the first quantization phase of balanced SID, the Quantized Auto-Encoder has hidden layers of size [2048, 1024, 512, 256] in its encoder and decoder with the ReLU activation function. The latent dimension is set to be 32. The codebook size is the same as K for each dataset. We use a batch size of 2048, a learning rate of 0.001, and a β of 0.25 to train the model using the Adagrad optimizer until full convergence. We choose the same k for the two datasets in the generation of balanced CID.

Sequence-to-Sequence Model. We implement our sequence-to-sequence prediction model based on the Switch Transformers [56] architecture. We use a model dimension of 256, an inner dimension of 512 with ReLU activation function, and 6 heads of dimension 64 in the self-attention layer. The model has 4 regular encoder layers and 4 sparse decoder layers, and we add behavior injection in the first two encoder and decoder layers. Each sparse layer has 5 of experts in the sparse FFN layer. We set the batch size to 512, learning rate to 0.001, and trained our model with the AdamW optimizer for 350000 steps on both datasets. The learning rate is tuned in $\{0.0008, 0.001, 0.0012\}$ and the weight decay is tuned in $\{0.005, 0.001\}$. The models with the best NDCG@10 performance on the validation set are selected to be tested on the test set.

3.3.2 Overall Performance

Target Behavior Item Prediction

We compare the two variants (using SID and CID proposed in Section 3.2.2, respectively) of our proposed MBGen with the baseline methods on the target behavior item prediction task and report the results in Table 3.2. We only use sequences that end with an interaction with target behavior for target behavior evaluation.

For baseline methods, we observe that all baselines that incorporate sequential information significantly outperform non-sequential baseline models (*i.e.*, MB-GCN, and MB-GMN), verifying that sequential patterns are a crucial part of the MBSR problem. Furthermore, we cannot finish training and evaluating the MB-GCN and MB-GMN models in a reasonable time

on the IJCAI dataset since it’s not designed for retrieval tasks on such a large-scale dataset. Different from the results reported in previous MBSR works, we observe that some sequential recommendation models (*e.g.*, SASRec, and MB-STR) outperform all the MBSR models. This could be caused by the following reasons: (1) Some previous works [258, 195] train the sequential recommendation baseline models using only the target behavior sequence, which undermines the performance of the sequential recommendation baselines. (2) Previous works [264] use the SASRec and GRU4Rec models that was trained with the original binary entropy loss instead of the cross entropy loss, which is known to provide a significant improvement over the original SASRec model [270]. (3) In our evaluation setting, the ground-truth item of each sequence is ranked among all the other items instead of some sampled negative items as in [264, 258, 195]. We also notice that the behavior-enhanced SASRec_B and BERT4Rec_B models perform worse than the original SASRec and Bert4Rec modes. This could be due to the increased sparsity of the data after we create additional items for each behavior type. Additionally, we observe that the generative recommendation method (*i.e.*, TIGER) significantly outperforms conventional sequential recommendation models (*e.g.*, SASRec and GRU4Rec) when retrieving from a large number of candidates in the IJCAI dataset. For the proposed MBGen model, both of its variants significantly outperform all baselines in both datasets by 30% to 70%.

Behavior-Specific Item Prediction

We compare the proposed MBGen model with the four baselines with the behavior-specific item prediction capability and present the results in Table 3.3. In this task, the model is given the behavior type of the last interaction that the user performs and is required to predict the interacted items. We note that all models perform worse on behavior-specific item prediction task than target behavior item prediction task. This follows our intuition the target behavior (*e.g.*, purchase) usually indicates a stronger preference than other behaviors (*e.g.*, click). Thus, it could be easier to predict interactions with target behavior than arbitrary behavior. The proposed MBGen model also performs best in the behavior-specific item prediction task among

Table 3.3. Performance comparison on the behavior-specific item prediction task.

Model	Retail		IJCAI	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
BERT4Rec _B	0.0021	0.0013	0.0037	0.0018
SASRec _B	0.0468	0.0580	0.0822	0.0946
PBAT	0.0346	0.0430	0.0799	0.0944
MB-STR	<u>0.0606</u>	<u>0.0715</u>	<u>0.1109</u>	<u>0.1257</u>
MBGen (SID)	0.2053	0.2131	0.3023	0.3317
MBGen (CID)	0.2030	0.2076	0.2972	0.3261

Table 3.4. Performance comparison on the behavior-item prediction task.

Model	Retail		IJCAI	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
BERT4Rec _B	0.0013	0.0007	0.0024	0.0010
SASRec _B	0.0299	0.0395	0.0287	0.0375
HSTU	<u>0.1579</u>	<u>0.1616</u>	<u>0.1607</u>	<u>0.1654</u>
MBGen (SID)	0.1712	0.1803	0.2155	0.2301
MBGen (CID)	0.1720	0.1800	0.2443	0.2730

all baselines.

Behavior-Item Prediction

We evaluate the performance of our proposed MBGen and other baselines on the behavior-item prediction task that requires the model to make predictions on both the behavior type and the item of the user’s next interaction. We count a prediction as correct if the prediction of both behavior and item matches the ground-truth behavior and item. As shown in Table 3.4, despite its lack of capability on the other two tasks, HSTU performs best among all baseline models. Our proposed MBGen model also outperforms all baseline methods in the behavior-item prediction task. We also note only a 10%-20% drop in performance compared to the behavior-specific prediction task, showing our model’s solid ability in the next-behavior prediction task in the MBSR problem.

Table 3.5. Ablation analysis on three MBSR tasks. The best performance is denoted in bold font, and underlined font is used to denote the best performance in ablation models if one variant of our proposed MBGen achieves the best performance. We use *PR* to denote the position routed sparse FFN layer, *BI* to denote the behavior injection module, and *IT* to denote the item tokenizer. N@5 and N@10 are short for NDCG@5 and NDCG@10, respectively.

Model	Retail						IJCAI					
	Target Behavior		Behavior Specific		Behavior-Item		Target Behavior		Behavior Specific		Behavior-Item	
	N@5	N@10	N@5	N@10	N@5	N@10	N@5	N@10	N@5	N@10	N@5	N@10
<i>PBA Transformer Modules Ablation</i>												
<i>w/o PR</i>	0.3765	0.3894	0.1977	0.2050	<u>0.1643</u>	<u>0.1730</u>	<u>0.3325</u>	<u>0.3778</u>	0.2897	<u>0.3188</u>	0.2122	0.2258
<i>w/o BI</i>	<u>0.3818</u>	<u>0.3970</u>	0.1946	0.2030	0.1593	0.1697	0.3138	0.3575	0.2756	0.3038	0.2029	0.2159
<i>w/o PR & BI</i>	0.3853	0.3999	<u>0.1979</u>	0.2059	0.1633	0.1729	0.3155	0.3607	0.2743	0.3036	0.1945	0.2085
<i>Item Tokenizer Ablation</i>												
HSTU	N/A	N/A	N/A	N/A	0.1579	0.1616	N/A	N/A	N/A	N/A	0.1607	0.1654
<i>w/o IT</i>	0.3565	0.3598	0.1946	0.1959	0.1585	0.1637	0.3318	0.3396	0.2875	0.2926	<u>0.2373</u>	<u>0.2506</u>
TIGER	0.2049	0.2263	N/A	N/A	N/A	N/A	0.2595	0.2788	N/A	N/A	N/A	N/A
RQ-VAE (128 ⁴)	0.3690	0.3861	0.1951	<u>0.2074</u>	0.1592	0.1715	0.2993	0.3265	0.2649	0.2833	0.1853	0.1947
RQ-VAE (256 ⁴)	0.3460	0.3634	0.1665	0.1799	0.1315	0.1445	0.3323	0.3684	<u>0.2909</u>	0.3156	0.2008	0.2149
<i>Ours</i>												
MBGen (SID)	0.3800	0.3954	0.2053	0.2131	0.1712	0.1803	0.3498	0.3949	0.3023	0.3317	0.2155	0.2301
MBGen (CID)	0.3815	0.3934	0.2030	0.2076	0.1720	0.1800	0.3454	0.3913	0.2972	0.3261	0.2443	0.2730

3.3.3 Ablation Study

We analyze how each of the proposed components affects final performance. Table 3.5 shows the performance of the two full variants of our proposed MBGen model and eight models with some key designs removed.

PBA Transformer Module Ablation

(1) *w/o* Position Routed Sparse FFN: The position routed sparse structure in the PBA Transformer model helps to scale up the model without an increase in inference cost. We see a degradation in performance due to the lack of model capacity, as expected.

(2) *w/o* Behavior Injection: Removing the behavior injection module undermines the model’s ability to capture the semantics of the behaviors at a token level.

(3) *w/o* Position Routed Sparse & FFN Behavior Injection: The ablation of both modules could largely hurt the model’s capacity and ability to capture fine-grained behavior patterns. This

observation is even more pronounced in the larger-scale IJCAI dataset.

Tokenizer Ablation

(1) HSTU Tokenizer: The tokenizer proposed in HSTU [271] tokenizes an interaction into an item token followed by a behavior token. As mentioned, we argue that this method doesn't adhere to the two-step nature of the MBSR problem. We observe a worse performance than the *w/o* item tokenizer variant due to its reversed behavior and item token order. Additionally, such a tokenizer also lacks the capability of conditioning next-item prediction on known behavior types.

(2) *w/o* Item Tokenizer: We remove the item tokenizer and replace the original 3 item tokens with one single item token. The model cannot model the token-level behavior-item interaction patterns without the item tokenizer. When the item tokenizer is removed, it's also harder to retrieve the next-item prediction from large amount of candidates. We do observe that this variant with the item tokenizer removed outperforms our proposed Semantic ID variant, and we hypothesize that this is caused by the complicated probability distribution when we retrieve top candidates using beam search. We detail this discussion in Section 3.3.4.

(3, 4, 5) RQ-VAE Tokenizer: We test the behavior-aware RQ-VAE tokenizer on two codebook sizes, which are both larger than the codebook size of the proposed balanced item tokenizer due to the above-mentioned code imbalance nature of the RQ-VAE tokenizer. Thus, an even smaller RQ-VAE codebook size is impractical due to the very serious collision problem. Both the code imbalance, the bigger codebook size, and more codebook layers make it hard for the model to perform next-item prediction. We perform quantitative analysis and further discuss the code distribution of the RQ-VAE tokenizer and our proposed balanced tokenizer in Section 3.3.4. Additionally, we also compare the behavior-aware RQ-VAE tokenizer with the original TIGER tokenizer without behavior-aware designs, which demonstrates the role that behavior token plays in our model. In addition to the incapability of behavior-specific and behavior-item predictions, we also observe a big performance decay in target behavior prediction

Table 3.6. Quantitative analysis on the code distribution of different item tokenizers. The lower the variance and collision, the more balanced the generated codes are.

Metrics	Retail		IJCAI	
	Balanced SID	RQ-VAE SID	Balanced SID	RQ-VAE SID
L1 Variance	90,093	152,325	3,651,624	7,562,833
L2 Variance	71	238	601	2,487
L3 Variance	0.2368	0.6778	0.024	1.15
L4 Collisions	N/A	51,083	N/A	220,263

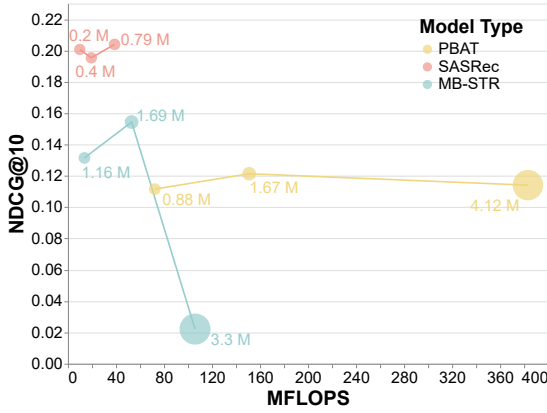


Figure 3.2. Baseline models scalability comparison on target behavior item prediction task on IJCAI dataset. The model computational budget is measured from one single forward call on inference.

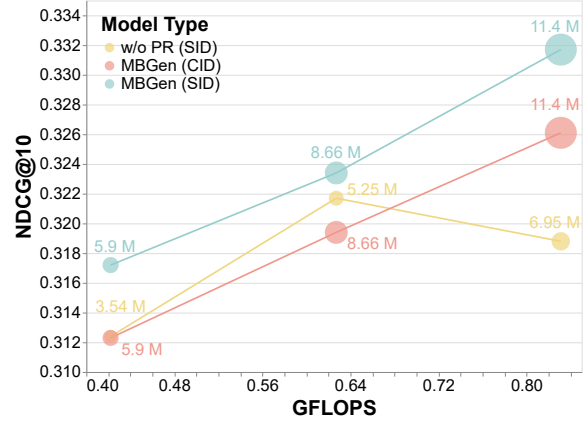


Figure 3.3. MBGen variants scalability comparison on behavior-specific item prediction task on IJCAI dataset. The model computational budget is measured from one single forward call on inference.

tasks. This shows the effectiveness of our designed behavior-aware tokenizer schema.

3.3.4 Further Analysis

Quantitative Analysis on Code Distribution

We perform a quantitative analysis on the empirical distribution, and the results are shown in Table 3.6. We omit the analysis for the Chunked IDs variant since it theoretically grants a perfectly balanced code distribution. For a fair comparison, we fit RQ-VAE models with the same codebook size as the codebook sizes of our proposed balanced semantic IDs. For the codebook layer m , we count the number of items with the same first m digits for each possible code combination. For instance, there are $64 \times 64 = 4096$ possible combinations on the first two

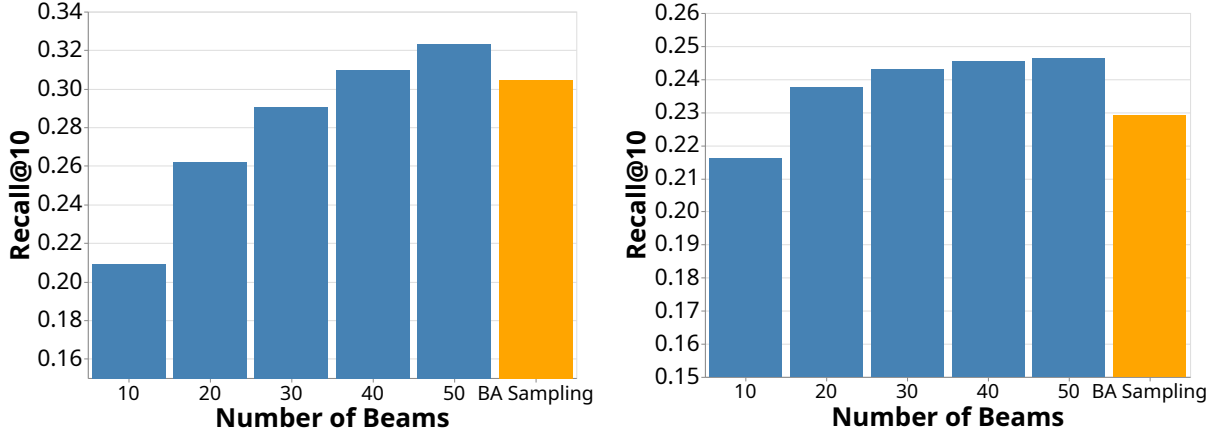


Figure 3.4. Comparison of retrieval performance with respect to different numbers of beams used in beam search along with the performance of proposed behavior aware sampling method on IJCAI (left) and Retail (right) datasets.

digits for the retail dataset. We calculate the histogram of the distribution of the first m digits. Then, we compare the variance of such empirical distribution (the smaller, the more balanced the code distribution). We observe that our proposed Balanced SID has a significantly more balanced code distribution in each codebook layer. Furthermore, we measure the number of collisions on the fourth layer in the RQ-VAE tokenizer; the result shows that more than half of the items in both datasets have some other items that share the same first three codes with them due to extremely unbalanced code assignment. In contrast, there are only three layers in our designed balanced item tokenizers.

Behavior-Aware Sampling

As shown in Figure 3.4, we observe that increasing the number of beams significantly improves the performance of our model on the behavior-item prediction task. We propose that this is caused by the inability to capture the complicated conditional probability distribution of the beam search algorithm. In both datasets, some behaviors (*e.g.*, purchase and add-to-favorite) appear much less often than others (*e.g.*, click). Such an unbalanced probability distribution can make it hard for the beam search to retrieve the behavior-item pairs that jointly have a high probability but with a rarer behavior type. To that end, we experiment with a behavior-aware

sampling method that achieves considerable improvement in the retrieval task without the need for more beams in the beam search. We first record the probability of each behavior and retrieve the same proportion of items conditioned on each behavior. For instance, if the probability of the four behaviors is $[0.3, 0.42, 0.18, 0.1]$, we use the beam search to get the top $[3, 4, 2, 1]$ results for each given behavior. We then combine the results as our top-10 behavior-item prediction. As shown in Figure 3.4, such a behavior-aware sampling method significantly outperforms the original beam search algorithm with 10 beams in both datasets.

Scalability Analysis

We show the relation between the model capacity and the model performance of baseline methods in Figure 3.2. We observe that the baseline models have a degradation of performance as we scale up the mode, especially the MB-STR and PBAT models. As in Figure 3.3, we observe that the variants of MBGen have better scalability when compared to the baseline models. Furthermore, we observe that our model with the position routed sparse FFN architecture performs better than the variant model without the sparse architecture with the same amount of computation consumption.

3.4 Conclusion

In this chapter, we propose MBGen, the first generative recommendation on the multi-behavior sequential recommendation problem. Different from previous models that only model the second step (next item prediction given next behavior) of the MBSR problem, we construct a unified multi-behavior generative recommendation paradigm that’s coherently trained on both steps of the MBSR problem. We further design a balanced behavior-aware item tokenizer that allows our model to learn from token-level fine-grained interaction patterns. Exploiting the heterogeneous nature of our constructed behavior-item sequence, we designed a position-routed sparse architecture to efficiently scale up the model. Extensive experiments are conducted with public datasets to show that our model significantly outperforms all baseline models in multiple

MBSR tasks.

Chapter 3, in part, is a reprint of the material as it appears in “Multi-Behavior Generative Recommendation.” by Zihan Liu*, Yupeng Hou*, and Julian McAuley, in Proceedings of the 33rd ACM International Conference on Information and Knowledge Management in 2024, referenced as [144]. The dissertation author was the primary investigator and author of this paper.

Chapter 4

ActionPiece: Contextually Tokenizing Action Sequences for Generative Recommendation

Observing the context-independent nature of existing action tokenization methods, this chapter asks whether making action tokenization context-aware would be more powerful, *i.e.*, whether the same action should be tokenized differently under different contexts.

Generative recommendation (GR) is an emerging paradigm where user actions are tokenized into discrete token patterns and autoregressively generated as predictions. However, existing GR models tokenize each action independently, assigning the same fixed tokens to identical actions across all sequences without considering contextual relationships. This lack of context-awareness can lead to suboptimal performance, as the same action may hold different meanings depending on its surrounding context. To address this issue, we propose ActionPiece to explicitly incorporate context when tokenizing action sequences. In ActionPiece, each action is represented as a *set* of item features. Given the action sequence corpora, we construct the vocabulary by merging feature patterns as new tokens, based on their co-occurrence frequency both within individual sets and across adjacent sets. Considering the unordered nature of feature sets, we further introduce set permutation regularization, which produces multiple segmentations of action sequences with the same semantics.¹

¹Our code is available at https://github.com/google-deepmind/action_piece.

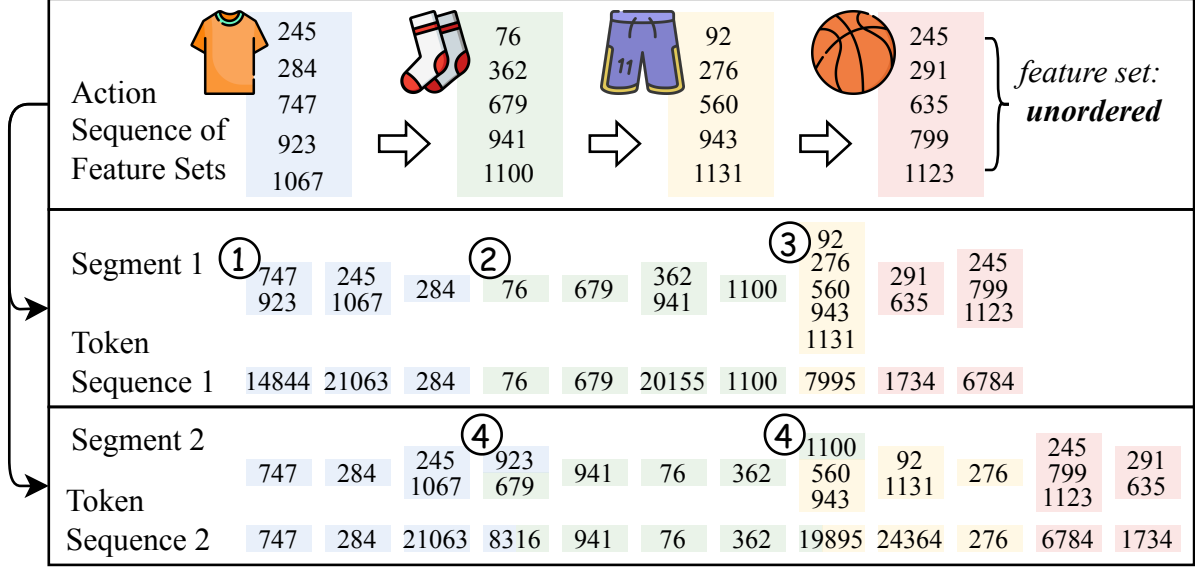


Figure 4.1. Illustration of the tokenization process of ActionPiece. Each action is represented as an unordered feature set. This figure presents two possible tokenized sequences, where features are grouped into different segments. The same action can be tokenized into different tokens depending on the surrounding context. A detailed case study can be found in Section 4.3.5.

4.1 Introduction

Generative recommendation (GR) [204, 174, 290, 271] is an emerging paradigm for the sequential recommendation task [82, 108]. By tokenizing the user actions (typically represented by the interacted items) into discrete tokens, GR models learn to autoregressively generate tokens, which are then parsed into recommended items. These tokens share a compact vocabulary that does not scale with the item pool size, improving model scalability, memory efficiency, and recommendation performance. The input action sequence is vital in understanding user intentions [82, 122, 108], which organizes a user’s historical interactions in chronological order. The same action (*e.g.*, purchasing the same item) may have different meanings in different action sequences. Evidence of taking a certain action can be found in the context, such as whether other items in the sequence share the same brand, color tone, or price range [280, 297, 87, 88, 266].

Despite the importance of contextual relations among actions, existing methods tokenize each action independently of its context (summarized in Table 4.1). The typical pipeline for

tokenizing action sequences involves two steps: (1) Tokenizing each action/item individually into a pattern of tokens; (2) Replacing each action in the input sequence with its corresponding token pattern. In this way, the tokens do not explicitly contain the context. Instead, they solely rely on the autoregressive model’s parameters being well-trained to generalize effectively in understanding the context, which challenges the capabilities of GR models. As a comparison, tokenization in language modeling also originates from context-independent methods, such as word-level tokenization [199, 5]. A decade of progress has led to most tokenization methods for modern large language models (LLMs) [162, 4, 206, 288] adopting context-aware approaches, including BPE [186] and Unigram tokenization [115], which tokenize the same word roots along with their adjacent context into different tokens.

In this chapter, we aim to make the first step towards context-aware tokenization for modeling *action sequences*. In analogy to how characters or bytes serve as the basic units in language modeling, we consider the associated features of an item as initial tokens. The idea is to iteratively find the most commonly co-occurring pairs of tokens among the training action sequences, then merge them into new tokens to represent segments of context. However, it’s non-trivial to achieve this. Unlike text, where characters naturally form a sequence, the features associated with an action form an unordered set [280, 297]. Thus, the proposed tokenization algorithm should be applied on *sequences of token sets*. We need to carefully consider which pairs of tokens should be counted, whether within a single set or between two adjacent sets, and how much weight should be given to these different types of relationships.

To this end, we propose **ActionPiece**, which enables the same actions to be tokenized into different tokens based on their surrounding context. **(1) Vocabulary construction** begins by initializing the vocabulary to include every unique feature as initial tokens. The vocabulary is then constructed by iteratively learning merge rules. Each merge rule specifies that a pair of tokens can be merged into a new token. In each iteration, we enumerate the training corpus to count the co-occurrence of existing tokens. Considering the structural differences between token pairs, *e.g.*, whether they occur within a single set or between two adjacent sets, we assign

Table 4.1. Comparison of different action tokenization methods for generative recommendation. “Contextual” denotes whether the same actions can be tokenized into different tokens based on the surrounding context. “Unordered” denotes whether the item features or semantic IDs are used in an order-agnostic manner.

Action Tokenization	Example	Contextual	Unordered
Product Quantization	VQ-Rec [88]	✗	✓
Hierarchical Clustering	P5-CID [96]	✗	✗
Residual Quantization	TIGER [174]	✗	✗
Text Tokenization	LMIndexer [104]	✗	✗
Raw Features	HSTU [271]	✗	✗
SentencePiece	SPM-SID [193]	✗	✗
ActionPiece	Ours	✓	✓

Algorithm 1. ActionPiece Vocabulary Construction

Input: Sequence corpus $\mathcal{S}'$, initial tokens $\mathcal{V}_0$, target size Q

Output: Merge rules $\mathcal{R}$, constructed vocabulary $\mathcal{V}$

```

1: Initialize vocabulary  $\mathcal{V} \leftarrow \mathcal{V}_0$  # Each initial token corresponds to one unique item feature
2:  $\mathcal{R} \leftarrow \emptyset$ 
3: while  $|\mathcal{V}| < Q$  do
4:   # Count: accumulate weighted token co-occurrences
5:    $\text{count}(\cdot, \cdot) \leftarrow \text{Count}(\mathcal{S}', \mathcal{V})$  # Algorithm 2
6:   # Update: merge a frequent token pair into a new token
7:    $\text{Select}(c_u, c_v) \leftarrow \arg \max_{(c_i, c_j)} \text{count}(c_i, c_j)$ 
8:    $\mathcal{S}' \leftarrow \text{Update}(\mathcal{S}', \{(c_u, c_v) \rightarrow c_{\text{new}}\})$  # Algorithm 3
9:    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(c_u, c_v) \rightarrow c_{\text{new}}\}$  # New merge rule
10:   $\mathcal{V} \leftarrow \mathcal{V} \cup \{c_{\text{new}}\}$  # Add new token to the vocabulary
11: end while
return  $\mathcal{R}, \mathcal{V}$ 

```

different weights to different pairs. **(2) Segmentation** refers to dividing raw features in action sequences into groups that can be replaced by tokens from the vocabulary. To fully exploit the unordered nature of the feature set of each action, we introduce set permutation regularization. By randomly permuting the features within each set, we can produce multiple token sequences of a single action sequence that preserve the same semantics. These variations act as natural augmentations for training data and enable inherent ensembling during model inference.

4.2 Method

In this section, we present **ActionPiece**, a context-aware method for tokenizing action sequences for generative recommendation. First, we formulate the task in Section 4.2.1. Then, we introduce the proposed tokenizer, covering vocabulary construction and segmentation, in Section 4.2.2. Finally, we describe the model training and inference process using ActionPiece-tokenized sequences in Section 4.2.3.

4.2.1 Problem Formulation

Given a user’s historical actions $S = \{i_1, i_2, \dots, i_t\}$, organized sequentially by their timestamps, the task is to predict the next item i_{t+1} the user will interact with.

Action as an unordered feature set. In the development of modern recommender systems, each item i_j is usually associated with a set of features $\mathcal{A}_j$ [280, 297, 28]. Assuming there are m features per item, the k -th feature of item i_j is denoted as $f_{j,k} \in \mathcal{F}_k$, where $\mathcal{F}_k$ is the collection of all possible choices for the k -th feature. Compared to representing actions using ordered semantic IDs (*e.g.*, those produced by RQ-VAE [174, 193]), the unordered set setting offers two key advantages: (1) It does not require a specific order among features, which aligns better with how items or actions are represented in most recommender systems; (2) It enables the inclusion of more general discrete and numeric features, such as *category*, *brand*, and *price* [164, 107].

Action sequence as a sequence of sets. Representing each item as an unordered set, the input action sequence can be written as $S' = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_t\}$, which is a chronologically ordered sequence of sets. There is no order within each set, but there are orders between the features from different sets. The tokenizer design should account for the ordered and unordered relationships among features.

Generative recommendation task. In this chapter, we aim to design a tokenizer that maps an input action sequence S' to a token sequence $C = \{c_1, c_2, \dots, c_l\}$, where l denotes the

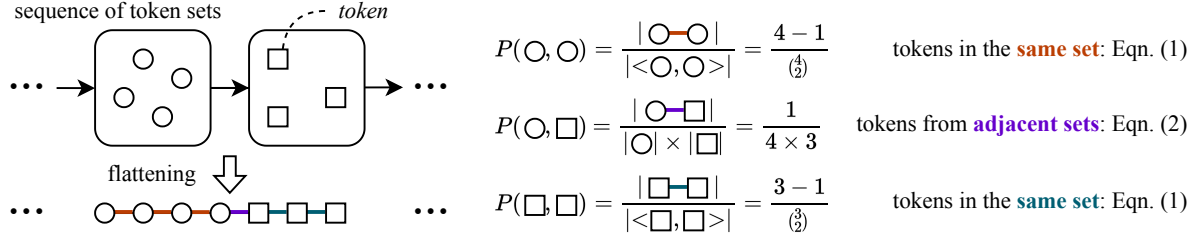


Figure 4.2. Illustration of how weights of co-occurring token pairs are counted during vocabulary construction. In this example, two adjacent sets in the sequence are considered: one with 4 tokens (represented as $\bigcirc$) and another with 3 tokens (represented as $\square$). Token pairs are counted within a single set ($\langle \bigcirc, \bigcirc \rangle$ and $\langle \square, \square \rangle$) and across the two adjacent sets ($\langle \bigcirc, \square \rangle$).

number of tokens in the sequence. Note that l is typically greater than the number of actions t . Next, we train a GR model to autoregressively generate tokens $\{c_{l+1}, \dots, c_q\}$, which can be parsed as next-item predictions $\hat{i}_{t+1}$.

4.2.2 Contextual Action Sequence Tokenizer

The proposed tokenizer is designed to transform action sequences (represented as sequences of feature sets) into token sequences. In the ActionPiece-tokenized sequences, each token corresponds to a set containing varying numbers of features. For example, a token can represent: (1) a subset of features from one item; (2) a single feature; (3) all features of one item; or (4) features from multiple items. We also label these four types of tokens in Figure 4.1. Below, we first describe how to construct the ActionPiece tokenizer’s vocabulary given a corpus of action sequences (Section 4.2.2). Then, we introduce how to segment action sequences into a new sequence of sets, where each set corresponds to a token from the constructed vocabulary (Section 4.2.2).

Vocabulary Construction on Action Sequence Corpus

Given a corpus of action sequences $\mathcal{S}'$, the goal of vocabulary construction is to create a vocabulary $\mathcal{V}$ of Q tokens. Each token represents a combination of features that frequently occur in the corpus. Similar to BPE [186], we construct the vocabulary using a bottom-up approach. The process starts with an **initial vocabulary** of tokens $\mathcal{V}_0$. The construction proceeds iteratively,

adding one new token to the vocabulary at each iteration until the predefined target size is reached. Each iteration consists of two consecutive steps: **count**, where the most frequently occurring token pair is identified, and **update**, where the corpus is modified by merging the selected pair into a new token. An algorithmic workflow is illustrated in Algorithm 1.

Vocabulary initialization. In BPE, each token represents a sequence of bytes. Thus, the most fundamental units—the initial tokens—are single bytes, which form the initial vocabulary of BPE. Similarly, each token in ActionPiece represents a set of features. Therefore, we initialize ActionPiece with a vocabulary in which each token represents a set containing one unique item feature. Formally, we denote the initial vocabulary as $\mathcal{V}_0 = \{c = \{f\} | f \in \mathcal{F}_1 \cup \dots \cup \mathcal{F}_m\}$. After initializing the vocabulary, each action sequence (of feature sets) can be represented as a sequence of token sets.

Count: context-aware token co-occurrence counting. In each iteration of vocabulary construction, the first step is to count the co-occurrence of token pairs in the corpus. These pairs capture important feature combinations, which are encoded by creating new tokens. There are two types of token co-occurrence within a sequence of sets: (1) two tokens exist within the same set, or (2) two tokens exist in adjacent sets in the sequence. Notably, the second type allows ActionPiece to explicitly include context information.

Weighted co-occurrence counting. In one-dimensional token sequences (e.g., text), all token pairs are typically treated equally. However, in sequences of token sets, token pairs vary based on their types and the sizes of their respective sets. To account for these differences, we propose assigning different weights to token pairs. To determine the weight for each token pair, we relate sequences of token sets to token sequences by randomly permuting the tokens within each set and flattening them into a single token sequence. Let $P(c, c')$ represent the expected probability that tokens c and c' are adjacent in the flattened sequence. For two tokens from the

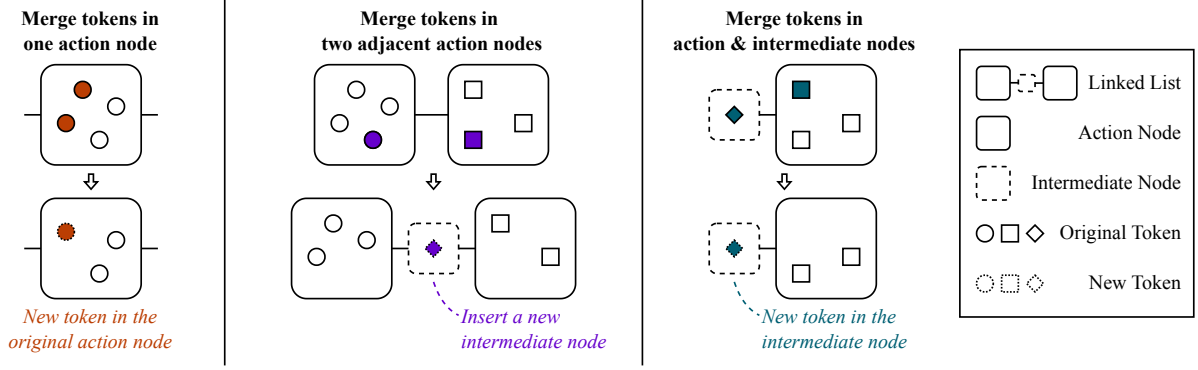


Figure 4.3. Illustration of how the linked list, which maintains the action sequence, is updated when merging two tokens into a new token. Three cases are considered: (1) both tokens are in the same action node; (2) the tokens are in two adjacent action nodes; (3) one token is in an action node, while the other is in an intermediate node.

same set, we have:

$$P(c_1, c_2) = P(c_2, c_1) = \frac{|\mathcal{A}_i| - 1}{\binom{|\mathcal{A}_i|}{2}} = \frac{2}{|\mathcal{A}_i|}, \quad c_1, c_2 \in \mathcal{A}_i, \quad (4.1)$$

and for two tokens from adjacent sets, we have:

$$P(c_1, c_3) = \frac{1}{|\mathcal{A}_i| \times |\mathcal{A}_{i+1}|}, \quad c_1 \in \mathcal{A}_i, c_3 \in \mathcal{A}_{i+1}. \quad (4.2)$$

By considering the probabilities of all adjacent token pairs in the flattened sequence as 1, the weights for token pairs in the original sequence of token sets correspond to the probabilities given in Equations (4.1) and (4.2). An illustration is shown in Figure 4.2.

Accumulating co-occurrence weights. The weights described above are calculated based solely on the co-occurrence type and the set size. They do not take into account the specific tokens being analyzed. Tokens c_i and c_j might appear in the same set in one sequence but in two adjacent sets in another sequence. By iterating through the corpus, we sum up the weights for each token pair whenever they appear together multiple times.

Update: corpus updating with action-intermediate nodes. The next step in each iteration is to merge the token pair with the highest accumulated co-occurrence weight. Since

token merging may change the set size, we use a double-ended linked list [308] to maintain each action sequence, where each node represents a set of tokens. Merging tokens within the same set is straightforward, *i.e.*, replacing the two tokens with a new one. However, merging tokens from two adjacent sets is more complex, *e.g.*, determining which set should include the new token.

Intermediate Node. We introduce the concept of “intermediate node” to handle tokens that combine features from multiple sets. Initially, all nodes in the maintained linked lists contain features specific to their corresponding actions. These nodes are referred to as “action nodes.”

(1) When tokens from two adjacent action nodes are being merged, we insert a new intermediate node between the two action nodes. The new token is stored in the intermediate node, and the merged tokens are removed from their respective action nodes;

(2) When merging tokens from an action node and an intermediate node, the new token replaces the original token in the intermediate node. The reason is that this new token also combines features from multiple actions. After the merge, the token from the action node is removed.

Following the above update rules ensures that there is at most one intermediate node between any two action nodes, and each intermediate node contains no more than one token. When calculating co-occurrence weights involving an intermediate node, it can simply be treated as a set of size 1.

Efficient implementation. Naively counting and updating the corpus requires a total time complexity of $O(QNLm^2)$, where Q is the target vocabulary size, N is the number of action sequences in the training corpus, and L is the average length of these sequences. However, it is unnecessary to count co-occurrences from scratch in each iteration. This is because only a small portion of the maintained linked lists is modified compared to the previous iteration.

Data structures. To address this, we propose creating inverted indices to map token pairs to all the linked lists that contain them. A global heap is maintained to return the token pair with the highest accumulated co-occurrence. The key challenge lies in updating these data structures. We carefully compute the changes in accumulated co-occurrences and update the inverted indices.

For the heap, we employ a lazy-update strategy. We insert the latest weights with a tag. When fetching a value from the heap, we check the tag to verify if the value is up-to-date. If it is not, we discard the value and fetch the next one.

Time complexity. Let $H = O(NLm)$ represent the maximal heap size. Using the proposed algorithm, we successfully reduce the original time complexity to $O(\log Q \log H \cdot NLm^2)$, achieving efficient vocabulary construction. In practice, the later iterations take significantly less time than the initial ones. This is expected and because tokens with higher accumulated co-occurrence weights typically appear frequently in the early stages. However, the overall construction time benefits from the reduced amortized complexity. Further details about the vocabulary construction algorithm are provided in Section 4.6.

Segmentation by Set Permutation Regularization

Segmentation is to convert original action sequences into a sequence of feature sets. Each set in the segmented sequence corresponds to a token in the vocabulary.

Naive segmentation. One segmentation strategy in ActionPiece involves applying the same technique used to construct the vocabulary. Specifically, this technique iteratively identifies token pairs with high priorities (represented by the IDs of tokens, where tokens added earlier may have higher priority). However, we observed that this strategy can lead to a bias, where only a subset of tokens in the vocabulary is frequently used (as shown empirically in Section 4.3.4).

Set permutation regularization (SPR). To address this issue and account for the unordered nature of sets, we propose *set permutation regularization*, which generates multiple segmentations for each action sequence. The key idea is to avoid enumerating all possible pairs between tokens in a set or adjacent sets. Instead, we generate a random permutation of each set and treat it as a one-dimensional sequence. By concatenating all the permutations, we create a long token sequence. This sequence can then be segmented using traditional BPE segmentation methods [186]. In this approach, different permutations can produce distinct segmented token sequences with the same semantics. These sequences serve as natural augmentations for model

Table 4.2. Comparison between the widely used text tokenization method, byte-pair encoding (BPE) [186], and the proposed context-aware action sequence tokenization method, ActionPiece.

Aspect	BPE	ActionPiece
Data Type	text sequences	action (unordered feature set) sequences
Token	a byte sequence	a feature set
Initial Vocabulary	single bytes	single-feature sets
Merging Unit	adjacent byte pairs	feature pairs within one set or between adjacent sets
Co-occurrence Weighting	raw frequency counting	probabilistic weighting (Figure 4.2)
Segmentation Strategy	greedy fixed-order merging	set permutation regularization (Algorithm 4)
Intermediate Structures	N/A	intermediate nodes for cross-action merges

training (Section 4.2.3) and enable inherent ensembling during model inference (Section 4.2.3).

4.2.3 Generative Recommendation Models

Training on Augmented Token Sequences

For an action sequence and its ground-truth next action in the training corpus, we tokenize them into token sequences C_{in} and C_{out} , respectively. Taking C_{in} as input, we train a Transformer encoder-decoder module [173] to autoregressively generate C_{out} (*e.g.*, next-token prediction objective [174]). During training, we tokenize the action sequence using the set permutation regularization described in Section 4.2.2 in each epoch. This approach naturally augments the training sequences, which empirically improves model performance, as shown in Section 4.3.3.

Inference-Time Ensembling

During model inference, we tokenize each action sequence q times using set permutation regularization. By passing these q tokenized sequences through the model, we obtain q output ranking lists (*e.g.*, using beam search for inference when TIGER [174] is the GR backbone). We then combine these ranking lists by averaging the scores of each predicted item. This approach applies data-level ensembling, which has been shown to enhance recommendation performance, as discussed in Section 4.3.4.

4.2.4 Discussion

Orders in action sequences. In recommender systems, user actions are typically represented as sets of features, such as the title and price of the associated item. These features typically have no inherent order within a single action. However, in sequential recommendation tasks, a user’s historical actions are usually ordered by timestamp to capture temporal behavioral dynamics. Building on this, we model action sequences as sequences of feature sets: while the features within each action remain unordered, the temporal ordering of actions is preserved. This evolving composition of features over time captures meaningful sequential patterns.

ActionPiece vs. BPE. While ActionPiece follows a similar algorithmic framework as BPE, the key distinction lies in the data formats they are designed to model. BPE operates on one-dimensional byte sequences, whereas ActionPiece is tailored for tokenizing sequences of feature sets. Modeling each action as an unordered set aligns better with the inherent structure of action sequences. For clarity, we summarize the key differences in Table 4.2.

Efficiency impact of SPR. Despite introducing set permutation regularization, the training efficiency remains comparable to existing methods such as TIGER. Feature permutation is performed on the CPU and runs asynchronously alongside TPU/GPU-based model updates, resulting in no noticeable degradation in training speed. At inference time, SPR introduces additional FLOPs due to the ensemble of augmented test cases. However, the overall latency remains comparable to baseline methods, as the augmented versions can be processed in parallel across multiple computing devices (*e.g.*, TPUs or GPUs). This parallelism offsets the added computation, enabling our method to maintain efficient inference despite the use of inference-time ensembling.

Table 4.3. Statistics of the processed datasets. “Avg. t ” denotes the average number of actions in an action sequence.

Datasets	#Users	#Items	#Actions	Avg. t
Sports	35,598	18,357	260,739	8.32
Beauty	22,363	12,101	176,139	8.87
CDs	75,258	64,443	1,022,334	14.58

Table 4.4. Performance comparison of different methods on the Amazon Reviews dataset [147]. The best and second-best performance is denoted in **bold** and underlined fonts. “R@K” and “N@K” are short for “Recall@K” and “NDCG@K”, respectively. “Improv.” denotes the percentage improvement of our method compared to the strongest baseline method.

Datasets	Metric	ID-based		Feature + ID			Generative					Improv.	
		BERT4Rec	SASRec	FDSA	S ³ -Rec	VQ-Rec	P5-CID	TIGER	LMIndexer	HSTU	SPM-SID		ActionPiece
Sports	R@5	0.0115	0.0233	0.0182	0.0251	0.0181	0.0287	0.0264	0.0222	0.0258	<u>0.0280</u>	0.0316 ± 0.0005	+12.86%
	N@5	0.0075	0.0154	0.0122	0.0161	0.0132	0.0179	<u>0.0181</u>	0.0142	0.0165	0.0180	0.0205 ± 0.0002	+11.71%
	R@10	0.0191	0.0350	0.0288	0.0385	0.0251	0.0426	0.0400	—	0.0414	<u>0.0446</u>	0.0500 ± 0.0007	+12.11%
	N@10	0.0099	0.0192	0.0156	0.0204	0.0154	0.0224	0.0225	—	0.0215	<u>0.0234</u>	0.0264 ± 0.0003	+12.82%
Beauty	R@5	0.0203	0.0387	0.0267	0.0387	0.0434	0.0468	0.0454	0.0415	0.0469	<u>0.0475</u>	0.0511 ± 0.0014	+7.58%
	N@5	0.0124	0.0249	0.0163	0.0244	0.0311	0.0315	0.0321	0.0262	0.0314	<u>0.0321</u>	0.0340 ± 0.0011	+5.92%
	R@10	0.0347	0.0605	0.0407	0.0647	<u>0.0741</u>	0.0701	0.0648	—	0.0704	0.0714	0.0775 ± 0.0017	+4.59%
	N@10	0.0170	0.0318	0.0208	0.0327	0.0372	<u>0.0400</u>	0.0384	—	0.0389	0.0399	0.0424 ± 0.0011	+6.00%
CDs	R@5	0.0326	0.0351	0.0226	0.0213	0.0314	0.0505	0.0492	—	0.0417	<u>0.0509</u>	0.0544 ± 0.0005	+6.88%
	N@5	0.0201	0.0177	0.0137	0.0130	0.0209	0.0326	0.0329	—	0.0275	<u>0.0337</u>	0.0359 ± 0.0004	+6.53%
	R@10	0.0547	0.0619	0.0378	0.0375	0.0485	<u>0.0785</u>	0.0748	—	0.0638	0.0778	0.0830 ± 0.0008	+5.73%
	N@10	0.0271	0.0263	0.0186	0.0182	0.0264	0.0416	0.0411	—	0.0346	<u>0.0424</u>	0.0451 ± 0.0005	+6.37%

4.3 Experiments

4.3.1 Experimental Setup

Datasets. We use three categories from the Amazon Reviews dataset [147] for our experiments: “Sports and Outdoors” (**Sports**), “Beauty” (**Beauty**), and “CDs and Vinyl” (**CDs**). Each user’s historical reviews are considered “actions” and are sorted chronologically as action sequences, with earlier reviews appearing first. To evaluate the models, we adopt the widely used leave-last-out protocol [108, 287, 174], where the last item and second-to-last item in each action sequence are used for testing and validation, respectively. The statistics of the processed datasets are shown in Table 4.3. More details about the datasets can be found in Section 4.8.

Compared methods. We compare the performance of ActionPiece with the following

methods: (1) ID-based sequential recommendation methods, including BERT4Rec [197], and SASRec [108]; (2) feature-enhanced sequential recommendation methods, such as FDSA [280], S³-Rec [297], and VQ-Rec [88]; and (3) generative recommendation methods, including P5-CID [96], TIGER [174], LMIndexer [104], HSTU [271], and SPM-SID [193], each representing a different action tokenization method (Table 4.1). A detailed description of these baselines is provided in Section 4.9.

Evaluation settings. Following Rajput et al. [174], we use Recall@ K and NDCG@ K as metrics to evaluate the methods, where $K \in \{5, 10\}$. Model checkpoints with the best performance on the validation set are used for evaluation on the test set. We run the experiments with five random seeds and report the average metrics.

Implementation details. Please refer to Section 4.10 for detailed implementation and hyperparameter settings.

4.3.2 Overall Performance

We compare ActionPiece with sequential recommendation and generative recommendation baselines, which use various action tokenization methods, across three public datasets. The results are shown in Table 4.4.

