Personalized Dense Retrieval on Global Index for Voice-enabled Conversational Systems

Masha Belyi, Charlotte Dzialo, Chaitanya Dwivedi
Prajit Reddy Muppidi, Kanna Shimizu

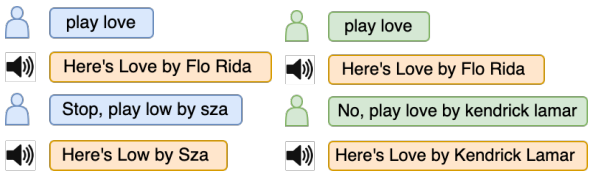
Amazon Alexa AI
{mashb, dzialocd, dwcahit, prajim, kannashi}@amazon.com

Abstract

Voice-controlled AI dialogue systems are susceptible to noise from phonetic variations and failure to resolve ambiguous entities. Typically, personalized entity resolution (ER) and/or query rewrites (QR) are deployed to recover from these error modes. Previous work in this field achieves personalization by constraining retrieval search space to personalized indices built from user's historical interactions with the device. While constrained retrieval achieves high precision, predictions are limited to entities in recent user history, which offers low coverage of future requests. Further, maintaining individual indices for a large number of users is memory intensive and difficult to scale. In this work, we propose a personalized entity retrieval system that is robust to phonetic noise and ambiguity but is not limited to a personalized index. We achieve this by embedding user listening preferences into a contextual query embedding used in retrieval. We demonstrate our model's ability to correct multiple error modes and show 91% improvement over baseline on the entity retrieval task. Finally, we optimize the end-to-end approach to fit within online latency constraints while maintaining gains in performance.

1 Introduction

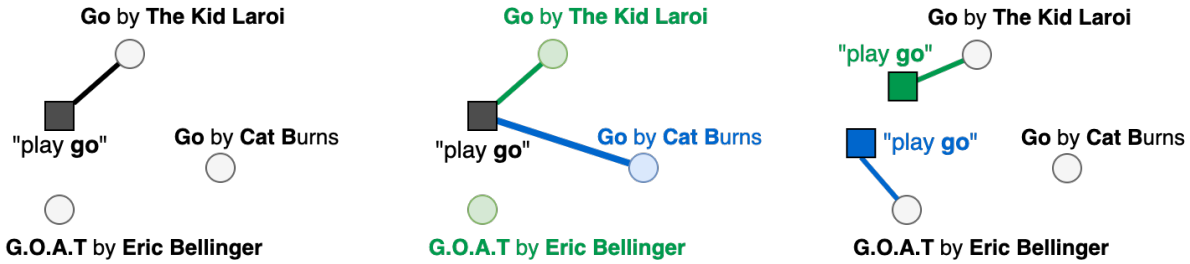
As conversational AI agents assert a ubiquitous presence in millions of households, the expectation for a seamless user experience grows. Users expect the AI agent to understand natural language queries and diverse accents, remember individual preferences, and function well in noisy environments. However, some interactions lead to user friction where a user does not get what they request. Friction primarily arises from (1) system errors and (2) ambiguity. System errors accumulate across various stages of the spoken dialog system pipeline, such as Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), and more. For example, the ASR model may confuse the user request "play low" with a phonetically similar request "play love" (Figure 1a). Ambiguity arises when a user's input is unclear due to abbreviations, or lack of context. A common scenario in the music domain is when a user requests a song without specifying the artist. As illustrated in Figure 1b, without context into a user's listening preferences, the system struggles to deliver the user's preferred track due to numerous matching entities in the catalog. These scenarios result in prolonged interactions where a user reformulates the query or abandons the request all together.



(a) ASR system error (b) Ambiguity error

Figure 1: Example (a) phonetically and (b) contextually ambiguous queries that require user reformulation for the system to resolve the request correctly.

In practice, these challenges are typically addressed through Query Rewriting (QR) (Ponnusamy et al., 2020; Chen et al., 2020b) and building robust Entity Resolution (Zhou et al., 2022) components. In search-based approaches, queries and candidate target entities are embedded in latent space where vector search is performed to retrieve the most relevant target per query. In QR, the output space is formed by historical user requests; in ER it comprises catalog entities. Personalization and contextualization are key for high-precision retrieval in entity-centric domains (Cho et al., 2021; Uma Naresh et al., 2022). Current approaches rely on user-specific indices to constrain the output search space to requests/entities a user has associated with in the past to achieve personalization. However, personalized indices offer low coverage of future queries (Uma Naresh et al., 2022), since



(a) Global index (b) Pesonalized Index (c) Our Approach

Figure 2: Different approaches to search-based retrieval visualized in a 2-d projection. Circles represent indexed entities and squares represent queries that are embedded in the same search space. Lines connect the query with its nearest entity neighbor. In (a) search is performed on a global index; the nearest entity to the query is retrieved. The output is agnostic to user preferences. In (b) search is performed on personalized indices; different entities are retrieved for users A (green) and B (blue) due to different index compositions. Our approach is illustrated in (c), where personalization is achieved via contextualized query embeddings, such that semantically identical queries may be positioned differently for different users in the embedding space.

users regularly explore new content through novel queries.

In this work we present an efficient personalized retrieval system that extends beyond the personalized index. Rather than using historical interactions to constrain the output (Cho et al., 2021), we embed them into continuous representations to form user embeddings that become inputs into the retrieval model. The user embeddings are joined with semantic query representations to form a contextual embedding that is subsequently used in retrieval. Figure 2 visually differentiates our work from existing global and personalized search-based retrieval approaches.

To form the user embeddings, we propose entity2vec; a domain-aware continuous entity representation learning method that captures item similarities beyond semantics and phonetics. This differentiates our approach from previous work in contextualization (Hao et al., 2022), where multi-turn dialogues are concatenated with the query as input into a semantic encoder. To illustrate, consider a sequence of queries in a user session:

"play dancing queen by abba"
"play i will survive by gloria gaynor"
"play bad girls"

Previous approach cannot leverage semantic signal in the sequence to disambiguate the final request for "bad girls", whereas our domain-aware embeddings would derive that user likes 70's disco music and resolve it to *Bad Girls by Donna Summers* as opposed to the more recent and popular song *Bad Girls by MIA*.

In Section 6, we demonstrate that our approach improves by 91% over an index-based personalized baseline. Further, we explain how we optimize the end-to-end system for runtime deployment.

2 Related Work

Query Rewriting (QR) in voice-enabled conversational AI systems is a popular way to refine ASR output into forms that can be accurately handled by downstream systems (Ponnusamy et al., 2020; Fan et al., 2021; Cho et al., 2021). Ponnusamy et al. (2020) propose rewrites based on a Markov Chain model trained on historical user reformulation patterns. Chen et al. (2020b) re-frame the problem as neural retrieval where queries and rewrite candidates are jointly encoded in vector space, followed by nearest-neighbor search on the query. The embedding-based search enables generalization to previously unseen queries. Fan et al. (2021) and Cho et al. (2021) improve precision by explicitly modeling diverse user preferences through personalized indices. However, the index is constrained to historical interactions which results in low recall ceiling when users request for new entities (Uma Naresh et al., 2022). Collaborative filtering to diversify the index is suggested in (Uma Naresh et al., 2022), but index size is still limited by memory constraints. Our approach overcomes this limitation by expanding search to the full catalog, while maintaining high precision and personalization power.