For the compared methods, we observe that those using item features generally outperform item ID-only methods. This indicates that incorporating features enhances recommendation performance. Among the methods leveraging item features (“Feature + ID” and “Generative”), generative recommendation models achieve better performance. These results further confirm that injecting semantics into item indexing and optimizing at a sub-item level enables generative models to better use semantic information and improve recommendation performance. Among all the baselines, SPM-SID achieves the best results. By incorporating the SentencePiece model [116], SPM-SID replaces popular semantic ID patterns within each item with new tokens, benefiting from a larger vocabulary.

Our proposed ActionPiece consistently outperforms all baselines across three datasets,

Table 4.5. Ablation analysis of ActionPiece. The recommendation performance is measured using NDCG@10. The best performance is denoted in **bold** fonts.

Variants	Sports	Beauty	CDs
<i>TIGER with varying vocabulary sizes</i>			
(1.1) TIGER - 192 (4×48)	0.0231	0.0362	N/A [†]
(1.2) TIGER - 768 (3×2^8)	0.0220	0.0378	0.0331
(1.3) TIGER - 1k (4×2^8)	0.0225	0.0384	0.0411
(1.4) TIGER-49k (6×2^{13})	0.0162	0.0317	0.0338
(1.5) TIGER-66k (4×2^{14})	0.0194	N/A [‡]	0.0319
<i>Vocabulary construction</i>			
(2.1) w/o tokenization	0.0215	0.0389	0.0346
(2.2) w/o context-aware	0.0258	0.0416	0.0429
(2.3) w/o weighted counting	0.0257	0.0412	0.0435
<i>Set permutation regularization</i>			
(3.1) only for inference	0.0192	0.0316	0.0329
(3.2) only for training	0.0244	0.0387	0.0422
(3.3) TIGER + SPR	0.0202	0.0330	0.0351
ActionPiece (40k)	0.0264	0.0424	0.0451

[†]not applicable because the number of conflicts among semantic ID prefixes (the first three tokens) in CDs exceeds 48.

[‡]not applicable because 2^{14} is larger than #items in Beauty.

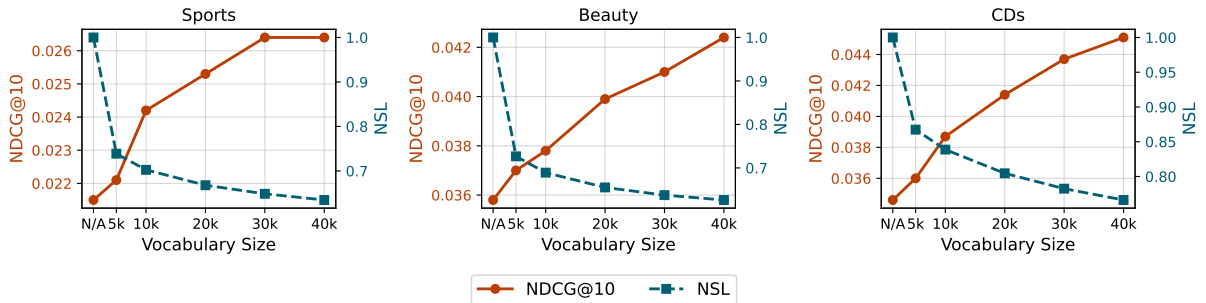


Figure 4.4. Analysis of recommendation performance (NDCG@10, ↑) and average tokenized sequence length (NSL, ↓) w.r.t. vocabulary size across three datasets. “N/A” indicates that ActionPiece is not applied, *i.e.*, action sequences are represented solely by initial tokens.

achieving a significant improvement in NDCG@10. It surpasses the best-performing baseline method by 6.00% to 12.82%. Unlike existing methods, ActionPiece is the first context-aware action sequence tokenizer, *i.e.*, the same action can be tokenized into different tokens depending

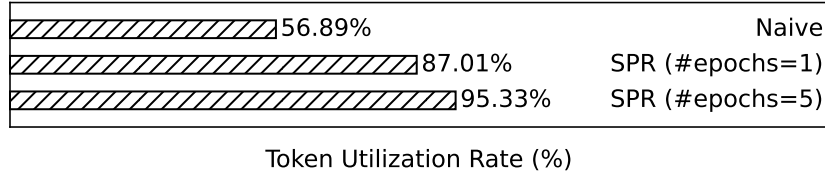


Figure 4.5. Analysis of token utilization rate (%) during model training w.r.t. segmentation strategy.

on its surrounding context. This allows ActionPiece to capture important sequence-level feature patterns that enhance recommendation performance.

4.3.3 Ablation Study

We conduct ablation analyses in Table 4.5 to study how each proposed technique contributes to ActionPiece.

(1) To examine whether the performance gain of ActionPiece stems from the choice of vocabulary size, we conduct an ablation study by varying the vocabulary size of TIGER. We increase the number of semantic ID digits per item ($4 \rightarrow 6$) and the number of candidate semantic IDs per digit ($2^8 \rightarrow 2^{13}$ or 2^{14}), resulting in two TIGER variants with vocabularies larger than ActionPiece. We also include two variants with reduced vocabulary sizes by decreasing the number of digits or candidates per digit. Despite the broader range of vocabulary sizes, all TIGER variants perform worse than ActionPiece, and in some cases, even worse than the original TIGER with 1,024 tokens. These results suggest that the performance improvement of ActionPiece is not simply due to scaling the vocabulary size up or down. Instead, they highlight the difficulty of effectively scaling vocabulary size in generative recommendation models, consistent with the observations from Zhang et al. [279].

(2) To evaluate the effectiveness of the proposed vocabulary construction techniques, we introduce the following variants: (2.1) *w/o tokenization*, which skips vocabulary construction, using item features directly as tokens; (2.2) *w/o context-aware*, which only considers co-occurrences and merges tokens within each action during vocabulary construction and segmentation; and (2.3) *w/o weighted counting*, which treats all token pairs equally rather than using the weights

defined in Equations (4.1) and (4.2). The results indicate that removing any of these techniques reduces performance, demonstrating the importance of these methods for building a context-aware tokenizer.

(3) To evaluate the effectiveness of SPR, we revert to naive segmentation, as described in Section 4.2.2, during model training and inference, respectively. The results show that replacing SPR with naive segmentation in either training or inference degrades performance. We also introduce an ablation variant that applies SPR to the existing GR model TIGER. The results indicate that SPR alone is insufficient to improve the GR model. When applied to action sequences tokenized by context-independent methods, SPR does not change token frequencies and merely disrupts the internal token order within RQ-VAE-based semantic IDs. In contrast, ActionPiece derives different tokens for the same item based on its neighboring context, improving token utilization and serving as an effective form of data augmentation.

4.3.4 Further Analysis

Performance and Efficiency w.r.t. Vocabulary Size

Vocabulary size is a key hyperparameter for language tokenizers [150, 35]. In this chapter, we investigate how adjusting vocabulary size affects the generative recommendation models. We use the normalized sequence length (NSL) [35] to measure the length of tokenized sequences, where a smaller NSL indicates fewer tokens per tokenized sequence. We experiment with vocabulary sizes in {N/A, 5k, 10k, 20k, 30k, 40k}, where “N/A” represents the direct use of item features as tokens. As shown in Figure 4.4, increasing the vocabulary size improves recommendation performance and reduces the tokenized sequence length. Conversely, reducing the vocabulary size lowers the number of model parameters, improving memory efficiency. This analysis demonstrates that adjusting vocabulary size enables a trade-off between model performance, sequence length, and memory efficiency.

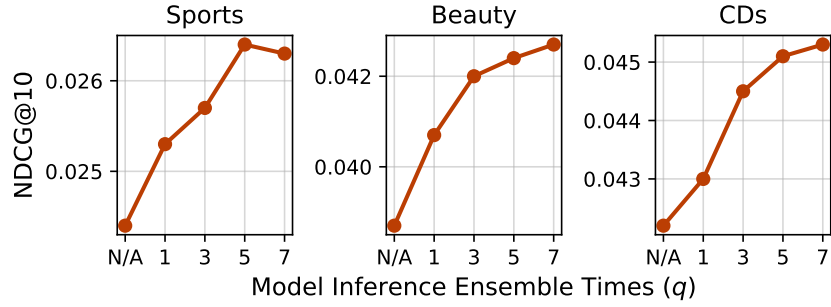


Figure 4.6. Analysis of performance (NDCG@10, $\uparrow$) w.r.t. the number of ensembled segments q during model inference.

Token Utilization Rate w.r.t. Segmentation Strategy

As described in Section 4.2.3, applying SPR augments the training corpus by producing multiple token sequences that share the same semantics. In Table 4.5, we observe that incorporating SPR significantly improves recommendation performance. One possible reason is that SPR increases token utilization rates. To validate this assumption, we segment the action sequences in each training epoch using two strategies: naive segmentation and SPR. As shown in Figure 4.5, naive segmentation uses only 56.89% of tokens for model training, limiting the model’s ability to generalize to unseen action sequences. In contrast, SPR achieves a token utilization rate of 87.01% after the first training epoch, with further increases as training progresses. These results demonstrate that the proposed SPR segmentation strategy improves the utilization of ActionPiece tokens, enabling better generalization and enhanced performance.

Performance w.r.t. Inference-Time Ensembles

As described in Section 4.2.3, ActionPiece supports inference-time ensembling by using SPR segmentation. We vary the number of ensembled segments, q , in $\{\text{N/A}, 1, 3, 5, 7\}$, where “N/A” indicates using naive segmentation during model inference. As shown in Figure 4.6, ensembling more tokenized sequences improves ActionPiece’s recommendation performance. However, the performance gains slow down as q increases to 5 and 7. Since a higher q also increases the computational cost of inference, this creates a trade-off between performance and computational budget in practice.

4.3.5 Case Study

To understand how GR models benefit from the unordered feature setting and context-aware action sequence tokenization, we present an illustrative example in Figure 4.1.

Each item in the action sequence is represented as a feature set, with each item consisting of five features. The features within an item do not require a specific order. The first step of tokenization leverages the unordered nature of the feature set and applies set permutation regularization (Section 4.2.2). This process arranges each feature set into a specific permutation and iteratively groups features based on the constructed vocabulary (Section 4.2.2). This results in different segments that convey the same semantics. Each segment is represented as a sequence of sets, where each set corresponds to a token in the vocabulary.

By examining the segments and their corresponding token sequences, we identify four types of tokens, as annotated in Figure 4.1: (1) a subset of features from a single item (token 14844 corresponds to features 747 and 923 of the T-shirt); (2) a set containing a single feature (feature 76 of the socks); (3) all features of a single item (token 7995 corresponds to all features of the shorts); and (4) features from multiple items (*e.g.*, token 8316 includes feature 923 from the T-shirt and feature 679 from the socks, while token 19895 includes feature 1100 from the socks as well as features 560 and 943 from the shorts). Notably, the fourth type of token demonstrates that the features of one action can be segmented and grouped with features from adjacent actions. This results in different tokens for the same action depending on the surrounding context, showcasing the context-aware tokenization process of ActionPiece.

4.4 Notations

We summarize the notations used in this chapter in Table 4.6.

Table 4.6. Notations and explanations.

Notation	Explanation
i, i_1, i_j	Item, item identifier, item ID
t	The number of actions in the input action sequence; the timestamp when the model makes a prediction
i_{t+1}	The ground-truth next item
$\hat{i}_{t+1}$	The predicted next item
$S = \{i_1, i_2, \dots, i_t\}$	The action sequence where each action is represented with the interacted item ID
$\mathcal{A}, \mathcal{A}_1, \mathcal{A}_j$	A set of item features or tokens
$m = \mathcal{A}_j $	The number of features associated with each item
$f_{j,k}$	The k -th feature of item i_j
$\mathcal{F}_k$	The collection of all possible choices for the k -th feature
$S' = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_t\}$	The action sequence where each action is represented with a set of item features
c, c_1, c_j	Input & generated tokens
l	The number of tokens in the token sequence
$C = \{c_1, c_2, \dots, c_l\}$	The token sequence tokenized from the input action sequence S'
$\{c_{l+1}, \dots, c_q\}$	The tokens generated by the GR model
$\mathcal{V}$	The vocabulary of ActionPiece tokenizer
$\mathcal{R}$	The merge rules of ActionPiece tokenizer
$\{(c_u, c_v) \rightarrow c_{\text{new}}\}$	One merge rule indicating two adjacent tokens c_u and c_v can be replaced by a token c_{new}
$Q = \mathcal{V} $	The size of ActionPiece vocabulary
$P(c, c')$	The probability that tokens c and c' are adjacent when flattening a sequence of sets into a token sequence
N	The number of action sequences in the training corpus
L	The average length of action sequences in the training corpus
H	Maximal heap size, $O(NLm)$
q	The number of segmentations produced using set permutation regularization during inference

4.5 Algorithmic Details

In this section, we provide detailed algorithms for vocabulary construction and segmentation.

4.5.1 Vocabulary Construction Algorithm

The overall procedure for vocabulary construction is illustrated in Algorithm 1. As described in Section 4.2.2, this process involves iterative **Count** (Algorithm 2) and **Update** (Algorithm 3) operations.

4.5.2 Segmentation with Set Permutation Regularization Algorithm

The detailed algorithm for segmenting action sequences into token sequences using set permutation regularization (SPR) is shown in Algorithm 4. In practice, we often run Algorithm 4 multiple times to augment the training corpus or ensemble recommendation outputs, as described

Algorithm 2. ActionPiece Vocabulary Construction – Count (Figure 4.2)

Input: Action sequence corpus $\mathcal{S}'$, current vocabulary $\mathcal{V}$

Output: Accumulated weighted token co-occurrences $\text{count}(\cdot, \cdot)$

```
1: for  $i \leftarrow 0$  to  $|\mathcal{V}|$ ,  $j \leftarrow 0$  to  $|\mathcal{V}|$  do
2:    $\text{count}(c_i, c_j) \leftarrow 0$ 
3: end for
4: for all sequence  $S' \in \mathcal{S}'$  do
5:    $t \leftarrow \text{length}(S')$  # Number of action nodes in sequence
6:   for  $k \leftarrow 0$  to  $t - 1$  do
7:      $\mathcal{A}_k \leftarrow S'[k]$  # Current action node
8:     # Process all unordered token pairs within  $\mathcal{A}_k$ 
9:     for all  $c_i, c_j \in \mathcal{A}_k, i \neq j$  do
10:       $\text{count}(c_i, c_j) \leftarrow \text{count}(c_i, c_j) + 2/|\mathcal{A}_k|$  # Weight of tokens within a single set (Equation (4.1))
11:       $\text{count}(c_j, c_i) \leftarrow \text{count}(c_j, c_i) + 2/|\mathcal{A}_k|$  # Symmetric update
12:    end for
13:    # Process all ordered token pairs between  $\mathcal{A}_k$  and  $\mathcal{A}_{k+1}$ 
14:    if  $k < t - 1$  then
15:       $\mathcal{A}_{k+1} \leftarrow S'[k+1]$ 
16:      for all  $c_i \in \mathcal{A}_k, c_j \in \mathcal{A}_{k+1}$  do
17:         $\text{count}(c_i, c_j) \leftarrow \text{count}(c_i, c_j) + 1/(|\mathcal{A}_k| \times |\mathcal{A}_{k+1}|)$  # Weight of tokens from two adjacent sets (Equation (4.2))
18:      end for
19:    end if
20:  end for
21: end for
return  $\text{count}(\cdot, \cdot)$ 
```

in Section 4.2.3.

4.6 Efficient Vocabulary Construction Implementation

To efficiently construct the ActionPiece vocabulary, we propose using data structures such as heaps with a lazy update trick, linked lists, and inverted indices to speed up each iteration of the construction process. The key idea is to avoid recalculating token co-occurrences in every iteration and instead update the data structures. The pseudocode is shown in Figure 4.7.

Algorithm 3. ActionPiece Vocabulary Construction – Update (Figure 4.3)

Input: Action sequence corpus $\mathcal{S}'$ before updating, current merge rule $\{(c_u, c_v) \rightarrow c_{\text{new}}\}$

Output: Updated action sequence corpus $\mathcal{S}'$

```
1: for all sequence  $S' \in \mathcal{S}'$  do
2:    $t \leftarrow \text{length}(S')$ 
3:   for  $k \leftarrow 0$  to  $t - 1$  do
4:      $\mathcal{A}_k \leftarrow S'[k]$ 
5:     # Merge tokens in one action node
6:     if  $c_u \in \mathcal{A}_k$  and  $c_v \in \mathcal{A}_k$  then
7:       Replace  $c_u$  and  $c_v$  in  $\mathcal{A}_k$  with  $c_{\text{new}}$ 
8:     end if
9:     # Merge tokens from two adjacent nodes
10:    if  $k < t - 1$  then
11:       $\mathcal{A}_{k+1} \leftarrow S'[k+1]$ 
12:      if  $c_u \in \mathcal{A}_k$  and  $c_v \in \mathcal{A}_{k+1}$  then
13:        if  $\mathcal{A}_k, \mathcal{A}_{k+1}$  are both action nodes then
14:          Create intermediate node  $M$  between  $\mathcal{A}_k$  and  $\mathcal{A}_{k+1}$ 
15:           $M \leftarrow \{c_{\text{new}}\}$  # Linked list:  $\mathcal{A}_k \rightarrow M \rightarrow \mathcal{A}_{k+1}$ 
16:           $\mathcal{A}_k \leftarrow \mathcal{A}_k \setminus c_u$ 
17:           $\mathcal{A}_{k+1} \leftarrow \mathcal{A}_{k+1} \setminus c_v$ 
18:        else if  $\mathcal{A}_k$  is intermediate node then
19:           $\mathcal{A}_k \leftarrow \{c_{\text{new}}\}$ 
20:           $\mathcal{A}_{k+1} \leftarrow \mathcal{A}_{k+1} \setminus c_v$ 
21:        else if  $\mathcal{A}_{k+1}$  is intermediate node then
22:           $\mathcal{A}_k \leftarrow \mathcal{A}_k \setminus c_u$ 
23:           $\mathcal{A}_{k+1} \leftarrow \{c_{\text{new}}\}$ 
24:        end if
25:      end if
26:    end if
27:  end for
28: end for
return  $\mathcal{S}'$ 
```

4.6.1 Data Structures

The data structures used in the proposed algorithm are carefully designed to optimize the efficiency of vocabulary construction. Here is a detailed discussion of their roles and implementations:

- **Linked list:** Each action sequence in the training corpus is stored as a linked list. This allows efficient local updates during token merging. When a token pair (c_u, c_v) is replaced

Algorithm 4. Segmentation via Set Permutation Regularization (SPR) (Section 4.2.2)

Input: Action sequence S , merge rules $\mathcal{R}$

Output: Segmented token sequences C

```
1:  $C \leftarrow []$  # Initialize permuted initial token sequence
2: for all token set  $\mathcal{A}_i \in S$  do
3:   Generate random permutation of  $\mathcal{A}_i$  as  $[c_1, c_2, \dots, c_{|\mathcal{A}_i|}]$ 
4:   Extend  $C$  with  $[c_1, c_2, \dots, c_{|\mathcal{A}_i|}]$  # Concatenate permutations
5: end for
6:
7: # Apply BPE [186] segmentation on permuted sequence
8: repeat
9:    $\mathcal{R}' \leftarrow \emptyset$  # Candidate merge rules
10:  for  $i \leftarrow 0$  to  $|C| - 1$  do
11:    if  $\{(c_i, c_{i+1}) \rightarrow c'\} \in \mathcal{R}$  then
12:       $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{(c_i, c_{i+1}) \rightarrow c'\}$ 
13:    end if
14:  end for
15:  Select  $\{(c_k, c_{k+1}) \rightarrow c'\} \in \mathcal{R}'$  with the smallest index among all merge rules  $\mathcal{R}$ 
16:   $C \leftarrow [c_1, \dots, c_{k-1}, c', c_{k+2}, \dots]$  # Replace  $(c_k, c_{k+1})$  with a new token  $c'$ 
17: until  $\mathcal{R}'$  is  $\emptyset$ 
return  $C$ 
```

by a new token c_{new} , only the affected nodes and their neighbors in the linked list need to be modified (as shown in Algorithm 3 and Figure 4.3).

- **Heap with lazy update trick:** A max-heap prioritizes token pairs by their co-occurrences. Instead of recalculating the heap entirely in each iteration, a “lazy update” strategy is employed: outdated entries (with mismatched co-occurrence counts) are retained but skipped during extraction. In the pseudocode, the loop checks if the top element is outdated via `is_outdated`. Invalid entries are discarded, and only valid ones are processed. Updated co-occurrences are pushed as new entries (with negative counts for max-heap emulation).
- **Inverted indices:** The `pair2head` dictionary maps token pairs to the sequences containing them. When a pair (c_u, c_v) is merged, the algorithm directly retrieves affected sequence IDs via `pair2head[(c_u, c_v)]`, avoiding a full corpus scan. After merging, the inverted

indices are incrementally updated: new token pairs (e.g., (c_{prev}, c_{new}) and (c_{new}, c_{next})) are added to `pair2head`, while obsolete pairs are removed. This enables targeted updates and ensures subsequent iterations efficiently access relevant sequences.

These structures collectively reduce time complexity by focusing computation on dynamically changing parts of the corpus and avoiding redundant global operations. The linked list enables localized edits, the heap minimizes priority recalculation, and the inverted indices eliminate brute-force searches, making the algorithm scalable to large corpora.

4.6.2 Time Complexity

The time complexity of the efficient vocabulary construction algorithm can be analyzed through two main components: **initialization** and **iterative merging**.

- **Initialization phase** involves building the initial max-heap to track co-occurrence frequencies. Given N input sequences (each with an average length of L), we count co-occurrences for all $O(m^2)$ token pairs within each set of size m . This requires $O(NLm^2)$ time.
- **Iterative merging phase** dynamically processes the involved sequences. The total number of such sequences across all iterations is approximately

$$O\left(\frac{N}{|\mathcal{V}_0|}\right) + O\left(\frac{N}{|\mathcal{V}_0|+1}\right) + \dots + O\left(\frac{N}{Q}\right) \simeq O(\log QN).$$

For each sequence, updating the linked list requires $O(Lm)$ time, counting co-occurrences takes $O(Lm^2)$ time, and inserting co-occurrences into the max-heap requires at most $O(Lm^2 \log H)$ time. Here, H represents the heap size, which is at most $O(NLm)$. Thus, the overall time complexity for iterative merging is

$$O(\log QN(Lm + Lm^2 + Lm^2 \log H)) = O(\log Q \log H \cdot NLm^2).$$

Therefore, the overall time complexity of our proposed vocabulary construction algorithm is $O(\log Q \log H \cdot NLm^2)$, where the iterative merging phase dominates. This complexity is significantly better than the naive vocabulary construction complexity of $O(QNLm^2)$.

4.7 Additional Discussions

4.7.1 Benefits of Set Permutation Regularization

SPR benefits the model from multiple perspectives:

- *Token utilization perspective.* SPR effectively prevents the features of a single action from being consistently merged into the most compressed (high-level) tokens. Instead, it allows the action to be tokenized into both high-level and low-level tokens, depending on the permutation and token merge rules. This increases the number of tokens actively involved during both training and inference. As shown in Figure 4.5 and discussed in Section 4.3.4, SPR significantly improves token utilization - from 56.89% to 95.33% by the 5th epoch - indicating that a greater proportion of tokens are trained after applying SPR.
- *Data augmentation perspective.* From the perspective of data augmentation, SPR enriches the token sequences available for model training. Without SPR, each action sequence can only be tokenized into a single, fixed token sequence. In contrast, SPR allows each action sequence to be tokenized in multiple ways (as shown in Figure 4.1). While these augmented sequences preserve the same semantic information, they expose the model to richer token patterns. Training on these diverse token sequences helps the model generalize better, as evidenced by the performance of variant (3.1) in Table 4.5.
- *Ensemble perspective.* SPR also enables inference-time augmentation. A given input action sequence can be augmented into multiple token sequences during inference. Each sequence may yield a different ranking of the next possible items. By ensembling these

recommendation results, overall performance can be enhanced, as demonstrated by variant (3.2) in Table 4.5 and further illustrated in Figure 4.6.

4.7.2 Connections Between ActionPiece and Feature Crossing

A core component of ranking models is automatically capturing feature-crossing patterns that are helpful for recommendation performance [187, 216, 217, 9], which is conceptually related to ActionPiece. The key difference lies in the level at which interactions are modeled. Prior works typically perform automatic feature crossing at the model level, learning interactions implicitly through the network architecture. In contrast, ActionPiece merges features at the data level.

Although we do not directly compare against these specific feature-crossing models, we include several baselines that follow a similar design philosophy. For example, HSTU in Table 4.4 and variant (2.1) in Table 4.5 use the same underlying item features as ActionPiece but input them in a flattened form, without merging. In these setups, feature interactions are expected to be learned by the autoregressive model via self-attention and feed-forward layers, *i.e.*, at the model level.

Our results demonstrate that ActionPiece, which performs data-level feature merging, consistently outperforms these methods in both recommendation quality (Tables 4.4 and 4.5) and efficiency (Figure 4.4). This is particularly evident in normalized sequence length (NSL): both HSTU and variant (2.1) yield NSL values of 1, reflecting longer sequences compared to the more compact sequences tokenized by ActionPiece.

4.8 Datasets

Categories. Among all the datasets, “Sports” and “Beauty” are two widely used benchmarks for evaluating generative recommendation models [174, 104, 96]. We conduct experiments on these benchmarks to ensure fair comparisons with existing results. Additionally, we introduce “CDs”, which contains about $4\times$ more interactions than “Sports”, making it a larger dataset for

evaluating the scalability of GR models. For “CDs”, since there are no publicly available results from generative recommendation methods to date, we closely followed the experimental settings used in public benchmarks like “Sports” and “Beauty” to ensure fair comparisons.

Sequence truncation length. Following Rajput et al. [174], we filter out users with fewer than 5 reviews and truncate action sequences to a maximum length of 20 for “Generative” methods, including ActionPiece. For “ID-based” and “Feature + ID” baselines, we set the maximum length to 50, as suggested in their original papers.

Item text features. Following Rajput et al. [174], Zheng et al. [290], Hou et al. [95], Sheng et al. [190], the first step for feature engineering is to combine multiple raw text features into a single sentence for each item. Then, we use a pretrained sentence embedding model to encode this sentence into a vector representation. In all our implementations, we concatenate *title*, *price*, *brand*, *feature*, *categories*, and *description*, and use sentence-t5-base [157] as the sentence embedding model.

- The encoded sentence embeddings of 768 dimension are directly used as textual item representations for UniSRec.
- We quantize the sentence embeddings using residual quantization (RQ) [174, 268, 293] into three codes, each with 256 candidates. To prevent conflicts, we add an extra identification code. These four codes together serve as the RQ-based semantic IDs for TIGER and SPM-SID.
- For other baselines that require item features, such as FDSA, S^3 -Rec, VQ-Rec, HSTU, and our method, we follow Hou et al. [88] and quantize the sentence embeddings using optimized product quantization (OPQ) [61]. Except for VQ-Rec, where the sentence embeddings are quantized into 32 codes as suggested in the original paper, we quantize the sentence embeddings into 4 codes for all other methods to ensure a fair comparison. The codebook size is 256 for each digit of code. For generative methods HSTU and ActionPiece, we also include an additional identification code to prevent conflicts. Note

that, unlike RQ-based semantic IDs, features produced by product/vector quantization do not require a specific order.

4.9 Baselines

We compare ActionPiece with the following representative baselines:

4.9.1 ID-Based Sequential Recommendation Methods

- **SASRec** [108] represents each item using its unique item ID. It encodes item ID sequences with a self-attentive Transformer decoder. The model is trained by optimizing a binary cross-entropy objective.
- **BERT4Rec** [197] also represents each item using its unique item ID. Unlike SASRec, it encodes sequences of item IDs with a bidirectional Transformer encoder. The model is trained using a masked prediction objective.

4.9.2 Feature-Enhanced Sequential Recommendation Methods

- **FDSA** [280] integrates item feature embeddings with vanilla attention layers to obtain feature representations. It then processes item ID sequences and feature sequences separately through self-attention blocks.
- **S³-Rec** [297] first employs self-supervised pre-training to capture the correlations between item features and item IDs. Then the checkpoints are loaded and fine-tuned for next-item prediction, using only item IDs.
- **VQ-Rec** [88] encodes text features into dense vectors using pre-trained language models. It then applies product quantization to convert these dense vectors into semantic IDs. The semantic ID embeddings are pooled together to represent each item. Since the experiments are not performed in a transfer learning setting, we omit the two-stage training strategy outlined in the original paper. Instead, we reuse the model architecture and train it from scratch using an in-batch contrastive loss with a batch size of 256.

4.9.3 Generative Recommendation Methods

Each generative recommendation baseline corresponds to an action tokenization method described in Table 4.1.

- **P5-CID** [96] is an extension of P5 [63], which formulates recommendation tasks in a text-to-text format. Building on P5, the authors explored several tokenization methods to index items for better recommendations. In this chapter, we use P5-CID as a representative hierarchical clustering-based action tokenization method. It organizes the eigenvectors of the Laplacian matrix of user-item interactions into a hierarchy and assigns cluster IDs at each level as item indices. When implementing this baseline method, we adopt the same model backbone as ActionPiece (encoder-decoder Transformers trained from scratch) and use the indices produced by P5-CID.
- **TIGER** [174] encodes text features similarly to VQ-Rec but quantizes them into semantic IDs using RQ-VAE. The model is then trained to autoregressively predict the next semantic ID and employs beam search for inference. We use a beam size of 50 in beam search to generate the top- K recommendations.
- **LMIndexer** [104] takes text as input and predicts semantic IDs. The text description of each item is first tokenized using a text tokenizer. The resulting text tokens are then concatenated to form input action sequences. The model is trained with self-supervised objectives to learn the semantic IDs of target items. The reported results in Table 4.4 are taken from the original paper. We do not report the results of LMIndexer on the large dataset “CDs” because it does not converge under similar computing budget as the other methods.
- **HSTU** [271] discretizes raw item features into tokens, treating them as input tokens for generative recommendation. The authors also propose a lightweight Transformer layer that improves both performance and efficiency. For action tokenization, we use the same

Table 4.7. Hyperparameter settings of ActionPiece for each dataset.

Hyperparameter	Sports	Beauty	CDs
learning_rate	0.005	0.001	0.001
warmup_steps	10,000	10,000	10,000
dropout_rate	0.1	0.1	0.1
weight_decay	0.15	0.15	0.07
vocabulary_size	40,000	40,000	40,000
n_inference_segments	5	5	5
beam_size	50	50	50
num_layers	4	4	4
d_model	128	128	256
d_ff	1,024	1,024	2,048
num_heads	6	6	6
d_kv	64	64	64
optimizer	adamw	adamw	adamw
lr_scheduler	cosine	cosine	cosine
train_batch_size	256	256	256
max_epochs	200	200	200
early_stop_patience	20	20	20

item features as our method and arrange them in a specific order to form the tokenized tokens of each item.

- **SPM-SID** [193] first tokenizes each item into semantic IDs. It then uses the SentencePiece model (SPM) [116] to merge important semantic ID patterns within each item into new tokens in the vocabulary. While the original paper introduces this method for ranking models, we adapt it for the generative recommendation task. Specifically, we concatenate the SPM tokens as inputs, feed them into the T5 model, and autoregressively generate SPM tokens as recommendations.

4.10 Implementation Details

Baselines. The results of BERT4Rec, SASRec, FDSA, S³-Rec, TIGER, and LMIndexer on the “Sports” and “Beauty” benchmarks are taken directly from existing papers [297, 174, 104]. For other results, we carefully implement the baselines and tune hyperparameters according to

the suggestions in their original papers. We implement BERT4Rec, SASRec, FDSA, and S³-Rec using the open-source recommendation library RecBole [285]. For other methods, we implement them ourselves with HuggingFace Transformers [235] and PyTorch [163]. We use FAISS [52] to quantize sentence representations.

ActionPiece. We use an encoder-decoder Transformer architecture similar to T5 [173]. We use four layers for both the encoder and decoder. The multi-head attention module has six heads, each with a dimension of 64. For the public benchmarks “Sports” and “Beauty”, we follow Rajput et al. [174] and set the token embedding dimension to 128 and the intermediate feed-forward layer dimension to 1024. This results in a total of 4.46M non-embedding parameters. For the larger “CDs” dataset, we use a token embedding dimension of 256 and an intermediate feed-forward layer dimension of 2048, leading to 13.11M non-embedding parameters. For model inference, we use beam search with a beam size of 50. Note that the baselines P5-CID, TIGER, and SPM-SID use the same model architecture, differing only in their action tokenization methods. For ActionPiece-specific hyperparameters, we set the number of segmentations produced using set permutation regularization during inference to $q = 5$. We tune the vocabulary size in $\{5k, 10k, 20k, 30k, 40k\}$.

Training. We train the GR models from scratch for up to 200 epochs, using early stopping if the model does not achieve a better NDCG@10 on the validation set for 20 consecutive epochs. The training batch size is set to 256. The learning rate is selected from $\{1 \times 10^{-3}, 3 \times 10^{-3}, 5 \times 10^{-3}\}$ with a warmup step of 10,000. We use a dropout rate of 0.1 and tune the weight decay from $\{0.07, 0.1, 0.15, 0.2\}$. For all methods implemented by us, we conduct five repeated experiments using random seeds $\{2024, 2025, 2026, 2027, 2028\}$. The model checkpoints with the best average NDCG@10 on the validation set are selected for evaluation on the test set, and we report these results. Each model is trained on a single 40G NVIDIA A100 GPU.

Inference. The inference process of ActionPiece follows the same procedure as TIGER. The decoder autoregressively generates token sequences representing the target items. During training, we use the original item features as labels, without any augmentation or token merging.

At inference time, for each augmented version of a test case, we apply beam search to generate top-ranked token sequences. The most probable sequences (*i.e.*, prefixes) are retained in the beam (with beam size specified in Table 4.7), and the model continues generating tokens one at a time until the target sequence length is reached.

4.11 Conclusion

In this chapter, we introduce ActionPiece, the first context-aware action sequence tokenizer for generative recommendation. By considering the surrounding context, the same action can be tokenized into different tokens in different sequences. We formulate generative recommendation as a task on sequences of feature sets and merge important feature patterns into tokens. During vocabulary construction, we propose assigning weights to token pairs based on their structures, such as those within a single set or across adjacent sets. To enable efficient vocabulary construction, we use double-ended linked lists to maintain the corpus and introduce intermediate nodes to store tokens that combine features across adjacent sets. Additionally, we propose set permutation regularization, which segments a single action sequence into multiple token sequences with the same semantics. These segments serve as natural augmentations for training and as ensemble instances for inference.

Chapter 4, in part, is a reprint of the material as it appears in “ActionPiece: Contextually Tokenizing Action Sequences for Generative Recommendation.” by Yupeng Hou, Jianmo Ni, Zhankui He, Noveen Sachdeva, Wang-Cheng Kang, Ed H. Chi, Julian McAuley, and Derek Zhiyuan Cheng, in International Conference on Machine Learning in 2025, referenced as [91]. The dissertation author was the primary investigator and author of this paper.

```

1 def vocab_construction_iteration(max_heap, vocab, rules, pair2head):
2     """Performs one iteration of efficient vocabulary construction.
3     Args:
4         max_heap (PriorityQueue): Max-heap storing (co_occurrence, (
5             c_u, c_v)) pairs.
6         vocab (List[Tuple]): Current vocabulary with merge rules.
7         rules (Dict): Merge rule {(c_u, c_v): c_new} mapping.
8         pair2head (Dict): Inverted indices that store mappings from
9             the token pair to all sequences that contain this token
10            pair.
11
12    """
13    # Get most frequent valid pair (c_u, c_v) from max-heap
14    # Efficient version of "Count" in Algorithm 1
15    # Avoid recalculating co-occurrences for each iteration, by
16    # maintaining them in a max-heap with the lazy update trick
17    while not max_heap.empty():
18        co_occurrence, (c_u, c_v) = max_heap.get()
19        if not is_outdated((c_u, c_v), co_occurrence): # Outdated
20            values are lazily removed.
21            break
22    # Create new token and update vocabulary
23    c_new = len(vocab)
24    vocab.append((c_u, c_v))
25    rules[(c_u, c_v)] = c_new
26    # Update sequences containing (c_u, c_v)
27    seq_ids = pair2head[(c_u, c_v)].copy() # IDs of affected
28    sequences
29    delta_counts = defaultdict(int)
30    for sid in seq_ids:
31        # Merge all (c_u, c_v) pairs in sequence
32        new_seq = merge_sequence(seqs[sid], c_u, c_v, c_new) #
33        Replace pairs
34        seqs[sid] = new_seq
35        # Calculate pair co-occurrence changes
36        new_freqs = count_pairs(new_seq) # Get token co-occurrences
37        of the updated sequence
38        delta = diff_counts(new_freqs, old_freqs[sid]) # Compute co
39        -occurrence differences
40        update_index(pair2head, delta, sid) # Update inverted index
41        old_counts[sid] = new_freqs
42        # Accumulate global co-occurrence changes
43        for p, cnt in delta.items():
44            delta_counts[p] += cnt
45    # Lazy update max-heap with new co-occurrences
46    for (c_u, c_v), delta in delta_counts.items():
47        if abs(delta) < eps: continue # eps: minimum update
48        threshold
49        all_pair_freqs[(c_u, c_v)] += delta # Global co-occurrences
50        max_heap.put( (-all_pair_freqs[(c_u, c_v)], (c_u, c_v)) )

```

Figure 4.7. Pseudocode for a single iteration of the efficient vocabulary construction algorithm, illustrating how a max-heap with lazy updates is used to track and merge frequent token pairs.

Chapter 5

Generating Long Semantic IDs in Parallel for Recommendation

This chapter asks how expressive semantic IDs can become when they are no longer constrained to very short sequences. It closes the tokenization-focused part of the dissertation by pairing longer semantic IDs with a modeling objective that keeps inference efficient.

Semantic ID-based recommendation models tokenize each item into a small number of discrete tokens that preserve specific semantics, leading to better performance, scalability, and memory efficiency. While recent models adopt a generative approach, they often suffer from inefficient inference due to the reliance on resource-intensive beam search and multiple forward passes through the neural sequence model. As a result, the length of semantic IDs is typically restricted (*e.g.*, to just 4 tokens), limiting their expressiveness. To address these challenges, we propose RPG, a lightweight framework for semantic ID-based recommendation. The key idea is to produce unordered, long semantic IDs, allowing the model to predict all tokens in parallel. We train the model to predict each token independently using a multi-token prediction loss, directly integrating semantics into the learning objective. During inference, we construct a graph connecting similar semantic IDs and guide decoding to avoid generating invalid IDs. Experiments show that scaling up semantic ID length to 64 enables RPG to outperform generative baselines by an average of 12.6% on the NDCG@10, while also improving inference

efficiency.¹

5.1 Introduction

A semantic ID represents an item as a sequence of discrete tokens from a shared vocabulary [88, 174, 271, 293]. Unlike traditional recommender systems that rely on unique item IDs [82, 108], semantic IDs eliminate the need for large embedding tables that scale with the total number of items. Early semantic ID-based recommendation models follow a retrieval paradigm, learning vector representations for items based on their semantic IDs and retrieving those closest to the user representation [88, 166]. Although effective, these models still require runtime memory proportional to the number of items. More recently, a generative paradigm has emerged [63, 174], where models are trained to autoregressively generate the next semantic ID in a token-by-token manner. These generative models make recommendations by decoding in the semantic ID space, resulting in runtime memory and latency independent of the number of items. Additionally, fine-grained semantics are explicitly integrated into the training objective by having the model generate sub-item tokens instead of entire items, resulting in strong performance.

Despite their promise, semantic ID-based generative models introduce significant **latency issue in model inference**. Compared to retrieval-based methods, where recommendations involve a single-step decoding process (*i.e.*, each token represents one item, and the next item is recommended by decoding the next token), generative models require multi-step decoding. For example, TIGER [174] autoregressively generates the next item’s semantic IDs in a token-by-token manner. In this multi-step decoding process, each step requires at least one batch of sequence model forwarding. To generate a ranking list of size K , TIGER employs beam search [67], maintaining the top b beams (semantic ID prefixes) and forwarding them through the neural encoder at each step. Because of autoregressive token generation and beam search, generative recommendation models require significantly more neural network forward passes for a single recommendation.

¹Our code is available at https://github.com/facebookresearch/RPG_KDD2025.

Moreover, the inference methods of existing generative recommendation models **limit the expressiveness of semantic IDs**. To ensure token-by-token autoregressive generation runs with reasonable latency, these models typically tokenize each item into a short sequence of semantic IDs. For example, TIGER uses 4 tokens per item, resulting in a decoding process of 4 steps per recommendation. However, representing an item with so few discrete tokens may be insufficient, especially as item features carry richer semantics. Retrieval-based methods like VQ-Rec [88] use longer semantic ID sequences (*e.g.*, 32 tokens per item), but applying such long sequences in generative models is impractical. Running beam search for 32 steps would be computationally prohibitive.

In this chapter, we aim to develop a semantic ID-based recommendation model that offers both efficient inference and expressive semantic IDs. Instead of generating semantic IDs token by token, we propose generating all tokens of the next item’s semantic ID in parallel. This approach allows us to decode the entire semantic ID in a single step. Since the number of decoding steps is independent of the semantic ID length, we can design expressive and long semantic IDs. However, applying this idea is non-trivial due to the following challenges: **(1) Sparse decoding space.** Generating all tokens in parallel removes sequential dependencies between tokens. However, it also means that predicting one token is independent of the predictions of other tokens. This lack of awareness makes it extremely difficult to map the predicted token combination to an existing semantic ID from the item pool. For example, if each semantic ID consists of 32 tokens, with each token selected from a codebook of 256, the total number of possible semantic IDs is $256^{32} \simeq 10^{77}$. Even if there are billions of items, the valid semantic IDs remain sparse compared to this vast decoding space ($10^9 \ll 10^{77}$). **(2) Efficient unordered decoding.** In recommendation tasks, we typically generate a top- K recommendation list for each input instance. Naively enumerating all candidate items would require runtime memory similar to retrieval-based methods, negating the benefits of generative approaches. Existing decoding methods, such as beam search, assume that tokens within a semantic ID follow a specific order. However, this assumption is unnecessary when generating tokens in parallel.

Therefore, a well-designed semantic ID decoding method is needed to handle unordered tokens efficiently.

To this end, we propose RPG, which stands for Recommendation with Parallel semantic ID Generation. To align with parallel decoding, we tokenize each item into an unordered long semantic ID (up to 64 tokens per item) using optimized product quantization [99, 62]. To enable the model to generate semantic IDs in parallel, we train it with a multi-token prediction (MTP) objective [65] instead of the conventional next-token prediction objective [174]. By applying MTP, sub-item semantics are explicitly incorporated into the training objective, similar to next-token prediction. Given MTP as the training objective, the prediction score for a candidate semantic ID is computed by summing the log probabilities of the ground-truth tokens across all digits. A key observation is that when two semantic IDs differ in only a few digits, their prediction scores should remain similar. Inspired by the widely used constrained beam search [85], which constructs a tree structure linking semantic prefixes that differ by only one digit, we propose a graph-constrained decoding method for unordered semantic IDs. This method connects similar semantic IDs (which have close prediction scores) by an edge in the graph structure. The decoding process follows an iterative graph propagation approach. Starting with a randomly initialized beam of semantic IDs, we propagate on the graph to identify semantic IDs that are close to the existing beam. We retain those with higher prediction scores and repeat the process for a predefined number of iterations to obtain the final top- K recommendations.

We show that the runtime memory and time complexities of RPG’s inference process are independent of the number of candidate items. Extensive experiments on three public benchmarks and a larger public dataset demonstrate that RPG benefits from expressive long semantic IDs and parallel generation, achieving the overall best performance among the compared baselines while being more efficient than existing semantic ID-based generative methods.

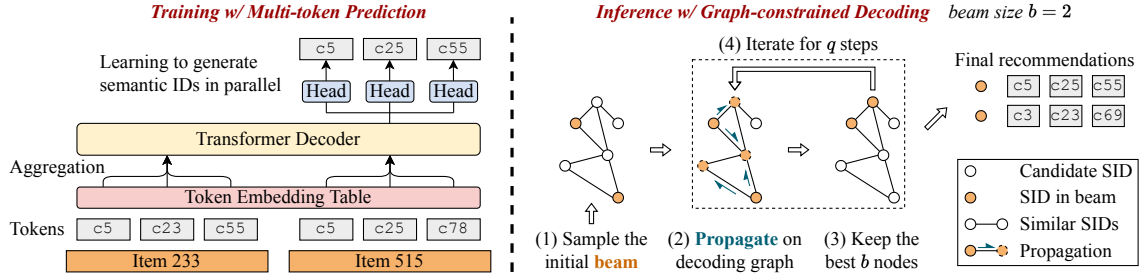


Figure 5.1. The overall framework of RPG.

5.2 Method

In this section, we introduce RPG, a recommendation model that generates each token of the next semantic ID in parallel. We formulate this task as a sequential recommendation problem. Given a user, RPG takes a sequence of previously interacted items, $s = \{i_1, i_2, \dots, i_{t-1}\}$, as input and predicts the next item i_t . First, we describe how RPG tokenizes and represents each item using a long semantic ID (Section 5.2.1). Next, we explain how RPG learns to generate semantic IDs in parallel by optimizing the multi-token prediction objective (Section 5.2.2). We then introduce graph-constrained decoding, which efficiently decodes unordered long semantic IDs during inference (Section 5.2.3). Finally, we discuss the connections between RPG and both retrieval-based and generative models. Figure 5.1 illustrates the overall framework of the proposed approach.

5.2.1 Long Semantic ID-based Item Representation

In recommender systems, each item is typically associated with multiple content features (*e.g.*, text descriptions and images). A common pipeline for item representation first encodes raw features into dense vector representations using a semantic encoder (*e.g.*, pretrained language models [157, 158]). Then, these vectors are tokenized into discrete tokens. Most existing semantic ID-based generative approaches use relatively short semantic IDs, typically with only four digits, due to efficiency concerns. In this chapter, we aim to design a model that generates each token in a semantic ID in parallel, enabling the use of longer semantic IDs

without significantly impacting efficiency. In this section, we describe how we convert raw features into semantic IDs and how we represent each item using long semantic IDs.

Long Semantic ID Construction

After obtaining semantic representations for each item [174, 290], we adopt product quantization instead of the commonly used residual quantization-based methods (*e.g.*, RQ-VAE [268, 174]). There are two key reasons for this choice: (1) predicting tokens in semantic IDs in parallel eliminates the need for sequential dependencies in RQ-based methods; (2) existing studies suggest that RQ-based semantic IDs suffer from unbalanced information distribution among tokens and poor scalability [279]. Concretely, following Hou et al. [88], we use optimized product quantization (OPQ) [99, 62] to produce a tuple of m tokens $(c_1, \dots, c_m)$ for each item i , where $c_1 \in \mathcal{C}^{(1)} = \{1, \dots, M\}$, $c_2 \in \mathcal{C}^{(2)} = \{M+1, \dots, 2M\}$, and so on. Here, M represents the codebook size for each digit of the semantic ID. Each token in the tuple is drawn from a different codebook, with each codebook corresponding to the semantic information of a subvector of the original semantic representation. In this chapter, we set the semantic ID length m to a maximum of 64, which is significantly longer than the typical length of 4 used in generative recommendation models.

Semantic ID Embedding Aggregation

After tokenizing each item into a semantic ID of length m , we consider how to represent each item in the input sequence. For each codebook, we assign a learnable token embedding table $\mathbf{E}_j \in \mathbb{R}^{M \times d}$. This allows us to obtain the corresponding token embeddings $(\mathbf{e}_{1,c_1}, \dots, \mathbf{e}_{m,c_m})$ for item i . Given that semantic IDs can be as long as 64 in this chapter, directly concatenating them may make the input token sequence too long, leading to efficiency and context length challenges [174]. To address this, we adopt an item-level input representation for sequence modeling. Specifically, we aggregate the token embeddings into a single item representation, $\mathbf{v}_i \in \mathbb{R}^d$, defined as $\mathbf{v}_i = \text{Aggr}(\mathbf{e}_{1,c_1}, \dots, \mathbf{e}_{m,c_m})$, where $\text{Aggr}(\cdot)$ is an aggregation function. In

practice, we can use mean or max pooling to aggregate the token embeddings efficiently [88].

5.2.2 Learning to Generate Semantic IDs in Parallel

Given the input item representation sequence $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{t-1}\}$ and the target semantic ID $(c_{t,1}, c_{t,2}, \dots, c_{t,m})$, we first introduce the learning objective that allows the model to generate tokens in parallel (Section 5.2.2). Then, we present an efficient method for logit calculation in models optimized with this objective.

Multi-token Prediction Objective

The key idea behind an efficient semantic ID-based recommendation model is to predict all tokens in the next semantic IDs in parallel. To achieve this, we draw inspiration from the multi-token prediction (MTP) objective [65]. First, we encode the item representation sequence using Transformer decoders, obtaining a sequence representation $\mathbf{s} \in \mathbb{R}^d$. We then formulate the problem as training the model to fit the probability $\mathbb{P}(c_{t,1}, \dots, c_{t,m} | \mathbf{s})$, which represents the likelihood that a given token combination appears in the next semantic IDs among all possible combinations. Since each token produced by OPQ encodes a separate subvector of the target item’s semantic representation, we assume that tokens in the tuple are conditionally independent given the input sequence $\mathbf{s}$. This allows us to factorize the probability as $\prod_{j=1}^m \mathbb{P}^{(j)}(c_{t,j} | \mathbf{s})$, where $\mathbb{P}^{(j)}(c_{t,j} | \mathbf{s})$ is the probability of the j -th token being in the next semantic ID. Formally, the MTP loss is defined as:

$$\mathcal{L} = - \sum_{j=1}^m \log \mathbb{P}^{(j)}(c_{t,j} | \mathbf{s}) = - \sum_{j=1}^m \log \frac{\exp(\mathbf{e}_{c_{t,j}}^\top \cdot \mathbf{g}_j(\mathbf{s}) / \tau)}{\sum_{c \in \mathcal{C}^{(j)}} \exp(\mathbf{e}_c^\top \cdot \mathbf{g}_j(\mathbf{s}) / \tau)}, \quad (5.1)$$

where τ is the temperature hyperparameter, $\mathbf{g}_j(\cdot)$ is a projection head (specifically, an MLP module [27, 11]) that maps sequence representations to the space of tokens in codebook $\mathcal{C}^{(j)}$.

Note that the MTP loss functions differently from the widely used cross-entropy loss defined in the item pool [108, 88]. MTP loss optimizes the probability of the target semantic

ID in the token space, *i.e.*, against all possible token combinations. In contrast, traditional item-based cross-entropy loss optimizes the probability of the ground-truth item ID in the item space, *i.e.*, against all other items. Additionally, similar to the next-token prediction loss used in generative recommendation models [174, 271], MTP directly integrates tokens in semantic IDs into the learning objective, enabling fine-grained semantic learning.

Efficient Logit Calculation

Given a model optimized with MTP loss, we propose an efficient logit calculation method. During inference, we compute $\mathcal{L}$ in Equation (5.1) as the logits. A straightforward approach is to calculate logits for each candidate item individually by enumerating token embeddings, computing logits for each token, and summing them. This results in a time complexity of $O(Nmd)$ for N candidate items. However, this method involves redundant dot product calculations between sequence representations and token embeddings. Popular tokens may undergo multiple dot product operations with the sequence representation. To address this inefficiency, we first cache the dot product between the sequence representation and all token embeddings:

$$\mathbf{p}^{(j)} = \text{softmax}(\mathbf{E}_j \cdot \mathbf{g}_j(\mathbf{s}) / \tau) \in \mathbb{R}^M, \quad (5.2)$$

where M is the codebook size, independent of the number of candidate items. Then, for a candidate semantic ID $(c_1, \dots, c_m)$, the logits can be computed as:

$$\sum_{j=1}^m \log \mathbf{p}_{c_j}^{(j)}, \quad (5.3)$$

achieving a reduced time complexity of $O(Mmd + Nm)$. This cached version is more efficient when $N > \frac{d}{d-1}M$. The choice of method depends on the serving scenario, and people can decide which inference approach works best for their needs.

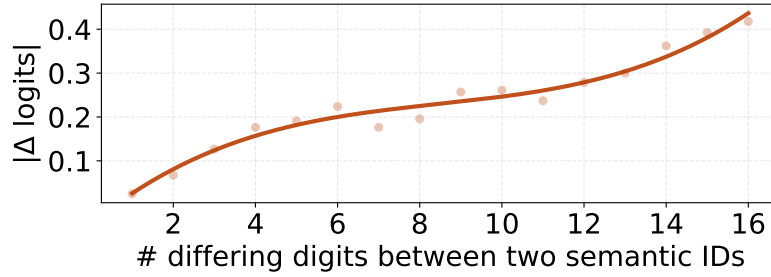


Figure 5.2. The relationship between the number of differing digits in two semantic IDs and the absolute difference in their predicted logits on the “Sports” dataset. As the number of differing digits increases, the model predicts increasingly different logits.

5.2.3 Next Semantic ID Decoding with Graph Constraints

For a model trained with the MTP objective, we aim to develop a decoding approach that is: (1) more efficient than naively enumerating all candidate items or applying beam search, and (2) more likely to generate valid semantic IDs rather than illegal token combinations.

Observation. The key insight lies in the unordered nature of PQ-based semantic IDs and logit calculation. Specifically, the logit of a candidate item is computed by summing the logits of each token in the corresponding semantic ID (Equation (5.3)). Consequently, when two semantic IDs differ by only a few digits, their logits should remain similar, as many token logits are identical. This observation is also empirically verified in Figure 5.2.

Overview. Inspired by constrained beam search [85], which uses a tree structure to connect similar semantic prefixes, we propose a graph-constrained semantic ID decoding method. This method links similar semantic IDs through a graph structure to guide the decoding process. The main idea is to start by sampling a random set of semantic IDs, propagate through the graph to explore new semantic IDs with slightly higher or lower logits, retain those with the best logits, and repeat this process for a fixed number of steps.

Building Graph to Connect Similar Semantic IDs

After training a model, we construct the *decoding graph*, which connects semantically similar semantic IDs. A node represents a valid semantic ID from the item pool. For any two nodes, $(c_{1,1}, \dots, c_{1,m})$ and $(c_{2,1}, \dots, c_{2,m})$, we compute their semantic similarity as $\sum_{j=1}^m \mathbf{e}_{j,c_{1,j}}^\top \cdot \mathbf{e}_{j,c_{2,j}}$. These similarity scores serve as edge weights in the graph. Since the initial graph is dense, it is inefficient and impractical. To address this, we sparsify it by retaining only the top k edges with the highest similarities for each node. Once built, the graph can be used throughout model inference until the model is retrained or the items are re-tokenized.

Decoding with Iterative Graph Propagation

Given an input sequence, we first cache the token logits following Equation (5.2).

(1) Sampling an initial beam. Decoding begins by sampling b semantic IDs from the item pool. We use the term “beam” by analogy with beam search, where b denotes the beam size.

(2) Propagating on the decoding graph. Similar to constrained beam search, the next step is to explore a larger set of semantic prefixes (or semantic IDs) and retain the best ones. Since neighboring nodes in the graph tend to have similar logits, we expand the beam by propagating through the graph, including all $b \times k$ neighbors of the current beam into a temporary set.

(3) Keeping the best b nodes. After expanding the beam, we compute their logits using the cached logits from Equation (5.3). The b nodes with the highest logits are then retained as the next beam.

(4) Iterating for q steps. We repeat steps (2) and (3) for q iterations to obtain the final results. Since each node has an edge to itself, this iterative beam update ensures that the average logits never decrease. In the worst case, if the beam remains unchanged for two consecutive steps, the logits will stay the same. The semantic IDs in the final beam are recommended as top K predictions.

Complexity Analysis

The decoding method has q steps, with each step involving at most $O(bk)$ nodes. This results in a time complexity of $O(Mmd + bqkm)$ by replacing $O(N)$ with $O(bqk)$ in Section 5.2.2, making it independent of the number of items. In practice, the graph propagation process can be further parallelized, *e.g.*, by using distributed computation techniques like MapReduce [39].

5.2.4 Discussion

In this section, we discuss the differences and connections between RPG and both retrieval-based and generative models.

Comparison of RPG and Retrieval-based Models

Compared to retrieval-based models like VQ-Rec [88], RPG offers two advantages that are typically exclusive to generative models: (i) its inference time complexity is independent of the total number of items, and (ii) it explicitly integrates sub-item level semantics into the learning objective. While the proposed graph-constrained decoding can be viewed as a form of approximate nearest neighbor (ANN) search, most existing ANN methods, such as those based on quantization codes [52], still require enumerating all items, resulting in time complexity proportional to the total number of items.

Comparison of RPG and Generative Models

Compared to generative models like TIGER [174], RPG reduces the number of sequence encoder forward passes per input instance from approximately $O(bm)$ to just $O(1)$, greatly improving inference efficiency (Section 5.3.3). Additionally, RPG benefits from a long semantic ID, which is empirically shown to be more expressive (Section 5.3.5).

Table 5.1. Statistics of the processed datasets. “Avg. t ” denotes the average number of interactions per input sequence.

Datasets	#Users	#Items	#Interactions	Avg. t
Sports	18,357	35,598	260,739	8.32
Beauty	22,363	12,101	176,139	8.87
Toys	19,412	11,924	148,185	8.63
CDs	75,258	64,443	1,022,334	14.58

Table 5.2. Performance comparison among baselines and RPG. The best performance score is denoted in **bold**. The second-best performance score is denoted in underline. RPG statistically outperforms the best baselines at a significance level of $p < 0.05$ by paired t-test.

Model	Sports and Outdoors			Beauty			Toys and Games			CDs and Vinyl		
	R@5	N@5	R@10	N@10	R@5	N@5	R@10	N@10	R@5	N@5	R@10	N@10
<i>Item ID-based</i>												
Caser	0.0116	0.0072	0.0194	0.0097	0.0205	0.0131	0.0347	0.0176	0.0166	0.0107	0.0270	0.0141
GRU4Rec	0.0129	0.0086	0.0204	0.0110	0.0164	0.0099	0.0283	0.0137	0.0097	0.0059	0.0176	0.0084
HGN	0.0189	0.0120	0.0313	0.0159	0.0325	0.0206	0.0512	0.0266	0.0321	0.0221	0.0497	0.0277
BERT4Rec	0.0115	0.0075	0.0191	0.0099	0.0203	0.0124	0.0347	0.0170	0.0116	0.0071	0.0203	0.0099
SASRec	0.0233	0.0154	0.0350	0.0192	0.0387	0.0249	0.0605	0.0318	0.0463	0.0306	0.0675	0.0374
FDSA	0.0182	0.0122	0.0288	0.0156	0.0267	0.0163	0.0407	0.0208	0.0228	0.0140	0.0381	0.0189
S ³ -Rec	0.0251	0.0161	0.0385	0.0204	0.0387	0.0244	0.0647	0.0327	0.0443	0.0294	0.0700	0.0376
<i>Semantic ID-based</i>												
RecJPQ	0.0141	0.0076	0.0220	0.0102	0.0311	0.0167	0.0482	0.0222	0.0331	0.0182	0.0484	0.0231
VQ-Rec	0.0208	0.0144	0.0300	0.0173	0.0457	0.0317	0.0664	0.0383	0.0497	0.0346	0.0737	0.0423
TIGER	<u>0.0264</u>	<u>0.0181</u>	0.0400	<u>0.0225</u>	0.0454	<u>0.0321</u>	0.0648	0.0384	<u>0.0521</u>	<u>0.0371</u>	0.0712	<u>0.0432</u>
HSTU	0.0258	0.0165	<u>0.0414</u>	0.0215	<u>0.0469</u>	0.0314	<u>0.0704</u>	<u>0.0389</u>	0.0433	0.0281	0.0669	0.0357
RPG	0.0314	0.0216	0.0463	0.0263	0.0550	0.0381	0.0809	0.0464	0.0592	0.0401	0.0869	0.0490
											0.0338	<u>0.0735</u>
												0.0415

5.3 Experiments

In this section, we empirically demonstrate the effectiveness and efficiency of the proposed method RPG.

5.3.1 Experimental Setup

Dataset. We conduct experiments using four categories from the Amazon Reviews dataset [147]: “Sports and Outdoors” (**Sports**), “Beauty” (**Beauty**), “Toys and Games” (**Toys**), and “CDs and Vinyl” (**CDs**). The first three categories are commonly used for benchmarking semantic ID-based generative recommendation methods [174, 96, 104]. Additionally, we include “CDs”, which has approximately four times more interactions than “Sports”, to evaluate performance on larger datasets.

Following previous studies [174, 88, 297], we treat users’ historical reviews as “interactions” and arrange them chronologically to form input interaction sequences, with earlier reviews appearing first. We adopt the widely used leave-last-out evaluation protocol [108, 287, 174], where the last item in each sequence is reserved for testing and the second-to-last item for validation. Table 5.1 presents the detailed statistics of the four datasets.

Baselines. We evaluate the performance of RPG by comparing it with the following item ID-based and semantic ID-based baselines.

- Caser [202] encodes the item ID sequence patterns using convolutional filters.
- GRU4Rec [82] is a recurrent neural network (RNN)-based approach proposed for session-based recommendation.
- HGN [146] applies the gating mechanism to RNN-based models.
- BERT4Rec [197] uses a bidirectional Transformer encoder to model item ID sequences and is trained with a Cloze-style objective.
- SASRec [108] uses a self-attentive Transformer decoder to model item ID sequences by optimizing a binary cross-entropy objective.

- FDSA [280] processes the item ID sequences and item feature sequences separately through self-attention blocks.

- S³-Rec [297] is first pretrained by optimizing self-supervised objectives to capture the correlations between item features and item IDs. Then the model is fine-tuned for next-item prediction objective, using only item IDs.

- VQRec [88] tokenizes each item using product quantization into semantic IDs. The semantic ID embeddings are pooled together to represent each item.

- TIGER [174] tokenizes each item using RQ-VAE. The model is then trained to autoregressively generate the next semantic ID token by token and employs beam search for inference.

- RecJPQ [166] applies joint product quantization to replace item embeddings with a concatenation of shared sub-embeddings.

- HSTU [271] discretizes raw item features into tokens, treating them as input tokens for generative recommendation. To ensure a fair comparison, we use the same 4-digit OPQ-tokenized semantic IDs as the item tokens.

Evaluation setting. We use Recall@ K and NDCD@ K as metrics to evaluate the methods, where $K \in \{5, 10\}$, following Rajput et al. [174]. Model checkpoints with the best performance on the validation set are used for evaluation on the test set.

Implementation details. We use most of the baseline results from Rajput et al. [174], while implementing other baselines mainly using RecBole [285]. We carefully tune the baselines’ hyperparameters following the original papers’ suggestions. For model training, we train all implemented models for a maximum of 150 steps with a batch size of 256. We tune the learning rate in $\{0.01, 0.003, 0.001\}$. We use an early stopping strategy to stop training if the validation results do not improve for 20 consecutive epochs.

We implement our method using HuggingFace transformers [235] and FAISS library [52]. To ensure a fair comparison with the baseline that has the largest number of parameters (TIGER, 13M), we use a 2-layer Transformer decoder with an embedding dimension of $d = 448$, a feed-forward layer dimension of 1024, and 4 attention heads, which achieve a similar num-

ber of parameters. We tune the temperature $\tau \in \{0.03, 0.05, 0.07\}$ and semantic ID length $m \in \{4, 8, 16, 32, 64\}$. For the semantic encoder model in this chapter, we mainly use OpenAI’s `text-embedding-3-large`² to show the expressive abilities of long semantic IDs, but also conduct additional experiments using `sentence-t5-base` [157] for fair comparison with benchmark results (Section 5.3.5). For model inference, we use a beam size $b = 10$, tune the number of edges per node $k \in \{50, 100, 500\}$ and iteration steps $q \in \{2, 3, 5\}$. Please refer to Section 5.5 for more details.

5.3.2 Overall Performance

We compare RPG with item ID-based and semantic ID-based baselines across three benchmarks and one larger public dataset. The results are shown in Table 5.2.

Across all datasets, methods leveraging semantic IDs generally outperform traditional item ID-based approaches. This aligns with the broader observation that incorporating item features and richer semantic information boosts recommendation quality. Within the semantic ID-based category, TIGER and HSTU consistently secure the second-best positions. These strong performances show the benefits of integrating semantic tokens into learning objectives over traditional methods.

Compared to all baselines, the proposed RPG achieves the best overall performance, ranking first in 11 out of 12 metrics. It outperforms the strongest baseline by an average of 12.6% on the NDCG@10 metric. Similar to generative models, RPG uses semantic tokens as predictive targets, learning sub-item semantics. By generating tokens in semantic IDs in parallel, RPG efficiently and effectively leverages long and expressive semantic IDs with up to 64 digits per item.

²<https://openai.com/index/new-embedding-models-and-api-updates/>

5.3.3 Inference Efficiency Analysis

One of the main advantages of RPG is the efficient inference method (Section 5.2.3) that has a complexity unrelated to the size of the item pool. We’ve conducted efficiency analyses from a complexity perspective in Section 5.2.3. In this section, we empirically demonstrate the efficiency of RPG. Based on the “Sports” dataset, we add varying numbers of dummy items to the item pool, achieving item pool sizes from 2×10^4 to 50×10^4 items. We then measure the required runtime memory (GB) and clock time (s) for model inference over one epoch on the test set. We illustrate the results in Figure 5.3. Note that all results are measured under a fixed set of hyperparameters, so the reported inference memory and time may not correspond to the best-performing configuration for each model. Here, runtime memory primarily refers to the GPU memory used to compute logits and generate the next tokens, including intermediate activations, rather than the total storage required to hold model parameters. A detailed breakdown of memory usage is provided in Table 5.8.

We observe that both retrieval-based baselines, SASRec and VQ-Rec, require increasing memory and inference time as the item pool expands. In contrast, the generative baseline TIGER and the proposed RPG maintain constant memory usage and inference time regardless of the total number of items. Moreover, RPG reduces runtime memory consumption by nearly $25\times$ and achieves an inference speedup of almost $15\times$ compared to TIGER. These results show that RPG inherits the scalability of generative models (*i.e.*, inference cost independent of the number of items), while being significantly more efficient than existing generative baselines in both memory and time.

5.3.4 Ablation Study

We conduct ablation analyses in Table 5.3 to study how each module contributes to the overall performance of RPG.

- (1) We first examine whether OPQ is a good choice for tokenizing items into long semantic

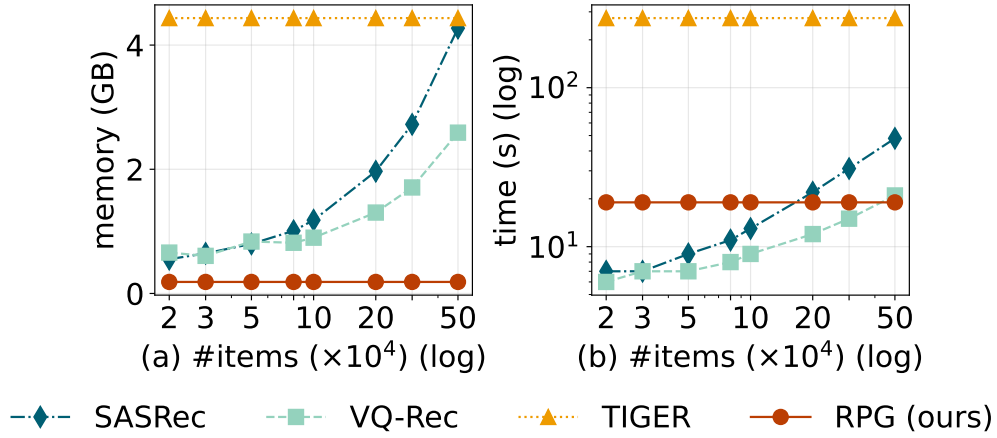


Figure 5.3. Comparison of runtime memory consumption and inference time w.r.t. the item pool size on the “Sports” dataset.

Table 5.3. Ablation analysis of RPG. The recommendation performance is measured using NDCG@10. The best performance is denoted in **bold** fonts.

Variants	Sports	Beauty	Toys	CDs
<i>Semantic ID Setting</i>				
(1.1) OPQ → Random	0.0179	0.0359	0.0288	0.0078
(1.2) OPQ → RQ	0.0242	0.0421	0.0458	0.0406
<i>Model Architecture</i>				
(2.1) no proj. head	0.0252	0.0423	0.0430	0.0361
(2.2) shared proj. head	0.0256	0.0424	0.0438	0.0368
<i>Model Inference</i>				
(3.1) beam search	0.0000	0.0000	0.0000	0.0000
(3.2) w/o graph constraints	0.0082	0.0214	0.0205	0.0183
RPG (ours)	0.0263	0.0464	0.0490	0.0415

IDs. We replace OPQ with (1.1) randomly assigned discrete tokens or (1.2) RQ-based semantic IDs, and in both cases, performance drops. The drop is especially significant when using random tokens, indicating that the multi-token prediction loss effectively learns the semantics embedded in the long semantic IDs and fails when there is no semantic meaning in the IDs.

(2) We then examine the designs of the projection head used in the MTP loss (Equation (5.1)). By default, we use a separate projection head for each digit of the semantic IDs. We implement

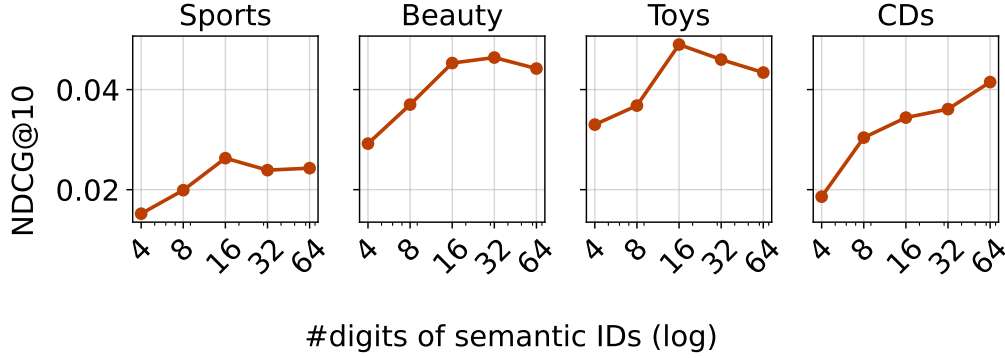


Figure 5.4. Scalability analysis of the performance (NDCG@10) of RPG w.r.t. semantic ID length.

two other variants: (2.1) removing the projection head and directly using the final hidden state of the sequence model to compute token logits, and (2.2) using a single shared projection head for all digits of the semantic IDs. Both variants lead to performance drops, highlighting the importance of mapping the sequence representation to different semantic spaces for decoding each digit of the tokens.

(3) We further experiment with two alternative inference methods: (3.1) beam search, which is commonly used in autoregressive generative models, and (3.2) inference without graph constraints, where the decoding graph is removed, and a similar number of semantic IDs are visited as in RPG, approximately 25% of the item pool (Table 5.6). Our results show that beam search struggles because OPQ-based semantic ID lacks sequential dependencies. As a result, it easily falls into local optima during long beam search steps. Additionally, removing the decoding graph significantly reduces performance, highlighting its importance in guiding the generation of long semantic IDs.

5.3.5 Further Analysis

Scalability of Semantic ID Lengths

The length of semantic IDs, m (*i.e.*, the number of digits in semantic IDs), is an important factor affecting recommendation efficiency and performance. We vary $m \in \{4, 8, 16, 32, 64\}$ to

examine how well the proposed method, RPG, scales with semantic ID length. From Figure 5.4, we observe that, in general, longer OPQ-based semantic IDs lead to better performance. However, for smaller datasets, they may converge earlier. The optimal length for the “Sports,” “Beauty,” and “Toys” datasets is 16, 32, and 16, respectively. For our largest dataset “CDs”, a semantic ID length of 64 achieves the best performance. These results demonstrate that our method scales effectively with increasing semantic ID length, particularly for larger datasets.

Expressive Ability Analysis

One assumption of using long semantic IDs is that they offer better expressive power than short ones. The original paper of TIGER indicates that they employ a Sentence-T5 semantic encoder (specifically, `sentence-t5-base` in our reproduction). In this experiment, we introduce a more powerful semantic encoder, OpenAI’s `text-embedding-3-large`, to examine whether the semantic ID design can be expressive and benefit from the improved semantic encoder. We primarily compare TIGER with our proposed method, RPG, using different semantic encoders and varying semantic ID lengths.

From Table 5.4, we observe that replacing `sentence-t5-base` with the strong OpenAI’s model `text-embedding-3-large` does not always improve TIGER’s performance when using 4-digit RQ-based semantic IDs. However, for RPG, a longer semantic ID performs better than a 4-digit one, even when using `sentence-t5-base`, and it already surpasses TIGER with the same encoder. Improving the encoder while maintaining long semantic IDs (up to 64 digits) further boosts performance. This demonstrates that the long semantic ID designed in RPG is more expressive and can benefit from a stronger semantic encoder.

Cold-Start Recommendation

Since RPG optimizes by directly predicting semantic IDs within the token space, it is natural to hypothesize that it has a similar cold-start recommendation ability to other semantic ID-based methods like VQ-Rec and TIGER. We conduct experiments to evaluate how well RPG

Table 5.4. Analysis of the expressive ability of short and long semantic IDs. The recommendation performance is measured using NDCG@10. “PLM” denotes the pretrained language model that encodes raw item features into representations. The best performance of each model is denoted in **bold**.

Model	PLM	#digits	Sports	Beauty	Toys	CDs
TIGER	sentence-t5-base	4	0.0225	0.0384	0.0432	0.0411
	text-emb-3-large	4	0.0243	0.0411	0.0390	0.0409
RPG	sentence-t5-base	4	0.0152	0.0292	0.0330	0.0186
	text-emb-3-large	4	0.0117	0.0235	0.0275	0.0175
	sentence-t5-base	16 ~ 64	0.0238	0.0429	0.0460	0.0380
	text-emb-3-large	16 ~ 64	0.0263	0.0464	0.0490	0.0415

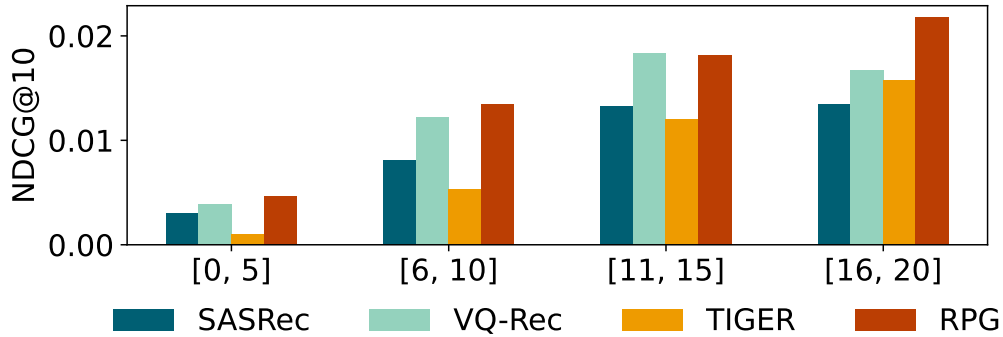


Figure 5.5. Cold-start analysis on the “Sports” dataset. Test items are divided into four groups based on the number of times they appear in the training set.

recommends long-tail items. We divide the test set items into four groups based on their occurrence in the training set: $[0, 5]$, $[6, 10]$, $[11, 15]$, and $[16, 20]$. As shown in Figure 5.5, RPG achieves the best overall cold-start recommendation performance, demonstrating the strong ability of the MTP objective to learn item semantics. The long semantic IDs also enhance the tokens’ semantic richness, further improving performance.

Hyperparameter Analysis

In this section, we tune the hyperparameters of the graph-constrained decoding method to analyze their impact on recommendation performance. The experiments are conducted on the “Sports” dataset with a default beam size of $b = 10$, number of edges per node $k = 50$, and

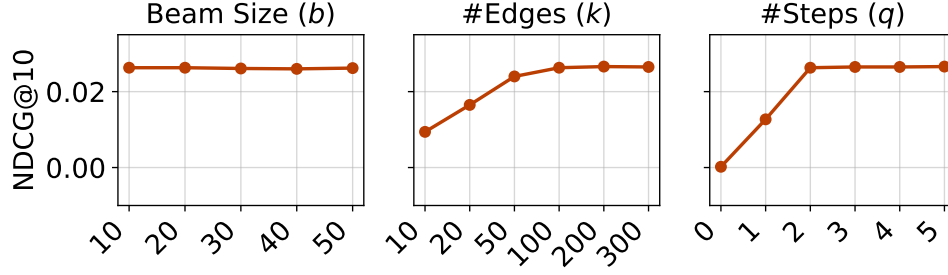


Figure 5.6. Model inference hyperparameter analysis. Each subplot shows the effect of varying one hyperparameter while keeping the others fixed.

iteration steps $q = 2$. The results are shown in Figure 5.6.

Beam size b . The beam size determines the number of semantic IDs retained in the beam at each step. In traditional beam search, a larger beam size brings the inference results closer to the global optimum, while a smaller one may lead to a local optimum. We vary the beam size as $b \in \{10, 20, 30, 40, 50\}$. From Figure 5.6, we observe that, using NDCG@10 as the recommendation performance metric, a beam size larger than 10 does not differ significantly from 10. These results suggest that it is generally safe to use a relatively small beam size (as long as it is larger than the K in the top- K metric) to improve efficiency.

Number of edges per node k . The parameter k controls the number of edges per node in the graph, determining how many candidate connections are considered at each decoding step. A higher k allows the model to explore more diverse paths, potentially improving recommendation performance. We evaluate k with values $\{10, 20, 50, 100, 200, 300\}$. From Figure 5.6, we observe a clear upward trend in NDCG@10 as k increases, with performance stabilizing beyond $k = 100$. This suggests that incorporating more edges enhances the model’s ability to identify relevant recommendations, but excessive values provide diminishing returns. Since increasing k also increases computation, we need to balance performance and efficiency by choosing an appropriate k .

Number of iteration steps q . The iteration steps q determine how many rounds of graph

propagation are performed. We evaluate q with values $\{0, 1, 2, 3, 4, 5\}$. As shown in Figure 5.6, increasing q significantly improves NDCG@10 from $q = 0$ to $q = 2$, indicating that additional iterations enhance recommendation performance. However, performance saturates after $q = 2$, suggesting that the beam may converge for $q > 2$. This indicates that a small number of decoding steps (*e.g.*, $q = 2$ or $q = 3$) is sufficient to reach optimal performance while maintaining efficiency.

5.4 Notations

We summarize the notations used throughout the chapter in Table 5.5.

5.5 Additional Implementation Details

All experiments were conducted on a single NVIDIA RTX 3090 GPU (24GB). For moderate-scale datasets (Sports, Beauty, Toys), each model training under a fixed hyperparameter setting takes less than 2 GPU hours. We tuned “learning rate”, “temperature”, and “#digit”, totaling 45 hyperparameter settings, resulting in under 90 GPU hours per dataset. For the large-scale dataset (CDs), each training takes under 8 hours, totaling about 180 GPU hours for all checkpoints reported. Regarding model inference, we evaluate all checkpoints on validation sets using the method described in Section 5.2.2. After identifying the best checkpoint, we further tune inference-time hyperparameters (k and q) over 9 combinations. This tuning process is lightweight and requires less than 1 GPU hour in total. We present the best hyperparameters for each dataset of RPG in Table 5.6.

5.6 Additional Discussion

What happens when equipping TIGER with long semantic IDs? To explore how existing semantic ID-based generative recommendation models perform with longer semantic IDs, we conduct experiments on the Sports dataset using TIGER [174] as the base model. We employ text-embedding-3-large as the PLM to generate semantic IDs and increase the

Table 5.5. Notations and explanations.

Notation	Explanation
i_1	The item IDs
N	The number of items
t	The length of interaction sequences
c_1	The individual tokens in semantic IDs
m	The length (number of tokens) of semantic IDs
$\mathcal{C}^{(1)}$	The vocabulary of tokens in semantic IDs
M	The codebook size for each digit of the semantic ID
$\mathbf{E}_j$	The embedding lookup table for semantic IDs' j th digit
d	The dimension for token and item embeddings
$\mathbf{e}_{1,c_1}$	The token embedding
$\mathbf{v}_i$	The item representation
$\mathbf{s}$	The user (sequence) representation
τ	The temperature hyperparameter
$g_j(\cdot)$	The projection head
$\mathbf{p}^{(j)}$	The log probability (<i>i.e.</i> , logit) of the next token
k	The number of edges for each node in the decoding graph
b	The beam size
q	The number of steps in the decoding process

Table 5.6. Best hyperparameters of RPG for each dataset.

Hyperparameter	Sports	Beauty	Toys	CDs
semantic ID length m	16	32	16	64
learning rate	0.003	0.01	0.003	0.001
temperature τ	0.03	0.03	0.03	0.03
beam size b	10	10	10	10
# edges per node k	100	100	50	500
# iteration steps q	2	3	5	3
<i>Number/ratio of visited items under the best hyperparameter.</i>				
# visited items	2×10^3	3×10^3	2.5×10^3	1.5×10^4
ratio of visited items	10.90%	24.79%	20.97%	23.28%

number of digits by applying more levels of residual quantization. As shown in Table 5.7, increasing the number of digits from 4 to 16 consistently degrades TIGER’s performance and significantly slows down inference. These results highlight the difficulty of scaling semantic ID length in existing models. In contrast, RPG not only handles long semantic IDs efficiently (Figure 5.3) but also benefits from increased ID length in terms of performance (Figure 5.4).

Table 5.7. Performance and inference time per epoch of TIGER w.r.t. #digit of semantic IDs. “OOM” denotes out-of-memory.

#digit	NDCG@10	Time (s)
4	0.0243	259
8	0.0219	788
16	0.0054	2,544
32	OOM	OOM

Table 5.8. Memory consumption breakdown. The term “fetched storage” refers to the amount of storage actually accessed to generate a ranking list for a user. The memory consumption of sequence encoders (*i.e.*, Transformer blocks) is omitted.

Method	NDCG@10	Storage	Fetched Storage
SASRec	0.0192	$O(Nd)$	$O(Nd)$
RecJPQ	0.0102	$O(Md + Nm)$	$O(Md + Nm)$
VQ-Rec	0.0173	$O(Mmd + Nm)$	$O(Mmd + Nm)$
RPG	0.0263	$O(Mmd + N(m + k))$	$O(Mmd + bqkm)$

Can multiple items share the same token set? Yes, this is possible. In RPG, each node in the decoding graph is initialized per item, rather than per unique token set. That said, since RPG uses long semantic IDs (*e.g.*, 64 tokens), the likelihood of two distinct items sharing the exact same token set is very low.

Why does RPG perform worse than TIGER when #digits = 4? As shown in Table 5.4, RPG underperforms TIGER when both use semantic IDs of length 4. However, comparing the two models under the same ID length is not particularly meaningful, given their fundamental differences in learning objectives, inference mechanisms, and semantic ID construction. When using their optimal semantic ID lengths, RPG consistently outperforms TIGER across all datasets while also achieving better inference efficiency (Figure 5.3).

Memory consumption breakdown. To better understand the memory footprint of RPG compared to existing methods, we provide a detailed breakdown in Table 5.8. RPG consumes more total storage, as it maintains a token embedding table, an item-token mapping table, and a decoding graph. However, during inference, RPG is the only model that avoids accessing the

entire storage but accesses a small subset of items, typically around 10-25%.

5.7 Conclusion

In this chapter, we propose RPG, an efficient and effective semantic ID-based recommendation approach. Instead of autoregressively generating tokens, our key idea is to predict each digit in parallel. This removes the sequential dependency between digits, allowing the use of unordered, long, and expressive semantic IDs. To enable parallel generation, we optimize the model with a multi-token prediction loss, explicitly integrating semantics into the learning objective. For inference, we leverage the observation that similar semantic IDs produce similar prediction logits. Based on this, we introduce a graph-constrained decoding method. By connecting semantic IDs with similar meanings, we iteratively refine the maintained beam of semantic IDs through graph propagation. Experimental results show that RPG is both effective and efficient in terms of memory consumption and inference speed.

Chapter 5, in part, is a reprint of the material as it appears in “Generating Long Semantic IDs in Parallel for Recommendation.” by Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, and Julian McAuley, in Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2 in 2025, referenced as [90]. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Inductive Generative Recommendation via Retrieval-based Speculation

The dissertation then moves from constructing semantic IDs to studying how generative recommendation models can use them. This chapter starts the modeling-focused part by examining whether semantic ID generation can support inductive recommendation for items that were not observed during training.

Generative recommendation (GR) is an emerging paradigm that tokenizes items into discrete tokens and learns to autoregressively generate the next tokens as predictions. While this token-generation paradigm is expected to surpass traditional transductive methods, potentially generating new items directly based on semantics, we empirically show that GR models predominantly generate items seen during training and struggle to recommend unseen items. In this chapter, we propose SpecGR, a plug-and-play framework that enables GR models to recommend new items in an inductive setting. SpecGR uses a *drafter* model with inductive capability to propose candidate items, which may include both existing items and new items. The GR model then acts as a *verifier*, accepting or rejecting candidates while retaining its strong ranking capabilities. We further introduce the guided re-drafting technique to make the proposed candidates more aligned with the outputs of generative recommendation models, improving verification efficiency. We consider two variants for drafting: (1) using an auxiliary drafter model for better flexibility, or (2) leveraging the GR model’s own encoder for parameter-efficient

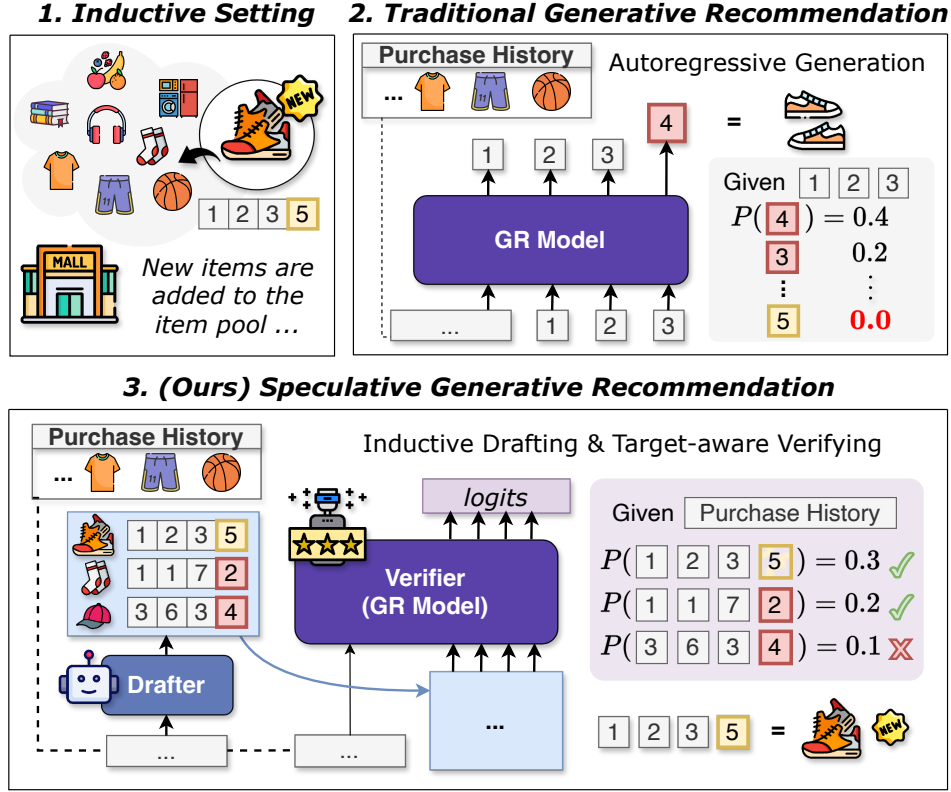


Figure 6.1. (1 & 2) GR models struggle to generate unseen items in an inductive setting. (3) SpecGR, a draft-then-verify framework, leverages GR models to verify candidates from an inductive drafter, enabling new-item recommendations.

self-drafting. Extensive experiments on three real-world datasets demonstrate that SpecGR exhibits both strong inductive recommendation ability and the best overall performance among the compared methods.¹

6.1 Introduction

Generative recommendation (GR) is an emerging paradigm for sequential recommendation tasks [174, 271, 290]. By tokenizing each item into a few discrete tokens (named semantic IDs), models are trained to autoregressively generate the next tokens, which are then parsed as predicted items. Compared to conventional sequential recommendation methods [108, 197], GR models scale up more easily and achieve better performance [174, 41], benefiting from the power of scaling laws [275, 271].

¹Our code is available at <https://github.com/Jamesding000/SpecGR>.

Unlike item ID-based transductive models such as SASRec [108], which cannot recommend new items due to the absence of their IDs in the trained model, GR models are expected to exhibit inductive capabilities by capturing semantic correlations and generating semantic tokens for previously unseen items. However, our empirical analysis shows that GR models systematically assign higher likelihoods to seen semantic ID sequences than to unseen ones. As shown quantitatively in Table 6.2, GR yields near-zero recommendation performance on unseen items. We attribute this to the models’ tendency to overfit to the semantic ID patterns present in the training data [257], making it unlikely for their outputs to match the semantic IDs of new items (Figure 6.1). In scenarios where up-to-date recommendations are critical (*e.g.*, news or short-video platforms), a flexible, on-the-fly inference framework is essential for the practical deployment of GR models.

In this chapter, we aim to develop *inductive* generative recommendation models that can recommend new items on-the-fly. Achieving this goal is non-trivial. In traditional recommendation systems, inductive recommendation [238] is typically achieved by incorporating side information and K-nearest neighbor (KNN) search. Although GR models can also tokenize items with side information into semantic IDs, as previously discussed, they struggle to generate unseen semantic ID patterns through autoregressive decoding [58, 174]. Recent efforts have explored blending GR outputs with items retrieved by non-GR methods [174, 257]. However, these approaches fail to fully exploit the modeling strengths of GR models, resulting in suboptimal performance.

To this end, we propose **SpecGR** (**Speculative Generative Recommendation**), an inductive generative recommendation framework that can be integrated with GR models in a plug-and-play manner. We extend the concept of the drafter-verifier framework in the original speculative decoding technique [119, 18, 81]. Rather than relying on a lightweight homologous model for inference acceleration, we explore the integration of models with different paradigms and capabilities. Specifically, we employ a KNN-based inductive model as the drafter to generate small batches of candidate items, while a GR model with stronger recommendation capabilities

serves as the verifier, responsible for accepting or rejecting these candidates. Instead of merely blending outputs from different models, SpecGR ensures that all final recommendations are ranked based on scores assigned by the GR verifier. In addition, we propose guided re-drafting to improve the quality of candidate items proposed by the drafter model, leveraging the semantic ID prefixes generated by the verifier GR model. Furthermore, to reduce the overhead of maintaining a separate drafter model, we introduce **SpecGR++** that enables the encoder of the GR model to serve as a drafter.

Extensive experiments are conducted on three public datasets. We split the training and evaluation sets chronologically using fixed timestamp cut-offs. This setup ensures that recommendation models are evaluated in a setting where new items appear over time. The experimental results demonstrate that SpecGR significantly improves the ability of GR models to recommend new items and achieves strong overall performance compared to existing methods.

6.2 Methodology

6.2.1 Problem Setup and Formulation

We follow the inductive sequential recommendation task [87, 266]. The input is a sequence of items $\{x_1, x_2, \dots, x_w\}$ ordered chronologically based on user interaction time. Each item $x \in \mathcal{S}$ has associated text features, such as title, description, and category. Here, w denotes the length of the item sequence. The task is to predict the next item of interest. Note that in the inductive setting, the target item may not appear in the training set, called new or unseen items in the following sections.

In this chapter, we focus on developing a general inductive recommendation framework for generative recommendation models. Typical GR models like TIGER [174] will tokenize each item x_i into a semantic ID pattern, $\text{ID}_i := [\langle c_1^i \rangle, \langle c_2^i \rangle, \dots, \langle c_l^i \rangle]$, where l denotes the number of digits of one item’s semantic ID pattern, and $\langle c \rangle$ denotes one digit of semantic ID. In this way, the input for GR models can be represented as follows by replacing the items in the original item

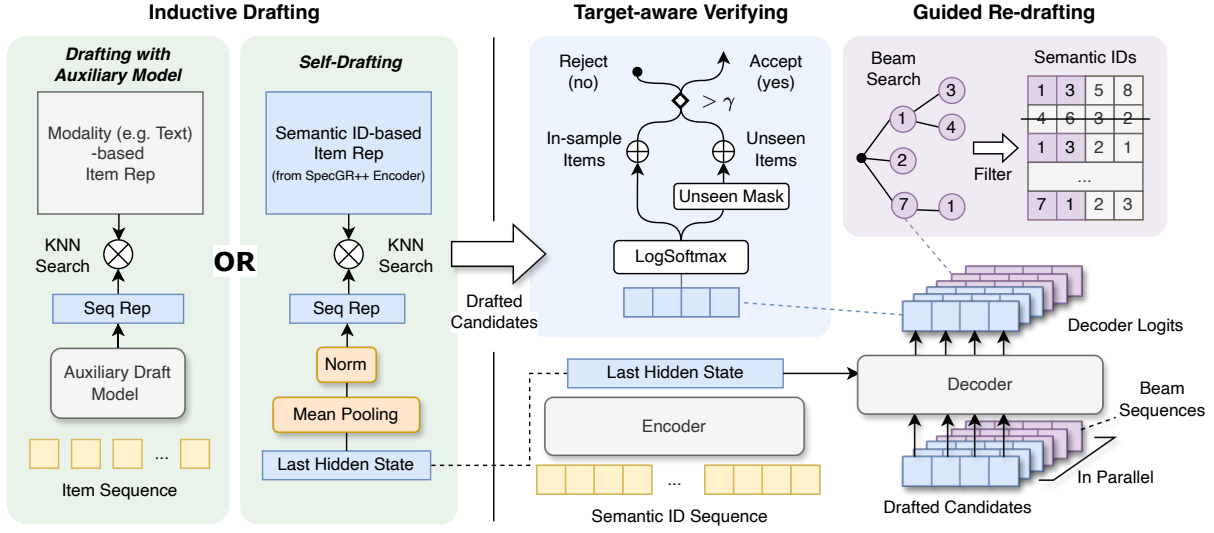


Figure 6.2. Illustration of the proposed SpecGR method, a draft-and-verify framework that iteratively performs drafting and verification until enough items are accepted. (a) **Inductive Drafting.** The inductive drafter first retrieves a set of candidates that contain new items. We present two drafting methods: using an auxiliary model or the GR’s encoder output (namely, self-drafting) for item retrieval. (b) **Target-aware Verifying.** The GR model accepts or rejects the candidates based on the likelihood of being the target. (c) **Guided Redrafting.** If not enough items are accepted, the GR filters the candidate space for the next drafting round based on the generated beam sequences.

sequence with the corresponding semantic IDs:

$$X = [\langle \text{bos} \rangle, \text{ID}_1, \text{ID}_2, \dots, \text{ID}_w, \langle \text{eos} \rangle], \quad (6.1)$$

where $\langle \text{bos} \rangle$ and $\langle \text{eos} \rangle$ are special tokens indicating the start and ending positions of a semantic ID sequence. Then the GR models are trained to generate K semantic ID patterns, which will be further parsed into recommended items with top- K probabilities. In the inductive setting, we assume that new items have been assigned semantic ID patterns. These new items can be parsed if the outputs of the GR models match the new semantic ID patterns.