Our model architecture is inspired by Cho et al. (2021) and Zhou et al. (2022), who fuse embeddings from multiple sources in encoder-based QR

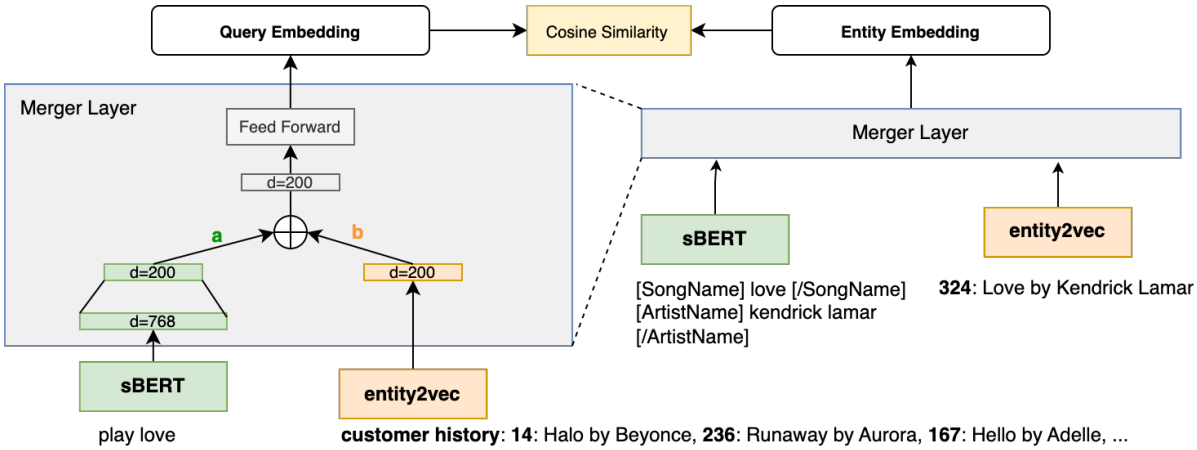


Figure 3: Dual encoder setup with shared encoder architecture. Semantic and domain signals are embedded with pre-trained encoders (SBERT and entity2vec). On the query side (left) we embed the query text and up to 50 recent entities that user interacted with. On the entity side (right), we embed the entity tokens and fetch the entity2vec embedding. Component embeddings are combined in the merger layer to form the final representation of query and entity. Model weights are optimized to maximize cosine similarity between groundtruth *(query,entity)* pairs.

and ER systems. We take a similar approach to merge user and query embeddings in our encoder, which we describe in section 3.2.

3 Model

We frame the problem as entity retrieval. Following a dual-encoder framework for dense retrieval (Gillick et al., 2019), we learn vector representations of queries and entities in a joint space. The encoder is optimized with a contrastive learning objective (Chen et al., 2020a), ensuring that queries and relevant entities are embedded closely in the latent vector space. At inference, we leverage FAISS (Johnson et al., 2019) to perform nearest neighbor search on a pre-computed global entity index to retrieve the closest candidates for an input query.

3.1 Encoder Architecture

Figure 3 shows our two-part encoder architecture. We detail each component of the encoder below.

Semantic Encoder. We leverage SBERT (Reimers and Gurevych, 2019), a pre-trained sentence encoder, to derive semantic representations of user queries. As in the original paper, we apply mean pooling on the token outputs of SBERT to form a 768-dimensional embedding for each input query and entity. The pre-trained SBERT is fine-tuned on domain data as described in Section 5.1.

Entity Encoder for Personalization. We leverage user interaction patterns to learn domain-aware entity representations. The goal of these representations is to capture domain knowledge, such that similar entities lie close together in the embedding space. Following the intuition of word2vec (Mikolov et al., 2013), we hypothesize that songs that appear together frequently in user-sessions are similar across some dimension. We propose entity2vec, a modified word2vec skip-gram model that operates at the entity level across user listening sessions instead of word level across sentences. Specifically, the model is trained to maximize the cosine similarity between target and context entities that appear together in user playback sessions. We train entity2vec with Gensim[1]. In Table 1 we qualitatively evaluate the resulting embeddings by inspecting nearest neighbors of select popular tracks and find that music entities from similar artists and genres have high cosine similarity.

In the dual-encoder model (Figure 3), entity2vec input on the entity side is a unique global catalogId. On the query side, we embed a maximum of 50 entities the user has recently engaged with, and compute their mean to generate what we term as a "user embedding". The motivation is that user embeddings should capture user listening preferences.

3.2 Merger Layer

The Merger Layer combines semantic and user/entity embeddings in the encoder model. We experiment with weighted sum fusion as in (Liu et al., 2018; Zhou et al., 2022) and concatenation

[1]https://radimrehurek.com/gensim/models/word2vec.html. Training parameters: dim=200, window_size=5, learning_rate=0.0025, negative_counts=5

Anchor	Nearest Neighbors
Kill Bill, SZA	Blind, SZA ǀ Sure Thing, Miguel ǀ Boy's a Liar, Pinkpatheress ǀ ...
Baby Shark, Pinkfong	The Alphabet Song, Cedarmont Kids ǀ We are the Dinosaurs, The Laurie Berkner Band ǀ Twinkle Twinkle Little Star, Super Simple Songs ǀ ...

Table 1: Nearest neighbors based on cosine similarity between entity2vec embeddings. The popular kid's song Baby Shark is closely associated with other kid's content. In contrast, a recently trending song by SZA is close to other hip-hop/r&b artists.

as in (Gillick et al., 2019). The weighted-sum approach leads to best results on our task (see comparison in Appendix B). A linear projection layer is applied to reduce all component embeddings to the same dimension, followed by an element-wise weighted sum. Rather than treating the weighted sum coefficients as hyper-parameters (Zhou et al., 2022; Liu et al., 2018), we let them update during training to converge to the optimal values given our objective function. Following Zhou et al. (2022), we pass the weighted sum output through 2 feed-forward layers to allow information to flow across the dimensions of the merged embedding.

4 Data

We build a dataset of voice search queries from user requests in a production system. For this work we target a subset of user utterances requesting music playback (e.g., "play flowers by miley cyrus"). Expansion to other domains is in scope for future work. All user data are de-identified.

4.1 Training

Rephrase Dataset. We use a heuristic rephrase detection algorithm as in (Cho et al., 2021; Fan et al., 2021) to construct a dataset of groundtruth (query, entity) pairs from user reformulations in multi-turn dialogues. When the production system fails to resolve a query correctly, users may choose to repeat their request until they get what they want (as in Figure 1). We find 2.5M such (query, rephrase) events in one week of production traffic and train our model to resolve the rephrase $entity$ from input query. We extract the song name and artist name announced to the user before playback begins to form the ground truth $entity$. For example, "play green green grass" and "put on the green grass song" rephrase utterances map to the entity *Green Green Grass by George Ezra*.

For embedding the queries we use the grapheme output of the ASR model as input into the semantic encoder. To form the user history, we embed up to 50 recently played entities by the same userId within 2 weeks of each request. To derive the semantic embedding of an entity, we feed the fine-tuned SBERT model a string containing track and artist names corresponding to the entity along with special tokens that demarcate track and artist boundaries (see Figure 3).

We train on 1 week of rephrase data (2.5M) and reserve 2 consecutive days for validation (400k) and testing (380k).

Entity Dataset. To train entity2vec, we build a dataset of user listening sessions. A session is a sequence of music entities played for a particular user, where session boundaries are characterized by 800s+ pause in playback. To reduce noise, we only consider entities that play for 30 seconds or longer. We keep sessions with at least 2 entities and have no upper limit on session length. We find 31 million such sessions in 1 month of English-speaking user interactions with our production system. For entity2vec training, we process the sessions into positive pairs of target and context entities. Negative pairs are generated by sampling random context entities from the vocabulary.

4.2 Inference

Entity Catalog. There are 1.5 million unique music entities in the Entity Dataset (Section 4.1), which cover 95% of all music entities in in our production system traffic. This set defines the output entity space that we search over during inference.

5 Experiments

5.1 Optimization Objective and Training

We train the end-to-end encoder on the Rephrase Dataset (Section 4.1). We tune the encoder with a contrastive objective (Chen et al., 2020a); given input query q_i and a set of candidate entities E, the task is to identify the ground-truth entity $e_{g.t.} \in E$.