6.2.2 Speculative Generative Recommendation

We begin by providing an overview of the proposed SpecGR framework, as illustrated in Figure 6.2. The framework consists of four components: (1) **Inductive Drafting**. Given the same input item sequence as the generative recommendation model, a drafter model with inductive recommendation capabilities proposes a set of candidate items as recommendation drafts. (2) **Target-aware Verifying**. The GR model, acting as a verifier, either accepts or rejects the candidates based on the probability that they could be targets of the input sequence. (3) **Guided Re-drafting**. If the number of accepted items does not meet the required recommendations K , the GR model guides the drafter to re-draft and propose the next set of candidate items. (4) **Adaptive Exiting**. Once K items are accepted, the framework exits and outputs the items, sorted by the scores given by the GR model. This process selects high-quality, unseen items through drafting and verifying, while maintaining the strong recommendation capabilities of generative recommendation models.

Inductive Drafting

Instead of expecting GR models to directly generate the semantic IDs of unseen items, we employ an inductive drafter model to first propose “recommendation drafts” that may include unseen items. Given the input item sequence X , the drafter model $D(\cdot)$ performs inductive drafting by recommending a set of δ candidates $\mathcal{Q} = D(X)$, where $|\mathcal{Q}| = \delta$. Note that in the original speculative decoding technique, the drafter model is considered an efficient approximation of the target model [119, 18], functioning as a homologous model to the verifier model. Here, we extend the above restriction in choosing drafter models. We do not require the drafter model to be a GR model but instead an inductive model to bring new capabilities to the following verifier model (in our case, the GR model). This approach allows high-quality unseen items to be introduced into the system, which are then verified by the more expressive generative model and finally included in the recommendations. Hence, the inductive drafter can be any inductive recommendation model, and we will discuss two effective instantiations in the subsequent Drafting Strategies

section.

Target-aware Verifying

While inductive drafters excel at recommending unseen items, they are not as effective as GR models in modeling input sequences and providing recommendations. Therefore, after obtaining candidates $\mathcal{Q}$ from the inductive drafting process, we use the generative model to verify them, rejecting items with low likelihood.

- **Target-aware likelihood for ranking.** Given a candidate item as a potential target, we use the GR model as a query-likelihood model (QLM) [306, 305, 160]. The QLM scores the query, *i.e.*, input sequence and the potential target, by measuring the likelihood of the model consecutively generating the tokens in the query. Previous studies have demonstrated that generative language models, such as T5 [173], exhibit robust zero-shot query-likelihood ranking performance in document retrieval tasks, without explicit fine-tuning for document ranking [306]. Accordingly, conditioning on the input sequence X , we adopt the conditional probability of generating the target semantic ID pattern as the verification score.

- **Likelihood score calculation.** However, naively applying the QLM for verification would result in low scores for unseen items. This happens because not all digits of semantic IDs are derived from item semantics. In addition to the tokens learned purely from item semantics, existing methods usually add an extra digit to avoid conflicts, known as the item identification token [174, 144]. For unseen items, the probability of generating this identification token lies outside the modeling distribution and therefore primarily consists of noise.

To provide a fair verification score for unseen items, we exclude the identification token and calculate only the probability of other digits. The target-aware verification score can be computed as:

$$V(x_t, X) = \begin{cases} \frac{1}{l} \sum_{i=1}^l \log P(c_i^t | c_{<i}^t, X) & \text{if } x_t \in \mathcal{I}, \\ \frac{1}{l-1} \sum_{i=1}^{l-1} \log P(c_i^t | c_{<i}^t, X) & \text{if } x_t \in \mathcal{I}^* \setminus \mathcal{I}, \end{cases} \quad (6.2)$$

where $V(\cdot)$ denotes the verifier model, which takes the input sequence X and the potential target item x_t as inputs, outputting the log-likelihood probability scores. l denotes the total number of digits in each semantic ID pattern, where the last digit is assumed to be the item identification token. $P(\cdot)$ denotes the backbone autoregressive model, which takes semantic ID sequences and outputs the likelihood scores. c_i^t refers to the i -th digit of the semantic ID for the target item x_t . The set $\mathcal{I}^* \setminus \mathcal{I}$ represents the unseen items.

To alleviate the bias caused by varying lengths of semantic IDs for unseen and existing items, we normalize the log-likelihood scores by the corresponding number of digits. After obtaining the likelihood score, we accept the items if $V(x_t, X) > \gamma$, where γ is a hyperparameter and can be tuned on the validation set.

Guided Re-drafting

If fewer than K items are accepted from the initial batch of drafted candidates, the drafter model $D(\cdot)$ must generate an additional batch of δ new candidates. Intuitively, since these candidates appear lower in the drafter’s ranking, their expected acceptance probability under the verifier $V(\cdot)$ is substantially reduced. To counteract this decline in acceptance rate, we introduce the *guided re-drafting* mechanism that steers subsequent candidate batches toward regions of the item space that better match the verifier’s scoring distribution.

Guided re-drafting operates by steering the drafter models using a set of semantic ID prefixes generated by the verifier models (GR models). Specifically, after verifying the j -th batch of recommendation drafts, the verifier model generates a set of beam sequences $\mathcal{B}_j$ using beam search, where each sequence is a j -digit semantic ID prefix. In the next draft-verify iteration, the drafter model is guided to propose only candidates $\mathcal{Q}_j$ whose prefixes match those in $\mathcal{B}_j$:

$$\mathcal{Q}_j = \{x_i | x_i \in D(X), (c_1^i, c_2^i, \dots, c_j^i) \in \mathcal{B}_j\}, \quad (6.3)$$

where $\mathcal{B}_j$ denotes the set of semantic ID prefixes, with a hyperparameter β as the set size.

Guided re-drafting happens along with the beam search decoding process of the GR model. It is important to note that the total number of draft-verify iterations will not exceed l , which corresponds to the maximum length of the semantic IDs, and is also equal to the maximum number of decoding steps.

Adaptive Exiting

SpecGR can adaptively terminate the draft-verify iterations based on the number of candidate items accepted by the verifier (GR) model. When the number of accepted items reaches K , the loop exits, avoiding the need to generate full-length sequences of l . This adaptive approach reduces inference time, as fewer generation steps are required. If, after the final iteration, there are still not enough accepted items, the beam sequences will be appended to the recommendation list until K is reached. As a result, even in the worst case, SpecGR does not incur additional time overhead compared to decoding with beam search. Finally, we rank the recommendation list by using the verification scores of the accepted items, along with the beam scores if items from beam sequences are included.

6.2.3 Drafting Strategies

In this section, we present two methods for drafting: by using an auxiliary draft model, and by reusing the encoder of generative recommendation models (namely SpecGR++).

Auxiliary Draft Model

The most straightforward way to draft is to introduce an auxiliary inductive recommendation model. An example is UniSRec [87], which uses modality-based item representations for KNN search. When new items are added, their representations can be directly incorporated into the item pool. The model can then retrieve new items if their modality-based representations are similar to the sequence representations.

Self-Drafting via GR Encoder

Despite the flexibility of using an auxiliary model as the drafter, issues such as communication latency and distribution shift may arise. Thus, we propose **SpecGR++**, which reuses the encoder module of the generative recommendation model to function as an inductive drafter model. The general idea is to encode both (1) the semantic IDs of a single item, and (2) the input semantic ID sequence of user history, using the same encoder module. Then we apply KNN search to retrieve semantic IDs of both existing and new items.

- **Semantic ID-based item and sequence representations.** To derive sequence representations, we use the same input format as our GR model (Equation (6.1)). To derive item representations, we format the semantic IDs of a single item x_i in the same way as the encoder’s input, *i.e.*, $[\langle \text{bos} \rangle, \text{ID}_i, \langle \text{eos} \rangle]$, where ID_i represents the semantic ID pattern of item x_i . To obtain the item and sequence representations, we take the last hidden state from the GR encoder and apply mean pooling.

- **Item–sequence contrastive pretraining.** Prior work shows that hidden states from generative models are not directly suitable as representations [157, 158]. To obtain strong inductive embeddings, we jointly train the GR encoder with a contrastive objective following [27, 87]. Given sequence embeddings $\mathbf{s}_j$ and next-item embeddings $\mathbf{i}_j$, the contrastive loss is:

$$\mathcal{L}_{\text{CL}} = -\frac{1}{B_{\text{emb}}} \sum_{j=1}^{B_{\text{emb}}} \log \frac{\exp(\mathbf{s}_j \cdot \mathbf{i}_j / \tau)}{\sum_{j'=1}^{B_{\text{emb}}} \exp(\mathbf{s}_j \cdot \mathbf{i}_{j'} / \tau)}. \quad (6.4)$$

We optimize $\mathcal{L}_{\text{CL}}$ jointly with the next-token generation loss:

$$\mathcal{L}_{\text{Gen}} = -\frac{1}{B_{\text{gen}}} \sum_{b=1}^{B_{\text{gen}}} \frac{1}{L_b} \sum_{t=1}^{L_b} \log P(c_{b,t} \mid c_{b,<t}, X_b). \quad (6.5)$$

To ensure sufficient in-batch negatives for the contrastive objective, we use a larger embedding batch (*i.e.*, $B_{\text{emb}} > B_{\text{gen}}$). The final multi-task loss is expressed as $\mathcal{L} = \lambda_1 \mathcal{L}_{\text{CL}} + \mathcal{L}_{\text{Gen}}$, where λ_1 is a hyperparameter balancing the two tasks.

Table 6.1. Performance comparison of different models. The best and the second-best performance is denoted in bold and underlined fonts, respectively. “R@K” is short for “Recall@K” and “N@K” is short for “NDCG@K”. “Improv.” denotes the improvement ratio of SpecGR compared to the best-performing baseline model.

Dataset	Metric	ID-based	Feature + ID		Modality-based			Generative			Ours		Improv.
		SASRec _{ID}	FDSA	S ³ -Rec	SASRec _T	UniSRec	Recformer	TIGER	TIGER _C	LIGER	SpecGR _{Aux}	SpecGR++	
Games	R@10	0.0186	0.0190	0.0195	0.0179	0.0225	0.0243	0.0222	0.0226	0.0139	0.0254	<u>0.0250</u>	+4.53%
	N@10	0.0093	0.0101	0.0094	0.0091	0.0115	0.0111	0.0114	0.0115	0.0068	0.0128	<u>0.0124</u>	+10.40%
	R@50	0.0477	0.0496	0.0473	0.0507	0.0621	<u>0.0740</u>	0.0584	0.0611	0.0635	0.0778	0.0717	+5.13%
	N@50	0.0162	0.0167	0.0154	0.0161	0.0200	0.0218	0.0193	0.0198	0.0172	0.0239	<u>0.0225</u>	+9.72%
Office	R@10	0.0093	0.0095	0.0100	0.0091	0.0119	0.0126	0.0132	0.0130	0.0059	0.0138	<u>0.0134</u>	+3.99%
	N@10	0.0047	0.0050	0.0052	0.0048	0.0062	0.0039	<u>0.0071</u>	0.0070	0.0029	0.0072	0.0070	+1.68%
	R@50	0.0217	0.0224	0.0234	0.0233	0.0322	<u>0.0340</u>	0.0308	0.0312	0.0268	0.0360	0.0332	+5.93%
	N@50	0.0074	0.0078	0.0080	0.0078	0.0105	0.0106	0.0109	0.0110	0.0072	0.0119	<u>0.0113</u>	+8.76%
Phones	R@10	0.0052	0.0067	0.0058	0.0072	0.0084	0.0074	0.0090	0.0087	0.0048	<u>0.0099</u>	0.0101	+11.90%
	N@10	0.0027	0.0035	0.0028	0.0037	0.0045	0.0036	0.0047	0.0046	0.0022	<u>0.0050</u>	0.0052	+10.64%
	R@50	0.0143	0.0184	0.0151	0.0188	0.0233	0.0236	0.0232	0.0233	0.0226	0.0285	<u>0.0275</u>	+20.64%
	N@50	0.0047	0.0060	0.0048	0.0062	0.0077	0.0070	0.0078	0.0078	0.0059	0.0090	0.0090	+14.80%

Table 6.2. Model performance breakdown on the “in-sample” and “unseen” subsets. The proportions of the test cases in each subset relative to the entire test data have been labeled. The best and second-best results are bolded and underlined.

Model	#Params. (M)	Games						Phones					
		Overall		In-Sample (39.7%)		Unseen (60.3%)		Overall		In-Sample (31.8%)		Unseen (68.2%)	
		R@50	N@50	R@50	N@50	R@50	N@50	R@50	N@50	R@50	N@50	R@50	N@50
UniSRec	2.90	0.0621	0.0200	0.1386	0.0461	0.0118	0.0029	0.0233	0.0077	0.0604	0.0211	0.0060	0.0014
Recformer	233.73	<u>0.0740</u>	0.0218	0.1082	0.0333	<u>0.0514</u>	<u>0.0142</u>	0.0236	0.0070	0.0340	0.0103	0.0188	0.0055
TIGER	13.26	0.0584	0.0193	<u>0.1472</u>	0.0486	-	-	0.0232	0.0078	<u>0.0730</u>	<u>0.0245</u>	-	-
TIGER _C	13.26	0.0611	0.0198	0.1447	<u>0.0482</u>	0.0061	0.0011	0.0233	0.0078	0.0691	0.0238	0.0019	0.0003
LIGER	13.26	0.0635	0.0172	0.0438	0.0160	0.0765	0.0179	0.0226	0.0059	0.0472	0.0107	<u>0.0111</u>	<u>0.0037</u>
SpecGR _{Aux}	16.16	0.0778	0.0239	0.1485	0.0457	0.0312	0.0096	0.0285	0.0090	0.0748	0.0237	0.0069	0.0021
SpecGR++	13.28	0.0717	<u>0.0225</u>	0.1323	0.0439	0.0318	0.0084	<u>0.0275</u>	0.0090	<u>0.0730</u>	0.0246	0.0063	0.0017

• **Learning-to-rank fine-tuning.** Following Li et al. [121], to further enhance the ranking ability of the semantic ID encoder, we continue to fine-tune the encoder using the cross-entropy loss $\mathcal{L}_{CE}$ on a larger batch of negative items (equivalent to $\mathcal{L}_{CL}$ in Equation (6.4) with $B_{emb} = |\mathcal{I}|$). To enable efficient large-batch training, the item representations are frozen at the beginning of the fine-tuning phase. The overall loss for fine-tuning can be written as $\mathcal{L}' = \lambda_2 \mathcal{L}_{CE} + \mathcal{L}_{Gen}$, where λ_2 is a hyperparameter.

6.3 Experiments

In this section, we present experimental results to answer the following Research Questions (**RQ**).

- **RQ1:** How does SpecGR perform compared with state-of-the-art baselines on overall and seen/unseen subsets?
- **RQ2:** Do all of SpecGR’s designs take effect?
- **RQ3:** How do key hyperparameters affect SpecGR’s performance and efficiency?
- **RQ4:** Is SpecGR an effective plug-and-play framework for different drafters and GR backbones?

6.3.1 Experimental Setup

Datasets

We use three categories, Video Games (**Games**), Office Products (**Office**), and Cell Phones and Accessories (**Phones**), from the Amazon Reviews 2023 dataset [95] as our experimental datasets. To assess the performance of SpecGR in real-world settings, we utilize preprocessed benchmarks² that exclude users and items with fewer than five interactions. The data is split into training, validation, and test sets based on predefined timestamp cut-offs. Notably, since the datasets are split by timestamps, the validation and test sets naturally include unseen items. This simulates a more realistic scenario in comparison to the widely-used leave-last-out splitting.

Compared Methods

We report results for two SpecGR variants: SpecGR_{Aux}, which uses UniSRec [87] as an auxiliary drafter, and SpecGR++, which uses its own encoder module for drafting. We

²huggingface.co/datasets/McAuley-Lab/Amazon-Reviews-2023/tree/main/benchmark/5core/timestamp_w_his

Table 6.3. Ablation study on SpecGR++ inference and training. The best and second-best results are bolded and underlined.

Variants	Games				Office				Phones			
	R@50	N@50	R@10	N@10	R@50	N@50	R@10	N@10	R@50	N@50	R@10	N@10
(1.1) <i>w/o</i> inductive drafting	0.0609	0.0202	0.0235	<u>0.0121</u>	0.0306	0.0109	<u>0.0132</u>	0.0070	0.0233	0.0080	0.0092	0.0049
(1.2) <i>w/o</i> likelihood score adjustment	<u>0.0712</u>	0.0221	0.0236	0.0119	0.0331	0.0103	0.0118	0.0057	0.0236	0.0081	0.0092	0.0049
(1.3) <i>w/o</i> guided re-drafting	0.0611	0.0202	0.0235	<u>0.0121</u>	0.0309	0.0110	<u>0.0132</u>	0.0070	0.0264	0.0086	0.0096	0.0050
(1.4) <i>w/o</i> item re-ranking	0.0703	0.0219	<u>0.0239</u>	0.0120	0.0334	0.0113	0.0131	0.0069	0.0264	0.0083	0.0093	0.0047
(1.5) <i>w/o</i> adaptive exiting	0.0694	0.0200	0.0203	0.0095	0.0313	0.0108	0.0126	0.0068	<u>0.0265</u>	0.0086	0.0095	0.0050
(2.1) TIGER for SpecGR++	0.0582	0.0192	0.0224	0.0114	0.0302	0.0105	0.0127	0.0067	0.0232	0.0078	0.0090	0.0047
(2.2) <i>w/o</i> contrastive pretraining	0.0581	0.0193	0.0221	0.0115	0.0313	0.0108	0.0126	0.0068	0.0234	0.0077	0.0093	0.0050
(2.3) <i>w/o</i> fine-tuning	0.0692	0.0225	0.0222	0.0111	0.0325	0.0110	0.0129	0.0068	0.0259	<u>0.0087</u>	<u>0.0098</u>	<u>0.0051</u>
SpecGR++	0.0717	0.0225	0.0250	0.0124	<u>0.0332</u>	0.0113	0.0134	0.0070	0.0275	0.0090	0.0101	0.0052

compare SpecGR against the following state-of-the-art methods: ID-based methods such as SASRec [108]; feature+ID-based methods such as FDSA [280] and S³-Rec [297]; modality-based methods such as UniSRec [87] and RecFormer [121]; and generative methods including TIGER, TIGER_C [174], and LIGER [257]. Notably, TIGER_C employs a heuristic strategy that mixes a fixed proportion of unseen items into TIGER’s recommendation list.

Evaluation Setting

We adopt Recall@ K and NDCG@ K as metrics to evaluate the compared methods, where $K \in \{10, 50\}$. In addition, based on whether the target items in the test set are existing items or new items (not shown in the training set), we split our test set into two subsets, named **In-Sample** and **Unseen**, respectively. The model checkpoints of all compared methods that have the best overall performance on the validation set will be evaluated on the test set.

Implementation Details

We use UniSRec [87] as the auxiliary drafter model for SpecGR_{Aux} and TIGER [174] as the GR backbone for both SpecGR variants. Input sequences are truncated to a maximum of 20 items, following Rajput et al. [174], and the same semantic ID tokenization process is applied. SpecGR++ is trained using a multi-task setup ($\lambda_1 = \lambda_2 = 6.0$) combining generation and contrastive objectives, followed by a learning-to-rank fine-tuning phase. We use $B_{\text{Emb}} = 2048$

and $B_{\text{Gen}} = 256$. Hyperparameters such as the draft threshold (γ), beam size (β), and draft size (δ) are tuned on validation splits.

6.3.2 Performance Analysis (RQ1)

Overall performance. We compare SpecGR with sequential and generative recommendation baselines across three public datasets; results are summarized in Table 6.1. ID-based and feature-based methods generally show poor performance, especially on sparse datasets (*e.g.*, Phones). Modality-based methods, such as UniSRec and Recformer, show improved performance by leveraging powerful text embeddings from Pretrained Language Models (PLMs). Generative recommendation models achieve the best results through autoregressive modeling of fine-grained semantic IDs. Among all models, SpecGR consistently achieves the best overall performance (*e.g.*, up to +14.8% in NDCG@50, and +20.64% in Recall@50). Notably, SpecGR++ attains both better parameter efficiency and comparable performance to SpecGR_{Aux}, highlighting the GR encoder’s effectiveness in learning robust semantic ID-based representations for inductive recommendation.

Subset Analysis. Next, we analyze the detailed performance breakdown on the in-sample and unseen subsets, as shown in Table 6.2. TIGER achieves strong in-sample performance but fails to generalize inductively on the unseen subset. SpecGR addresses this limitation by integrating inductive drafting with GR-based verification, greatly improving inductive generalization without compromising in-sample quality. Recformer is the best-performing modality-based method. However, its LLM backbone has significantly larger model size compared to other baselines. While LIGER attains higher performance on unseen items via dense retrieval-based candidate blending, its heuristic combination of candidates introduces irrelevant items, yielding a suboptimal trade-off and degraded overall recommendation quality. In contrast, SpecGR employs GR’s target-aware likelihood scores to filter inductive candidates, resulting in strong inductive ability and the best overall performance.

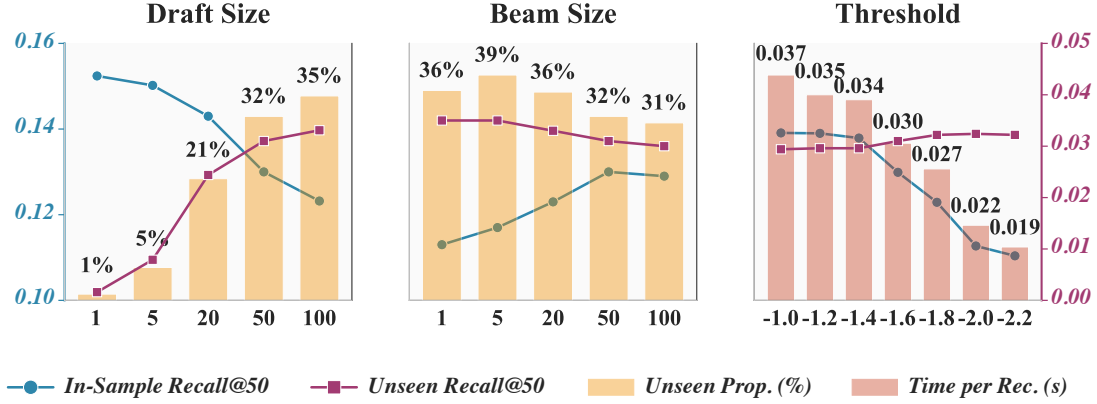


Figure 6.3. Impact of hyperparameters on SpecGR’s performance and efficiency. (Left, middle): Bars show the proportion of unseen items in recommendations. (Right): Bars represent inference latency in seconds. Lines depict the trade-off between in-sample and unseen Recall@50.

6.3.3 Ablation Study (RQ2)

SpecGR Inference Framework. First, we assess the contributions of inference components in SpecGR++, as summarized in Table 6.3. Inductive drafting (1.1) and likelihood score adjustment (1.2) substantially enhance inductive generalization by incorporating meaningful unseen candidates. Guided re-drafting (1.3) boosts recommendation quality when initial drafting provides insufficient accepted candidates. Removing verification-based re-ranking (1.4) and adaptive exiting (1.5) results in performance degradation, confirming the importance of these modules in maintaining a balance between in-sample accuracy and inductive capability.

SpecGR++ Training Paradigm. Next, we analyze training paradigm variants for SpecGR++. Directly using TIGER’s encoder states without dedicated representation learning (2.1) severely limits inductive recommendation. Contrastive pretraining (2.2) significantly improves representation quality, and subsequent fine-tuning (2.3) further refines performance, validating our two-stage training design.

Table 6.4. NDCG@50 on the unseen subset (U-N@50) and overall test set (O-N@50) for different GR backbones and drafter configurations on the **Video Games** dataset.

GR	Drafter	U-N@50	O-N@50	Improvement
TIGER	Baseline	—	0.0193	—
	GR Encoder (SpecGR++)	0.0084	0.0225	(+16.6%)
	Semantic-KNN	<u>0.0085</u>	<u>0.0231</u>	(+19.7%)
	UniSRec	0.0096	0.0239	(+23.8%)
DSI	Baseline	—	0.0198	—
	GR Encoder (SpecGR++)	0.0061	<u>0.0217</u>	(+9.6%)
	Semantic-KNN	0.0049	<u>0.0217</u>	(+9.6%)
	UniSRec	<u>0.0058</u>	0.0220	(+11.1%)

6.3.4 Hyperparameter Analysis (RQ3)

We analyze how model hyperparameters affect recommendation behaviors and performance. We conduct hyperparameter analyses of **SpecGR++** on the **Video Games** dataset, with results shown in Figure 6.3. The base hyperparameters are $\delta = 50$, $\gamma = -1.6$, and $\beta = 50$, with specific parameters adjusted in each plot while keeping others fixed.

- **Draft size** (δ) controls the proportion of unseen items in recommendations, as all unseen items originate from drafting. Increasing draft size enhances inductive performance but may degrade in-sample metrics due to the fixed number of accepted candidates. The optimal value is selected via hyperparameter tuning on the validation set.

- **Beam size** (β) controls the search space for guided re-drafting. As shown in Figure 6.3, increasing beam size improves in-sample performance but reduces inductive ability.

- **Threshold** (γ) controls the acceptance rate of drafted candidates, impacting the number of decoding steps needed for K recommendations. As shown in Figure 6.3, it governs the performance-efficiency trade-off. Lower thresholds degrade in-sample performance due to overly easy candidate acceptance. We select γ using the elbow criterion that balances marginal performance gain against additional latency.

6.3.5 Plug-and-Play Framework (RQ4)

To evaluate SpecGR’s plug-and-play capability, we integrate it with multiple inductive drafters (SemanticKNN, UniSRec, and the GR encoder (SpecGR++)) and multiple GR backbones (TIGER and DSI [204]). Notably, DSI employs hierarchical K-means tokenization to derive item semantic IDs. As shown in Table 6.4, both GR backbones are originally unable to generate unseen items. Integrating SpecGR improves overall performance by approximately 15% on average, while enabling strong inductive recommendation. This improvement holds across lightweight retrieval (Semantic-KNN), modality-based models (UniSRec), and self-drafting using the GR encoder, demonstrating that SpecGR is robust to different drafting paradigms and input modalities. Thus, SpecGR is model-agnostic to drafter choice and serves as a plug-and-play framework that equips any semantic-ID-based GR model with inductive capability.

6.4 Additional Implementation Details

6.4.1 SpecGR++ Training Details

We train SpecGR++ with Distributed Data Parallel (DDP) using 4 GPUs (NVIDIA RTX A6000, 48GB each, Ubuntu 22.04.5 LTS). The random seed is fixed to 42 for reproducibility. During pretraining, we compute the generation loss $\mathcal{L}_{\text{Gen}}$ with a batch size of 256, and the contrastive loss $\mathcal{L}_{\text{CL}}$ with an effective batch size of 2048 (negatives gathered across GPUs). We train for up to 200,000 steps with early stopping, selecting learning rates from $\{0.001, 0.0003\}$. For fine-tuning, SpecGR++ is trained for 15 epochs with a batch size of 256 and a learning rate of 10^{-4} . The draft threshold γ is tuned between -1.1 and -1.8 (step size 0.1), and beam size β and draft size δ are set to 50 or K . For SpecGR_{Aux}, it is trained on a single GPU with the same hyperparameters but without distributed negative gathering.

Table 6.5. Average validation Recall@50 for SpecGR++ across different training embedding batch sizes.

Batch Size	Avg. R@50
1024	0.0598
2048	0.0692
4096	0.0769

6.4.2 Embedding Batch Size Ablation

During SpecGR++ training, to ensure the model receives sufficient negative training signals, we leverage much larger batch sizes for embedding task than generative task. We report the performance against the effective batch size, which equals the batch size $\times$ number of gpus in the distributed setting Table 6.5. We can see that larger embedding batch size positively affects the performance of embeddings from SpecGR’s encoder.

6.4.3 Baseline Model Reproductions

We reproduce TIGER by following the instructions provided in its original implementation [174]. For Recformer, we utilize pretrained checkpoints from a popular reproduction repository³ and fine-tune them on our processed datasets. We implement LIGER by closely following the method details, pseudocode, and hyperparameter configurations outlined in the original paper [127]. For item text embeddings, we use representations obtained from a pretrained T5 sentence encoder. All other baseline models (e.g., SASRec variants and modality-based methods) are implemented using the open-source RecBole library [285]. We conduct thorough hyperparameter tuning for all baseline models and report results using the best-performing configuration for fair comparison.

³<https://github.com/AaronHeee/RecFormer>

Table 6.6. Statistics of the datasets. “New%” denotes the proportion of interactions with unseen target items. “#Items” and “#Inter.” are in thousands (K).

Dataset	Items		Train	Valid		Test	
	#Items	New %	#Inter.	#Inter.	New %	#Inter.	New %
Games	25.6	10.3	645.3	33.1	27.9	41.5	60.3
Office	77.6	15.1	1230.2	136.1	16.2	211.3	59.4
Phones	111.5	15.1	1841.5	232.9	33.0	297.4	68.3

6.4.4 TIGER_C Implementation

Following Rajput et al. [174], during inference, we use TIGER to generate top- K candidates and retrieve unseen items whose semantic prefix (*i.e.*, semantic IDs without the last identifier token) appears in this list. A hyperparameter ε controls the maximum proportion of unseen items included. After retrieving x unseen candidates, where $x \leq \varepsilon \cdot K$, we append them after the first $K - x$ generated candidates to finalize the recommendation. We tune ε on each dataset and report the best results in Table 6.3.

6.4.5 SpecGR++ Training Details

6.5 Case Study: Guided Redrafting

From Table 6.3, guided re-drafting leverages the generative backbone to refine redrafted candidates, enhancing overall performance. To analyze its mechanism and impact, we conduct a case study under a fixed threshold ($\gamma = -1.4$). First, Table 6.7 reports the comparison of acceptance rate per decoding step between naive drafting (w/o) and guided re-drafting (w). Under a strict threshold, we can see that the step-1 acceptance rate is low, highlighting the need for re-drafting. Guided re-drafting consistently increases the number of accepted items in later steps, improving verification efficiency. Furthermore, Table 6.8 presents the impact of guided re-drafting on overall and subset-specific performance under a fixed threshold. As shown, guided re-drafting yields improvements across all categories, particularly in the unseen subset, where

Table 6.7. Acceptance rate (%) per decoding step with and without guided re-drafting when $\gamma = -1.4$.

Step	1	2	3	4
w/o	10.34	5.93	4.25	2.63
w	10.34	6.62	5.51	5.75

Table 6.8. Performance comparison with and without guided re-drafting when $\gamma = -1.4$.

	Overall		Unseen		In-Sample	
	R@50	N@50	R@50	N@50	R@50	N@50
w/o	0.0702	0.0218	0.0288	0.0076	0.1331	0.0434
w	0.0721	0.0223	0.0317	0.0082	0.1333	0.0436

recall and NDCG scores rise by approximately 10%. This validates that leveraging redrafted tokens to constrain the drafting scope results in higher-quality candidates in subsequent steps.

6.6 Ablation: Comparison with Ensemble Methods

We compare SpecGR_{Aux} with ensemble-based variants combining GR and modality-based models, as shown in Table 6.9. The *Score-based Ensemble* linearly combines TIGER’s likelihood scores and UniSRec’s ranking scores. The *Ranking-based Ensemble* averages item ranking positions from both models to mitigate score scale mismatches. The *2-Stage Ensemble* selects top- K items using UniSRec and re-ranks them with TIGER’s scores. All ensemble methods have unseen performance bounded by the two base models, TIGER and UniSRec, since simple score aggregation or re-ranking does not introduce inductive capabilities. In contrast, SpecGR uses guided re-drafting to effectively leverage the backbone’s modeling capability for enhancing unseen candidates’ quality. The results illustrate the effectiveness of the proposed SpecGR methods.

Table 6.9. Performance comparison of SpecGR_{Aux} (UniSRec drafter) against ensemble variants. The best and second-best scores are bolded and underlined.

Model	Overall		Unseen		In-Sample	
	R@50	N@50	R@50	N@50	R@50	N@50
<i>Single Model</i>						
TIGER	0.0584	0.0193	–	–	0.1472	<u>0.0486</u>
UniSRec	0.0621	0.0200	<u>0.0118</u>	<u>0.0029</u>	0.1386	0.0461
<i>Ensemble Method</i>						
Score-based	0.0571	0.0191	0.0050	0.0009	0.1333	0.0456
Ranking-based	<u>0.0678</u>	<u>0.0218</u>	0.0056	0.0011	0.1624	0.0532
2-Stage	0.0621	0.0205	<u>0.0118</u>	0.0026	0.1386	0.0477
SpecGR (UniSRec)	0.0778	0.0239	0.0312	0.0096	<u>0.1485</u>	0.0457

6.7 Inference Speed Acceleration

Generative recommendation (GR) models rely on autoregressive next-token generation, which incurs high inference latency. SpecGR addresses this by employing speculative retrieval-based drafting, where candidate items proposed by the drafter can be efficiently verified by the GR verifier, significantly reducing the number of required autoregressive decoding steps.

6.7.1 Drafting and Verification Latency

To clearly illustrate SpecGR’s efficiency, we report detailed inference latency results (in seconds) separated into drafting (D) and verification (V) stages across various draft sizes. As shown in Table 6.10, SpecGR++ achieves remarkably low drafting latency due to its efficient encoder-based drafting mechanism, demonstrating significant overall speed improvements compared to TIGER.

Table 6.10. Empirical inference latency (seconds) across drafting (D) and verification (V) stages.

Draft Size	20 (D/V)	50 (D/V)	100 (D/V)
SpecGR (UniSRec)	0.0110 / 0.0261	0.0112 / 0.0249	0.0112 / 0.0227
SpecGR++	0.0003 / 0.0261	0.0003 / 0.0249	0.0003 / 0.0235
TIGER	- / 0.0403	- / 0.0403	- / 0.0403

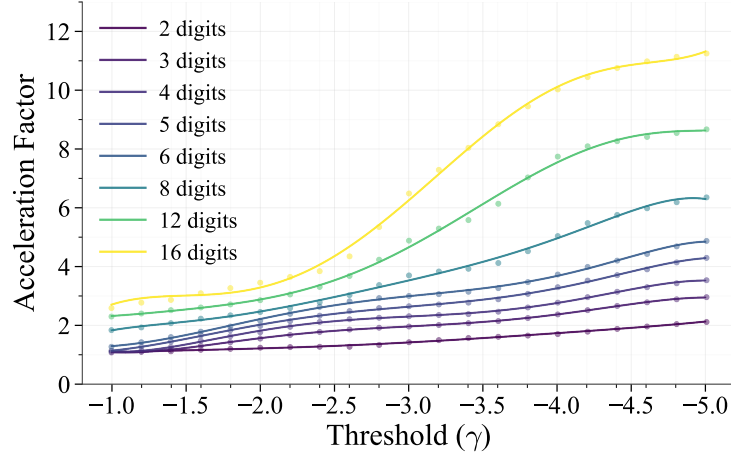


Figure 6.4. Inference speed acceleration factor w.r.t. different numbers of semantic ID digits.

6.7.2 Acceleration w.r.t. Generation Length

We analyze how SpecGR reduces inference latency as the generation length increases. Specifically, we illustrate the acceleration factor w.r.t. the number of digits in semantic IDs. We train SpecGR++ with a TIGER backbone across different semantic ID lengths (ranging from 2 to 16 digits) and report the acceleration factor relative to the TIGER baseline. The acceleration factor is defined as the inverse ratio of inference time. As shown in Figure 6.4, in standard GR settings with 4-digit semantic IDs, SpecGR achieves a $1.7\times$ inference speedup while maintaining strong overall performance (see the Experiments section). Moreover, longer semantic IDs lead to even higher acceleration as fewer decoding steps are needed per item.

6.7.3 Theoretical Time Complexity

We further analyze the expected time complexity of SpecGR, where each item is represented by l semantic tokens. Let K be the number of items to recommend, δ the draft size per iteration, and p the acceptance probability of a drafted item. We denote the runtime for a single forward pass in the GR encoder and decoder as $C_{encoder}$ and $C_{decoder}$, respectively. A standard GR model (e.g., TIGER) must decode all l tokens for K beams, yielding a complexity:

$$\text{Cost}_{\text{TIGER}} = \mathcal{O}(C_{encoder} + l \cdot C_{decoder}).$$

For SpecGR, inductive drafting requires one forward pass and a KNN search, denoted as C_{draft} . Verification and beam search happen in parallel with a cost of $C_{decoder}$ per iteration. The expected number of iterations to obtain K valid recommendations is bounded by:

$$T = \mathcal{O}\left(\frac{K}{\delta p}\right) \leq \mathcal{O}(l).$$

Thus, SpecGR’s expected time complexity is:

$$\text{Cost}_{\text{SpecGR}} = \mathcal{O}(C_{draft} + C_{encoder} + T \cdot C_{decoder}),$$

with an expected acceleration factor:

$$\frac{C_{encoder} + l \cdot C_{decoder}}{C_{draft} + C_{encoder} + \frac{K}{\delta p} \cdot C_{decoder}}.$$

6.7.4 Empirical Acceptance Rates and Speedup

We empirically measure the acceptance probability p for various drafting methods and report their corresponding acceleration factors in Table 6.11. For the SemanticKNN drafter, item sequences are encoded using mean text embeddings from the most recent five items, and top- K items are retrieved via KNN search. Across all methods, SpecGR consistently outperforms traditional GR models.

Table 6.11. Empirically measured acceptance rate p and expected acceleration factor for different drafters at $\gamma = -1.4$.

Drafter	Acceptance Rate (p)	Acceleration Factor
SpecGR++ Encoder	0.44	1.72
UniSRec	0.35	1.38
SemanticKNN	0.20	1.11

6.7.5 Scalability Analysis

SpecGR exhibits favorable scalability in multiple dimensions compared to traditional GR models:

- **Model Size.** As the GR model scales, assuming $C_{encoder} \approx C_{decoder}$, SpecGR maintains a constant acceleration factor $\frac{l+1}{\frac{K}{\delta p}+1}$, demonstrating robust scalability.
- **Semantic ID Length.** The acceleration factor grows linearly as the semantic ID length increases, i.e., $\mathcal{O}\left(\frac{l \cdot \delta \cdot p}{K}\right)$, highlighting SpecGR’s advantage in scenarios requiring long sequences, as quantitatively shown in Figure 6.4.

6.8 Additional Further Analysis

We briefly discuss SpecGR’s capabilities compared to traditional methods. Leveraging the draft-then-verify framework, SpecGR seamlessly integrates into diverse recommendation scenarios, adapting to dynamic needs (Table 6.13). For instance, when deployed as a ranker model, it exploits autoregressive generation for efficient subset ranking. Additionally, its tunable hyperparameters enable real-time controllable inductive ability and controllable inference speed. See the section Other New Capabilities for broader implications.

6.8.1 Subset Ranking

In this section, we assess new capability of SpecGR in efficient subset ranking compared to traditional subset ranking methods for GR models. Since most deep learning-based recommendation methods unavoidably introduce high latency, they are typically used as ranking models rather than retrieval models [34, 89]. In the subset ranking setting, the model of interest is applied as a ranker to rank a given subset of items ($\mathcal{I}_r \subset \mathcal{I}, |\mathcal{I}_r| \ll |\mathcal{I}|$). Subset ranking is out-of-the-box for traditional sequential recommendation models by selecting a subset of candidates when performing KNN search. However, for generative recommendation models,

subset ranking presents a challenge as they inherently recommend items by searching for the top K decoding paths across the entire item space.

Batch scoring (BS)

A simple approach to address this is through batch scoring, which involves splitting the item subset into fixed-size batches and scoring them consecutively with the generative model. However, this method grows linearly with batch size and is impractical for large subsets, as shown in Figure 6.5.

Constrained beam search (CBS)

An enhanced method is to use constrained beam search [3, 168, 38]. Constrained beam search constructs a trie using all allowed prefixes to restrict the search space at each decoding step. However, this approach introduces significant computational overhead. The time complexity for CBS is $\mathcal{O}(T \cdot n_b \cdot 2^C)$, where T is the sequence length, n_b is the beam size, and C is the number of constraints [24]. The large computational overhead and exponential growth with the number of constraints makes it inefficient for large-scale ranking tasks, as shown quantitatively in Figure 6.5.

SpecGR for subset ranking

SpecGR effectively addresses this issue by restricting the drafter model’s range to a specified subset. This ensures that all recommendations originate from within the subset. As demonstrated in Figure 6.5, SpecGR achieves a $3.5\times$ speedup for subset sizes $< 10^4$ compared to TIGER with constrained beam search (denoted as CBS). Moreover, SpecGR maintains a time complexity that is bounded by its full ranking complexity as the retrieval size increases, making it a highly efficient solution for subset ranking tasks.

6.8.2 Other New Capabilities

As a direct impact of the adjustable hyperparameters analyzed in the Experiments section, SpecGR possesses two other new capabilities: controllable inductive ability and controllable

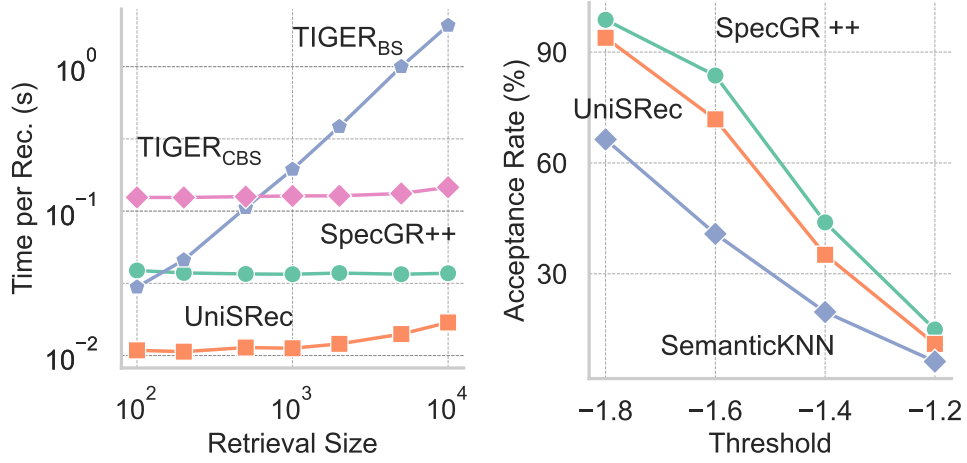


Figure 6.5. (Left) Inference latency comparison for subset ranking. Both x- and y-axis use log scale. (Right) Acceptance rate comparison for different drafting strategies.

Table 6.12. Comparison of different models based on parameter efficiency and training time.

Model	Trainable (M)			Non-trainable (M)	Training Time (h)
	Total	Non-emb	Emb		
SASRec _{ID}	7.24	0.10	7.13	0	3.6
UniSRec	2.90	2.90	0	85.62	18.3
Recformer	233.73	106.32	127.41	0	226.0
TIGER	13.26	13.11	0.15	0	16.2
TIGER _C	13.26	13.11	0.15	0	16.2
SpecGR _{Aux}	16.16	16.02	0.15	85.62	34.5
SpecGR++	13.28	13.13	0.15	14.27	42.8

inference speed.

Controllable inductive ability allows platforms to dynamically adjust their recommendation strategy based on seasonal demand, favoring new items during certain periods while prioritizing established products at others. For instance, e-commerce platforms could increase the inductive ability of SpecGR to promote new items, and reduce it during clearance sales.

Controllable inference speed enables platforms to trade off model performance for faster inference speed during high-traffic periods, ensuring responsive user experiences.

Table 6.13. Comparison of SpecGR against existing models across different scenarios and model capabilities.

Model	Recommendation Scenario		Model Capability		
	Efficient Subset Ranking	Inductive Recommendation	Autoregressive Generation	Controllable Inductive Ability	Controllable Inference Speed
SASRec _{ID}	✓	✗	✗	✗	✗
UniSRec	✓	✓	✗	✗	✗
Recformer	✓	✓	✗	✗	✗
TIGER	✗	✗	✓	✗	✗
TIGER _C	✗	✓	✓	✓	✗
SpecGR _{Aux}	✓	✓	✓	✓	✓
SpecGR++	✓	✓	✓	✓	✓

In summary, as shown in Table 6.13, SpecGR inherits the architecture of generative recommendation, allowing effective scaling on large datasets. It also extends high performance and low inference speed to broader real-life recommendation settings, adapting to specific data characteristics and recommendation needs.

6.9 Further discussion on SpecGR++ Architecture

SpecGR++ utilizes its encoder as the self-drafter, effectively eliminating the need for maintaining a separate draft model for drafting, resulting in a more integrated method during inference. In this section, we will study the additional advantages of the SpecGR++ compared to the SpecGR with an auxiliary model.

6.9.1 Parameter Efficiency and Speed

First, SpecGR++ uses intermediate encoder outputs for drafting, resulting in nearly no additional computational cost and parameter sizes compared to GR. We report the total number of parameters and training time required for different methods in Table 6.12. As we can see, SpecGR inherits GR’s advantages for scaling for the larger dataset as it assigns most of the parameters into non-embedding layers. Due to the additional embedding training tasks, SpecGR++ training time is slightly longer than training a drafter model and TIGER, and is 2.6x

more GPU hours than training a TIGER. However, we believe that it is worthwhile to consider the acceleration during inference time. We’ve also provided a distributed training implementation in the released code.

6.9.2 Unified Representation Space

Because both the drafter and verifier use the encoder’s representation, we observe a higher acceptance rate for the self-drafter compared to drafting with auxiliary models. This leads to better recommendation efficiency, where less decoding step is required.

Notably, we also observe a slight increase in generative performance after SpecGR++ pretraining compared to generation-only training (*e.g.*, TIGER). Recent studies have shown similar results, indicating that with a unified representation space, a model can maintain high performance in both generative and embedding tasks in NLP [155]. Our study further confirms that generation and representation are not conflicting tasks in the recommendation setting but rather two complementary approaches to solving the same problem. We look forward to seeing future research in recommendation systems that explores the unification and overlap between generative recommendation and representational recommendation.

6.10 Conclusion

In this chapter, we propose SpecGR, a plug-and-play framework that extends the capability of generative recommendation models for inductive recommendation. Our method, inspired by speculative decoding, leverages an inductive model as a drafter to propose candidate items and uses the GR model as a verifier to ensure that only high-quality candidates are recommended. We further propose two drafting strategies: (1) using an auxiliary model for flexibility, and (2) using the GR model’s own encoder for parameter-efficient self-drafting. Extensive experiments on three public datasets demonstrate strong inductive and overall recommendation performance for SpecGR.