Empirically, we find that fine-tuning SBERT separately performs better than tuning the encoder and merger layers at the same time (Appendix Table 8). Thus, we first fine-tune SBERT with in-batch softmax loss (eq 1), where the candidate set E is constrained mini-batch entities. Since we can't guarantee that all targets in batch are unique, we use (Khosla et al., 2020)'s formulation of softmax contrastive loss which generalizes to an arbitrary num-

Baseline Model	r@1	r@5	r@10
global, SBERT	+249.47%	+179.99%	+148.20%
global, fine-tuned	+109.78%	+69.87%	+50.90%
personalized	+91.09%	+162.38%	+176.20%

Table 2: Retrieval results on rephrase test set relative to baselines. In **global, SBERT**, search is performed over the entire entity catalog from section 4.2 using a pre-trained SBERT to generate query and entity embeddings. We fine-tune SBERT following section 4.1 in **global, fine-tuned** and perform search over the same entity catalog. **personalized** is our own implementation of (Fan et al., 2021) using SBERT from **global, fine-tuned** with personalized catalogs constructed from up to 1 month of historical user utterances. P-value computed from bootstrap confidence intervals is <0.01 for all reported results.

ber of in-batch positives. Using cosine similarity as the scoring function $s(q, e) = cos(f(q), f(e))$, where $f(.)$ denotes a forward pass through our encoder, the loss for a batch of size N is given by:

$$L_S = \sum_{i=1}^{N} \frac{-1}{|P_i|} \sum_{p \in P_i} log \frac{exp(s(q_i, e_p)/\tau)}{\sum_{j=1}^{N} exp(s(q_i, e_j)/\tau)} \quad (1)$$

Here, P_i is the set of "positive" batch indices such that $e_p = e_i \forall p \in P_i$, and τ is a scalar temperature parameter that we set to 0.1 based on findings in Khosla et al. (2020) and Chernyavskiy et al. (2022).

Next, we adopt triplet loss with a hard margin for training the merger layers. Here, negative entities e^- are explicitly sampled for each input rephrase pair (q_i, e_i^+) to form triplets (q_i, e_i^+, e_i^-). The loss function to minimize is:

$$L_T = \sum_{i=1}^{N} max(0, \lambda - s(q_i, e_i^+) + s(q_i, e_i^-)) \quad (2)$$

where λ is the margin hyper-parameter that we tune to $\lambda = 0.25$ on the validation set. To form the triplets we sample 2 random negative catalog entities for each positive (q, e) pair our dataset.

For all experiments we train with Adam optimizer with initial learning rate 5e-5 and batch size=1024 distributed across 8 NVIDIA v100 GPU cores. The output dimension is fixed to 200. We train for up to 3 epochs with early stopping on validation loss. Our best model learns coefficients of 0.8 and 0.2 corresponding to the semantic and entity embedding weights (**a** and **b** in Figure 3).

5.2 Evaluation

We evaluate the model on the task of retrieving the correct target entity from a global catalog given an input user utterance and listening history. Our most competitive baseline is our own implementation of (Fan et al., 2021), which comprises semantic search over de-identified personalized catalogs constructed from up to 1 month of historical user interactions. For a fair comparison with our model, we generate semantic embeddings with our fine-tuned SBERT. We also compare against a global baseline where we run semantic search over a global catalog.

We report recall@k for values of k in {1, 5, 10}, measuring the fraction of test set for which the target entity is in the top k model predictions.

history size	target entity in history?			
	no		yes	
	% of test	r@1	% of test	r@1
0	22.47%	+3.90%	-	-
1-10	18.34%	-2.66%	6.29%	**+49.41%**
10-20	19.28%	-1.55%	10.67%	+38.13%
20-30	10.96%	+4.57%	6.72%	+39.24%
30+	3.30%	**+16.66%**	2.02%	+39.99%
total	74.28%	0 (relative)	25.71%	+41.36%

Table 3: Model performance as function of user history size and composition. **r@1** reported relative to average **r@1** when target is not in history. When user history is not available, we build a user embedding from the top 50 popular tracks in our catalog.

6 Results & Discussion

Table 2 compares our results against baselines. Our model achieves >100% and 91% relative improvement in recall@1 over the global and personalized baselines, respectively. Interestingly, relative performance gains over the (Fan et al., 2021) personalized index baseline grow for higher values of k. This result highlights a major advantage of our model whose predictions are not limited to a fixed size user catalog, leading to higher recall on future requests. In Table 3 we show how our model generalizes to predict entities beyond those present in user history, and that the generalization power increases with larger history size. At the same time, when target is present in recent history, the model performs best when history size is small.

We observe that fine-tuning SBERT is crucial to adapt the semantic encoder to our entity-centric domain (**global, fine-tuned** vs **global, SBERT** in Table 2). User requests in production are typically short, comprising an action verb followed by the desired entity (e.g., "play baby shark"). Fine-tuning on the query-entity matching task teaches the encoder to differentiate between entities and verbs

query	user history	model predictions
play go	1035, Tiesto | Beautiful Dat, Trinix | In the Name of Love, Martin Garrix | ...	**Go, Cat Burns** | Go!, Lil Yachty | Go, The Kid Laroi
play go	All I want, Olivia Rodrigo | Cigarettes, Juice WRLD | Love, Kendrick Lamar | ...	**Go, The Kid Laoroi** | Goat, Lil Tjay | Go, Cat Burns
play go	Trap Queen, Fetty Wap | Love & War, Kodak Black | Im so Awesome, Kodak Black | ...	**Goat, Eric Bellinger** | Go, Cat Burns | Go, The Kid Laroi
play drinkin mexico	Something in the Orange, Zach Bryan | Oh My Dayum!, The Gregory Brothers | About You, The 1975 | ...	**Beer in Mexico, Kenny Chesney** | Stick That in your Country Song, Eric Church | Caught Up in The Country, Rodney Arkins | ...

Table 4: Top 3 model predictions for input user query and history. Top 3 rows demonstrate how user history influences the order of top retrieved entities for a fixed request. The bottom 2 rows are examples of ASR and ER error correction, respectively.

in the query. For example, the query "play phone booth" moved closer to song *Payphone* and further away from *Phone Play* post-fine-tuning on our task.

In Table 4 we demonstrate how model predictions on the ambiguous request "play go" vary based on user history (e.g., *Go by The Kid Laroi* given hip hop preferences vs. *Go by Cat Burns* given electronic dance preferences). We also highlight an example of successful recovery from phonetic variations (go → *Goat by Eric Bellinger*) and semantic aliasing ("play drinkin mexico" → *Beer in Mexico*).

6.1 Online Inference

Upon profiling the runtime of the retrieval model at each step we find that encoder forward pass and vector similarity search are prohibitively slow on our deployment hardware (r4.8xlarge CPU instance on AWS cloud). We perform the following optimizations to reduce latency and enable real time online inference.

Knowledge distillation. We replace SBERT (109M parameters) used in the semantic encoder with MiniLM (22M parameters) (Wang et al., 2020) which is distilled from BERT-base and follow the same training process as described in section 4.1. As a result, we reduce encoder forward pass runtime from 44 ms/query to 14 ms/query while maintaining 71% improvement over baseline. More details can be found in Appendix in Table 9.

Similarity search. By combining inverted index (IVF) with product quantization as in (Herve et al., 2011) for approximate vector-search, we reduce the average search speed from 239ms to 6ms per query with marginal performance trade-off in recall@1. Detailed results are reported in Appendix Table 10.

6.2 Phonetic signal

Similar to previous work (Zhou et al., 2022; Cho et al., 2021), we experiment with ingesting phonetic signal into our model using a transformer based phonetic encoder. However, contrary to previous reports, we find that a SBERT with subword tokenization is effective at recognizing phonetic variations as well as semantics and a separate phonetic encoder does not improve performance on our task. We report results in Appendix A.