Chapter 6, in part, is a reprint of the material as it appears in “Inductive Generative

Recommendation via Retrieval-Based Speculation.” by Yijie Ding, Jiacheng Li, Julian McAuley, and Yupeng Hou, in Proceedings of the AAAI Conference on Artificial Intelligence in 2026, referenced as [46]. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Expressiveness Limits of Autoregressive Semantic ID Generation in Generative Recommendation

After Chapter 6 extends generative recommendation to new-item recommendation, this chapter examines a limitation that arises from the autoregressive decoding process itself.

Generative recommendation (GR) models generate items by autoregressively producing a sequence of discrete tokens that jointly index the target item. However, this autoregressive generation process also induces a structured decoding space whose impact on model expressiveness remains underexplored. Specifically, token-by-token generation can be viewed as traversing a decoding tree induced by semantic ID tokens, where leaf nodes correspond to candidate items. We observe that the item probabilities produced by GR models are strongly correlated with this tree structure: items that are close in the tree tend to receive similar probabilities for any given user, making it difficult to distinguish among them based on user-specific preferences. We further show theoretically that such structural correlations prevent GR models from representing even simple patterns that can be well captured by conventional collaborative filtering models. To mitigate this issue, we propose Latte, a simple modification that injects a latent token before each semantic ID, reshaping the decoding space from a single tree into multiple latent-token-conditioned trees. This design creates multiple paths with varying tree distances between items, relaxing tree-induced probability coupling and yielding an average of 3.45%

relative improvement on NDCG@10.¹

7.1 Introduction

Generative recommendation (GR) [174, 290, 41, 76] tokenizes items as sequences of discrete tokens (semantic IDs or SIDs [204, 228, 174]). Unlike traditional models that score user-item preferences via representation similarity [82, 108], GR models autoregressively generate SID tokens, scoring an item as the product of its tokens' conditional probabilities. While existing research has largely focused on improving how items are tokenized [220, 302, 91], the autoregressive generation process itself is often taken for granted, despite directly determining how item scores are composed and constrained.

In this chapter, we take the first step towards understanding the expressive power of this autoregressive token generation process of GR models. Specifically, we investigate whether there exist specific user-item preference patterns that GR models struggle to express. Note that when calculating the probability of an item (*i.e.*, a sequence of tokens), items sharing the same initial tokens also share common terms in the probability calculation, leading to correlated item probabilities. This motivates us to formulate a *decoding tree* view of the autoregressive process, where generating a token corresponds to traversing one level down the tree, and each leaf node represents a candidate item (with the path corresponding to a valid semantic ID). Based on this formulation, we investigate the relationship between learned item probabilities and the decoding tree structure.

Through empirical analysis, we find that if two items have a short distance in the decoding tree (*i.e.*, they share a long prefix), their predicted probabilities are strongly correlated for any given user. Intuitively, this implies that if a GR model predicts Alice prefers item A over item B (where A and B are close in the tree), it is highly likely to predict that Bob also prefers A over B. This contradicts the fundamental assumption of personalized recommendation that preferences should be user-specific. Driven by this observation, we theoretically demonstrate that there

¹Our code is available at <https://github.com/hyp1231/Latte>.

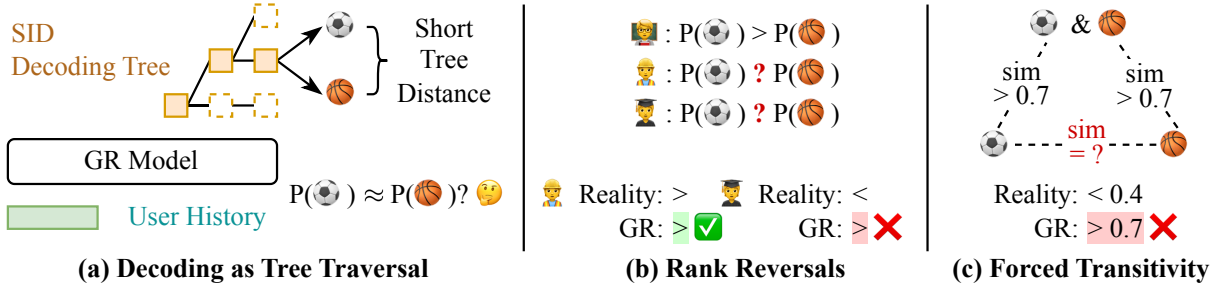


Figure 7.1. Illustration of the expressive limitations induced by autoregressive SID generation. (a) Generating a SID can be viewed as traversing a decoding tree, where items sharing longer prefixes have shorter tree distances and tend to receive highly correlated generation probabilities. (b) shows that GR tends to assign consistent relative preference scores across users, while (c) shows that the tree structure can force two items to be similar even when their true similarity is low.

exist simple user-item preference patterns (Section 7.3.3) and item-item similarity relationships (Section 7.3.4) that GR models struggle to represent, thereby limiting their expressive power.

To alleviate this issue, we propose **Latte**, a simple yet effective modification to the standard SID generation approach. We introduce a small set of additional tokens, named *latent tokens*. During training, we inject a randomly sampled latent token before the target semantic ID. The concatenated sequence then serves as the new optimization target. This modification effectively reshapes the decoding structure from a single universal tree to multiple trees rooted at a shared hyper-root, where the second level corresponds to the latent tokens. This design enables multiple paths with varying tree distances between any pair of items, thus relaxing the strong correlation between tree structure and item probabilities in existing GR models. Experiments on benchmarks validate the effectiveness of the proposed method, leading to an average of 3.45% relative improvement on NDCG@10.

Finally, we provide a further exploration of binding latent tokens with inductive biases. Specifically, in a multimodal scenario where each token corresponds to a modality of item features, we bind each latent token to a specific permutation of the semantic ID. Empirically, we show that latent tokens associated with better-performing permutations indeed receive higher selection frequency, leading to improved overall recommendation performance.

7.2 Preliminaries

Semantic ID.

A SID refers to a sequence of discrete tokens that jointly index an item. Formally, a SID can be represented as a tuple $(c^{(1)}, c^{(2)}, \dots, c^{(m)})$, where m denotes the length of the SID. Each token $c^{(j)}$ is selected from a compact vocabulary $\mathcal{C}^{(j)}$ of size M that is shared across all items.

Generative recommendation.

GR models aim to predict the next item i_t with which a user will interact, given the historical interaction sequence $(i_1, i_2, \dots, i_{t-1})$. In this framework, each item is tokenized into its corresponding semantic ID. Formally, given the semantic IDs for historical interactions $u = (c_1^{(1)}, \dots, c_{t-1}^{(m)})$, the model is trained to generate the target semantic ID $(c_t^{(1)}, c_t^{(2)}, \dots, c_t^{(m)})$ in a token-by-token manner. Therefore, the user-item preference score is often defined as the joint probability of generating the target Semantic ID:

$$\mathbb{P}(i_t | u) = \prod_{j=1}^m \mathbb{P}(c_t^{(j)} | c_t^{(1)}, \dots, c_t^{(j-1)}, u). \quad (7.1)$$

7.3 Limitations of Autoregressive SID Generation

In this section, we analyze the expressive limitations of the autoregressive semantic ID generation process in generative recommendation models. We first formulate the generation process as a tree traversal procedure in Section 7.3.1. Next, we empirically demonstrate a strong correlation between the decoding tree structure and item generation probabilities in Section 7.3.2. Based on these observations, we provide a theoretical analysis showing that such structural correlations constrain the expressive power of generative recommendation (GR) models, causing them to struggle with expressing specific patterns of user-item preferences (Section 7.3.3) and item-item similarities (Section 7.3.4).

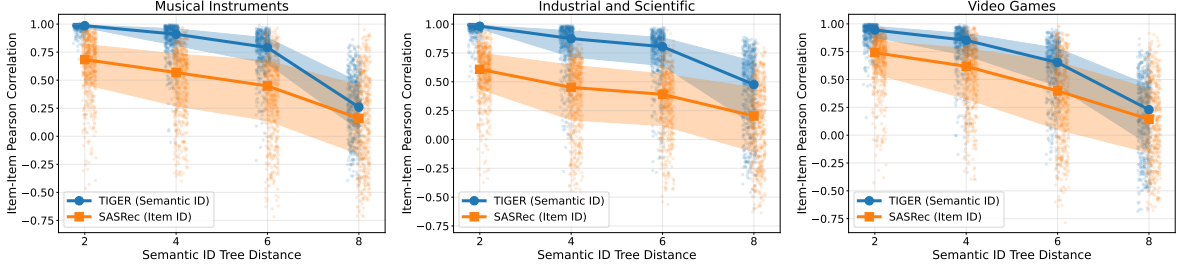


Figure 7.2. Correlation between tree distance and item generation probability similarity.

7.3.1 Generation as Tree Traversal

Generative recommendation models predict the next item by autoregressively generating a sequence of semantic ID tokens. We can formulate this generation process as traversal along a decoding tree induced by the set of valid semantic IDs. We define the decoding tree as follows:

Definition 7.3.1 (Decoding Tree). Let $\mathcal{T}$ be a decoding tree of depth m formed by all valid semantic IDs. The root represents the empty sequence, and each node at depth j corresponds to a unique prefix $(c^{(1)}, \dots, c^{(j)})$. An edge connects a node to its child if the child’s prefix extends the parent’s by one valid token. Consequently, each leaf node uniquely represents an item.

Under this formulation, the autoregressive generation of a semantic ID is equivalent to traversing from the root to a leaf node in the decoding tree. We adopt this perspective because it links the number of shared prefix tokens between two items to a well-defined distance metric:

Definition 7.3.2 (Tree Distance). Consider two items i and i' with semantic IDs $(c^{(1)}, \dots, c^{(m)})$ and $(c'^{(1)}, \dots, c'^{(m)})$, respectively. Let k denote the length of the longest common prefix shared by these IDs. The tree distance between i and i' is defined as $d_{\mathcal{T}}(i, i') = 2(m - k)$, which corresponds to the number of edges on the unique path connecting the corresponding leaf nodes in $\mathcal{T}$.

7.3.2 Tree Distance vs. Item Generation Probability

Motivation.

Based on Equation (7.1), consider two items i and i' with a tree distance of $2(m - k)$, implying they share a prefix of length k . The item generation probabilities can be factored as

follows:

$$\begin{aligned} \mathbb{P}(i \mid u) &= \prod_{j=1}^k \mathbb{P}(c^{(j)} \mid c^{(1)}, \dots, c^{(j-1)}, u) \\ &\quad \cdot \prod_{j=k+1}^m \mathbb{P}(c^{(j)} \mid c^{(1)}, \dots, c^{(j-1)}, u), \end{aligned} \quad (7.2)$$

$$\begin{aligned} \mathbb{P}(i' \mid u) &= \prod_{j=1}^k \mathbb{P}(c'^{(j)} \mid c'^{(1)}, \dots, c'^{(j-1)}, u) \\ &\quad \cdot \prod_{j=k+1}^m \mathbb{P}(c'^{(j)} \mid c'^{(1)}, \dots, c'^{(j-1)}, u), \end{aligned} \quad (7.3)$$

where $c^{(j)} = c'^{(j)}$ for all $j \leq k$ (the shared prefix). The highlighted terms in Equations (7.2) and (7.3) are identical, as they depend solely on the common prefix. This factorization suggests that items with smaller tree distances (longer shared prefixes) will likely exhibit more similar generation probabilities, largely independent of the user's preference.

Empirical validation.

To validate this hypothesis, we compare TIGER [174], a representative GR model, with SASRec [108], a standard item ID-based model, across three public datasets (detailed settings are provided in Section 7.5.1). We uniformly sample 1,024 pairs of items across varying tree distances ($\{2, 4, 6, 8\}$). For each pair (i, i') , we randomly sample 512 users $\{u_k\}_{k=1}^{512}$, compute the generation probabilities $\mathbb{P}(i \mid u_k)$ and $\mathbb{P}(i' \mid u_k)$ for each sampled user, and calculate the Pearson correlation coefficient between these two probability sequences.

Correlation analysis.

The empirical results demonstrate that item generation probabilities are highly correlated when items are close in the decoding tree. Furthermore, the strength of this correlation increases monotonically as tree distance decreases. For instance, we observe a Pearson correlation of ~ 1.0 for items with a tree distance of 2, which drops to > 0.8 for a distance of 4. Formally, we encode this observation into the following assumption for our theoretical analysis:

Assumption 7.3.3. For any pair of items i and i' , let $\rho(i, i')$ denote the Pearson correlation coefficient between their generation probabilities $\mathbb{P}(i \mid u)$ and $\mathbb{P}(i' \mid u)$ across the user population.

We assume that for any threshold δ , the probability $\mathbb{P}(\rho(i, i') > \delta)$ is monotonically decreasing w.r.t. the tree distance $d_{\mathcal{T}}(i, i')$.

7.3.3 Limitation on User-Item Preference Modeling

In this section, we analyze how the structural property observed in Section 7.3.2 constrains the model's ability to express diverse user-item preference patterns. We intuitively formulate the limitation as the inability to express *rank reversals* between items that are close in the tree.

Rank reversals.

Let u and u' be two independent users drawn from the population $\mathcal{U}$. We define the rank reversal event $\mathcal{R}(i, i')$ for a pair of items i and i' as the scenario where users have opposite relative preferences:

$$\begin{aligned} \mathcal{R}(i, i') \triangleq & \{ \mathbb{P}(i | u) > \mathbb{P}(i' | u) \wedge \mathbb{P}(i | u') < \mathbb{P}(i' | u') \} \\ & \cup \{ \mathbb{P}(i | u) < \mathbb{P}(i' | u) \wedge \mathbb{P}(i | u') > \mathbb{P}(i' | u') \}. \end{aligned} \quad (7.4)$$

Rank reversals are essential for personalized recommendation; without them, the relative order of i and i' would be identical for all users, reducing to a non-personalized ranking. Specifically, we have the following theorem bounding the probability of rank reversals based on the correlation of generation probabilities.

Theorem 7.3.4 (Correlation Suppresses Rank Reversals). *Let σ^2 denote the variance of the generation probabilities (assumed equal for i and i' , see Section 7.8). The rank reversal probability is bounded by:*

$$\mathbb{P}(\mathcal{R}(i, i')) \leq \frac{4\sigma^2(1 - \rho(i, i'))}{\mu^2 + 2\sigma^2(1 - \rho(i, i'))}, \quad (7.5)$$

where $\mu = |\mathbb{E}[\mathbb{P}(i | u) - \mathbb{P}(i' | u)]|$ is the expected difference in generation probabilities and $\rho(i, i')$ is the correlation defined in Theorem 7.3.3.

A detailed proof is provided in Section 7.8.

Implication.

Theorem 7.3.4 implies that as $\rho(i, i') \rightarrow 1$, the rank-reversal probability $\mathbb{P}(\mathcal{R}(i, i')) \rightarrow 0$. Combining with Theorem 7.3.3, we conclude that for items with small tree distance $d_{\mathcal{T}}(i, i')$, GR is structurally constrained to assign similar relative ranking of i and i' across all users. This limits the model's ability to capture users whose preferences diverge from the majority trend.

7.3.4 Limitation on Item-Item Similarity Modeling

Beyond user-item preferences, a powerful recommender system should also capture complex item-item similarities. To analyze this, we adopt a collaborative filtering view [182, 184], where item-item similarity is induced by the inner product of the user-item preference matrices. A critical property of such similarity in real-world scenarios is that it is *not necessarily transitive*. Consider a scenario with three items and two user groups G_A and G_B :

- Item i_1 : Preferred by G_A .
- Item i_2 : Preferred by both G_A and G_B .
- Item i_3 : Preferred by G_B .

In this case, a flexible model should capture that i_1 is similar to i_2 (co-preferred by G_A) and i_2 is similar to i_3 (co-preferred by G_B), while simultaneously reflecting that i_1 and i_3 are dissimilar (disjoint user base). However, the strong correlation between the tree structure and item generation probabilities identified in Section 7.3.2 hinders the model's ability to represent such intransitive similarity relationships. We formalize this limitation in the following theorem:

Theorem 7.3.5 (Forced Transitivity). *Based on Assumption 7.3.3, suppose high similarity (correlation $> \tau$) implies small tree distance $d_{\mathcal{T}} \leq \delta$, and conversely $d_{\mathcal{T}} \leq \delta$ implies correlation $> \tau$. Then, if the model captures similarities for both (i_1, i_2) and (i_2, i_3) (correlation $> \tau$), it is structurally forced to assign correlation $> \tau$ to (i_1, i_3) , preventing the representation of their dissimilarity.*

A detailed proof is provided in Section 7.9.

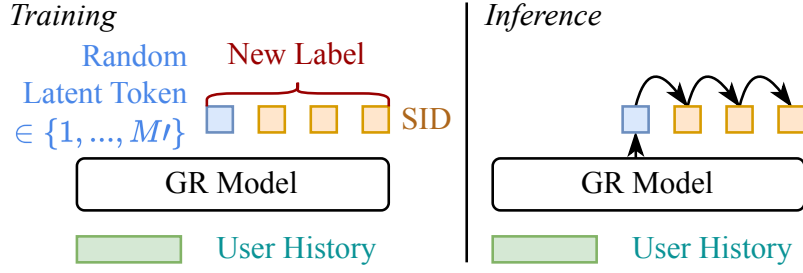


Figure 7.3. Overall framework of Latte.

Implication.

The proof leverages the fact that the tree distance $d_{\mathcal{T}}$ satisfies the ultrametric inequality: $d_{\mathcal{T}}(i_1, i_3) \leq \max(d_{\mathcal{T}}(i_1, i_2), d_{\mathcal{T}}(i_2, i_3))$. This structural property implies that items sharing common similar neighbors in the decoding tree are forced to be close to each other. Consequently, generative recommendation models may struggle to distinguish items that are locally similar to the same set of items but distinct from each other (*e.g.*, sharing different features or preferred by different user groups), effectively limiting the model’s expressiveness.

7.4 Alleviating Expressive Limits

Having analyzed the expressive limitations of standard autoregressive semantic ID generation in GR, we now present **Latte** (Figure 7.3), a simple yet effective modification designed to relax the constraints imposed by the decoding tree structure and enhance model expressiveness. The core idea is to condition the model to predict an additional latent token prior to generating the semantic ID tokens. This effectively introduces a super-root node that connects multiple copies of the original decoding trees, allowing for multiple paths with varying tree distances between any pair of items (leaf nodes of the decoding tree). Below, we detail the training (Section 7.4.1) and inference (Section 7.4.2) processes of Latte, and discuss how this straightforward adjustment improves the expressive power of GR models (Section 7.4.3).

7.4.1 Training: Sampling Latent Tokens

Latte uses a small set of additional discrete tokens, termed latent tokens, denoted as $\mathcal{L} = \{\ell_1, \ell_2, \dots, \ell_{M'}\}$, where $M' \ll M$ represents the number of latent tokens. During training, for each target semantic ID $(c_t^{(1)}, c_t^{(2)}, \dots, c_t^{(m)})$, we randomly sample a latent token $\ell \in \mathcal{L}$ and prepend it to the semantic ID, thereby creating an augmented target sequence $(\ell, c_t^{(1)}, c_t^{(2)}, \dots, c_t^{(m)})$. The GR model is then trained to generate this augmented sequence autoregressively using the standard next-token prediction loss.

7.4.2 Inference: Generating Latent Tokens

At inference time, we impose no constraints on the selection of latent tokens. Instead, we allow the model to generate autoregressively, initiating generation with a latent token followed by the semantic ID tokens. Consequently, the user-item preference score is reformulated as:

$$\mathbb{P}(i_t | u) = \text{Agg}_{\ell \in \mathcal{L}} \left(\mathbb{P}(\ell | u) \cdot \prod_{j=1}^m \mathbb{P}(c_t^{(j)} | \ell, c_t^{(1:j-1)}, u) \right),$$

where Agg denotes the aggregation operator, which can be instantiated via operations such as summation (sum) or maximization (max). In practice, following prior work [174, 290], we employ beam search during inference to efficiently approximate this aggregation.

7.4.3 Improved Expressive Power via Latent Tokens

The analysis in Section 7.3 demonstrates that the fixed tree distance $d_{\mathcal{T}}$ enforces a high correlation between the generation probabilities of structurally close items. Latte effectively alleviates this limitation by introducing latent tokens $\mathcal{L}$, thereby expanding the single decoding tree into a forest of $|\mathcal{L}|$ trees. This structural modification enables dynamic distances between items.

Dynamic tree distance.

In standard GR, tree distance $d_{\mathcal{T}}(i, i')$ remains constant. In contrast, Latte employs an aggregation operation (*e.g.*, $\text{Agg} = \max$) that allows the generation process for a user u to select a specific latent path. Let $\ell^*(i, u) = \arg \max_{\ell} \mathbb{P}(\ell \mid u) \mathbb{P}(i \mid \ell, u)$ denote the dominant latent token for generating item i given user u . The “effective” structural distance between i and i' thus becomes context-dependent:

$$d_{\text{eff}}(i, i'; u) = \begin{cases} d_{\mathcal{T}}(i, i') & \text{if } \ell^*(i, u) = \ell^*(i', u), \\ 2(m+1) & \text{if } \ell^*(i, u) \neq \ell^*(i', u). \end{cases} \quad (7.6)$$

By assigning distinct latent tokens to items, the model can significantly increase the effective distance. This implies that the probability computations for structurally close items diverge at an earlier stage (see Equations (7.2) and (7.3)). This capability enables the model to reflect lower correlations even for items with similar SIDs (empirically validated in Section 7.5.3). Proof can be found in Section 7.9.4.

7.5 Experiments

7.5.1 Experimental Setup

Datasets.

Following previous works [131, 291], we conduct experiments on three categories of the Amazon Reviews 2023 dataset [95]: **Instruments**, **Scientific**, and **Games**. Each user’s review history is grouped into a single sequence and sorted chronologically. We adopt the widely used leave-one-out strategy [108, 174] for data splitting, using the most recent interaction for testing, the second most recent for validation, and the remainder for training. Data statistics are summarized in Table 7.7.

Table 7.1. Performance comparisons between different methods and the proposed method Latte. The best and second-best results are highlighted in **bold** and underlined font, respectively. “ Δ ” indicates the performance gain of Latte over the best baseline. “*” denotes statistically significant improvements ($p < 0.05$) over the best baseline according to a paired t-test.

Methods	Instrument				Scientific				Game			
	R@5	R@10	N@5	N@10	R@5	R@10	N@5	N@10	R@5	R@10	N@5	N@10
GRU4Rec	0.0324	0.0501	0.0209	0.0266	0.0202	0.0338	0.0129	0.0173	0.0499	0.0799	0.0320	0.0416
BERT4Rec	0.0307	0.0485	0.0195	0.0252	0.0186	0.0296	0.0119	0.0155	0.0460	0.0735	0.0298	0.0386
SASRec	0.0333	0.0523	0.0213	0.0274	0.0259	0.0412	0.0150	0.0199	0.0535	0.0847	0.0331	0.0438
FMLP-Rec	0.0339	0.0536	0.0218	0.0282	0.0269	0.0422	0.0155	0.0204	0.0528	0.0857	0.0338	0.0444
HSTU	0.0343	0.0577	0.0191	0.0271	0.0271	0.0429	0.0147	0.0198	0.0578	0.0903	0.0334	0.0442
FDSA	0.0347	0.0545	0.0230	0.0293	0.0262	0.0421	0.0169	0.0213	0.0544	0.0852	0.0361	0.0448
S ³ -Rec	0.0317	0.0496	0.0199	0.0257	0.0263	0.0418	0.0171	0.0219	0.0485	0.0769	0.0315	0.0406
TIGER	0.0370	0.0564	0.0244	0.0306	0.0264	0.0422	0.0175	0.0226	0.0559	0.0868	0.0366	0.0467
LETTER	0.0372	0.0580	0.0246	0.0313	0.0279	0.0435	<u>0.0182</u>	0.0232	0.0563	0.0877	0.0372	0.0473
ActionPiece	0.0383	<u>0.0615</u>	0.0243	0.0318	<u>0.0284</u>	<u>0.0452</u>	<u>0.0182</u>	<u>0.0236</u>	0.0591	0.0927	0.0382	0.0490
PSID	<u>0.0390</u>	0.0602	<u>0.0256</u>	<u>0.0325</u>	0.0278	0.0445	0.0181	0.0235	<u>0.0599</u>	<u>0.0939</u>	<u>0.0391</u>	<u>0.0500</u>
Latte	0.0401*	0.0618	0.0261*	0.0331*	0.0304*	0.0470*	0.0196*	0.0249*	0.0618*	0.0958*	0.0406*	0.0515*
Δ	+2.82%	+0.49%	+1.95%	+1.85%	+7.04%	+3.98%	+7.69%	+5.51%	+3.17%	+2.02%	+3.84%	+3.00%

Evaluation details.

We adopt PSID [278] with RQ-Kmeans [106] item tokenization as our base model. Please refer to Section 7.10 for detailed implementation and evaluation settings.

7.5.2 Main Results

We compare the proposed Latte method with all baselines in Table 7.1. From the results, we can see that Latte consistently outperforms its base model, PSID, and all other baselines across all datasets and metrics. Specifically, Latte achieves an average of 3.45% relative improvement on NDCG@10 with only one simple modification, *i.e.*, generating an additional latent token before semantic IDs, demonstrating the effectiveness of relaxing the constraints discussed in Section 7.3.

7.5.3 Tree-Structure Correlation Analysis

To quantitatively compare structural constraints across different models, we analyze the association between the decoding tree structure and the item probabilities. Specifically, we adopt the definition of item-item similarity introduced in Section 7.3.2, defined as the Pearson

Table 7.2. Kendall’s rank correlation between tree distance and item-item similarity. **Bold** numbers indicate the fewest constraints.

Model	Instruments ($\uparrow$)	Scientific ($\uparrow$)	Games ($\uparrow$)
TIGER	-0.7170	-0.6137	-0.6462
PSID	-0.6225	-0.4611	-0.6072
Latte	-0.6030	-0.4451	-0.5958

correlation of item generation probabilities across the sampled user population. Since tree distances typically take only a few values (*e.g.*, $\{2, 4, 6, 8\}$), we employ Kendall’s rank correlation coefficient [110] for this measurement. This metric is more robust for evaluating monotonic associations between ranked variables. A stronger association (approaching -1 or 1) implies that item similarities are heavily dominated by the decoding tree structure. Conversely, a value closer to 0 is desirable, as it indicates the model’s capacity to learn flexible item relationships that are not strictly bound by the underlying tree structure.

As shown in Table 7.2, Latte consistently achieves lower absolute Kendall correlation values compared to its base model, PSID, while using the exact same semantic IDs. This suggests that generating an additional latent token effectively relaxes the structural constraints imposed by the decoding tree. Notably, there is a large numerical gap between TIGER and PSID, which may be attributed to TIGER’s greater tree depth (4 *vs.* 3). However, because these two models use different sets of semantic IDs, they are not directly comparable. We include the TIGER results as a reference only; how to fairly compare structural constraints across models with different semantic IDs remains an open research question.

7.5.4 In-Depth Analysis

Performance w.r.t. Item Tokenization Method

To investigate the generalizability of Latte across different item tokenization strategies, we conduct experiments using three representative tokenizers: OPQ [88, 17, 90], RQ-VAE [174, 220, 302, 291], and RQ-KMeans [106, 41]. As shown in Table 7.3, Latte consistently outperforms the base model PSID across all tokenization methods and datasets. This result demonstrates

Table 7.3. Performance comparison across different tokenization methods (NDCG@10). Δ denotes Latte’s relative improvement over the base model PSID. All improvements are statistically significant ($p < 0.05$) according to the paired t-test.

Tokenization	Model	Instruments	Scientific	Games
OPQ	PSID	0.0313	0.0232	0.0478
	Latte	0.0318	0.0235	0.0493
	Δ	+1.68%	+1.37%	+3.24%
RQ-VAE	PSID	0.0325	0.0241	0.0490
	Latte	0.0331	0.0247	0.0502
	Δ	+2.02%	+2.76%	+2.55%
RQ-KMeans	PSID	0.0325	0.0235	0.0500
	Latte	0.0331	0.0249	0.0515
	Δ	+2.02%	+5.90%	+3.11%

Table 7.4. Performance comparison of different aggregation methods (NDCG@10).

Aggregation Method	Instruments	Scientific	Games
PSID	0.032462	0.023531	0.049994
Latte (Agg = sum)	0.033134	<u>0.024301</u>	<u>0.051476</u>
Latte (Agg = max)	<u>0.033114</u>	0.024920	0.051547

the broad applicability of our proposed method. Interestingly, while the base model performs best when paired with RQ-VAE, the introduction of latent tokens allows RQ-KMeans to achieve superior performance. We hypothesize that tokenizers not heavily optimized for the target corpus, such as RQ-KMeans, which lacks the end-to-end training of RQ-VAE, may offer greater optimization headroom.

Performance w.r.t. Inference Aggregation Method

In Section 7.4.2, we introduced two aggregation methods for combining the scores of generations associated with different latent tokens that point to the same target item: sum and max. We compare their performance in Table 7.4. We observe that both aggregation methods outperform the base model, while the performance difference between the two is relatively small. These results demonstrate that both strategies are effective and that the choice of aggregation method is robust.

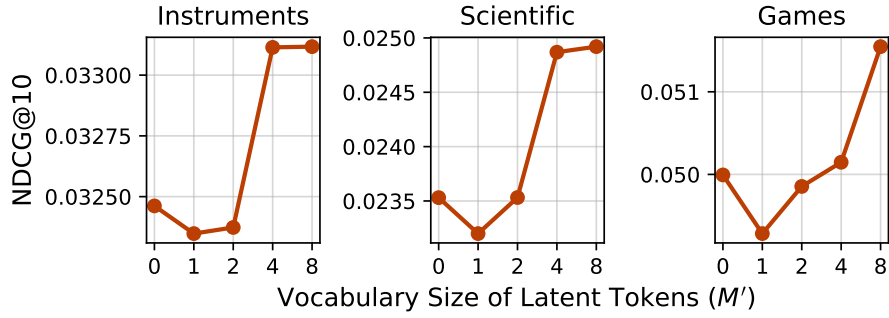


Figure 7.4. Analysis of model performance w.r.t. the number of latent tokens M' .

Performance w.r.t. Latent Token Vocabulary Size

We investigate the impact of latent token vocabulary size on model performance. Specifically, we tune the number of introduced latent tokens $M' \in \{0, 1, 2, 4, 8\}$, where 0 denotes the base model PSID. As shown in Figure 7.4, we observe that introducing only one or two latent tokens usually leads to a performance drop due to accumulated errors in latent token prediction. However, when the vocabulary size increases to 4 and 8, the models outperform the base model. Overall, these results suggest that a moderate number of latent tokens can effectively improve model performance.

7.6 Incorporating Inductive Bias into Latent Tokens

In Section 7.4, we proposed generating latent tokens prior to semantic IDs to enhance model expressiveness. While effective, these latent tokens are sampled uniformly at random during training. In this section, we explore the possibility of anchoring latent tokens to specific inductive biases.

7.6.1 Latent Tokens as SID Permutation Indicators

We consider a multimodal setting where each token corresponds to a specific modality of item features. Existing works typically adopt a fixed modality order to organize an item’s semantic IDs [48, 303, 269, 274], largely based on human heuristics. However, user preferences

Table 7.5. Performance comparison on MPD dataset with different modality orders.

Method	Recall@10	NDCG@10
<i>Base: Fixed Modality Order</i>		
playlist $\rightarrow$ tag $\rightarrow$ metadata	0.1966	0.1266
playlist $\rightarrow$ metadata $\rightarrow$ tag	0.1972	0.1268
tag $\rightarrow$ playlist $\rightarrow$ metadata	0.1948	0.1255
tag $\rightarrow$ metadata $\rightarrow$ playlist	0.1939	0.1250
metadata $\rightarrow$ playlist $\rightarrow$ tag	0.1933	0.1247
metadata $\rightarrow$ tag $\rightarrow$ playlist	0.1942	0.1248
<i>Ours: Dynamic Modality Ordering</i>		
Agg = max	<u>0.2060</u>	<u>0.1335</u>
Agg = sum	0.2107	0.1353

over different modalities may vary. In addition, identifying a globally optimal modality ordering is non-trivial. While a straightforward approach would involve enumerating all possible permutations and training separate models, such a strategy is computationally exhaustive. We are therefore interested in whether latent tokens can enable the model to adaptively select modality orders for different users, or alternatively, whether this mechanism can facilitate the discovery of globally best orderings in a purely data-driven manner.

Specifically, we bind each latent token to a specific permutation of the SID sequence (representing a unique ordering of modalities). During training, we continue to sample latent tokens uniformly; however, each token now dictates a specific permutation of the SIDs. We permute the SIDs accordingly before concatenating them with the latent token. During inference, the model generates latent tokens autonomously, followed by the SIDs. The only modification is that the generated SIDs are de-permuted back to their original order based on the preceding latent token.

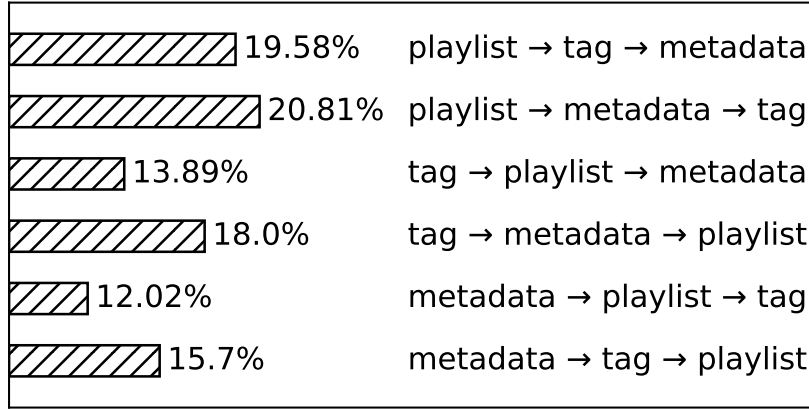


Figure 7.5. Distribution of modality orders corresponding to the generated permutation tokens.

7.6.2 Experiments and Analysis

Dataset.

We evaluate our approach on the Million Playlist Dataset (MPD) [19]. Following prior work [111], we filter out songs lacking tags or metadata and exclude playlists with fewer than six songs. The dataset is split into training, validation, and test sets using an 8:1:1 ratio.

Data processing.

We incorporate three modalities: *playlist*, *tag*, and *metadata*. For the playlist modality, we leverage collaborative information from song co-occurrences by training a Word2Vec-style [151, 48] embedding model on the training set. For the remaining modalities, *tags* represent categorical features (*e.g.*, genre), while *metadata* comprises text-rich descriptions. For simplicity, we follow Doh et al. (2025) [49] and extract embeddings using a pretrained text encoder [255]. Each embedding is clustered into 1,024 groups via *k*-means clustering, where the centroid indices serve as the semantic ID tokens. Since there are three modalities, we maintain a latent token vocabulary of size $3! = 6$.

Results.

We compare our latent token-based method against six baseline models that utilize fixed modality orders. As shown in Table 7.5, our method consistently outperforms all baselines,

Table 7.6. Notations and explanations.

Notation	Explanation
i, i_t	item, item at time t
$(i_1, i_2, \dots, i_{t-1})$	historical interaction sequence
u	user’s historical interactions represented as semantic IDs
$c^{(j)}$	the j -th token in a semantic ID
$(c^{(1)}, c^{(2)}, \dots, c^{(m)})$	semantic ID, a sequence of discrete tokens indexing an item
$c_t^{(1:j-1)}$	Shorthand for the token sequence $(c_t^{(1)}, c_t^{(2)}, \dots, c_t^{(j-1)})$
m	the length (depth) of semantic IDs
$\mathcal{C}^{(j)}$	vocabulary for the j -th position in semantic IDs
M	size of the token vocabulary $\mathcal{C}^{(j)}$
$\mathbb{P}(i_t u)$	probability of item i_t given user history u
$\mathbb{P}(c_t^{(j)} c_t^{(1)}, \dots, c_t^{(j-1)}, u)$	conditional probability of generating the j -th token
$\mathcal{T}$	decoding tree induced by valid semantic IDs
$d_{\mathcal{T}}(i, i')$	tree distance between item i and i' in the decoding tree $\mathcal{T}$
$\rho(i, i')$	Pearson correlation coefficient between generation probabilities of i and i'
σ^2	Variance of item generation probability $\mathbb{P}(i u)$ across users
$\mathcal{R}(i, i')$	Rank reversal event between items i and i' (see Section 7.3.3)
μ	Expected difference in generation probabilities $ \mathbb{E}[\mathbb{P}(i u) - \mathbb{P}(i' u)] $
M'	Number of latent tokens in Latte

demonstrating the effectiveness of introducing modality-order flexibility via latent tokens. In Figure 7.5, we observe that the model generates a non-uniform distribution of latent tokens. Notably, the two most frequent permutations (both of which prioritize the playlist modality) correspond to the best-performing fixed-order baselines in Table 7.5. This suggests that latent tokens with inductive bias can automatically identify more effective modality sequences. Overall, these results validate the potential of anchoring latent tokens to specific inductive biases. While the permutation-based approach discussed here is a straightforward example, we believe many other grounding strategies merit future exploration.

7.7 Notation

We summarize the notations used throughout the chapter in Table 7.6 for easy reference.

Table 7.7. Statistics of the datasets. “Avg. t ” denotes the average length of user interaction sequences.

Dataset	#Users	#Items	#Interactions	Avg. t
Instruments	57,439	24,587	511,836	8.91
Scientific	50,985	25,848	412,947	8.10
Games	94,762	25,612	814,586	8.60

7.8 Proofs for Section 7.3.3 (User-Item Preference Limitation)

In this section, we provide the detailed proofs for the claims made in Section 7.3.3 regarding the limitation of autoregressive SID generation in modeling diverse user preferences.

7.8.1 Formal Setup

Let u be a random user drawn from the user population $\mathcal{U}$. For a fixed item pair (i, i') , we consider the generation probabilities as random variables:

$$X \triangleq \mathbb{P}(i | u), \quad Y \triangleq \mathbb{P}(i' | u), \quad D \triangleq X - Y.$$

Let $\mu \triangleq \mathbb{E}[D]$ be the expected difference in generation probability (*e.g.*, reflecting global trend difference) and $\rho(i, i') \triangleq \text{Corr}(X, Y)$ be the Pearson correlation coefficient between the generation probabilities of i and i' (as illustrated in Figure 7.2).

For two independent users $u, u' \stackrel{\text{iid}}{\sim} \mathcal{U}$, we define the rank reversal event $\mathcal{R}(i, i')$ as the event where the two users have opposite relative preferences for items i and i' . Formally:

$$\mathcal{R}(i, i') \triangleq \left\{ \mathbb{P}(i | u) > \mathbb{P}(i' | u), \mathbb{P}(i | u') < \mathbb{P}(i' | u') \right\} \cup \left\{ \mathbb{P}(i | u) < \mathbb{P}(i' | u), \mathbb{P}(i | u') > \mathbb{P}(i' | u') \right\}.$$

In terms of the difference variable D , let $D_u = \mathbb{P}(i | u) - \mathbb{P}(i' | u)$ and $D_{u'} = \mathbb{P}(i | u') - \mathbb{P}(i' | u')$. Then $\mathcal{R}(i, i') = \{D_u > 0, D_{u'} < 0\} \cup \{D_u < 0, D_{u'} > 0\}$.

Assumption 7.8.1 (Comparable scale across users). For the fixed pair (i, i') , we assume the variance of generation probabilities is comparable: $\text{Var}(X) = \text{Var}(Y) = \sigma^2$ for some $\sigma^2 > 0$.

7.8.2 Proof of Rank Reversal Bound

Lemma 7.8.2 (Reversal probability decomposition). *Under the independence of u and u' ,*

$$\mathbb{P}(\mathcal{R}(i, i')) = 2\mathbb{P}(D > 0)\mathbb{P}(D < 0).$$

Proof. By definition and independence:

$$\begin{aligned}\mathbb{P}(\mathcal{R}(i, i')) &= \mathbb{P}(D_u > 0, D_{u'} < 0) + \mathbb{P}(D_u < 0, D_{u'} > 0) \\ &= \mathbb{P}(D_u > 0)\mathbb{P}(D_{u'} < 0) + \mathbb{P}(D_u < 0)\mathbb{P}(D_{u'} > 0).\end{aligned}$$

Since u and u' are i.i.d., D_u and $D_{u'}$ are identically distributed as D . Thus,

$$\mathbb{P}(\mathcal{R}(i, i')) = 2\mathbb{P}(D > 0)\mathbb{P}(D < 0).$$

□

Lemma 7.8.3 (Gap variance). *Under Assumption 7.8.1, the variance of the gap D is:*

$$\text{Var}(D) = 2\sigma^2(1 - \rho(i, i')).$$

Proof. Using $\text{Var}(X) = \text{Var}(Y) = \sigma^2$ and $\text{Cov}(X, Y) = \rho(i, i')\sigma^2$, we have:

$$\text{Var}(D) = \text{Var}(X - Y) = \text{Var}(X) + \text{Var}(Y) - 2\text{Cov}(X, Y) = 2\sigma^2 - 2\rho(i, i')\sigma^2 = 2\sigma^2(1 - \rho(i, i')).$$

□

Theorem 7.8.4 (High correlation implies few cross-user rank reversals). *Under Assumption 7.8.1, let $\mu = \mathbb{E}[D]$ and $\rho = \rho(i, i')$. Then:*

$$\mathbb{P}(\mathcal{R}(i, i')) \leq \frac{4\sigma^2(1-\rho)}{\mu^2 + 2\sigma^2(1-\rho)}.$$

Proof. From Lemma 7.8.2, $\mathbb{P}(\mathcal{R}) = 2\mathbb{P}(D > 0)\mathbb{P}(D < 0) \leq 2\min\{\mathbb{P}(D > 0), \mathbb{P}(D < 0)\}$. Without loss of generality, assume $\mu \geq 0$. Then $\mathbb{P}(D < 0)$ is the tail probability.

$$\mathbb{P}(\mathcal{R}(i, i')) \leq 2\mathbb{P}(D \leq 0) = 2\mathbb{P}(D - \mu \leq -\mu).$$

By Cantelli's inequality [13]:

$$\mathbb{P}(D - \mu \leq -\mu) \leq \frac{\text{Var}(D)}{\text{Var}(D) + \mu^2}.$$

Calculating the bound:

$$\mathbb{P}(\mathcal{R}(i, i')) \leq \frac{2\text{Var}(D)}{\text{Var}(D) + \mu^2}.$$

Substituting $\text{Var}(D) = 2\sigma^2(1-\rho)$ from Lemma 7.8.3:

$$\mathbb{P}(\mathcal{R}(i, i')) \leq \frac{4\sigma^2(1-\rho)}{\mu^2 + 2\sigma^2(1-\rho)}.$$

□

7.9 Proofs for Section 7.3.4 (Item-Item Similarity Limitation)

In this section, we provide the theoretical justification for the limitation on item-item similarity modeling discussed in Section 7.3.4.

7.9.1 Tree Distance as an Ultrametric

Lemma 7.9.1 (Ultrametric inequality). *The tree distance $d_{\mathcal{T}}$ defined in Definition 7.3.2 satisfies the ultrametric inequality. For any three items $i_1, i_2, i_3 \in \mathcal{I}$:*

$$d_{\mathcal{T}}(i_1, i_3) \leq \max(d_{\mathcal{T}}(i_1, i_2), d_{\mathcal{T}}(i_2, i_3)). \quad (7.7)$$

Proof. Let v_1, v_2, v_3 be the leaf nodes corresponding to items i_1, i_2, i_3 in the decoding tree $\mathcal{T}$. The tree distance $d_{\mathcal{T}}(i, i')$ is determined by the depth of the lowest common ancestor (LCA) of the two leaves. specifically, if the tree has depth m , then $d_{\mathcal{T}}(i, i') = 2(m - \text{depth}(\text{LCA}(i, i')))$.

Consider the computed LCAs for the pairs (i_1, i_2) and (i_2, i_3) . In any tree structure, the path intersection property dictates that:

$$\text{depth}(\text{LCA}(i_1, i_3)) \geq \min(\text{depth}(\text{LCA}(i_1, i_2)), \text{depth}(\text{LCA}(i_2, i_3))).$$

Substituting the distance definition into this inequality:

$$\begin{aligned} 2(m - \frac{d_{\mathcal{T}}(i_1, i_3)}{2}) &\geq \min\left(2(m - \frac{d_{\mathcal{T}}(i_1, i_2)}{2}), 2(m - \frac{d_{\mathcal{T}}(i_2, i_3)}{2})\right) \\ m - \frac{d_{\mathcal{T}}(i_1, i_3)}{2} &\geq m - \max\left(\frac{d_{\mathcal{T}}(i_1, i_2)}{2}, \frac{d_{\mathcal{T}}(i_2, i_3)}{2}\right) \\ \frac{d_{\mathcal{T}}(i_1, i_3)}{2} &\leq \max\left(\frac{d_{\mathcal{T}}(i_1, i_2)}{2}, \frac{d_{\mathcal{T}}(i_2, i_3)}{2}\right). \end{aligned}$$

Multiply by 2, we obtain:

$$d_{\mathcal{T}}(i_1, i_3) \leq \max(d_{\mathcal{T}}(i_1, i_2), d_{\mathcal{T}}(i_2, i_3)).$$

□

7.9.2 Correlation and Item Similarity

Before proving the main theorem, we establish the connection between the Pearson correlation coefficient $\rho(i, i')$ and item-item similarity in collaborative filtering.

In collaborative filtering [182, 184], item-item similarity is commonly modeled as the inner product of item representations derived from the user-item preference matrix. Let $\mathbf{P} \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{U}|}$ denote the matrix where entry $P_{i,u} = \mathbb{P}(i | u)$ represents the generation probability of item i for user u . Each row $\mathbf{P}_{i,:}$ captures how different users respond to item i .

The Pearson correlation coefficient between items i and i' is defined as:

$$\rho(i, i') = \text{Corr}(\mathbf{P}_{i,:}, \mathbf{P}_{i',:}) = \frac{\mathbb{E}_u[(P_{i,u} - \bar{P}_i)(P_{i',u} - \bar{P}_{i'})]}{\sigma_i \sigma_{i'}},$$

where $\bar{P}_i = \mathbb{E}_u[P_{i,u}]$ is the mean generation probability and σ_i is the standard deviation. This correlation coefficient is mathematically equivalent to the inner product of the normalized preference vectors:

$$\rho(i, i') = \left\langle \frac{\mathbf{P}_{i,:} - \bar{P}_i}{\sigma_i}, \frac{\mathbf{P}_{i',:} - \bar{P}_{i'}}{\sigma_{i'}} \right\rangle.$$

This formulation reveals that high correlation $\rho(i, i') \approx 1$ indicates that items i and i' have similar user preference patterns, which is connected to item-item similarity in collaborative filtering. Conversely, low or negative correlation suggests dissimilar preference patterns. This connection allows us to interpret the structural constraints imposed by the tree distance (via Assumption 7.3.3) as constraints on the model's ability to capture flexible item-item similarities.

7.9.3 Proof of Limitation on Intransitive Similarity

Here we provide the proof for Theorem 7.3.5 in Section 7.3.4, building on the established relationship between correlation and item similarity.

Proof of Theorem 7.3.5. By Assumption 7.3.3, the correlation $\rho(i, i')$ is monotonically related to the tree distance $d_{\mathcal{T}}(i, i')$. Assume the model successfully captures the similarities for the first

Table 7.8. Best hyperparameters for the base model PSID across three datasets.

Hyperparameter	Instruments	Scientific	Games
Beam size	500	500	500
Learning rate	3×10^{-3}	1×10^{-3}	3×10^{-3}

Table 7.9. Best hyperparameters for our model Latte across three datasets.

Hyperparameter	Instruments	Scientific	Games
Number of latent tokens M'	4	8	8
Beam size	500	500	500
Learning rate	3×10^{-3}	3×10^{-3}	3×10^{-3}

two pairs:

$$1. i_1 \sim i_2 \implies \rho(i_1, i_2) > \tau \implies d_{\mathcal{T}}(i_1, i_2) \leq \delta.$$

$$2. i_2 \sim i_3 \implies \rho(i_2, i_3) > \tau \implies d_{\mathcal{T}}(i_2, i_3) \leq \delta.$$

Using the ultrametric property from Lemma 7.9.1:

$$d_{\mathcal{T}}(i_1, i_3) \leq \max(d_{\mathcal{T}}(i_1, i_2), d_{\mathcal{T}}(i_2, i_3)) \leq \max(\delta, \delta) = \delta.$$

Since $d_{\mathcal{T}}(i_1, i_3) \leq \delta$, by the bidirectional relationship between correlation and tree distance, we have:

$$\rho(i_1, i_3) > \tau.$$

This implies i_1 and i_3 must exhibit a high level of similarity (correlation $> \tau$), making it impossible for the model to treat them as dissimilar (which would require $\rho \leq \tau$). Thus, the tree structure imposes a transitivity constraint on the learned similarities. $\square$

7.9.4 Expressiveness of Latte

We now show that Latte relaxes the rank-reversal constraint in Theorem 7.3.4. Recall that for a pair of items (i, i') , the rank-reversal probability in standard GR is bounded by

$$\mathbb{P}(\mathcal{R}(i, i')) \leq B(\rho) \triangleq \frac{4\sigma^2(1-\rho)}{\mu^2 + 2\sigma^2(1-\rho)}, \quad (7.8)$$

where $\rho = \rho(i, i')$ denotes the correlation between the generation probabilities of i and i' across users. As discussed in Section 7.3.2, structurally close items in standard GR tend to have large ρ , which makes $B(\rho)$ small and suppresses cross-user rank reversals.

Proposition 7.9.2 (Latte relaxes the rank-reversal constraint). *Consider a pair of structurally close items (i, i') with correlation ρ in standard GR. Suppose Latte uses M' latent tokens and the dominant latent token for each item is approximately uniformly distributed, i.e.,*

$$\mathbb{P}(\ell^*(i, u) = \ell^*(i', u)) = \frac{1}{M'}.$$

Let ρ_{low} denote the correlation between two items whose generation paths diverge at the latent-token level, with $\rho_{\text{low}} < \rho$. Then the effective correlation between i and i' under Latte satisfies

$$\rho_{\text{Latte}} \approx \frac{1}{M'}\rho + \left(1 - \frac{1}{M'}\right)\rho_{\text{low}} < \rho, \quad (7.9)$$

for any $M' > 1$. Consequently, when $\mu > 0$, Latte loosens the rank-reversal bound:

$$B(\rho_{\text{Latte}}) > B(\rho). \quad (7.10)$$

Proof. In standard GR, the tree distance $d_{\mathcal{T}}(i, i')$ between two items is fixed. Thus, for structurally close items, their generation probabilities share a long prefix in the decoding tree, inducing a high correlation ρ across users.

In Latte, the effective generation path also depends on the latent token selected for each item and user. Let

$$\ell^*(i, u) = \arg \max_{\ell} \mathbb{P}(\ell | u) \mathbb{P}(i | \ell, u)$$

denote the dominant latent token for generating item i given user u . For the pair (i, i') , there are two cases.

First, if $\ell^*(i, u) = \ell^*(i', u)$, the two items are generated under the same latent path. In this case, their effective tree distance remains the original distance $d_{\mathcal{T}}(i, i')$, and their correlation remains close to the standard GR correlation ρ .

Second, if $\ell^*(i, u) \neq \ell^*(i', u)$, the two generation paths diverge immediately after the root through different latent tokens. Their effective distance becomes

$$d_{\text{eff}}(i, i'; u) = 2(m + 1),$$

which is the maximum distance in the latent-augmented decoding tree. By Theorem 7.3.3, larger tree distance corresponds to lower correlation in generation probabilities. We denote this lower correlation by ρ_{low} , where $\rho_{\text{low}} < \rho$.

Since the latent tokens are approximately uniformly assigned, the probability that two items use the same dominant latent token is $1/M'$, while the probability that they use different dominant latent tokens is $1 - 1/M'$. Therefore, the expected effective correlation under Latte can be written as

$$\rho_{\text{Latte}} \approx \frac{1}{M'} \rho + \left(1 - \frac{1}{M'}\right) \rho_{\text{low}}.$$

Because $M' > 1$ and $\rho_{\text{low}} < \rho$, we have

$$\begin{aligned} \rho_{\text{Latte}} - \rho &= \frac{1}{M'} \rho + \left(1 - \frac{1}{M'}\right) \rho_{\text{low}} - \rho \\ &= \left(1 - \frac{1}{M'}\right) (\rho_{\text{low}} - \rho) < 0. \end{aligned} \tag{7.11}$$

Thus, $\rho_{\text{Latte}} < \rho$.

It remains to show that this lower correlation gives a looser rank-reversal bound. Taking the derivative of $B(x)$ in Equation (7.8), we obtain

$$\begin{aligned} B'(x) &= \frac{\partial}{\partial x} \frac{4\sigma^2(1-x)}{\mu^2 + 2\sigma^2(1-x)} \\ &= -\frac{4\sigma^2\mu^2}{(\mu^2 + 2\sigma^2(1-x))^2}. \end{aligned} \tag{7.12}$$

When $\mu > 0$, we have $B'(x) < 0$, so $B(x)$ is strictly decreasing in x . Since $\rho_{\text{Latte}} < \rho$, it follows that

$$B(\rho_{\text{Latte}}) > B(\rho).$$

Therefore, Latte relaxes the rank-reversal constraint imposed by high structural correlation. Intuitively, by allowing structurally close items to take different latent paths, Latte reduces their effective correlation and gives the model more flexibility to express user-specific rank reversals. $\square$

Uniform latent-token generation.

In the above analysis, we assume that the latent tokens are approximately uniformly selected during generation. This assumption is motivated by our training design, where latent tokens are uniformly sampled and prepended before the semantic IDs. Under this assumption, when the latent vocabulary size is M' , each latent token is selected with probability approximately $1/M'$. Therefore, for two items i and i' , the probability that they share the same dominant latent path is approximately $1/M'$, while the probability that they diverge at the latent-token level is approximately $1 - 1/M'$. For the Games dataset with $M' = 8$, the expected generation probability of each latent token is thus $1/8 = 0.125$, as shown in Table 7.10.

Table 7.10. Latent-token generation probabilities on the Games dataset under the uniform latent-token generation assumption.

Latent Token	Generation Probability
ℓ_1	0.1244
ℓ_2	0.1253
ℓ_3	0.1248
ℓ_4	0.1243
ℓ_5	0.1259
ℓ_6	0.1254
ℓ_7	0.1251
ℓ_8	0.1248

7.10 Implementation Details

Compared models.

We compare Latte with the following baselines: (1) sequential recommendation models that primarily based on item IDs (and feature IDs), including GRU4Rec [82], BERT4Rec [197], SASRec [108], FMLP-Rec [298], HSTU [271], FDSA [280], and S³-Rec [297]; and (2) generative recommendation models based on semantic IDs, including TIGER [174], LETTER [220], ActionPiece [91], and PSID [278].

We adhere to the exact same experimental settings and directly adopt the reported results for most baselines from prior works [291, 296]. For the base model, PSID [278], we refer to the officially released code² to implement the model.

Evaluation.

We evaluate all models using the widely adopted Recall@K (R@K) and NDCG@K (N@K) metrics, with $K \in \{5, 10\}$, following prior works [174, 291]. We tune hyperparameters based on NDCG@10 performance on the validation set and select the best-performing checkpoint for testing.

Latte. We build Latte upon our base model, PSID [278]. Specifically, we adopt the architecture of the representative model TIGER [174], maintaining the same configuration: a T5-style

²<https://github.com/wangshanyw/PurelySemanticIndexing>

Table 7.11. Statistics of structurally coupled items under PSID with RQ K-Means tokenization. We report the average number of items within tree distance 2 and 4 from each target item, as well as the ratio of test instances whose target item has at least one item at tree distance 2.

Dataset	#Items at Dist. 2	#Items at Dist. 4	Ratio with Dist.-2 Items
Instruments	6.29	142.02	83.98%
Scientific	7.39	153.97	85.25%
Games	6.98	179.41	84.17%

encoder-decoder, 4 stacked Transformer blocks in both the encoder and decoder, and an embedding dimension of 128. Regarding tokenization, we follow the recommendations of Ju et al. (2025) [106] to primarily use RQ K-Means, though we also experiment with other tokenizers such as RQ-VAE and OPQ (see Table 7.3). To prevent semantic ID conflicts, we employ the ESM algorithm as proposed in the PSID paper [278]. The resulting semantic ID length is $m = 3$ with a vocabulary size of $M = 256$ for each token position. The number of introduced latent tokens is tuned from the set $\{2, 4, 8\}$ based on validation performance. For the Latte results reported in Table 7.1, we use max as the aggregation method.

Training and inference. We tune the learning rate for both the base model PSID and our model, Latte, from the set $\{1 \times 10^{-3}, 3 \times 10^{-3}\}$. We use a weight decay of 0.05. Models are trained for up to 150 epochs, employing early stopping with a patience of 20 epochs. Hyperparameters are selected based on the best NDCG@10 performance on the validation set, and the best-performing checkpoint is used for testing. During inference, we tune the beam size from $\{50, 100, 500\}$, also based on validation performance. All experiments were conducted on a single NVIDIA A6000 GPU.

Best hyperparameters for both PSID and Latte across the three datasets are summarized in Tables 7.8 and 7.9.

7.11 Discussion

Why should semantically similar items not always exhibit correlated generation probabilities?

We emphasize that semantic similarity and correlated generation probabilities are related but distinct. Assigning similar semantic IDs to semantically similar items is a common design choice in current GR models, as it allows related items to share token prefixes. However, this design choice becomes limiting when shared prefixes force semantically similar items to have highly correlated generation probabilities across users. In practice, users may have opposite preferences toward semantically similar items. For example, in the Games dataset, “Pokémon Scarlet (Switch)” and “Pokémon Sun (3DS)” have a tree distance of 2, meaning that only their last semantic ID token differs. While some users may prefer the newer game and favor Scarlet, others may prefer the older style and favor Sun. Standard GR models struggle to represent such ranking reversals, as discussed in Section 7.3.3, and tend to assign one item a consistently higher probability than the other for most users. This behavior conflicts with the personalized nature of recommendation, where even semantically similar items should be distinguishable based on user-specific preferences.

How significant is the identified issue? To assess how frequently the identified structural coupling occurs in practice, we analyze the base PSID model with RQ K-Means tokenization, where each item is represented by three tokens. For each target item in the test set, we count the average number of other items within tree distance 2 and 4. As shown in Table 7.11, each target item is associated with around 6–7 closely coupled items at tree distance 2 and 140–180 items at tree distance 4. We further compute the ratio of test instances whose target item has at least one other item at tree distance 2. The ratio is consistently above 80% across all datasets, indicating that the coupling effect is not a rare corner case, but a pervasive structural issue in autoregressive SID generation.

Is the latent token design necessary or optimal?

We do not claim that the latent token design in Latte is necessary or optimal. Rather, our goal is to analyze the expressive limitations of current GR models and show that these limitations can be alleviated by relaxing the rigid decoding tree structure. Latte is one simple instantiation of this idea: by introducing an additional latent token before the semantic ID tokens, it provides multiple latent-conditioned generation paths and therefore reduces the structural coupling induced by a single fixed tree. We view Latte as an initial attempt rather than a final solution. We hope this analysis can motivate future work on more expressive and efficient designs for semantic ID-based generative recommendation.

When does Latte degrade to the base model? Latte alleviates prefix coupling by introducing multiple latent-conditioned generation paths. Although two items remain coupled when they are generated under the same latent token, different latent tokens do not simply reweight the same shared prefix probability. For example, given two latent tokens l_A and l_B , the probabilities of generating the same semantic prefix are

$$P(l_A | x)P(\text{prefix} | x, l_A),$$
$$P(l_B | x)P(\text{prefix} | x, l_B).$$

Even ignoring the path weights $P(l_A | x)$ and $P(l_B | x)$, the conditional prefix probabilities can differ because they are conditioned on different latent tokens, i.e., $P(\text{prefix} | x, l_A) \neq P(\text{prefix} | x, l_B)$ in general. Thus, Latte provides additional flexibility beyond assigning different weights to the same prefix. It would reduce to the base model **only in the extreme case** where different latent tokens induce identical representations or identical conditional generation distributions.

Does Latte introduce significant inference overhead? Latte does not require $|\mathcal{L}|$ independent beam searches over latent tokens. Instead, it uses a single beam search process, where the model first predicts a latent token and then continues generating the semantic ID tokens. Therefore,

Table 7.12. Inference time for one epoch on the Instruments dataset.

Model	Time
TIGER	98s
PSID	66s
Latte ($M' = 4$)	76s
Latte ($M' = 8$)	76s
Latte ($M' = 16$)	78s

Table 7.13. Comparison with noise-based data augmentation baselines. We report NDCG@10 results.

Model	Instruments	Scientific	Games
PSID	0.0325	0.0235	0.0500
Dropout	0.0322	0.0239	0.0499
Swap	0.0300	0.0208	0.0462
Latte	0.0331	0.0249	0.0515

introducing latent tokens only adds one extra decoding step, rather than multiplying the inference cost by the number of latent tokens. In practice, the aggregation is performed over the candidates retained by beam search, rather than over all possible latent tokens. To quantify this overhead, we compare the inference time for one epoch on the Instruments dataset. TIGER uses 4 semantic ID tokens per item, PSID uses 3 semantic ID tokens per item, and Latte uses 1 latent token followed by 3 semantic ID tokens. As shown in Table 7.12, Latte moderately increases the inference cost compared with its base model PSID, but remains more efficient than TIGER.

Does Latte mainly improve performance by introducing noise?

To examine whether Latte’s improvement mainly comes from a noise-like regularization effect, we compare it with two data augmentation baselines: Dropout [171] and Swap [250]. During each training epoch, these baselines randomly drop or swap a certain ratio of tokens in the input sequence, with the ratio tuned from $\{0.1, 0.2, 0.3\}$. As shown in Table 7.13, these augmentation methods can sometimes improve over the base PSID model, but they remain consistently worse than Latte. We further compute Kendall’s rank correlation between tree distance and item-item similarity, following the analysis in Table 7.2. As shown in Table 7.14, only Latte

Table 7.14. Kendall’s rank correlation between tree distance and item-item similarity for noise-based augmentation baselines. Less negative values indicate weaker structural coupling.

Model	Instruments	Scientific	Games
PSID	-0.6225	-0.4611	-0.6072
Dropout	-0.6248	-0.4687	-0.6025
Swap	-0.6277	-0.4887	-0.6096
Latte	-0.6030	-0.4451	-0.5958

substantially improves the correlation, indicating that its gains do not merely come from injecting noise, but from alleviating the structural probability coupling discussed in Section 7.3.

7.12 Limitations

Latte is proposed as a simple example to alleviate the expressive limitations discussed in this chapter, rather than as an optimal solution. It still has several known limitations. First, Latte introduces additional inference cost, since it requires one extra decoding step to generate the latent token before the original semantic ID tokens. Although this overhead is moderate in our experiments, it may still matter in latency-sensitive recommendation systems. Second, Latte may degenerate to the base model in extreme cases where different learned latent token embeddings become nearly identical, causing different latent-conditioned paths to induce similar generation distributions. Third, the latent token design increases the flexibility of the decoding process but does not completely remove the structural constraints imposed by semantic ID generation. Items generated under the same latent-conditioned path may still exhibit prefix coupling. Despite these limitations, they do not affect our main findings on the expressive limitations of current GR models, including rank reversal (Section 7.3.3) and forced transitivity (Section 7.3.4). We hope these findings motivate future work on more expressive and efficient designs for semantic ID-based generative recommendation.

7.13 Conclusion

In this chapter, we identify inherent expressiveness limitations in generative recommendation models that arise from their unique autoregressive decoding process. We empirically demonstrate that GR models tend to assign similar probabilities to items that are close in the decoding tree induced by semantic ID tokens. We further provide theoretical evidence that such structural correlations prevent GR models from capturing simple user-item preference (rank reversals, see Section 7.3.3) and item-item similarity (forced transitivity, see Section 7.3.4) patterns. To mitigate this issue, we propose a simple yet effective modification to the standard GR framework. By generating latent tokens before the semantic IDs, the decoding tree is reshaped to allow multiple paths with varying tree distances between any pair of items, thereby relaxing the structural constraints imposed by the original fixed decoding tree. Extensive experiments on public datasets validate the effectiveness of our proposed method, leading to an average of 3.45% relative improvement of NDCG@10. We further investigate binding latent tokens with inductive biases. In particular, order-indicating latent tokens help the model identify effective modality orderings, resulting in improved overall performance.

Chapter 7, in part, is a reprint of the material as it appears in “Expressiveness Limits of Autoregressive Semantic ID Generation in Generative Recommendation.” by Yupeng Hou, Haven Kim, Clark Mingxuan Ju, Eduardo Escoto, Neil Shah, and Julian McAuley, referenced as [94]. The dissertation author was the primary investigator and author of this manuscript.

Chapter 8

On the Memorization and Generalization of Generative Recommendation

As the final technical chapter, this chapter steps back from individual architectures to ask what benefits semantic ID-based generative recommendation provides against conventional item ID-based models. We aim to answer the question through the lens of generalization and memorization.

A widely held hypothesis for why generative recommendation (GR) models outperform conventional item ID-based models is that they generalize better. However, there is few systematic way to verify this hypothesis beyond a superficial comparison of overall performance. To address this gap, we categorize each data instance based on the specific capability required for a correct prediction: either memorization (reusing item transition patterns observed during training) or generalization (composing known patterns to predict unseen item transitions). Extensive experiments show that GR models perform better on instances that require generalization, whereas item ID-based models perform better when memorization is more important. To explain this divergence, we shift the analysis from the item level to the token level and show that what appears to be item-level generalization often reduces to token-level memorization for GR models. Finally, we show that the two paradigms are complementary. We propose a simple memorization-aware indicator that adaptively combines them on a per-instance basis, leading to improved

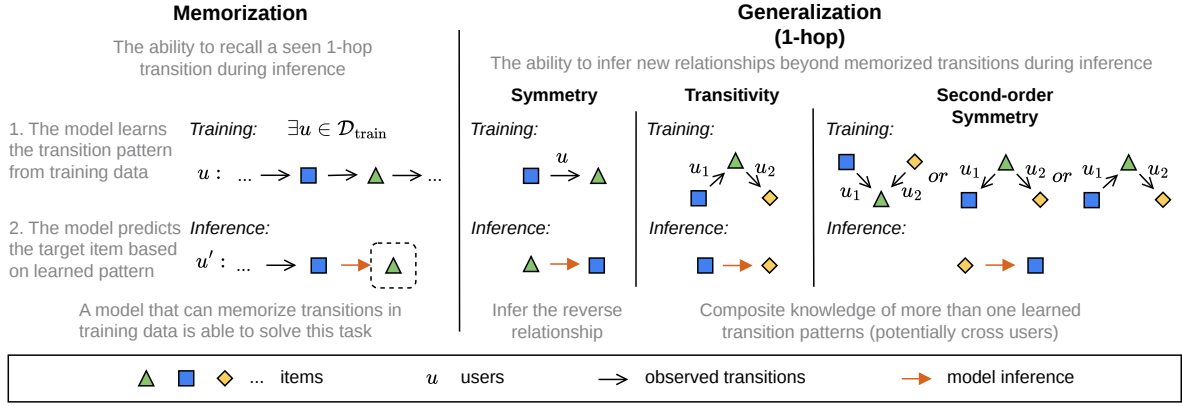


Figure 8.1. Illustrated definitions for memorization vs. generalization. We define memorization and different sub-categories of generalization based on (1) the transition patterns observed in training data, and (2) the patterns required to infer.

overall recommendation performance.¹

8.1 Introduction

Generative recommendation (GR) [174, 290, 41, 76] has recently emerged as a promising paradigm for sequential recommendation. Compared with conventional models such as SASRec [108], a key difference is that GR models tokenize each item as a sequence of sub-item tokens (*e.g.*, semantic IDs [204, 174]) rather than a single unique item ID. However, the advantage of GR models has typically been observed in terms of overall performance, that is, GR models correctly predict more data instances than conventional methods [174, 41]. This naturally raises the question of which types of data instances are better handled by generative recommendation models.

We hypothesize that each data instance requires different levels of generalization and memorization for correct prediction, leading to the performance discrepancies observed between GR and item ID-based models. To investigate this, we propose an analytical framework that categorizes each data instance by the primary model capability it requires (either memorization or generalization) based on the underlying data patterns. We then analyze model performance on

¹Our code is available at <https://github.com/Jamesding000/MemGen-GR>.

each category separately. Nevertheless, conducting such analyses requires two key components: identifying the data patterns of interest in the context of sequential recommendation, and designing reasonable methods to categorize instances.

(1) Data patterns. Since the task is framed as predicting the next item from a user’s history, a natural starting point is to focus on the target items. While prior work often studies cold-start items (*i.e.*, items that are rare or unseen during training) as out-of-distribution cases that require generalization [193, 257, 46], this target-centric view ignores the interaction between the history and the target. Even when a target item is popular, the transition from the given history to that item may be rare in the training data. Predicting such transitions can therefore still require generalization.

(2) Categorization. We require a principled method to determine whether a given instance primarily relies on memorization or generalization. Prior studies, such as those based on counterfactual memorization [273, 68, 175, 64], are usually computationally expensive, as they require frequent model retraining on datasets that exclude specific data points. This makes them impractical for recommendation settings with large-scale user interaction logs [41, 271]. Another line of work categorizes instances by measuring representation similarities between training instances and the predictions [98, 167, 170]. However, these methods are mainly adopted in tasks without a clear ground truth, such as language modeling. In contrast, recommendation is typically evaluated with a well-defined ground-truth target item for each instance [108, 77].

Given these considerations, we treat *item transitions* (from a historical item to the target) rather than single target items as the data patterns of interest (Figure 8.1). To categorize data instances, we examine whether the item transitions required for the correct prediction have been observed in the training data (memorization), or if they can be composed or inferred from observed patterns (generalization). Using this categorization, we explicitly partition the test data into subsets reflecting different capabilities and evaluate model performance on each, thereby distinguishing the contributions of memorization and generalization to overall performance.

To this end, we benchmark two representative models for each paradigm: TIGER [174]

as the semantic ID-based GR model and SASRec [108] as the item ID-based conventional model. By evaluating performance on memorization and generalization subsets across seven real-world datasets, we find that GR models indeed excel on generalization-related subsets, while generally underperforming item ID-based models on memorization-related subsets. This observation motivates us to investigate the mechanism behind the generalization capability of GR models. We then shift our analysis from item transition patterns to sub-item token transition patterns. From this perspective, a substantial fraction of target item transitions that would be regarded as item-level generalization can instead be interpreted as token-level memorization. This effectively explains the source of the GR models’ generalization capability.

Finally, we show that these two paradigms are complementary. We introduce an adaptive ensembling method that combines a GR model with an item ID-based model. The ensemble assigns instance-specific weights based on whether each data instance primarily requires memorization or generalization, as predicted by an indicator. Experimental results show that this adaptive ensembling strategy consistently improves overall performance over both individual models and naive fixed-weight ensembles.

8.2 Defining Memorization and Generalization

In this section, we describe our proposed framework for analyzing memorization and generalization in sequential recommendation. We first outline the task definition and notation in Section 8.2.1. In what follows, we present our criteria for attributing data instances as memorization-based (Section 8.2.2) or generalization-based (Section 8.2.3), relying on the item transition patterns they contain. Subsequently, we extend the generalization criteria to encompass multi-hop generalization in Section 8.2.4. Finally, we discuss the remaining uncategorized instances in Section 8.2.5.

8.2.1 Task Definition

Sequential recommendation. A user is represented by a sequence of historical item interactions $u = [i_1, i_2, \dots, i_{t-1}]$, where $i \in \mathcal{I}$. The goal is to predict the next item i_t that the user will interact with. The recommendation models are trained on a set of user interaction sequences $\mathcal{D}_{\text{train}}$. For a data instance (u, i_t) not present in the training set, we attribute it to memorization or generalization based on its constituent item transition patterns.

Item transition. As discussed in Section 8.1, we treat item transitions as the fundamental data patterns for studying memorization and generalization. Specifically, we define an item transition as a directed pair of items $[i_s \rightarrow i_t]$, where $i_s, i_t \in u$ and $s < t$. We further define the *hop count* of the item transition based on the distance between i_s and i_t in the user’s history. For example, if $s = t - 2$, the hop count is 2. Our framework categorizes each data instance (u, i_t) based on the set of item transitions $\{[i_s \rightarrow i_t], i_s \in u\}$ it contains.

8.2.2 Memorization-Related Data Instance

We define a data instance as *memorization-related* if the 1-hop item transition $[i_{t-1} \rightarrow i_t]$ has been observed in the training data, regardless of which user’s history it appears in. Under this condition, it is possible for a model to correctly predict the target item only by memorizing the training data.

Def. (Memorization). A data instance (u, i_t) is *memorization-related* if:

$$(u, i_t) \in \mathcal{D}_{\text{mem}} \iff \exists u' \in \mathcal{D}_{\text{train}} \text{ s.t. } [i_{t-1} \rightarrow i_t] \subseteq u'.$$

8.2.3 Generalization-Related Data Instance

A data instance is defined as *generalization-related* if: (1) it is not memorization-related; and (2) it contains at least one item transition that can be inferred or composed from observed

Multi-hop Generalization

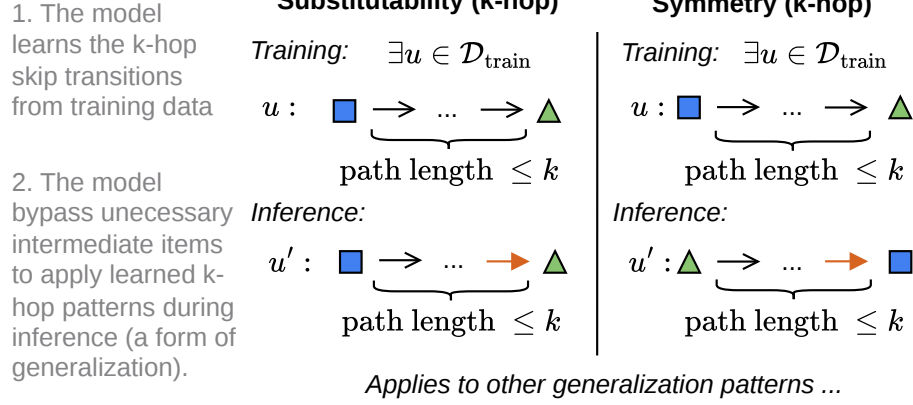


Figure 8.2. Illustration of multi-hop generalization. The same extension applies to the other generalization types.

transitions in the training data. We categorize generalization into multiple types based on specific inference or composition methods. Note that a single data instance may satisfy multiple generalization types.

For simplicity, we first focus on 1-hop item transitions $[i_s \rightarrow i_t]$ where $s = t - 1$, introducing three possible 1-hop generalization types: transitivity, symmetry, and 2nd-order symmetry.

Transitivity implies that the model can infer $[i_{t-1} \rightarrow i_t]$ by bridging two observed transitions via an intermediate item x .

Def. (Transitivity). A data instance $(u, i_t) \notin \mathcal{D}_{\text{mem}}$ satisfies *transitivity* if:

$$\exists x \in \mathcal{I}, \exists u', u'' \in \mathcal{D}_{\text{train}}, [i_{t-1} \rightarrow x] \subseteq u' \wedge [x \rightarrow i_t] \subseteq u''.$$

Symmetry allows the model to infer a transition $[i_{t-1} \rightarrow i_t]$ if its reverse $[i_t \rightarrow i_{t-1}]$ has been observed.

Def. (Symmetry). A data instance $(u, i_t) \notin \mathcal{D}_{\text{mem}}$ satisfies *symmetry* if:

$$\exists u' \in \mathcal{D}_{\text{train}}, [i_t \rightarrow i_{t-1}] \subseteq u'.$$

Table 8.1. Dataset statistics.

Dataset	Users	Items	Interactions	Sparsity	AvgLen
Sports	35,599	18,358	296,337	99.95%	8.32
Beauty	22,364	12,102	198,502	99.93%	8.88
Science	50,986	25,849	412,947	99.97%	8.10
Music	57,440	24,588	511,836	99.96%	8.91
Office	223,309	77,552	1,800,878	99.99%	8.06
Steam	47,762	10,972	599,620	99.89%	12.55
Yelp	30,432	20,034	316,354	99.95%	10.40

2nd-Order Symmetry encompasses complex symmetric relations where i_{t-1} and i_t are related via an intermediate item x in non-transitive ways.

Def. (2nd-Order Symmetry). A data instance $(u, i_t) \notin \mathcal{D}_{\text{mem}}$ satisfies *2nd-order symmetry* if $\exists x \in \mathcal{I}, \exists u', u'' \in \mathcal{D}_{\text{train}}$ such that one of the following holds:

- (1) $[x \rightarrow i_{t-1}] \subseteq u' \wedge [x \rightarrow i_t] \subseteq u''$ (common cause);
- (2) $[i_{t-1} \rightarrow x] \subseteq u' \wedge [i_t \rightarrow x] \subseteq u''$ (common effect);
- (3) $[i_t \rightarrow x] \subseteq u' \wedge [x \rightarrow i_{t-1}] \subseteq u''$ (reverse path).

8.2.4 Multi-Hop Generalization

Although Section 8.2.3 focuses on 1-hop item transitions for simplicity, our proposed criteria naturally extend to multi-hop transitions. Specifically, we define multi-hop generalization types (transitivity, symmetry, and 2nd-order symmetry) by applying the same logic to multi-hop item transitions (as defined in Section 8.2.1). If a data instance involves multiple item transitions with different hop counts that satisfy the generalization criteria, we use the minimum hop count for categorization.

Substitutability. Beyond the types introduced in Section 8.2.3, one might consider extending the definition of memorization to multi-hop item transitions. However, we argue that “multi-hop

memorization” is effectively a form of generalization. It requires the model to have strong generalization capabilities to bypass unnecessary intermediate items and select the appropriate multi-hop transition for prediction. Therefore, we define *substitutability* as a unique generalization type involving only multi-hop item transitions.

Def. (Substitutability). A data instance $(u, i_t) \notin \mathcal{D}_{\text{mem}}$ satisfies *substitutability* if:

$$\exists k \geq 2, \exists u' \in \mathcal{D}_{\text{train}} \text{ s.t. } [i_{t-k} \rightarrow \cdots \rightarrow i_t] \subseteq u'.$$

Table 8.2. Performance breakdown by memorization or generalization categories. We report NDCG@10 for each dataset. **Bold** indicates the best performing model. “Mem.” stands for “memorization”, and “UC” stands for “uncategorized”. Note that a single data instance may fall into multiple generalization categories, so the sum of the ratios of the generalization subsets can exceed that of “All”. In contrast, the proportions of memorization, generalization, and uncategorized instances sum to 100%.

Dataset	Model	Mem.	All	Generalization																UC
				Substitutability				Symmetry				Transitivity				2nd-Symmetry				
				2	3	4	1	2	3	4	1	2	3	4	1	2	3	4		
Sports (2014)	Ratio	5.1	84.7	5.4	4.7	4.2	1.4	3.3	3.6	3.4	15.1	26.1	18.0	11.1	25.1	32.9	15.8	7.9	10.2	
	SASRec	.2816	.0128	.0732	.0446	.0279	.0767	.0556	.0414	.0368	.0433	.0120	.0046	.0016	.0321	.0072	.0020	.0012	.0004	
	TIGER	.1656	.0179	.1059	.0650	.0415	.0963	.0762	.0650	.0591	.0499	.0215	.0086	.0023	.0405	.0140	.0015	.0013	.0003	
Beauty (2014)	Ratio	8.6	81.3	6.7	5.5	4.4	2.1	3.6	3.4	2.8	16.2	22.6	16.7	10.8	24.1	30.8	15.2	7.9	10.1	
	SASRec	.3793	.0134	.0814	.0395	.0235	.0773	.0535	.0510	.0391	.0461	.0105	.0044	.0026	.0342	.0076	.0018	.0008	.0001	
	TIGER	.2456	.0210	.1046	.0648	.0381	.1156	.1026	.0696	.0437	.0551	.0235	.0113	.0059	.0468	.0167	.0029	.0015	.0015	
Science (2023)	Ratio	4.0	84.2	4.6	4.1	3.5	2.0	3.7	3.5	2.9	16.5	24.8	17.2	10.8	26.3	30.7	15.4	8.3	11.8	
	SASRec	.2452	.0132	.0902	.0510	.0339	.1073	.0722	.0436	.0346	.0435	.0118	.0039	.0021	.0329	.0070	.0014	.0009	.0002	
	TIGER	.2348	.0177	.1114	.0688	.0447	.1033	.1040	.0619	.0459	.0512	.0189	.0070	.0037	.0400	.0123	.0030	.0011	.0003	
Music (2023)	Ratio	6.9	89.8	7.5	6.4	5.6	3.1	6.3	5.9	5.0	30.4	30.3	15.0	7.1	43.3	30.7	10.4	4.2	3.3	
	SASRec	.1973	.0173	.0818	.0513	.0319	.0920	.0678	.0502	.0294	.0394	.0095	.0033	.0013	.0314	.0057	.0011	.0011	.0000	
	TIGER	.2020	.0205	.0976	.0574	.0376	.0879	.0814	.0581	.0396	.0434	.0144	.0043	.0014	.0359	.0087	.0014	.0007	.0004	
Office (2023)	Ratio	4.4	86.3	4.5	4.1	3.6	1.9	3.7	3.4	2.9	22.4	27.4	16.4	9.5	33.1	30.1	13.5	6.8	9.3	
	SASRec	.2405	.0097	.0682	.0412	.0282	.0729	.0612	.0398	.0273	.0277	.0062	.0019	.0012	.0212	.0037	.0012	.0006	.0002	
	TIGER	.2719	.0154	.1092	.0667	.0433	.0914	.1025	.0649	.0470	.0384	.0136	.0040	.0020	.0311	.0089	.0017	.0007	.0001	
Steam	Ratio	37.4	61.5	18.7	11.9	7.1	7.5	12.7	9.3	6.0	51.0	8.0	1.6	0.5	54.9	5.2	0.9	0.4	1.1	
	SASRec	.3855	.0133	.0271	.0129	.0092	.0306	.0216	.0139	.0091	.0144	.0079	.0079	.0112	.0143	.0057	.0018	.0051	.0000	
	TIGER	.3885	.0157	.0366	.0149	.0073	.0305	.0331	.0169	.0109	.0175	.0079	.0038	.0000	.0168	.0077	.0007	.0000	.0000	
Yelp	Ratio	4.2	92.8	6.4	7.8	6.9	2.7	7.1	6.5	5.8	22.6	33.5	18.5	9.7	38.7	36.8	12.0	4.4	3.0	
	SASRec	.2487	.0233	.0858	.0714	.0435	.0724	.0855	.0484	.0300	.0436	.0215	.0133	.0139	.0439	.0120	.0016	.0010	.0007	
	TIGER	.1402	.0213	.0898	.0620	.0445	.0659	.0861	.0479	.0389	.0436	.0213	.0112	.0055	.0356	.0150	.0034	.0021	.0000	

8.2.5 Uncategorized Data Instance

Given a maximum hop count (*e.g.*, 4 in our experiments, see Section 8.3), any data instance that is neither memorization-related nor generalization-related is labeled as “uncategorized.” Such instances may involve items unseen during training, exhibit higher-order transition patterns, require capabilities beyond the scope of memorization and generalization, or be inherently unpredictable based on historical data alone. In our experiments, we also analyze model performance on these uncategorized instances.

8.3 Performance Breakdown: Item IDs vs. Semantic IDs

In this section, we present the empirical results for memorization and generalization capabilities for GR and item ID-based models.

8.3.1 Experiment Setup

Datasets. We conduct experiments on seven public datasets that are widely used in evaluating GR models [174, 131, 257, 220]: “Sports and Outdoors” (**Sports**) and “Beauty” (**Beauty**) from the Amazon Reviews 2014 collection [147]; “Industrial and Scientific” (**Science**), “Musical Instruments” (**Music**), and “Office Products” (**Office**) from the Amazon 2023 collection [95]; **Steam** [108] and **Yelp**². The statistics of processed datasets have been reported in Table 8.1. We adopt the standard leave-last-out data split, using the last and second-to-last items of each sequence for testing and validation, respectively.

Models. We benchmark two models: TIGER [174], representing the generative recommendation paradigm, and SASRec [108], representing the conventional sequential recommendation paradigm. Note that, for a fair comparison, we optimize the SASRec model using cross-entropy loss and treat all items as negative samples, following Liu et al. [131], rather than sampling a single negative item per instance as in Rajput et al. [174].

²<https://business.yelp.com/data/resources/open-dataset/>

Implementation details. We fine-tune the learning rate over $\{1e-3, 3e-3\}$ and train for a maximum of 150 epochs with early stopping. The checkpoint achieving the best validation performance is selected for testing.

Data categorization. We partition test instances into: **memorization**, **generalization** (if memorization is not satisfied), and **uncategorized** (if none of the above is satisfied). Note that the above three categories are mutually exclusive. However, a data instance may exhibit multiple generalization types, each associated with several possible hop distances. Following Occam’s razor, we annotate an instance with all applicable types but retain only the minimum hop distance for each type.

8.3.2 Performance Analysis

In this section, we analyze the performance comparison between SASRec and TIGER, broken down by the memorization and generalization categories defined in Section 8.2.

SASRec memorizes, TIGER generalizes. As illustrated in Table 8.2, TIGER generally underperforms SASRec on memorization subsets (*e.g.*, **−43.6%** on **Yelp**, **−41.2%** on **Sports**, **−35.2%** on **Beauty**, and comparable on others), while consistently outperforming SASRec on generalization subsets (*e.g.*, **+58.8%** on **Office**, **+56.7%** on **Beauty**, and **+39.8%** on **Sports**). This trade-off suggests that SASRec relies more on memorizing observed patterns, while TIGER is more effective at composing learned item transitions for generalization. Both models achieve substantially higher performance on memorization than on generalization overall, reflecting the intrinsic difficulty of generalizing beyond observed transitions. Moreover, both models exhibit near-zero performance on the uncategorized subset while achieving reasonable performance on the others. This supports the validity of our data attribution and suggests that the uncategorized instances are indeed difficult to predict, consistent with our hypothesis in Section 8.2.5.

Generalization categories. Comparing performance across generalization categories, we observe that both models achieve higher performance on *Substitutability* and *Symmetry* than on

Transitivity and *2nd-Symmetry*. We attribute this to differences in the difficulty of the various generalization types. Substitutability and Symmetry require induction from only a single training example, whereas Transitivity and second-order Symmetry require composing knowledge from multiple examples, representing a structurally more complex form of generalization.

Generalization hops. Within each generalization category, both models perform monotonically worse as hop distance increases. This shows that nearby item transitions pose a stronger influence than distant ones. In low-hop settings, SASRec can sometimes outperform TIGER. But its performance drops faster as the hop distance grows. The decline is even sharper in more difficult categories such as Transitivity and 2nd-Symmetry. This suggests that SASRec mainly generalizes over local context, while TIGER remains more robust for longer-hop generalization.

Data ratio analysis. Finally, we examine the proportion of test instances in each category. In all datasets, memorization cases form a much smaller portion than generalization cases. This suggests that pure memorization is limited, and effective recommendation requires substantial generalization capability. Among the generalization categories, most instances require combining information from multiple training examples, whereas only a small fraction can be inferred from a single training instance (Substitutability and Symmetry). Uncategorized instances consistently account for less than 10% of the data, indicating that most test transitions can be explained by other categories.

8.4 Mechanism Analysis: A Token-Level Lens

Semantic ID-based generative recommendation models generally outperform item ID-based models on generalization-related subsets (Section 8.3). This raises a question: *Why does GR generalize better yet memorize worse than item ID-based models?* In this section, we investigate the underlying mechanisms of GR models through a token-level lens.

We first introduce the concept of *prefix n -gram memorization* (Section 8.4.1), and demonstrate that item-level generalization can often be interpreted as token-level memorization within

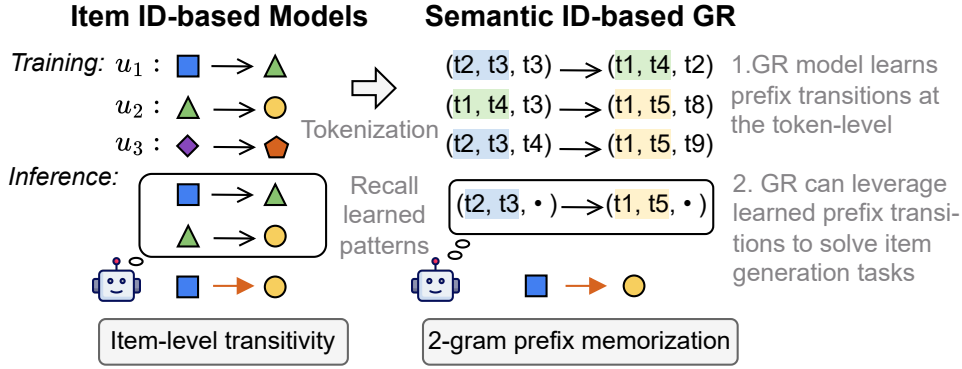


Figure 8.3. Illustration of how item-level generalization can be reduced to token-level memorization for GR models.

the semantic ID space (Section 8.4.2). Next, we characterize models’ behavior using this new lens (Section 8.4.3), and find that: (1) GR generalization performance improves when the underlying token transitions are more frequently observed in the training data. (2) Different item transitions can share the same memorized prefix, which can decrease GR’s ability to memorize a specific item transition. Finally, to further validate our hypothesis, we design a controlled study to vary the token memorization ratio and measure its direct impact on the generalization-memorization trade-off (Section 8.4.4).

8.4.1 Prefix N-Gram Memorization

Motivation. Unlike item ID-based models, GR models represent items as sequences of discrete semantic ID tokens shared across items. This allows the model to anchor predictions on sub-item level transition patterns. However, quantifying memorization behavior at a token level is non-trivial. Directly attributing the effect of a single token on another token is difficult because token-to-token correlations are dense and highly dependent on the contexts [68]. For LLMs, people often assess memorization via n -gram correlation between context and target text, reflecting the model’s ability to memorize the n -gram ‘knowledge’ and generate the corresponding n -gram ‘answer’ [133, 221]. Drawing inspiration from this, we propose quantifying memorization in GR by considering the *prefix n-grams* of the context-target item pair. Since semantic IDs encode

hierarchical (coarse-to-fine) semantic information, focusing on transitions from one prefix to another prefix captures the most prominent semantic dependencies (Figure 8.3).

Token prefix. Let $\text{tok}(i) = [z_1, z_2, \dots, z_L]$ denote the semantic-ID tokenization of item i , where L is the number of tokens in the semantic ID. For a prefix length $n \leq L$, define the n -gram prefix operator $\text{pref}_n(i) \triangleq [z_1, \dots, z_n]$.

Prefix n -gram memorization. We define token-level memorization by considering only the semantic ID prefixes of items in the transitions. A test instance is considered *n -gram prefix-memorizable* if the first n tokens (the n -gram prefix) of both items in the target transition $[i_s \rightarrow i_t]$ occur in the training set, even when the exact items differ.

Def. (1-hop Prefix N-Gram Memorization). A data instance (u, i_t) satisfies *1-hop prefix n -gram memorization* if:

$$\begin{aligned} \exists u' \in \mathcal{D}_{\text{train}}, \exists s \geq 2 \text{ s.t. } & [j_{s-1} \rightarrow j_s] \subseteq u', \\ & \text{pref}_n(i_{t-1}) = \text{pref}_n(j_{s-1}), \\ & \text{pref}_n(i_t) = \text{pref}_n(j_s). \end{aligned}$$

Analogous to the multi-hop generalization framework, this definition naturally extends to k -hop transitions. Notably, when $k = 1$, token prefix memorization can be viewed as a relaxed form of *memorization*, whereas for $k > 1$, it is analogous to the definition of *substitutability* (see Section 8.2.4). For the following sections, we refer to prefix n -gram memorization as *token memorization* for brevity.

8.4.2 From Item Generalization to Token Memorization

We examine how item-level generalization can be reduced to token memorization for GR models, following the definition in Section 8.4.1. Unless otherwise specified, the following experiments on all datasets aggregate all token memorizations with $\text{hop} \leq 4$, and utilize a 256×3

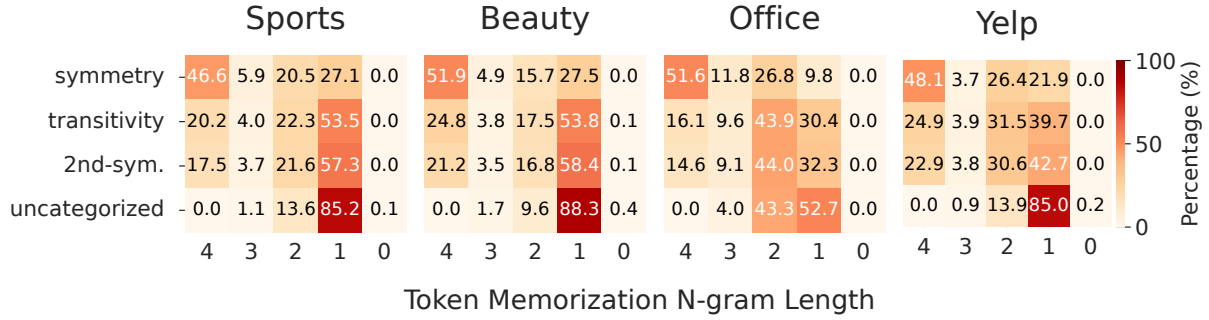


Figure 8.4. Token memorization ratios for each item-level generalization category. The X-axis represents the prefix length for token memorization.

semantic ID quantization followed by one identifier token, consistent with Rajput et al. [174].

Item generalization instances often reduce to token memorization for GR models. Figure 8.4 illustrates the reduction of item-level generalization categories into token-level prefix-gram memorization. We observe that a non-trivial fraction of instances reduce to 1-, 2-, and 3-gram prefix memorization. For example, on average, more than 5% of item-level generalization transitions (symmetry, transitivity, and 2nd-symmetry) can also be explained as 3-hop prefix memorization. Notably, the vast majority of test instances ($> 99\%$) across all item-level categories admit at least 1-gram prefix memorization. This demonstrates that for many test instances where the item-level transition is unseen, the training set nevertheless contains matching prefix transitions, allowing the model to leverage prefix memorization for inference.