7 Conclusion

In this work we present a novel approach for personalized search-based entity retrieval on noisy queries in voice-operated AI dialogue systems. We achieve personalization by encoding historical user preferences in a contextual query vector representation, followed by vector search on a global entity catalog. We empirically confirm that our approach significantly improves on existing baselines that rely on fixed-size historical indices to guide model output to personalized predictions.

Future work includes incorporating more contextual features in the user embeddings. For example, user preferences may vary based on time of day, day of week, and seasonality trends. Similarly including external knowledge such as lyrics, release dates, etc., in the entity embeddings will be explored.

Limitations

Music is a dynamic domain with new content released on a weekly basis. To keep up with changing trends, the proposed system must be retrained at a frequent cadence to learn embeddings for newly released entities and adapt to changing listening patterns. Another direction for future work is to explore methods for approximating embeddings for new releases to close the coverage gap between model re-trainings. For example, new releases from known artists can be positioned close to other entities from the same artist in the embedding space.

References

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020a. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Zheng Chen, Xing Fan, Yuan Ling, Lambert Mathias, and Chenlei Guo. 2020b. Pre-training for query rewriting in a spoken language understanding system. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7969–7973.

Anton Chernyavskiy, Dmitry Ilvovsky, Pavel Kalinin, and Preslav Nakov. 2022. Batch-softmax contrastive loss for pairwise sentence scoring tasks. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 116–126, Seattle, United States. Association for Computational Linguistics.

Eunah Cho, Ziyan Jiang, Jie Hao, Zheng Chen, Saurabh Gupta, Xing Fan, and Chenlei Guo. 2021. Personalized search-based query rewrite system for conversational AI. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 179–188, Online. Association for Computational Linguistics.

Xing Fan, Eunah Cho, Xiaojiang Huang, and Chenlei Guo. 2021. Search based self-learningquery rewrite system in conversational ai. In *2nd International Workshop on Data-Efficient Machine Learning (DeMaL)*.

Daniel Gillick, Sayali Kulkarni, Larry Lansing, Alessandro Presta, Jason Baldridge, Eugene Ie, and Diego Garcia-Olano. 2019. Learning dense representations for entity retrieval. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 528–537, Hong Kong, China. Association for Computational Linguistics.

Jie Hao, Yang Liu, Xing Fan, Saurabh Gupta, Saleh Soltan, Rakesh Chada, Pradeep Natarajan, Chenlei Guo, and Gokhan Tur. 2022. CGF: Constrained generation framework for query rewriting in conversational AI. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 475–483, Abu Dhabi, UAE. Association for Computational Linguistics.

Jegou Herve, Douze Matthijs, and Schmid Cordelia. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 117–128.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:18661–18673.

Hairong Liu, Mingbo Ma, Liang Huang, Hao Xiong, and Zhongjun He. 2018. Robust neural machine translation with joint textual and phonetic embedding. *arXiv preprint arXiv:1810.06729*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pragaash Ponnusamy, Alireza Roshan Ghias, Chenlei Guo, and Ruhi Sarikaya. 2020. Feedback-based self-learning in large-scale conversational AI agents. In *Proceedings of the AAAI conference on artificial intelligence*, pages 13180—13187.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Niranjan Uma Naresh, Ziyan Jiang, Ankit Ankit, Sungjin Lee, Jie Hao, Xing Fan, and Chenlei Guo. 2022. PENTATRON: PErsonalized coNText-aware transformer for retrieval-based cOnversational uNderstanding. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 90–98, Abu Dhabi, UAE. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Xiaozhou Zhou, Ruying Bao, and William M. Campbell. 2022. Phonetic Embedding for ASR Robustness in Entity Resolution. In *Proc. Interspeech 2022*, pages 3268–3272.

A Phonetic Encoder

We inject phonetic signal as input to teach the end-to-end model to recognize phonetic variations for ASR error correction. We pre-train a phonetic BERT (PBERT) on user queries with a masked language modeling (MLM) objective. Training data for the phonetic encoder comprises phoneme string outputs from the ASR model (e.g., "p l eI @ t l { n t @ s" for the request "play atlantis"). As in (Cho et al., 2021), we introduce word boundaries (e.g., "pleI @tl{nt@s") and train a sub-word phoneme tokenizer to make the phoneme token length comparable to query length. As with SBERT, we mean-pool over the token output of the last layer to build the final embedding.