Token memorization ratio reflects item-level difficulty. Across categories, symmetry exhibits a higher ratio of 4-gram memorization, largely due to its overlap with item-level substitutability. In contrast, transitivity and 2nd-order symmetry mostly reduce to short prefix memorization (2–3 grams), yielding weaker prefix-transition support from training and making these tasks harder. Furthermore, uncategorized instances reduce almost exclusively to 1-gram memorization, representing the weakest form of prefix-transition support. Overall, these findings show that the ratio of token memorization directly reflects the item-level task difficulty and the model performance trends observed in Section 8.3.

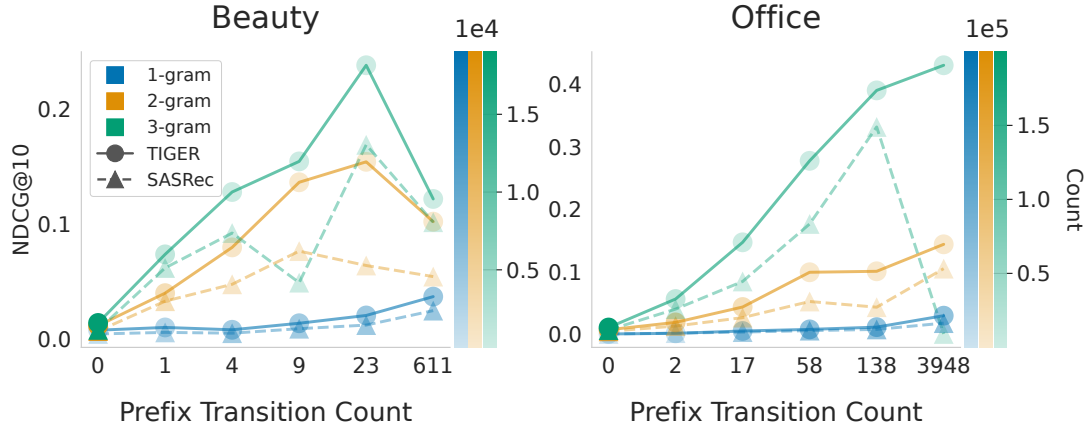


Figure 8.5. Test NDCG@10 against prefix transition counts on 2 datasets. Transition counts are grouped by quantiles.

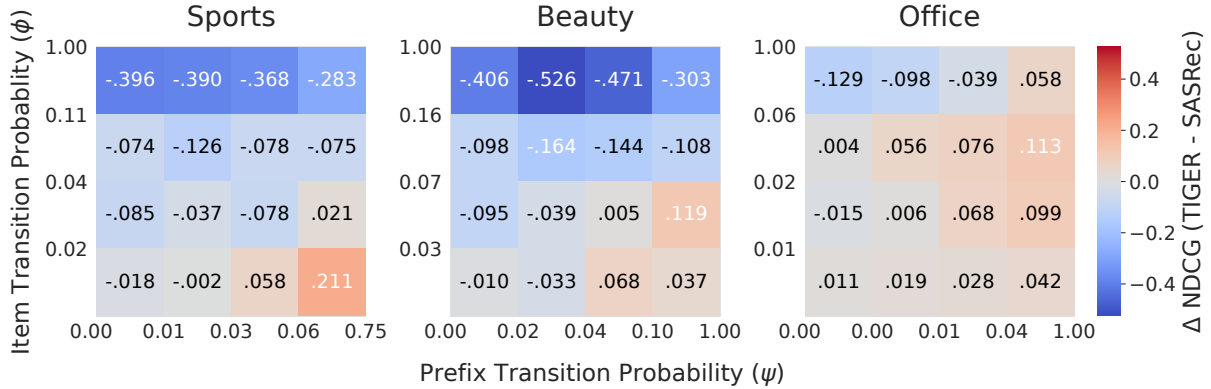


Figure 8.6. TIGER Advantage breakdown by prefix transition probability and item transition probability. Both axes are grouped by quantiles.

8.4.3 Explaining Performance Trade-off via Token Memorization

Having established that item-level generalization often reduces to token-level memorization, we now investigate whether this mechanism explains the performance trade-off: GR generalizes better but memorizes worse at item level. We categorize test instances through token memorization and show that *token memorization enables better generalization* (Section 8.4.3), and *token memorization dilutes item memorization* (Section 8.4.3).

Token Memorization Enables Better Generalization

To analyze how TIGER’s performance correlates with token memorization support at a finer granularity, we quantify prefix support as the count of prefix-transition occurrences in the training data:

$$C_n(i_{t-k}, i_t) = C(\text{pref}_n(i_{t-k}) \rightarrow \text{pref}_n(i_t)),$$

where $C(\cdot)$ is the count function. Given an item-level generalization instance (u, i_t) , the token memorization support $C_n(u, i_t) = \sum_{k=1}^K C_n(i_{t-k}, i_t)$ is the sum of prefix supports of all the k -hop transitions to the target item.

More token memorization support correlates with better generalization. We aggregate token memorization support for hop $k \leq 4$, and plot the TIGER and SASRec models’ NDCG@10 on all item generalization instances (without item memorization support) against $C_n(u, i_t)$ in Figure 8.5. First, we observe that supported instances consistently achieve substantially higher NDCG@10 than non-supported instances for both TIGER and SASRec. TIGER’s gain over SASRec is substantially larger on prefix-supported instances, indicating that TIGER benefits from prefix-transition support that SASRec does not explicitly model. Furthermore, TIGER’s performance gap increases with both increasing support count $C_n(u, i_t)$ and prefix length n , indicating that more token memorization support correlates with better generalization performance.

Token Memorization Dilutes Item Memorization

Conversely, we investigate why this same mechanism might degrade performance on item memorization tasks. We hypothesize a dilution effect: while SASRec directly optimizes the specific item transition $i_{t-1} \rightarrow i_t$, TIGER predicts through prefix transitions that are shared by many items. Thus, its ability to memorize specific item transitions can be diluted. To verify this, we evaluate TIGER’s relative performance (ΔNDCG) against two metrics: *item transition probability* (ϕ), the probability of predicting target item based on all item transitions, and *prefix transition probability* (ψ), the probability of predicting target item’s prefix based on all prefix

Table 8.3. Experiment configurations and token memorization ratio across different semantic ID (SID) configurations.

SID Configuration		Budget	Token Memorization Ratio (%)					
Len (L)	Size (V)	TFLOPs	$n = 5$	$n = 4$	$n = 3$	$n = 2$	$n = 1$	$n = 0$
$L = 2$	1024	5.00×10^{15}	–	–	–	15.06	84.47	0.47
	4096	5.00×10^{15}	–	–	–	15.06	83.93	1.01
$L = 3$	256	7.50×10^{15}	–	–	15.06	16.57	68.36	0.01
	4096	7.50×10^{15}	–	–	15.06	5.40	79.54	0.01
$L = 4$	64	1.00×10^{16}	–	15.06	7.68	46.26	31.01	0.00
	1024	1.00×10^{16}	–	15.06	0.92	21.34	62.68	0.00
$L = 5$	64	1.33×10^{16}	15.06	1.22	5.74	46.08	31.89	0.00
	256	1.33×10^{16}	15.06	0.20	1.09	16.69	66.96	0.00

transitions. For a transition $i_{t-1} \rightarrow i_t$, these quantities are estimated empirically by:

$$\phi = \frac{C(i_{t-1} \rightarrow i_t)}{C(i_{t-1} \rightarrow \cdot)}, \quad \psi = \frac{C(\text{pref}_n(i_{t-1}) \rightarrow \text{pref}_n(i_t))}{C(\text{pref}_n(i_{t-1}) \rightarrow \cdot)},$$

where C is the count function and $C(x \rightarrow \cdot)$ is the total number of outgoing transitions from the context x .

Token memorization dilutes item memorization. We visualize how TIGER’s NDCG gain over SASRec (*i.e.*, ΔNDCG) breaks down by item transition probability ϕ and prefix transition probability ψ . As shown in Figure 8.6, across all datasets, we observe significant NDCG loss when instances have high ϕ but low ψ . In this case, TIGER spreads probability mass across many items sharing the same prefix transition, causing difficulty for the model to memorize a specific item transition pattern. Conversely, TIGER can even outperform SASRec when ψ is high, indicating that TIGER’s item memorization performance is strong only when token memorization aligns. Across the datasets, token memorization usually leads to more dilution cases, causing TIGER to underperform SASRec on memorization tasks.

8.4.4 Mechanism Validation

In this section, we test our hypothesis that increasing token memorization ratio improves item-level generalization but can reduce item-level memorization. Our goal is to vary the token memorization ratio and measure the resulting changes in performance.

Experiment setup. To manipulate the token memorization ratio in a systematic way, we vary the *codebook size*. The intuition is that denser codebooks (smaller codebook sizes V) induce more prefix sharing across items, which increases the total number of token memorization instances. We consider SID lengths $L \in \{2, 3, 4, 5\}$ (including identifier tokens), and for each fixed L we evaluate two codebook sizes V . In Table 8.3 we report the resulting token memorization ratios at each prefix length; as expected, smaller V yields a larger fraction of instances with prefix-transition support.

Controlled factors. We only compare configurations *within the same SID length L* , since optimizing different lengths can have different difficulty and longer SID length often leads to optimization issues [90]. We also fix the recommendation model size and match the training compute budget within each L . For each L , we estimate the training compute to ensure convergence, defined as both memorization and generalization metrics no longer increasing on the validation set. We report (i) validation training dynamics of NDCG@10 and (ii) final test NDCG@10 for each configuration.

Smaller codebook sizes lead to better generalization but worse memorization. From Figure 8.7, we observe a consistent pattern: smaller codebook size V (denser codebooks, higher token memorization ratio) leads to better item-level generalization, but degraded memorization performance. Averaged across settings, the smaller V improves generalization by **+10.24%** relative compared to the larger variant, while memorization decreases by **-7.62%** relative. This provides consistent evidence across all tested codebook sizes that increasing token memorization ratio improves generalization while degrading memorization.

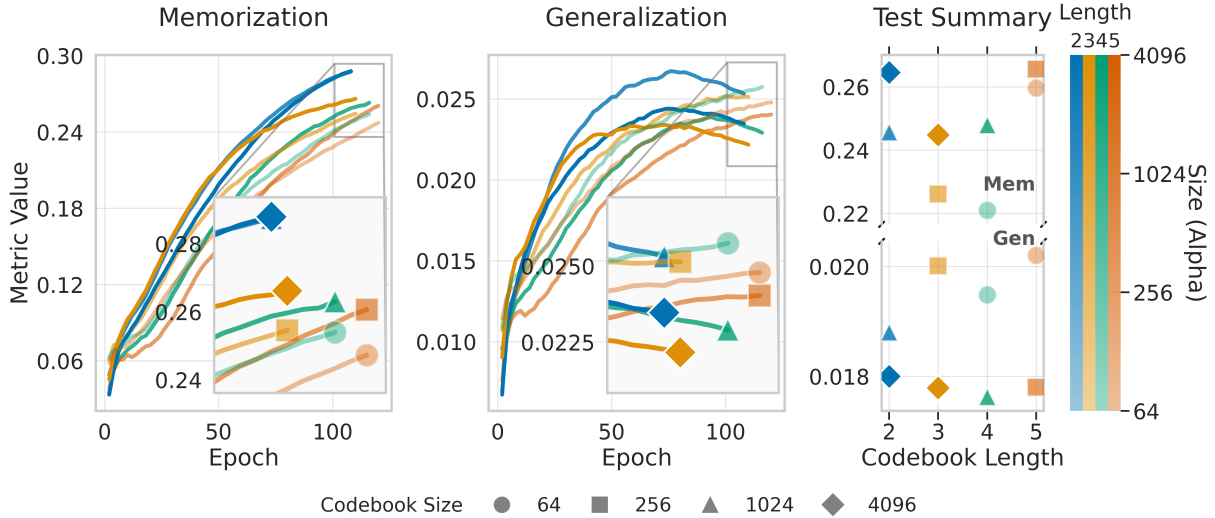


Figure 8.7. Left/Middle: Val NDCG@10 training dynamics on the Memorization and Generalization subsets. Right: Test NDCG@10 for different codebook configurations.

Table 8.4. Performance comparison of ensemble strategies. The best performance is bolded.

Type	Method	Sports (2014)		Beauty (2014)		Scientific (2023)		Instruments (2023)		Office (2023)		Steam		Yelp	
		R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10
Single-Model	SASRec	0.0318	0.0253	0.0566	0.0436	0.0262	0.0209	0.0319	0.0291	0.0241	0.0190	0.1738	0.1525	0.0362	0.0321
	TIGER	0.0318	0.0237	0.0542	0.0383	0.0303	0.0243	0.0357	0.0323	0.0306	0.0254	0.1773	0.1551	0.0281	0.0257
Ensemble	Fixed-weight	0.0530	0.0291	0.0838	0.0471	0.0483	0.0260	0.0635	0.0343	0.0456	0.0261	0.2008	0.1579	0.0630	0.0351
	Adaptive	0.0537	0.0296	0.0841	0.0476	0.0485	0.0261	0.0638	0.0344	0.0461	0.0264	0.2008	0.1579	0.0637	0.0352

Regularization effect in training dynamics. From Figure 8.7, we further observe that for large V , generalization performance peaks early and then degrades as training proceeds. In contrast, for small V , generalization remains stable or continues to improve until convergence. This suggests that denser tokenization has a stronger data-level regularization effect. It encourages TIGER to rely more on shared prefix-transition structure, rather than overfitting to noisier, item-specific transitions.

8.5 Memorization-Aware Adaptive Ensemble

So far, we have shown that GR models excel on instances requiring generalization, whereas item ID-based models perform better on memorization. Motivated by this observation, we investigate whether it is possible to combine the strengths of both paradigms. We propose

a hybrid framework that dynamically weights the predictions of a GR model and an item-based model for each given input, depending on the primary capability the input may require. We first introduce a simple, training-free indicator to estimate the likelihood of memorization (Section 8.5.1), followed by an empirical validation of their correlation with ground-truth categories (Section 8.5.2). Finally, we present experimental results demonstrating that this adaptive model ensemble strategy achieves improved overall performance (Section 8.5.3).

8.5.1 Adaptive Ensemble Indicators

The core idea of our adaptive ensemble strategy is to estimate how likely a data instance can be solved via memorization, and use this estimate to adjust the relative weights between the item ID-based model and the GR model for each instance. A key challenge is that the memorization criterion defined in Section 8.2.2 cannot be directly applied at inference time, since the target item is unavailable. We therefore introduce a practical indicator that estimates the memorization likelihood of a data instance (u, i_t) and converts it into an ensemble weight.

Confidence-based indicator. The intuition behind this approach is that memorization-related instances tend to lie closer to the training data distribution. As a result, an item ID-based model that effectively memorizes training patterns should produce more confident predictions on such instances. We therefore use the prediction confidence of the item ID-based model as a proxy for memorization likelihood. While confidence is often defined as the distance between prediction logits and a uniform distribution, the recommendation task typically involves a very large item space where most items receive negligible probability. We thus adopt the maximum softmax probability (MSP) [207] as a practical confidence score, defined as:

$$s_{\text{Conf}}(u) = \max_{j \in \mathcal{I}} P_{\text{ID}}(i_t = j \mid u), \quad (8.1)$$

where $P_{\text{ID}}(\cdot \mid u)$ denotes the predicted probability distribution of the ID-based model given user history u . We then transform $s_{\text{Conf}}(u)$ into an ensemble weight that controls the contribution of

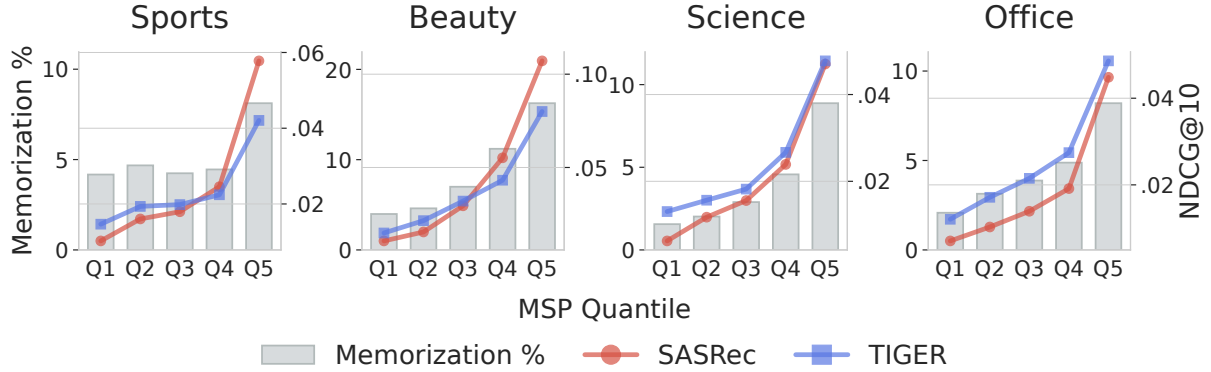


Figure 8.8. Correlation between the MSP indicator, proportion of memorization-related instances (bars), and model performance of SASRec and TIGER (lines).

each model:

$$\alpha(u) = \text{sigmoid}(-q(s_{\text{Conf}}(u) - \tau)), \quad (8.2)$$

where q and τ are hyperparameters.

8.5.2 Validation of the Indicator

In this section, we empirically validate whether the proposed MSP indicator provides a meaningful estimate of memorization likelihood and can therefore support our adaptive ensemble strategy. Using the same datasets and models as in Section 8.3, we group data instances into bins according to quantiles of their indicator values. For each bin, we examine both the proportion of memorization-related instances and the performance of the two paradigms.

Correlation with memorization. We can see that as the indicator value increases, the proportion of memorization-related instances rises monotonically. This trend suggests that the indicator provides a meaningful estimate of memorization likelihood.

Performance crossover. We also observe a performance crossover between the generative model TIGER and the ID-based model SASRec as the indicator value increases. At lower indicator values, TIGER achieves better performance, whereas at higher values SASRec becomes increasingly competitive and may outperform TIGER. This crossover pattern is consistent

with our hypothesis: instances with high indicator values tend to be memorization-related and are better handled by ID-based models, while instances with low values favor the stronger generalization capability of GR models.

8.5.3 Performance of Adaptive Ensemble

Having validated the effectiveness of the proposed MSP indicators, we proceed to evaluate the overall performance of the memorization-aware adaptive model ensemble strategy on real-world datasets.

Experimental settings. We use the same datasets and base models as in Section 8.3. We use Equation (8.2) to map the raw MSP values to an instance-specific ensemble weight $\alpha(u) \in [0, 1]$. The final prediction is obtained by combining the scores produced by the two base models, SASRec and TIGER, according to this weight. We compare our adaptive ensemble against the individual base models and a *Fixed-weight Ensemble* baseline, which assigns a global constant weight α_{static} to SASRec for all instances. Hyperparameters are tuned on the validation set, with $q \in \{1, 5, 9, 13\}$, $\tau \in \{0, 0.1, \dots, 0.5\}$, and $\alpha_{\text{static}} \in \{0, 0.1, \dots, 1.0\}$.

Recommendation results. We present the performance comparison in Table 8.4 and make the following observations:

- The proposed adaptive ensemble strategy generally outperforms both the individual base models and the fixed-weight ensemble baseline. These results provide empirical evidence that item ID-based models and GR models offer complementary strengths.
- The improvement over baselines is more prominent on datasets with a stronger model cross-over effect (see Figure 8.8). The greater the comparative advantage each base model holds in its respective domain, the more the memorization-aware indicator can leverage this specialization.

Table 8.5. Qualitative case studies from Beauty (2014). Case 1 illustrates how item-level generalization reduces to token memorization in GR models. Case 2 illustrates how token memorization dilutes item-level memorization. ϕ and ψ denote the item transition probability and the prefix transition probability, respectively (Section 8.4). The count for target-specific transitions and total outgoing transitions are shown in the parentheses.

	Case 1: Item-level generalization reduces to token memorization	Case 2: Token memorization dilutes item memorization
Example ID	19159	5873
Context item i_{t-1}	Vitamin C Serum	Emu Oil Skin Repair
Target item i_t	Argan Oil Treatment	Facial Moisturizer
Item-level Category	3-hop transitivity, 3-hop 2nd-symmetry	memorization
Prefix transition	$[139, 310] \rightarrow [159, 310]$	$[196, 310] \rightarrow [234, 310]$
Item trans. prob. ϕ	– (unseen)	1.000 (1/1)
Prefix trans. prob. ψ	0.21 (23/111)	0.011 (1/90)
SASRec NDCG@10	0.0	1.0
TIGER NDCG@10	1.0	0.0

Table 8.6. NDCG@10 on memorization (M) and generalization (G) subsets for additional GR and item ID-based models. Values are reported as M / G.

Model	Sports	Beauty	Science	Music	Office	Steam	Yelp
LRURec	.2580/.0100	.3474/.0142	.2698/.0109	.2121/.0142	.2649/.0077	.3978/.0108	.3028/.0256
PSID	.1635/.0175	.2432/.0188	.2143/.0162	.1962/.0187	.2682/.0157	.3871/.0143	.1303/.0201

Table 8.7. Fine-grained comparison of fixed-weight and adaptive ensembles on memorization (M) and generalization (G) subsets. $\bar{\alpha}$ denotes the average ensemble weight assigned to SASRec. Values are reported as M / G.

	Sports	Beauty	Science	Music	Office	Steam	Yelp
Fixed	.2684/.0180	.3813/.0176	.2664/.0182	.2208/.0213	.2862/.0155	.3958/.0158	.2447/.0268
Adaptive	.2651/.0188	.3733/.0190	.2671/.0183	.2211/.0214	.2899/.0156	.3958/.0159	.2424/.0270
$\bar{\alpha}$	.459/.426	.536/.520	.423/.400	.401/.396	.297/.256	.439/.407	.581/.559

8.6 Performance Comparison on Other Item ID-based and GR Models

To verify that the memorization vs. generalization performance gap extends beyond TIGER and SASRec, we evaluate two additional models: PSID [278], a GR model that resolves

SID collisions via non-nearest-neighbor centroid assignment, and LRURec [267], an item ID-based model built on linear recurrent units. As shown in Table 8.6, PSID exhibits a memorization vs. generalization profile similar to TIGER, while LRURec behaves consistently with SASRec. Moreover, across all seven datasets, PSID generalizes better but memorizes worse than LRURec. These results indicate that our findings are not tied to a specific tokenization scheme or backbone, and extend to other GR and item ID-based models.

8.7 Adaptive Ensemble Performance Breakdown

To examine whether the adaptive ensemble’s improvement is driven by memorization-aware weighting rather than a globally tuned mixing ratio, we report a fine-grained breakdown on memorization and generalization subsets in Table 8.7, along with the average ensemble weight $\bar{\alpha}$ assigned to SASRec on each subset.

The adaptive ensemble improves over the fixed-weight baseline on most generalization subsets, with mixed effects on memorization. More importantly, $\bar{\alpha}$ is consistently higher on memorization subsets than on generalization subsets across all datasets. This confirms that the indicator behaves as intended: it up-weights SASRec on memorization-related instances, compensating for the GR model’s relative weakness on this category.

8.8 Qualitative Analysis

To further validate the explanation made in Section 8.4.3, we present two case studies from the Beauty (2014) dataset: item-level generalization realized via token-level memorization, and token-level dilution of item-level memorization, as shown in Table 8.5.

Case 1: Item-level generalization via token-level memorization. The exact item transition (Vitamin C Serum $\rightarrow$ Argan Oil Hair Treatment) is unseen in training, and therefore the item ID-based model has to infer through item-level generalization. However, the training datasets contain 23 parallel transitions whose 2-gram prefixes match the test case, where prefix [139,

310] semantically encodes “Vitamin C/anti-aging serum” and [159, 310] encodes “Argan-based hair/skin treatment.” Therefore, TIGER is able to leverage the 2-gram prefix memorization to correctly predict the target item.

Case 2: Token-level dilution of item-level memorization. The 1-hop item transition (Emu Oil Skin Repair $\rightarrow$ Organic Facial Moisturizer) appears in every training occurrence of the context item ($\phi = 1.0$), so SASRec memorizes it directly. For TIGER, however, the context prefix [196, 310] appears in many training transitions whose target prefixes span a wide range of product categories (e.g., cleansers, masks, serums), with only $\psi = 0.011$ going to the target prefix [234, 310]. As a result, these prefix collisions with other distracting transitions with different behavioral patterns dilute the probability for generating the specific target.

8.9 Discussions on Categorization Method

8.9.1 Design Rationales

Defining generalization can be more challenging than the memorization case, where the model is only required to recall an learned item transition. Our idea is to consider the most fundamental patterns that the model should meaningfully derive from the existing transitions, regardless of specific model architectures. Considering the case for movie recommendation, flipping the order of user actions (e.g., watching Iron Man then Captain America) often yields equally valid recommendations; thus, inferring this symmetric relationship (or symmetry) is an important type of generalization. Prior studies, such as CL4SRec [249], explicitly use “item reordering” for data augmentation, making the model robust to handle the flexible order of interactions in the real world recommendation. Based on similar notions, we categorize generalizations into induction (substitutability, symmetry) from single existing transitions or composition (transitivity, 2nd-symmetry) from multiple existing transitions.

Advantages. The main advantages of the proposed pattern-based framework are its simplicity and scalability to large recommendation scenarios, since the categorization is very computa-

tionally efficient compared to alternatives such as counterfactual memorization. Moreover, our experiments show strong evidence that the simple categorization effectively captures the consistent difference in GR and item ID-based models.

8.9.2 Limitations

Despite the advantages discussed in Section 8.9.1, we acknowledge two limitations of our categorization framework, which we leave to future works.

- **Existence-based definition.** Our categorization is based on whether a memorization or generalization pattern exists in the training data, rather than on its occurrence frequency. A transition appearing once is treated identically to one appearing thousands of times, even though the latter likely provides a stronger learning signal.
- **No causal attribution.** The rule-based categorization does not establish a causal connection between training data and model predictions. A memorization-related instance is not guaranteed to be predicted correctly, nor does an uncategorized instance necessarily mean the model cannot.

8.10 Conclusion

In this chapter, we presented a systematic study comparing semantic ID-based generative recommendation models with traditional item ID-based models through the lens of memorization and generalization. We proposed a framework to categorize data instances based on the item transition patterns they contain. Extensive experiments across various datasets revealed that generative recommendation models excel on generalization-related instances, while item ID-based models perform better on memorization-related instances. To explain this, we analyzed the models at the token level, finding that item-level generalization in generative models often reduces to token-level memorization. Finally, we show that the two paradigms are complementary by

demonstrating that adaptively adjusting ensemble weights using a memorization-aware indicator leads to improved performance.

Chapter 8, in part, is a reprint of the material as it appears in “On the Memorization and Generalization of Generative Recommendation.” by Yijie Ding, Zitian Guo, Jiacheng Li, Letian Peng, Shuai Shao, Wei Shao, Xiaoqiang Luo, Luke Simon, Jingbo Shang, Julian McAuley, and Yupeng Hou, accepted for publication at the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining in 2026, referenced as [45]. The dissertation author was the primary investigator and author of this paper.

Chapter 9

Related Work

9.1 Recommender Systems

9.1.1 Sequential Recommendation

Sequential recommendation [82, 108] predicts the next item that a user interacts with given the user’s interaction history. Sequential behavior patterns are useful for identifying user needs and providing recommendations. The sequential recommendation task usually takes a sequence of items as input. Early attempts adopt probability models like Markov chains [183], while modern approaches adopt models like Recurrent Neural Networks (RNN) [82, 122, 267], Convolutional Neural Network (CNN) [202], Graph Neural Networks (GNN) [239, 16], and Transformers [108, 197]. More recently, due to its effective attention mechanism in sequential modeling tasks [51, 206], Transformer [208]-based recommendation models [297, 197, 108, 281, 86] achieves best performance in sequential recommendation tasks.

9.1.2 Generative Recommendation

A new generative recommendation paradigm shows better performance and scalability than the conventional Transformer-based recommender systems on sequential recommendation tasks. Unlike conventional sequential recommendation paradigms [183, 108, 197, 82, 202, 146], where each item is typically indexed by a unique ID, generative recommendation tokenizes each item as a sequence of sub-item tokens [204, 174, 271, 41, 76, 290, 92, 40, 93]. A generative

model then operates over these token sequences and autoregressively generates tokens that index the next item. By autoregressively generating the next tokens as recommendations, this generative paradigm offers benefits such as memory efficiency [174, 257, 46, 90], scalability [271, 143], and easier alignment with LLMs [290, 104, 200, 120].

Item tokenization methods are crucial for the performance of generative recommendation methods [174, 165, 166, 104, 132]. A common practice is to leverage the text features associated with each item. These text features will then be encoded into semantically rich item tokens to facilitate the model’s decision process. TIGER [174] generates item tokens with sentence embedding [42, 157] through RQ-VAE [268] using descriptive text associated with each item. GPTRec [165] quantizes item representations from pretrained Truncated Singular Value Decomposition (SVD) models to generate item tokens. Jin et al. [104] proposed a language model based indexer that could be applied to multiple information retrieval tasks. Such item tokenization techniques also make the generative paradigm more feasible to integrate other modalities, *e.g.*, text, image, and audio [226, 123, 290, 222, 63].

Existing work on generative recommendation primarily focuses on developing improved tokenization methods, such as exploring alternative algorithms [220, 302, 90, 96, 104, 131, 278, 292, 291, 225, 213, 251] or more effectively leveraging heterogeneous features during the tokenization stage [224, 143, 91, 303, 132, 211, 296, 261, 227, 232, 253, 145]. To better understand the underlying mechanisms of GR, recent studies have investigated its cold-start capabilities [257, 46], scaling behavior [135], and generalization capability [45].

9.1.3 LLMs for Recommendation

In recent years, the development of Large Language Models (LLMs) [288, 1, 206] has enabled their application across a wide range of recommendation scenarios due to their well-learned world knowledge from web-scale internet data, which allow them to easily understand cold-start items. One direct application, the ‘LLM for Rec’ methods, mainly involves prompting [136, 36, 89], finetuning [272, 6], retrieval-augmented generation (RAG) [43, 33, 236], and

conversational dialogue systems [80, 198]. While these approaches require minimal pre-training, they suffer from high inference latency and hallucinations. To mitigate these issues, recent methods use LLMs to generate modality-based item representations [87, 121, 88] and train backbone models on these frozen representations, achieving state-of-the-art performance in cold-start scenarios.

9.1.4 Multi-Behavior Recommendation

Multi-behavior recommendation models incorporate behavior types of history user-item interactions to predict the items that the user will interact with. Early methods mainly exploit MF-based models for multi-behavior recommendations [114, 289, 192]. Later, some deep learning methods [20, 59], especially GNN-based methods [21, 245, 282, 103, 256], are also proved to be effective. However, the above-mentioned methods do not include temporal sequential patterns in their model, which is vital for next-item prediction.

Recent works have started to study the multi-behavior sequential recommendation (MBSR) problem. Early MBSR models [69, 70, 23] usually take sequential input of each behavior as separate subsequences and aggregate the representation encoded from each subsequence to predict the next interaction. However, this aggregation method ignores the sequential pattern between different behaviors. Thus, other approaches aim to directly integrate the sequence of behaviors into the item sequence [264, 195, 258]. MB-STR [264] routes each input item to different sets of feed-forward and linear transformation layers inside the Transformer model according to the corresponding behavior type. PBAT [195] considers the user’s personalized interaction pattern of behaviors. MBHT [258] uses hypergraph learning to uncover item-wise multi-behavior correlations. A recent work, HSTU [271], utilizes actions (analogous to behaviors) to unify the retrieval and rank tasks in generative recommender systems.

9.1.5 Cold-Start and Inductive Recommendation

The item cold-start problem (ICS) [283], a longstanding challenge in recommender systems, requires models to recommend newly added items with limited or no historical interaction data, and therefore external knowledge have to leveraged. In scenarios with a few interactions, studies focused on data-efficient learning to quickly improve cold-start performance [231, 299]. Meta-learning [223] is a widely adopted approach that pretrains models on warm interactions to capture general interaction patterns and then adapts parameters to new cold-start items using limited data. Common approaches use meta-learning that helps models generalize better from limited interactions, via gradient-based optimization [118, 57], task adaptation [129, 242], or memory-augmented modules [50, 295].

In more challenging ICS scenarios with no interaction data, studies focus on deriving content-based item representations while still maintaining strong performance on warm items. Early works explored perturbation training strategies to reduce reliance on interactions and encourage models to learn generalized recommendations solely from content features [209, 304]. Others formulated item representation learning as an explicit knowledge alignment task, bridging warm interaction-based and content-based representations through methods such as contrastive learning [231, 299] and generative adversarial networks (GANs) [22, 2]. Inductive recommendation particularly tackles the challenge of recommending new items without any associated interactions. Existing methods leverage side information like tags or descriptions [164, 304, 121], modality representations [87, 88, 189], and behavior patterns [238, 237].

In recent years, graph neural networks [243] have gained attention for modeling user-item interaction graphs in recommendation, with its power in modeling higher-order relationships beyond direct interactions and content features. However, a key challenge remains in effectively propagating information to cold-start items, where interaction signals are sparse. One typical methods trains the model to infer probable edges on the original use item interaction graph for cold-start items to enable information aggregation. Typical training tasks involves masking and

reconstruction [112], maximizing mutual information [219], and uncertainty estimation [140]. On the other hand, another stream of work enrich the existing interaction graph with more fine-grained connections to propagate information to cold-start items. The constructed edges includes semantic links [14, 139], multi-view feature extractions [294], and knowledge relations [53].

To mitigate data sparsity for cold-start item recommendation, researchers have explored using cross-domain interactions to learn generalizable patterns and transfer knowledge from data-rich domains [300, 301, 249]. However, a key challenge lies in designing effective transfer techniques that account for discrepancies between domains. Early works primarily relied on overlapping information between the source and target domain, including items [192, 301] or attributes [203]. To address distributional differences, distribution alignment [141, 218] and invariant representation learning [240, 74] has been proposed to build robust item representations to facilitate knowledge transfer. Recently, with the rise of pretrained language models (PLMs), text representations have proven to be powerful semantic bridges, reducing the need for explicit entity overlaps between domains [44, 63, 87]. To learn universal representations from text, many approaches leverage pretraining across multiple domains, followed by fine-tuning on domain-specific tasks to enable seamless cross-domain transfer [87, 88, 121].

9.1.6 Expressiveness of Recommendation Models

As recommender systems enter the era of deep learning, understanding the expressive power of various architectures has emerged as a critical research direction. Existing literature can be broadly categorized into two primary threads. The first focuses on the expressiveness of the underlying backbone architectures upon which recommendation models are built, such as graph neural networks [252, 153, 188, 12]. The second thread examines the dot-product-based scoring functions, like connecting expressiveness with embedding dimensionality [161, 234, 149, 178]. However, unlike traditional two-tower models, generative recommendation produces predictions by decoding sequences rather than retrieving single vectors. Consequently, the insights derived from retrieval-based paradigms may not hold in this new context.

9.2 Action Tokenization

9.2.1 Semantic Encoders in Recommendation

Researchers have long sought to better leverage semantically rich textual features such as titles and descriptions. Early works in content-based recommendation [152, 66] extracted keywords and learned their collaborative relations between users and items [181, 210, 25, 212]. With advances in natural language processing (NLP), later studies began directly encoding text features using pretrained models to better capture language semantics [26, 87, 266]. However, such models often suffer from domain gaps between their pretraining data and recommendation scenarios [88]. Recently, the emergence of LLMs with superior generalization capabilities has inspired researchers to explore using LLMs as semantic encoders in recommendation [230, 180, 179, 189].

9.2.2 Semantic ID-Based Recommendation

Semantic ID refers to a sequence of discrete tokens that jointly index an item while retaining some semantics [174, 88, 290, 193]. Existing methods for constructing semantic IDs tokenize item features such as text [290, 104], images [132], and collaborative signals [165, 224, 225, 143] using techniques like quantization [174, 220, 302, 292, 291], hierarchical clustering [96, 191], or joint training with recommendation models [130]. By indexing items with semantic IDs, one can maintain a fixed vocabulary of tokens independent of the number of items, enabling scalable [271, 143] and memory-efficient models [174, 204, 166].

Early semantic ID-based recommendation models were primarily retrieval-based. Each item’s semantic ID was embedded, and the closest aggregated semantic ID embeddings to the user representation were retrieved [88, 166, 257]. With the rise of generative models and insights from scaling laws in related fields, such as large language models (LLMs) [288] and generative retrieval [204, 125, 126], generative recommendation has emerged as a promising paradigm for sequential recommendation [174, 271, 128, 91, 92]. In generative recommendation,

tokens in semantic IDs directly serve as input for the sequence encoder. Models are then trained to autoregressively generate the tokens, which can be then parsed as predicted items. These generative models explicitly integrate semantics into the learning objective and align more naturally with LLMs [290, 104, 200, 120].

Existing research has developed different action tokenization techniques, such as hierarchical clustering [96, 191], quantization [174, 220, 302], or jointly training with recommendation models [130]. Other works incorporate additional modalities like collaborative filtering [165, 224, 225, 143, 132] and natural language [290, 104, 89, 276].

9.3 LLMs, Generative Models, and More General Machine Learning Models

9.3.1 Benchmarking Embeddings and LLMs for Recommendation

To comprehensively evaluate text embedding models across domains and tasks, researchers have developed benchmarks that aggregate multiple public datasets. An early example is BEIR [205], which evaluates text embedding models on information retrieval tasks. MTEB [154] and MMTEB [55] further expand the evaluation scope beyond retrieval to broader embedding applications such as classification and clustering. Recently, BRIGHT [194] focuses on retrieval tasks that require reasoning to identify relevant documents.

Existing efforts to benchmark LLMs for recommendation can be grouped into three categories: (1) benchmarking their real-world web interaction capabilities as shopping agents [260, 214]; (2) benchmarking their shopping-related knowledge and understanding of user intent [105]; and (3) benchmarking LLMs themselves as recommendation models [137, 134, 101, 138].

9.3.2 Speculative Decoding and Efficient Inference

Large language models (LLMs) [1, 288] have revolutionized a wide range of applications, yet suffer from high inference latency due to autoregressive decoding and large model size. Speculative decoding was proposed to accelerate LLM inference by leveraging a lightweight, ho-

mologous drafter model to generate future token sequences, which are then verified by a stronger target model, achieving lower latency and lossless performance [119, 18]. Subsequent efforts have developed efficient drafting strategies [11, 81, 54], improved draft quality to maximize acceptance rate [247, 65], and applied the framework to diverse domains [229, 37]. Recently, Lin et al. [127] extended speculative decoding to accelerate top-K sequence generation in the GR setting.

9.3.3 Tokenization for Large Language Models

Tokenization is the process of transforming raw text into discrete token sequences [116]. Early word-level methods are context-independent and struggle to tokenize out-of-vocabulary words [199, 5]. Consequently, subword-level tokenization has gradually become the more mainstream choice. The vocabularies of these subword-level tokenizers are constructed iteratively, either bottom-up (starting with a small vocabulary and merging commonly occurring token pairs as new tokens) [241, 186], or top-down (starting with a large vocabulary and pruning tokens to minimize likelihood decrease) [115, 262]. Once the vocabulary is built, the text can be segmented either using the same method employed during vocabulary construction or based on additional objectives [78, 169, 84, 185].

9.3.4 Memorization and Generalization

The study of model memorization and generalization has long been a central concern in machine learning, spanning from classical machine learning models [72, 259, 10] to modern large language models [15, 221, 102]. A key challenge of conducting these analyses is how to characterize the influence of training data on model behavior. One line of work approaches this question through *counterfactual memorization*, which measures the causal effect of removing individual training instances on task performance [273, 68, 175, 64]. However, these methods are often computationally expensive and difficult to scale [68, 170, 246]. Other approaches define generalization as robustness to local input perturbations (*e.g.*, input clues [47] or task

rules [248, 7, 31]).

Chapter 10

Conclusion and Future Outlook

10.1 Summary of Contributions

In this dissertation, we present our research on action tokenization, which studies how to effectively convert human-perceivable actions into machine-readable token sequences.

We mainly consider two data formats: semantic representations, *i.e.*, continuous vectors, and semantic IDs, *i.e.*, discrete token sequences. Our contribution BLAIR shows that semantic representations can already support powerful action tokenization methods, paving the way for further study of semantic ID-based approaches, which are more memory-efficient and better suited for building deep, large-scale generative models.

Our contributions around semantic IDs are organized into three major thrusts: tokenizing actions into semantic IDs, modeling with semantic IDs, and understanding semantic IDs.

In the first thrust, we propose methods that produce semantic IDs that either capture more useful information or become more expressive given the same input. Our contribution MBGen formulates multi-behavior recommendation as an autoregressive generation task over concatenated behavior-type tokens and item-action tokens, enabling training data from broader behavior types to be used even when they differ from the target behavior. Our contribution ActionPiece challenges the context-independent nature of existing action tokenization methods and proposes a context-aware approach in which the same action can be tokenized differently depending on its context. Our contribution RPG extends semantic IDs from relatively short

sequences to much longer ones while maintaining efficient inference, showing that longer semantic IDs can provide greater expressiveness.

In the second thrust, we propose methods that make better use of given semantic IDs to support specific capabilities. Our contribution SpecGR finds that semantic ID-based generative recommendation methods struggle to generate semantic IDs that have not been seen during training. We therefore leverage the strong ranking capability of generative recommendation models to rank candidates proposed by an inductive drafter model. Our contribution Latte identifies an inherent expressiveness limitation in the structural decoding process, where items with similar semantic IDs tend to receive correlated generation probabilities. This limits personalization in recommendation results. To address this issue, we introduce a latent token before the actual decoding tree, thereby increasing the effective tree distance between originally correlated items.

In the final thrust, we systematically analyze the discrepancy between generative recommendation models and conventional item ID-based recommendation models through the lens of memorization and generalization, aiming to understand what benefits semantic IDs and action tokenization bring to recommendation tasks. We find that generative recommendation models generally generalize better across datasets, but may sometimes memorize worse. We further show that this discrepancy comes from how different models tokenize action data, demonstrating that action tokenization fundamentally affects recommendation performance.

10.2 Future Directions

In addition to the topics discussed in this dissertation, several future directions remain interesting and potentially worth further investigation:

1. *Gaps between metrics on constructed semantic IDs and downstream recommendation performance.* At present, the most reliable way to determine whether a set of constructed semantic IDs is better than an existing one is still to train downstream recommendation models and compare their final performance. Several metrics have been proposed to

measure the quality of constructed semantic IDs, such as token utilization rate and entropy. However, these metrics may be useful for identifying poor semantic IDs, but they are often less effective at distinguishing among stronger semantic ID constructions. More fundamentally, it remains unclear whether developing intrinsic metrics on semantic IDs that reliably inform downstream recommendation performance is merely difficult, or whether it is impossible. The community would therefore benefit from further studies that clarify when such metrics are possible, what information they must capture, and what limitations they may inevitably face.

2. *Inference efficiency.* Latency is one of the key bottlenecks hindering the broad adoption of generative recommendation models. Compared with traditional nearest-neighbor search, which can often be performed through a single dot product over candidate representations, generative recommendation models typically require autoregressive generation with constrained beam search, involving multiple model forward passes. This issue becomes even more severe when each forward pass is expensive, as in billion-scale LLM-aligned generative models. Although generative recommendation has the potential to reduce the latency overhead introduced by coordinating multiple stages in traditional recommendation pipelines, and may benefit from improved parallelism after sufficient scaling, more practical exploration is needed to better understand these trade-offs. Another potential solution is distillation, which uses cheaper architectures to reduce the cost of model forwarding, as in Guo et al. [71].
3. *LLM-native generative recommendation.* A key motivation for studying generative recommendation is the possibility of scaling up recommendation models and aligning them with generative models from other modalities, such as LLMs. Existing work has explored aligning LLMs with semantic IDs by constructing cross-modal data and performing post-training to introduce semantic IDs as new tokens. This approach works to some extent, but introducing a new modality at the post-training stage remains challenging. A

more promising pathway may be to incorporate generalizable user behavior data directly during pre-training, leading to large generative models that understand both user behaviors and other modalities. This could ultimately support LLM-native, or more broadly multimodal-model-native, personalized assistants and recommender systems.

4. *Action tokenization methods beyond semantic IDs.* Semantic IDs are a useful solution at the current stage, but they may not be the final form of tokenizing action data into token sequences. They involve a chain of processing steps, such as encoding and quantization, that are difficult to optimize end-to-end. In particular, clustering and quantization often introduce information loss. Semantic representation-based generative models have traditionally been considered difficult to build because they require optimization and alignment over a large vocabulary of encoded representations. However, with recent progress in continuous vector sequence modeling and improved alignment techniques between semantic spaces and generative models, such approaches may become increasingly powerful for action tokenization.

Bibliography

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Floren-
cia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat,
et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Manal A Alshehri and Xiangliang Zhang. Generative adversarial zero-shot learning
for cold-start news recommendation. In *Proceedings of the 31st ACM International
Conference on Information & Knowledge Management*, pages 26–36, 2022.
- [3] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Guided open vocab-
ulary image captioning with constrained beam search. *arXiv preprint arXiv:1612.00576*,
2016.
- [4] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu
Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav
Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin
Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy,
Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm
Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford,
Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim
Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders
Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez,
Misha Khalman, Jakub Sygnowski, and et al. Gemini: A family of highly capable
multimodal models. *arxiv:2312.11805*, 2023.
- [5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by
jointly learning to align and translate. In *ICLR*, 2015.
- [6] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. TALL-
Rec: An effective and efficient tuning framework to align large language model with
recommendation. In *RecSys*, 2023.
- [7] Joshua Barron and Devin White. Too big to think: Capacity, memorization, and general-
ization in pre-trained transformers. *arXiv preprint arXiv:2506.09099*, 2025.

- [8] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.
- [9] Weijie Bian, Kailun Wu, Lejian Ren, Qi Pi, Yujing Zhang, Can Xiao, Xiang-Rong Sheng, Yong-Nan Zhu, Zhangming Chan, Na Mou, et al. CAN: feature co-action network for click-through rate prediction. In *WSDM*, pages 57–65, 2022.
- [10] Sam Buchanan, Druv Pai, Yi Ma, and Valentin De Bortoli. On the edge of memorization in diffusion models. *arXiv preprint arXiv:2508.17689*, 2025.
- [11] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.
- [12] Xuheng Cai, Lianghao Xia, Xubin Ren, and Chao Huang. How expressive are graph neural networks in recommendation? In *CIKM*, pages 173–182, 2023.
- [13] Francesco Paolo Cantelli. Sui confini della probabilita. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*, pages 47–60, 1929.
- [14] Yi Cao, Sihao Hu, Yu Gong, Zhao Li, Yazheng Yang, Qingwen Liu, and Shouling Ji. Gift: Graph-guided feature transfer for cold-start video click-through rate prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 2964–2973, 2022.
- [15] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models, 2023. *URL* <https://arxiv.org/abs/2202.07646>, 2202, 2023.
- [16] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. Sequential recommendation with graph neural networks. In *SIGIR*, 2021.
- [17] Ben Chen, Xian Guo, Siyuan Wang, Zihan Liang, Yue Lv, Yufei Ma, Xinlong Xiao, Bowen Xue, Xuxin Zhang, Ying Yang, Huangyu Dai, Xing Xu, Tong Zhao, Mingcan Peng, Xiaoyang Zheng, Chao Wang, Qihang Zhao, Zhixin Zhai, Yang Zhao, Bochao Liu, Jingshan Lv, Xiao Liang, Yuqing Ding, Jing Chen, Chenyi Lei, Wenwu Ou, Han Li, and Kun Gai. Onesearch: A preliminary exploration of the unified end-to-end generative framework for e-commerce search. *arXiv preprint arXiv:2509.03236*, 2025.
- [18] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

- [19] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys challenge 2018: automatic music playlist continuation. In *RecSys*, pages 527–528, 2018. doi: 10.1145/3240323.3240342. URL <https://doi.org/10.1145/3240323.3240342>.
- [20] Chong Chen, Min Zhang, Yongfeng Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. Efficient heterogeneous collaborative filtering without negative sampling for recommendation. In *AAAI*, 2020.
- [21] Chong Chen, Weizhi Ma, Min Zhang, Zhaowei Wang, Xiuqiang He, Chenyang Wang, Yiqun Liu, and Shaoping Ma. Graph heterogeneous multi-relational recommendation. In *AAAI*, 2021.
- [22] Hao Chen, Zefan Wang, Feiran Huang, Xiao Huang, Yue Xu, Yishi Lin, Peng He, and Zhoujun Li. Generative adversarial framework for cold-start item recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2565–2571, 2022.
- [23] Hong Chen, Yudong Chen, Xin Wang, Ruobing Xie, Rui Wang, Feng Xia, and Wenwu Zhu. Curriculum disentangled recommendation with noisy multi-feedback. In *NeurIPS*, 2021.
- [24] Siyuan Chen, Hanshen Yu, Jamal Yagoobi, and Chenhui Shao. Reinforcement learning constrained beam search for parameter optimization of paper drying under flexible constraints. *arXiv preprint arXiv:2501.12542*, 2025.
- [25] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *JMLR*, 13(1):3619–3622, 2012.
- [26] Ting Chen, Liangjie Hong, Yue Shi, and Yizhou Sun. Joint text embedding for personalized content-based recommendation. *arXiv preprint arXiv:1706.01084*, 2017.
- [27] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, pages 1597–1607, 2020.
- [28] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.
- [29] Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, et al. On the representation collapse of sparse mixture of experts. In *NeurIPS*, 2022.

- [30] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *JMLR*, 24(240):1–113, 2023.
- [31] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.
- [32] Pierre Colombo, Nathan Noiry, Ekhine Irurozki, and Stéphan Cléménçon. What are the best systems? new perspectives on NLP benchmarking. In *NeurIPS*, 2022.
- [33] Emile Contal and Garrin McGoldrick. Ragsys: Item-cold-start recommender as rag system. *arXiv preprint arXiv:2405.17587*, 2024.
- [34] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*, pages 191–198, 2016.
- [35] Gautier Dagan, Gabriel Synnaeve, and Baptiste Roziere. Getting the most out of your tokenizer for pre-training and domain adaptation. In *ICML*, 2024.
- [36] Sunhao Dai, Ninglu Shao, Haiyuan Zhao, Weijie Yu, Zihua Si, Chen Xu, Zhongxiang Sun, Xiao Zhang, and Jun Xu. Uncovering chatgpt’s capabilities in recommender systems. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1126–1132, 2023.
- [37] Valentin De Bortoli, Alexandre Galashov, Arthur Gretton, and Arnaud Doucet. Accelerated diffusion models via speculative sampling. *arXiv preprint arXiv:2501.05370*, 2025.
- [38] Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. *arXiv preprint arXiv:2010.00904*, 2020.
- [39] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
- [40] Yashar Deldjoo, Zhankui He, Julian McAuley, Anton Korikov, Scott Sanner, Arnau Ramisa, René Vidal, Maheswaran Sathiamoorthy, Atoosa Kasirzadeh, and Silvia Milano. A review of modern recommender systems using generative models (gen-recsys). In *KDD*, pages 6448–6458, 2024.
- [41] Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965*, 2025.

- [42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019.
- [43] Dario Di Palma. Retrieval-augmented recommender system: Enhancing recommender systems with large language models. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1369–1373, 2023.
- [44] Hao Ding, Yifei Ma, Anoop Deoras, Yuyang Wang, and Hao Wang. Zero-shot recommender systems. *arXiv preprint arXiv:2105.08318*, 2021.
- [45] Yijie Ding, Zitian Guo, Jiacheng Li, Letian Peng, Shuai Shao, Wei Shao, Xiaoqiang Luo, Luke Simon, Jingbo Shang, Julian McAuley, and Yupeng Hou. On the memorization and generalization of generative recommendation. In *KDD*, 2026.
- [46] Yijie Ding, Jiacheng Li, Julian McAuley, and Yupeng Hou. Inductive generative recommendation via retrieval-based speculation. In *AAAI*, 2026.
- [47] Albérick Euraste Djiré, Abdoul Kader Kaboré, Earl T Barr, Jacques Klein, and Tegawendé F Bissyandé. Memorization or interpolation? detecting llm memorization through input perturbation analysis. *arXiv preprint arXiv:2505.03019*, 2025.
- [48] Seunghoon Doh, Keunwoo Choi, and Juhan Nam. Talkplay: Multimodal music recommendation with large language models. *arXiv preprint arXiv:2502.13713*, 2025.
- [49] Seunghoon Doh, Keunwoo Choi, and Juhan Nam. Talkplay-tools: Conversational music recommendation with llm tool calling. *arXiv preprint arXiv:2510.01698*, 2025.
- [50] Manqing Dong, Feng Yuan, Lina Yao, Xiwei Xu, and Liming Zhu. Mamo: Memory-augmented meta-optimization for cold-start recommendation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 688–697, 2020.
- [51] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [52] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.
- [53] Yuntao Du, Xinjun Zhu, Lu Chen, Ziquan Fang, and Yunjun Gao. Metakg: Meta-learning on knowledge graph for cold-start recommendation. *IEEE Transactions on Knowledge*

and *Data Engineering*, 35(10):9850–9863, 2022.

- [54] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.
- [55] Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Sibli, Dominik Krzemiński, Genta Indra Winata, et al. Mmteb: Massive multilingual text embedding benchmark. In *ICLR*, 2025.
- [56] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.
- [57] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.
- [58] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.
- [59] Chen Gao, Xiangnan He, Dahua Gan, Xiangning Chen, Fuli Feng, Yong Li, Tat-Seng Chua, and Depeng Jin. Neural multi-task recommendation from multi-behavior data. In *ICDE*. IEEE, 2019.
- [60] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *EMNLP*, pages 6894–6910, 2021.
- [61] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2946–2953, 2013.
- [62] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(4):744–755, 2014.
- [63] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *RecSys*, pages 299–315, 2022.
- [64] Bishwamittra Ghosh, Soumi Das, Qinyuan Wu, Mohammad Aflah Khan, Krishna P Gummadi, Evimaria Terzi, and Deepak Garg. Rethinking memorization measures and their implications in large language models. *arXiv preprint arXiv:2507.14777*, 2025.
- [65] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel

- Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.
- [66] Prem Gopalan, Laurent Charlin, and David M Blei. Content-based recommendations with poisson factorization. In *NIPS*, volume 27, 2014.
 - [67] Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.
 - [68] Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.
 - [69] Yulong Gu, Zhuoye Ding, Shuaiqiang Wang, Lixin Zou, Yiding Liu, and Dawei Yin. Deep multifaceted transformers for multi-objective ranking in large-scale e-commerce recommender systems. In *CIKM*, 2020.
 - [70] Long Guo, Lifeng Hua, Rongfei Jia, Binqiang Zhao, Xiaobo Wang, and Bin Cui. Buying or browsing?: Predicting real-time purchasing intent using attention-based deep network with multiple behavior. In *SIGKDD*, 2019.
 - [71] Zitian Guo, Yupeng Hou, Clark Mingxuan Ju, Neil Shah, and Julian McAuley. Mlps are efficient distilled generative recommenders. *arXiv preprint arXiv:2605.12617*, 2026.
 - [72] Junlin Han, Huangying Zhan, Jie Hong, Pengfei Fang, Hongdong Li, Lars Petersson, and Ian Reid. What images are more memorable to machines? *arXiv preprint arXiv:2211.07625*, 2022.
 - [73] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.
 - [74] Jia He, Rui Liu, Fuzhen Zhuang, Fen Lin, Cheng Niu, and Qing He. A general cross-domain recommendation framework via bayesian neural network. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1001–1006. IEEE, 2018.
 - [75] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016.
 - [76] Ruining He, Lukasz Heldt, Lichan Hong, Raghunandan H. Keshavan, Shifan Mao, Nikhil Mehta, Zhengyang Su, Alicia Tsai, Yueqi Wang, Shao-Chuan Wang, Xinyang Yi, Lexi Baugher, Baykal Cakici, Ed H. Chi, Cristos Goodrow, Ningren Han, He Ma, Rómer Rosales, Abby Van Soest, Devansh Tandon, Su-Lin Wu, Weilong Yang, and Yilin Zheng. PLUM: adapting pre-trained language models for industrial-scale generative recommendations. *arXiv preprint arXiv:2510.07784*, 2025.

- [77] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.
- [78] Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. Dynamic programming encoding for subword segmentation in neural machine translation. In *ACL*, pages 3042–3051, 2020.
- [79] Zhankui He, Zhouhang Xie, Rahul Jha, Harald Steck, Dawen Liang, Yesu Feng, Bodhisattwa Majumder, Nathan Kallus, and Julian McAuley. Large language models as zero-shot conversational recommenders. In *CIKM*, 2023.
- [80] Zhankui He, Zhouhang Xie, Rahul Jha, Harald Steck, Dawen Liang, Yesu Feng, Bodhisattwa Prasad Majumder, Nathan Kallus, and Julian McAuley. Large language models as zero-shot conversational recommenders. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pages 720–730, 2023.
- [81] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*, 2023.
- [82] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.
- [83] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [84] Valentin Hofmann, Hinrich Schuetze, and Janet Pierrehumbert. An embarrassingly simple method to mitigate undesirable properties of pretrained language model tokenizers. In *ACL*, 2022.
- [85] Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *ACL*, pages 1535–1546, 2017.
- [86] Yupeng Hou, Binbin Hu, Zhiqiang Zhang, and Wayne Xin Zhao. Core: Simple and effective session-based recommendation within consistent representation space. In *SIGIR*, 2022.
- [87] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Towards universal sequence representation learning for recommender systems. In *KDD*, 2022.
- [88] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-

- quantized item representation for transferable sequential recommenders. In *WWW*, 2023.
- [89] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. Large language models are zero-shot rankers for recommender systems. In *ECIR*, 2024.
 - [90] Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, and Julian McAuley. Generating long semantic ids in parallel for recommendation. In *KDD*, 2025.
 - [91] Yupeng Hou, Jianmo Ni, Zhankui He, Noveen Sachdeva, Wang-Cheng Kang, Ed H Chi, Julian McAuley, and Derek Zhiyuan Cheng. Actionpiece: Contextually tokenizing action sequences for generative recommendation. In *ICML*, 2025.
 - [92] Yupeng Hou, An Zhang, Leheng Sheng, Zhengyi Yang, Xiang Wang, Tat-Seng Chua, and Julian McAuley. Generative recommendation models: Progress and directions. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 13–16, 2025.
 - [93] Yupeng Hou, An Zhang, Leheng Sheng, Zhengyi Yang, Xiang Wang, Tat-Seng Chua, and Julian McAuley. Generative recommendation models: Progress and directions. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 13–16, 2025.
 - [94] Yupeng Hou, Haven Kim, Clark Mingxuan Ju, Eduardo Escoto, Neil Shah, and Julian McAuley. Expressiveness limits of autoregressive semantic id generation in generative recommendation. *arXiv preprint arXiv:2605.06331*, 2026.
 - [95] Yupeng Hou, Jiacheng Li, Xiangjun Fu, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation: Benchmarking llms as semantic encoders. In *ACL*, 2026.
 - [96] Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. How to index item ids for recommendation foundation models. In *SIGIR-AP*, pages 195–204, 2023.
 - [97] Junjie Huang, Duyu Tang, Wanjuan Zhong, Shuai Lu, Linjun Shou, Ming Gong, Daxin Jiang, and Nan Duan. Whiteningbert: An easy unsupervised sentence embedding approach. In *Findings of EMNLP*, pages 238–244, 2021.
 - [98] Hamish Ivison, Muru Zhang, Faeze Brahman, Pang Wei Koh, and Pradeep Dasigi. Large-scale data selection for instruction tuning. *arXiv preprint arXiv:2503.01807*, 2025.
 - [99] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.
 - [100] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary,

- Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [101] Chumeng Jiang, Jiayin Wang, Weizhi Ma, Charles L. A. Clarke, Shuai Wang, Chuhan Wu, and Min Zhang. Beyond utility: Evaluating LLM as recommender. In *WWW*, pages 3850–3862, 2025.
 - [102] Minhao Jiang, Ken Ziyu Liu, Ming Zhong, Rylan Schaeffer, Siru Ouyang, Jiawei Han, and Sanmi Koyejo. Investigating data contamination for pre-training language models. *arXiv preprint arXiv:2401.06059*, 2024.
 - [103] Bowen Jin, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. Multi-behavior recommendation with graph convolutional networks. In *SIGIR*, 2020.
 - [104] Bowen Jin, Hansi Zeng, Guoyin Wang, Xiusi Chen, Tianxin Wei, Ruirui Li, Zhengyang Wang, Zheng Li, Yang Li, Hanqing Lu, Suhang Wang, Jiawei Han, and Xianfeng Tang. Language models as semantic indexers. In *ICML*, 2024.
 - [105] Yilun Jin, Zheng Li, Chenwei Zhang, Tianyu Cao, Yifan Gao, Pratik Jayarao, Mao Li, Xin Liu, Ritesh Sarkhel, Xianfeng Tang, Haodong Wang, Zhengyang Wang, Wenju Xu, Jingfeng Yang, Qingyu Yin, Xian Li, Priyanka Nigam, Yi Xu, Kai Chen, Qiang Yang, Meng Jiang, and Bing Yin. Shopping MMLU: A massive multi-task online shopping benchmark for large language models. In *NeurIPS*, 2024.
 - [106] Clark Mingxuan Ju, Liam Collins, Leonardo Neves, Bhuvish Kumar, Louis Yufeng Wang, Tong Zhao, and Neil Shah. Generative recommendation with semantic ids: A practitioner’s handbook. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 6420–6425, 2025.
 - [107] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *RecSys*, page 43–50, 2016.
 - [108] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *ICDM*, pages 197–206, 2018.
 - [109] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.
 - [110] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.
 - [111] Haven Kim, Yupeng Hou, and Julian McAuley. Fusid: Modality-fused semantic ids for generative music recommendation. *arXiv preprint arXiv:2601.08764*, 2026.

- [112] Jinri Kim, Eungi Kim, Kwangeun Yeo, Yujin Jeon, Chanwoo Kim, Sewon Lee, and Joonseok Lee. Content-based graph reconstruction for cold-start item recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1263–1273, 2024.
- [113] Walid Krichene and Steffen Rendle. On sampled metrics for item recommendation. In *SIGKDD*, pages 1748–1757, 2020.
- [114] Artus Krohn-Grimberghe, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. Multi-relational matrix factorization using bayesian personalized ranking for social network data. In *WSDM*, 2012.
- [115] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *ACL*, pages 66–75, 2018.
- [116] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP*, 2018.
- [117] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham M. Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In *NeurIPS*, 2022.
- [118] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1073–1082, 2019.
- [119] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.
- [120] Guanghan Li, Xun Zhang, Yufei Zhang, Yifan Yin, Guojun Yin, and Wei Lin. Semantic convergence: Harmonizing recommender systems via two-stage alignment and behavioral semantic tokenization. In *AAAI*, 2025.
- [121] Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. Text is all you need: Learning language representations for sequential recommendation. In *SIGKDD*, pages 1258–1267, 2023.
- [122] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *CIKM*, pages 1419–1428, 2017.
- [123] Lei Li, Yongfeng Zhang, and Li Chen. Prompt distillation for efficient llm-based recommendation. In *CIKM*, pages 1348–1357, 2023.

- [124] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. Embedding-based product retrieval in taobao search. In *KDD*, pages 3181–3189, 2021.
- [125] Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. From matching to generation: A survey on generative information retrieval. *arXiv preprint arXiv:2404.14851*, 2024.
- [126] Yongqi Li, Xinyu Lin, Wenjie Wang, Fuli Feng, Liang Pang, Wenjie Li, Liqiang Nie, Xiangnan He, and Tat-Seng Chua. A survey of generative search and recommendation in the era of large language models. *arXiv preprint arXiv:2404.16924*, 2024.
- [127] Xinyu Lin, Chaoqun Yang, Wenjie Wang, Yongqi Li, Cunxiao Du, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Efficient inference for large language model-based generative recommendation. *arXiv preprint arXiv:2410.05165*, 2024.
- [128] Xinyu Lin, Haihan Shi, Wenjie Wang, Fuli Feng, Qifan Wang, See-Kiong Ng, and Tat-Seng Chua. Order-agnostic identifier for large language model-based generative recommendation. In *SIGIR*, 2025.
- [129] Xixun Lin, Jia Wu, Chuan Zhou, Shirui Pan, Yanan Cao, and Bin Wang. Task-adaptive neural process for user cold-start recommendation. In *Proceedings of the Web Conference 2021*, pages 1306–1316, 2021.
- [130] Enze Liu, Bowen Zheng, Cheng Ling, Lantao Hu, Han Li, and Wayne Xin Zhao. End-to-end learnable item tokenization for generative recommendation. In *SIGIR*, 2025.
- [131] Enze Liu, Bowen Zheng, Cheng Ling, Lantao Hu, Han Li, and Wayne Xin Zhao. Generative recommender with end-to-end learnable item tokenization. In *SIGIR*, pages 729–739, 2025.
- [132] Han Liu, Yinwei Wei, Xuemeng Song, Weili Guan, Yuan-Fang Li, and Liqiang Nie. Mmgrec: Multimodal generative recommendation with transformer model. *arXiv preprint arXiv:2404.16555*, 2024.
- [133] Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. In *COLM*, 2024.
- [134] Jiao Liu, Zhu Sun, Shanshan Feng, and Yew-Soon Ong. Language model evolutionary algorithms for recommender systems: Benchmarks and algorithm comparisons. *CoRR*, abs/2411.10697, 2024.
- [135] Jingzhe Liu, Liam Collins, Jiliang Tang, Tong Zhao, Neil Shah, and Clark Mingxuan Ju.

- Understanding generative recommendation with semantic ids from a model-scaling view. *arXiv preprint arXiv:2509.25522*, 2025.
- [136] Junling Liu, Chao Liu, Peilin Zhou, Renjie Lv, Kang Zhou, and Yan Zhang. Is chatgpt a good recommender? a preliminary study. *arXiv preprint arXiv:2304.10149*, 2023.
 - [137] Junling Liu, Chao Liu, Peilin Zhou, Qichen Ye, Dading Chong, Kang Zhou, Yueqi Xie, Yuwei Cao, Shoujin Wang, Chenyu You, and Philip S. Yu. Llmrec: Benchmarking large language models on recommendation task. *CoRR*, abs/2308.12241, 2023.
 - [138] Qijiong Liu, Jieming Zhu, Lu Fan, Kun Wang, Hengchang Hu, Wei Guo, Yong Liu, and Xiao-Ming Wu. Benchmarking llms in recommendation tasks: A comparative evaluation with conventional recommenders. *CoRR*, abs/2503.05493, 2025.
 - [139] Siwei Liu, Iadh Ounis, Craig Macdonald, and Zaiqiao Meng. A heterogeneous graph neural model for cold-start recommendation. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 2029–2032, 2020.
 - [140] Taichi Liu, Chen Gao, Zhenyu Wang, Dong Li, Jianye Hao, Depeng Jin, and Yong Li. Uncertainty-aware consistency learning for cold-start item recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2466–2470, 2023.
 - [141] Weiming Liu, Xiaolin Zheng, Jiajie Su, Longfei Zheng, Chaochao Chen, and Mengling Hu. Contrastive proxy kernel stein path alignment for cross-domain cold-start recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11216–11230, 2023.
 - [142] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.
 - [143] Zihan Liu, Yupeng Hou, and Julian McAuley. Multi-behavior generative recommendation. In *CIKM*, 2024.
 - [144] Zihan Liu, Yupeng Hou, and Julian McAuley. Multi-behavior generative recommendation. In *CIKM*, 2024.
 - [145] Xinchun Luo, Jiangxia Cao, Tianyu Sun, Jinkai Yu, Rui Huang, Wei Yuan, Hezheng Lin, Yichen Zheng, Shiyao Wang, Qigen Hu, et al. Qarm: Quantitative alignment multi-modal recommendation at kuaishou. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 5915–5922, 2025.
 - [146] Chen Ma, Peng Kang, and Xue Liu. Hierarchical gating networks for sequential recom-

- mendation. In *KDD*, 2019.
- [147] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, pages 43–52, 2015.
 - [148] Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. SFR-Embedding-Mistral: Enhance Text Retrieval with Transfer Learning. <https://www.salesforce.com/blog/sfr-embedding/>, 2024.
 - [149] Aditya Menon, Sadeep Jayasumana, Ankit Singh Rawat, Seungyeon Kim, Sashank Reddi, and Sanjiv Kumar. In defense of dual-encoders for neural ranking. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 15376–15400. PMLR, 17–23 Jul 2022. URL <https://proceedings.mlr.press/v162/menon22a.html>.
 - [150] Meta AI. Introducing Meta Llama 3: The most capable openly available LLM to date, 2024. URL <https://ai.meta.com/blog/meta-llama-3/>.
 - [151] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
 - [152] Raymond J Mooney and Lorie Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204, 2000.
 - [153] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2021. URL <https://arxiv.org/abs/1810.02244>.
 - [154] Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. In *EACL*, pages 2014–2037, 2023.
 - [155] Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *ICLR*, 2025.
 - [156] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP*, pages 188–197, 2019.
 - [157] Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B. Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In *Findings of ACL*, pages 1864–1874, 2022.
 - [158] Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernandez Abrego, Ji Ma, Vincent

- Zhao, Yi Luan, Keith Hall, Ming-Wei Chang, et al. Large dual encoders are generalizable retrievers. In *EMNLP*, pages 9844–9855, 2022.
- [159] Yongxin Ni, Yu Cheng, Xiangyan Liu, Junchen Fu, Youhua Li, Xiangnan He, Yongfeng Zhang, and Fajie Yuan. A content-driven micro-video recommendation dataset at scale. *arXiv preprint arXiv:2309.15379*, 2023.
- [160] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. From doc2query to docttttquery. *Online preprint*, 6(2), 2019.
- [161] Naoto Ohsaka and Riku Togashi. Curse of “low” dimensionality in recommender systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’23, page 537–547. ACM, July 2023. doi: 10.1145/3539618.3591659. URL <http://dx.doi.org/10.1145/3539618.3591659>.
- [162] OpenAI. Introducing ChatGPT, 2022. URL <https://openai.com/index/chatgpt/>.
- [163] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32, 2019.
- [164] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web: methods and strategies of web personalization*, pages 325–341. Springer, 2007.
- [165] Aleksandr V Petrov and Craig Macdonald. Generative sequential recommendation with gptrec. *arXiv preprint arXiv:2306.11114*, 2023.
- [166] Aleksandr V Petrov and Craig Macdonald. Recjqpq: training large-catalogue sequential recommenders. In *WSDM*, 2024.
- [167] Pouya Pezeshkpour, Sarthak Jain, Byron C. Wallace, and Sameer Singh. An empirical comparison of instance attribution methods for NLP. In *NAACL*, pages 967–975, 2021.
- [168] Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation, 2018. URL <https://arxiv.org/abs/1804.06609>.
- [169] Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. Bpe-dropout: Simple and effective subword regularization. In *ACL*, pages 1882–1892, 2020.
- [170] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

- [171] Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. Contrastive learning for representation degeneration problem in sequential recommendation. In *WSDM*, 2022.
- [172] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.
- [173] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67, 2020.
- [174] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Mahesh Sathiamoorthy. Recommender systems with generative retrieval. In *NeurIPS*, 2023.
- [175] Vikas Raunak, Arul Menezes, and Marcin Junczys-Dowmunt. The curious case of hallucinations in neural machine translation. *arXiv preprint arXiv:2104.06683*, 2021.
- [176] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *NeurIPS*, 2019.
- [177] Chandan K Reddy, Lluís Màrquez, Fran Valero, Nikhil Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. Shopping queries dataset: A large-scale esci benchmark for improving product search. *arXiv preprint arXiv:2206.06588*, 2022.
- [178] Nils Reimers and Iryna Gurevych. The curse of dense low-dimensional information retrieval for large index sizes. In *ACL*, pages 605–611, 2021.
- [179] Xubin Ren and Chao Huang. Easyrec: Simple yet effective language models for recommendation. *arXiv preprint arXiv:2408.08821*, 2024.
- [180] Xubin Ren, Wei Wei, Lianghao Xia, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. Representation learning with large language models for recommendation. In *WWW*, pages 3464–3475, 2024.
- [181] Steffen Rendle. Factorization machines. In *ICDM*, pages 995–1000. IEEE, 2010.
- [182] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.

- [183] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, 2010.
- [184] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2010.
- [185] Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. Tokenization is more than compression. In *EMNLP*, 2024.
- [186] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725, 2016.
- [187] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *KDD*, pages 255–262, 2016.
- [188] Yifei Shen, Yongji Wu, Yao Zhang, Caihua Shan, Jun Zhang, B Khaled Letaief, and Dongsheng Li. How powerful is graph convolution for recommendation? In *CIKM*, pages 1619–1629, 2021.
- [189] Leheng Sheng, An Zhang, Yi Zhang, Yuxin Chen, Xiang Wang, and Tat-Seng Chua. Language models encode collaborative signals in recommendation. In *ICLR*, 2024.
- [190] Leheng Sheng, An Zhang, Yi Zhang, Yuxin Chen, Xiang Wang, and Tat-Seng Chua. Language representations can be what recommenders need: Findings and potentials. In *ICLR*, 2025.
- [191] Zihua Si, Zhongxiang Sun, Jiale Chen, Guozhang Chen, Xiaoxue Zang, Kai Zheng, Yang Song, Xiao Zhang, Jun Xu, and Kun Gai. Generative retrieval with semantic tree-structured item identifiers via contrastive learning. In *SIGIR-AP*, 2024.
- [192] Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.
- [193] Anima Singh, Trung Vu, Nikhil Mehta, Raghunandan Keshavan, Maheswaran Sathiamoorthy, Yilin Zheng, Lichan Hong, Lukasz Heldt, Li Wei, Devansh Tandon, Ed H. Chi, and Xinyang Yi. Better generalization with semantic ids: A case study in ranking for recommendations. In *RecSys*, 2024.
- [194] Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Serkan Ö. Arik, Danqi Chen, and Tao Yu. BRIGHT: A realistic and challenging benchmark for reasoning-intensive retrieval. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

- [195] Jiajie Su, Chaochao Chen, Zibin Lin, Xi Li, Weiming Liu, and Xiaolin Zheng. Personalized behavior-aware transformer for multi-behavior sequential recommendation. In *Proceedings of the 31st ACM International Conference on Multimedia*, 2023.
- [196] Jianlin Su, Jiarun Cao, Weijie Liu, and Yangyiwen Ou. Whitening sentence representations for better semantics and faster retrieval. *CoRR*, abs/2103.15316, 2021.
- [197] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*, pages 1441–1450, 2019.
- [198] Ruixuan Sun, Xinyi Li, Avinash Akella, and Joseph A Konstan. Large language models as conversational movie recommenders: A user study. *arXiv preprint arXiv:2404.19093*, 2024.
- [199] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.
- [200] Juntao Tan, Shuyuan Xu, Wenyue Hua, Yingqiang Ge, Zelong Li, and Yongfeng Zhang. Idgenrec: Llm-recsys alignment with textual id learning. In *SIGIR*, 2024.
- [201] Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016.
- [202] Jiayi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*, 2018.
- [203] Jie Tang, Sen Wu, Jimeng Sun, and Hang Su. Cross-domain collaboration recommendation. In *KDD*, pages 1285–1293, 2012.
- [204] Yi Tay, Vinh Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Prakash Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. Transformer memory as a differentiable search index. In *NeurIPS*, 2022.
- [205] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.
- [206] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

2023.

- [207] Roman Vashurin, Ekaterina Fadeeva, Artem Vazhentsev, Lyudmila Rvanova, Daniil Vasilev, Akim Tsvigun, Sergey Petrakov, Rui Xing, Abdelrahman Sadallah, Kirill Grishchenkov, et al. Benchmarking uncertainty quantification methods for large language models with lm-polygraph. *Transactions of the Association for Computational Linguistics*, 13:220–248, 2025.
- [208] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, volume 30, 2017.
- [209] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. Dropoutnet: Addressing cold start in recommender systems. *Advances in neural information processing systems*, 30, 2017.
- [210] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456, 2011.
- [211] Dongsheng Wang, Yuxi Huang, Shen Gao, Yifan Wang, Chengrui Huang, and Shuo Shang. Generative next poi recommendation with semantic id. In *KDD*, 2025.
- [212] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *KDD*, pages 1235–1244, 2015.
- [213] Huanjie Wang, Xinchun Luo, Honghui Bao, Zhang Zixing, Lejian Ren, Yunfan Wu, Hongwei Zhang, Liwei Guan, and Guang Chen. Pit: A dynamic personalized item tokenizer for end-to-end generative recommendation. *arXiv preprint arXiv:2602.08530*, 2026.
- [214] Jiangyuan Wang, Kejun Xiao, Qi Sun, Huaipeng Zhao, Tao Luo, Jiandong Zhang, and Xiaoyi Zeng. Shoppingbench: A real-world intent-grounded shopping benchmark for llm-based agents. *CoRR*, abs/2508.04266, 2025.
- [215] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In *ACL*, pages 11897–11916, 2024.
- [216] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD’17*, pages 1–7. 2017.
- [217] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *WWW*, pages 1785–1797, 2021.

- [218] Tianxin Wang, Fuzhen Zhuang, Zhiqiang Zhang, Daixin Wang, Jun Zhou, and Qing He. Low-dimensional alignment for cross-domain recommendation. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 3508–3512, 2021.
- [219] Wenbo Wang, Ben Chen, Bingquan Liu, Xinxin Wang, Luwei Yang, Wen Jiang, Wei Ning, and Jian Guan. Mutual information assisted graph convolution network for cold-start recommendation. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6785–6789. IEEE, 2024.
- [220] Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Learnable tokenizer for llm-based generative recommendation. In *CIKM*, 2024.
- [221] Xinyi Wang, Antonis Antoniadis, Yanai Elazar, Alfonso Amayuelas, Alon Albalak, Kexun Zhang, and William Yang Wang. Generalization vs memorization: Tracing language models’ capabilities back to pretraining data. *arXiv preprint arXiv:2407.14985*, 2024.
- [222] Xinyuan Wang, Liang Wu, Liangjie Hong, Hao Liu, and Yanjie Fu. Llm-enhanced user-item interactions: Leveraging edge information for optimized recommendations. *arXiv preprint arXiv:2402.09617*, 2024.
- [223] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.
- [224] Ye Wang, Jiahao Xun, Minjie Hong, Jieming Zhu, Tao Jin, Wang Lin, Haoyuan Li, Linjun Li, Yan Xia, Zhou Zhao, and Zhenhua Dong. Eager: Two-stream generative recommender with behavior-semantic collaboration. In *KDD*, page 3245–3254, 2024.
- [225] Yidan Wang, Zhaochun Ren, Weiwei Sun, Jiyuan Yang, Zhixiang Liang, Xin Chen, Ruobing Xie, Su Yan, Xu Zhang, Pengjie Ren, Zhumin Chen, and Xin Xin. Content-based collaborative generation for recommender systems. In *CIKM*, 2024.
- [226] Yidan Wang, Zhaochun Ren, Weiwei Sun, Jiyuan Yang, Zhixiang Liang, Xin Chen, Ruobing Xie, Su Yan, Xu Zhang, Pengjie Ren, et al. Enhanced generative recommendation via content and collaboration integration. *arXiv preprint arXiv:2403.18480*, 2024.
- [227] Yuhao Wang, Junwei Pan, Xinhang Li, Maolin Wang, Yuan Wang, Yue Liu, Dapeng Liu, Jie Jiang, and Xiangyu Zhao. Empowering large language model for sequential recommendation via multimodal embeddings and semantic ids. In *CIKM*, pages 3209–3219, 2025.
- [228] Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen,

- Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, et al. A neural corpus indexer for document retrieval. In *NeurIPS*, 2022.
- [229] Zilong Wang, Zifeng Wang, Long Le, Huaixiu Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, et al. Speculative rag: Enhancing retrieval augmented generation through drafting. *arXiv preprint arXiv:2407.08223*, 2024.
- [230] Wei Wei, Xubin Ren, Jiabin Tang, Qinyong Wang, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. Llmrec: Large language models with graph augmentation for recommendation. In *WSDM*, pages 806–815, 2024.
- [231] Yinwei Wei, Xiang Wang, Qi Li, Liqiang Nie, Yan Li, Xuanping Li, and Tat-Seng Chua. Contrastive learning for cold-start recommendation. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 5382–5390, 2021.
- [232] Zhipeng Wei, Kuo Cai, Junda She, Jie Chen, Minghao Chen, Yang Zeng, Qiang Luo, Wencong Zeng, Ruiming Tang, Kun Gai, et al. Oneloc: Geo-aware generative recommender systems for local life service. *arXiv preprint arXiv:2508.14646*, 2025.
- [233] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.
- [234] Orion Weller, Michael Boratko, Iftekhhar Naim, and Jinhyuk Lee. On the theoretical limitations of embedding-based retrieval, 2025. URL <https://arxiv.org/abs/2508.21038>.
- [235] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *EMNLP*, 2020.
- [236] Junda Wu, Cheng-Chun Chang, Tong Yu, Zhankui He, Jianing Wang, Yupeng Hou, and Julian McAuley. Coral: collaborative retrieval-augmented large language models improve long-tail recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3391–3401, 2024.
- [237] Le Wu, Yonghui Yang, Lei Chen, Defu Lian, Richang Hong, and Meng Wang. Learning to transfer graph embeddings for inductive graph based recommendation. In *SIGIR*, pages 1211–1220, 2020.
- [238] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Junchi Yan, and Hongyuan Zha. Towards open-world recommendation: An inductive model-based collaborative filtering approach. In *ICML*, pages 11329–11339. PMLR, 2021.

- [239] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *AAAI*, 2019.
- [240] Tao Wu, Ellie Ka-In Chio, Heng-Tze Cheng, Yu Du, Steffen Rendle, Dima Kuzmin, Ritesh Agarwal, Li Zhang, John Anderson, Sarvjeet Singh, et al. Zero-shot heterogeneous transfer learning from recommender systems to cold-start search retrieval. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2821–2828, 2020.
- [241] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [242] Zhenchao Wu and Xiao Zhou. M2eu: Meta learning for cold-start recommendation via enhancing user preference estimation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1158–1167, 2023.
- [243] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.
- [244] Lianghao Xia, Chao Huang, Yong Xu, Peng Dai, Bo Zhang, and Liefeng Bo. Multiplex behavioral relation learning for recommendation via memory augmented transformer network. In *SIGIR*, 2020.
- [245] Lianghao Xia, Yong Xu, Chao Huang, Peng Dai, and Liefeng Bo. Graph meta network for multi-behavior recommendation. In *SIGIR*, pages 757–766, 2021.
- [246] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.
- [247] Bin Xiao, Chunan Shi, Xiaonan Nie, Fan Yang, Xiangwei Deng, Lei Su, Weipeng Chen, and Bin Cui. Clover: Regressive lightweight speculative decoding with sequential knowledge. *arXiv preprint arXiv:2405.00263*, 2024.
- [248] Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. On memorization of large language models in logical reasoning. 2024. URL <https://arxiv.org/abs/2410.23123>.

- [249] Ruobing Xie, Qi Liu, Liangdong Wang, Shukai Liu, Bo Zhang, and Leyu Lin. Contrastive cross-domain recommendation in matching. In *KDD*, pages 4226–4236, 2022.
- [250] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. Contrastive learning for sequential recommendation. In *ICDE*, 2022.
- [251] Zhouhang Xie, Bo Peng, Zhankui He, Ziqi Chen, Alice Han, Isabella Ye, Benjamin Coleman, Naveen Sachdeva, Fernando Pereira, Julian McAuley, et al. Agentictagger: Structured item representation for recommendation with llm agents. *arXiv preprint arXiv:2602.05945*, 2026.
- [252] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019. URL <https://arxiv.org/abs/1810.00826>.
- [253] Yi Xu, Moyu Zhang, Chenxuan Li, Zhihao Liao, Haibo Xing, Hao Deng, Jinxin Hu, Yu Zhang, Xiaoyi Zeng, and Jing Zhang. Mmq: Multimodal mixture-of-quantization tokenization for semantic id generation and user behavioral adaptation. *arXiv preprint arXiv:2508.15281*, 2025.
- [254] An Yan, Zhankui He, Jiacheng Li, Tianyang Zhang, and Julian McAuley. Personalized showcases: Generating multi-modal explanations for recommendations. In *SIGIR*, pages 2251–2255, 2023.
- [255] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [256] Chen Yang, Yupeng Hou, Yang Song, Tao Zhang, Ji-Rong Wen, and Wayne Xin Zhao. Modeling two-way selection preference for person-job fit. In *RecSys*, 2022.
- [257] Liu Yang, Fabian Paischer, Kaveh Hassani, Jiacheng Li, Shuai Shao, Zhang Gabriel Li, Yun He, Xue Feng, Nima Noorshams, Sem Park, et al. Unifying generative and dense retrieval for sequential recommendation. *arXiv preprint arXiv:2411.18814*, 2024.
- [258] Yuhao Yang, Chao Huang, Lianghao Xia, Yuxuan Liang, Yanwei Yu, and Chenliang Li. Multi-behavior hypergraph-enhanced transformer for sequential recommendation. In *SIGKDD*, 2022.
- [259] Zitong Yang, Michal Lukasik, Vaishnavh Nagarajan, Zonglin Li, Ankit Rawat, Manzil Zaheer, Aditya K Menon, and Sanjiv Kumar. Resmem: Learn what you can and memorize the rest. *Advances in Neural Information Processing Systems*, 36:60768–60790, 2023.
- [260] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022.

- [261] Wencai Ye, Mingjie Sun, Shaoyun Shi, Peng Wang, Wenjin Wu, and Peng Jiang. Das: Dual-aligned semantic ids empowered industrial recommender system. In *CIKM*, pages 6217–6224, 2025.
- [262] Shaked Yehezkel and Yuval Pinter. Incorporating context into subword vocabularies. In *EACL*, pages 623–635, 2023.
- [263] Yelp Inc. Yelp Dataset. <https://business.yelp.com/data/resources/open-dataset/>, 2020.
- [264] Enming Yuan, Wei Guo, Zhicheng He, Huifeng Guo, Chengkai Liu, and Ruiming Tang. Multi-behavior sequential transformer recommender. In *SIGIR*, 2022.
- [265] Guanghu Yuan, Fajie Yuan, Yudong Li, Beibei Kong, Shujie Li, Lei Chen, Min Yang, Chenyun Yu, Bo Hu, Zang Li, et al. Tenrec: A large-scale multipurpose benchmark dataset for recommender systems. In *NeurIPS*, volume 35, pages 11480–11493, 2022.
- [266] Zheng Yuan, Fajie Yuan, Yu Song, Youhua Li, Junchen Fu, Fei Yang, Yunzhu Pan, and Yongxin Ni. Where to go next for recommender systems? id-vs. modality-based recommender models revisited. In *SIGIR*, 2023.
- [267] Zhenrui Yue, Yueqi Wang, Zhankui He, Huimin Zeng, Julian J. McAuley, and Dong Wang. Linear recurrent units for sequential recommendation. In *WSDM*, 2024.
- [268] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2021.
- [269] Jianyang Zhai, Zi-Feng Mai, Chang-Dong Wang, Feidiao Yang, Xiawu Zheng, Hui Li, and Yonghong Tian. Multimodal quantitative language for generative recommendation. *arXiv preprint arXiv:2504.05314*, 2025.
- [270] Jiaqi Zhai, Zhaojie Gong, Yueming Wang, Xiao Sun, Zheng Yan, Fu Li, and Xing Liu. Revisiting neural retrieval on accelerators. In *SIGKDD*, pages 5520–5531, 2023.
- [271] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *ICML*, 2024.
- [272] Chao Zhang, Haoxin Zhang, Shiwei Wu, Di Wu, Tong Xu, Yan Gao, Yao Hu, and Enhong Chen. Notellm-2: Multimodal large representation models for recommendation. *arXiv preprint arXiv:2405.16789*, 2024.
- [273] Chiyuan Zhang, Daphne Ippolito, Katherine Lee, Matthew Jagielski, Florian Tramèr, and

- Nicholas Carlini. Counterfactual memorization in neural language models. *Advances in Neural Information Processing Systems*, 36:39321–39362, 2023.
- [274] Fuwei Zhang, Xiaoyu Liu, Dongbo Xi, Jishen Yin, Huan Chen, Peng Yan, Fuzhen Zhuang, and Zhao Zhang. Multi-aspect cross-modal quantization for generative recommendation, 2025. URL <https://arxiv.org/abs/2511.15122>.
- [275] Gaowei Zhang, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, and Ji-Rong Wen. Scaling law of large sequential recommendation models. In *RecSys*, 2024.
- [276] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. Recommendation as instruction following: A large language model empowered recommendation approach. *ACM Trans. Inf. Syst.*, 2025.
- [277] Mingzhu Zhang, Ruiping Yin, Zhen Yang, Yipeng Wang, and Kan Li. Advances and challenges of multi-task learning method in recommender system: A survey. *arXiv preprint arXiv:2305.13843*, 2023.
- [278] Ruohan Zhang, Jiacheng Li, Julian McAuley, and Yupeng Hou. Purely semantic indexing for LLM-based generative recommendation and retrieval. *arXiv preprint arXiv:2509.16446*, 2025.
- [279] Taolin Zhang, Junwei Pan, Jinpeng Wang, Yaohua Zha, Tao Dai, Bin Chen, Ruisheng Luo, Xiaoxiang Deng, Yuan Wang, Ming Yue, et al. Towards scalable semantic representation for recommendation. *arXiv preprint arXiv:2410.09560*, 2024.
- [280] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, Xiaofang Zhou, et al. Feature-level deeper self-attention network for sequential recommendation. In *IJCAI*, pages 4320–4326, 2019.
- [281] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, Xiaofang Zhou, et al. Feature-level deeper self-attention network for sequential recommendation. In *IJCAI*, 2019.
- [282] Weifeng Zhang, Jingwen Mao, Yi Cao, and Congfu Xu. Multiplex graph neural networks for multi-behavior recommendation. In *CIKM*, pages 2313–2316, 2020.
- [283] Weizhi Zhang, Yuanchen Bei, Liangwei Yang, Henry Peng Zou, Peilin Zhou, Aiwei Liu, Yinghui Li, Hao Chen, Jianling Wang, Yu Wang, et al. Cold-start recommendation towards the era of large language models (llms): A comprehensive survey and roadmap. *arXiv preprint arXiv:2501.01945*, 2025.
- [284] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3

- embedding: Advancing text embedding and reranking through foundation models. *CoRR*, abs/2506.05176, 2025.
- [285] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, et al. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In *CIKM*, pages 4653–4664, 2021.
 - [286] Wayne Xin Zhao, Yupeng Hou, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, et al. Recbole 2.0: Towards a more up-to-date recommendation library. In *CIKM*, pages 4722–4726, 2022.
 - [287] Wayne Xin Zhao, Zihan Lin, Zhichao Feng, Pengfei Wang, and Ji-Rong Wen. A revisiting study of appropriate offline evaluation for top-n recommendation algorithms. *ACM Transactions on Information Systems*, 41(2):1–41, 2022.
 - [288] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.
 - [289] Zhe Zhao, Zhiyuan Cheng, Lichan Hong, and Ed H Chi. Improving user topic interest profiles by behavior factorization. In *WWW*, pages 1406–1416, 2015.
 - [290] Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. Adapting large language models by integrating collaborative semantics for recommendation. In *ICDE*, pages 1435–1448, 2024.
 - [291] Bowen Zheng, Enze Liu, Zhongfu Chen, Zhongrui Ma, Yue Wang, Wayne Xin Zhao, and Ji-Rong Wen. Pre-training generative recommender with multi-identifier item tokenization. *arXiv preprint arXiv:2504.04400*, 2025.
 - [292] Bowen Zheng, Hongyu Lu, Yu Chen, Wayne Xin Zhao, and Ji-Rong Wen. Universal item tokenization for transferable generative recommendation. *arXiv preprint arXiv:2504.04405*, 2025.
 - [293] Bowen Zheng, Junjie Zhang, Hongyu Lu, Yu Chen, Ming Chen, Wayne Xin Zhao, and Ji-Rong Wen. Enhancing graph contrastive learning with reliable and informative augmentation for recommendation. In *KDD*, 2025.
 - [294] Jiawei Zheng, Qianli Ma, Hao Gu, and Zhenjing Zheng. Multi-view denoising graph auto-encoders on heterogeneous information networks for cold-start recommendation. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data*

mining, pages 2338–2348, 2021.

- [295] Yujia Zheng, Siyi Liu, Zekun Li, and Shu Wu. Cold-start sequential recommendation via meta learner. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4706–4713, 2021.
- [296] Qiyong Zhong, Jiajie Su, Yunshan Ma, Julian McAuley, and Yupeng Hou. Pctx: Tokenizing personalized context for generative recommendation. *arXiv preprint arXiv:2510.21276*, 2025.
- [297] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *CIKM*, pages 1893–1902, 2020.
- [298] Kun Zhou, Hui Yu, Wayne Xin Zhao, and Ji-Rong Wen. Filter-enhanced MLP is all you need for sequential recommendation. In *TheWebConf*, 2022.
- [299] Zhihui Zhou, Lilin Zhang, and Ning Yang. Contrastive collaborative filtering for cold-start item recommendation. In *Proceedings of the ACM Web Conference 2023*, pages 928–937, 2023.
- [300] Feng Zhu, Chaochao Chen, Yan Wang, Guanfeng Liu, and Xiaolin Zheng. Dtcdr: A framework for dual-target cross-domain recommendation. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1533–1542, 2019.
- [301] Feng Zhu, Yan Wang, Chaochao Chen, Jun Zhou, Longfei Li, and Guanfeng Liu. Cross-domain recommendation: challenges, progress, and prospects. *arXiv preprint arXiv:2103.01696*, 2021.
- [302] Jieming Zhu, Mengqun Jin, Qijiong Liu, Zexuan Qiu, Zhenhua Dong, and Xiu Li. Cost: Contrastive quantization based semantic tokenization for generative recommendation. In *RecSys*, 2024.
- [303] Jing Zhu, Mingxuan Ju, Yozen Liu, Danai Koutra, Neil Shah, and Tong Zhao. Beyond unimodal boundaries: Generative recommendation with multimodal semantics. *arXiv preprint arXiv:2503.23333*, 2025.
- [304] Ziwei Zhu, Shahin Sefati, Parsa Saadatpanah, and James Caverlee. Recommendation for new users and new items via randomized training and mixture-of-experts transformation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1121–1130, 2020.

- [305] Shengyao Zhuang, Hang Li, and Guido Zuccon. Deep query likelihood model for information retrieval. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28–April 1, 2021, Proceedings, Part II* 43, pages 463–470. Springer, 2021.
- [306] Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. Open-source large language models are strong zero-shot query likelihood models for document ranking. *arXiv preprint arXiv:2310.13243*, 2023.
- [307] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.
- [308] Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In *Findings of ACL*, pages 598–614, 2023.