We experiment with different encoder configurations: vocabulary size=10,000, hidden dimensions {128, 512, 768}, {4, 8, 12} attention heads, and {2, 4} transformer layers. We evaluate PBERT on a downstream homophone binary classification task before using it in the end-2-end model. We opt for a smaller architecture (small-BERT (Turc et al., 2019), 18M parameters) to ensure that our end-to-end model latency is not significantly affected.

In the merger layer, we add phonetic embedding derived from PBERT with the semantic and entity embedding, as described in Figure 4. Table 5 compares end-to-end model performance with and without phonetic encoder. We observe that recall@k is similar between SBERT + PBERT + entity2vec end-to-end model and SBERT + entity2vec model.

Model	recall@1	recall@5	recall@10
SBERT + pBERT + entity2vec	+89.88%	+162.88%	+176.64%
SBERT + entity2vec	+91.09%	+162.38%	+176.20%
pBERT + entity2vec	+66.35%	+135.46%	+149.72%

Table 5: Performance comparison with and without phonetic encoder on rephrase test set. Metrics reported relative to personalized baseline (our implementation of (Fan et al., 2021)).

B Multi-modal fusion methods

For merging the semantic, phonetic, and entity2vec embeddigs, we experiment with weighted sum fusion as in (Liu et al., 2018) and concatenation as in (Gillick et al., 2019).

The weighted sum approach is described in Section 3.2. For concatenation, outputs of each component encoder (entity2Vec, SBERT, and/or PBERT) are joined to form one large embedding combining the dimensionality of all the inputs. We pass the concatenated embedding through a feed forward layer to reduce the dimension back to 200, followed by another 2 fully connected layers on top to match the architecture of the weighted sum merger.

Model	Fusion	r@1	r@5	r@10
pBERT + E2V	concat	+60.70%	+131.29%	+145.64%
	sum	**+66.35%**	**+135.46%**	**+149.72%**
SBERT + E2V	concat	+85.07%	+158.63%	+172.36%
	sum	**+91.09%**	**+162.38%**	**+176.20%**
SBERT + pBERT + E2V	concat	+80.53%	+156.46%	+171.32%
	sum	**+89.88%**	**+162.88%**	**+176.64%**

Table 6: Comparison of concatenation (**concat**) and weighted-sum (**sum**) fusion methods for different combinations of semantic (SBERT), and entity (E2V) embeddings. All metrics reported relative to personalized baseline (our implementation of (Fan et al., 2021)).

Results in Table 6 indicate that weighted sum fusion consistently outperforms concatenation.

C Encoder training method

To train the end-to-end model described in Section 3.1, we first try tuning the pre-trained SBERT encoder and Merger Layer weights at the same time. However, we find that fine-tuning SBERT on our rephrase dataset separately yields a better performing model. Results are reported in Table 8. Thus, for all experiments reported in this paper we split training into 3 steps: first we and train entity2vec; next we fine-tune pre-trained SBERT/PBERT on the rephrase dataset; finally, we freeze all encoder weights and fine-tune the merger layers on the same rephrase dataset.

D Encoder runtime optimization

Tables 9 and 10 present detailed results from encoder and vector search optimization detailed in Section 6.1. Replacing SBERT (109M parameters) with MiniLM (22M parameters) paired with optimized approximate nearest neighbor search results in total 20ms/query average inference time. This is 14x improvement in speed over the SBERT + exhaustive search baseline.

E Performance Over Time

User music preferences vary over time due to seasonality, changing trends, and new releases. In Table 11 we report how model performance regresses over time. We observe 721bps regression in recall@1 over 3 months, indicating that regular retraining is necessary to keep up with user listening habits and incorporate new releases.

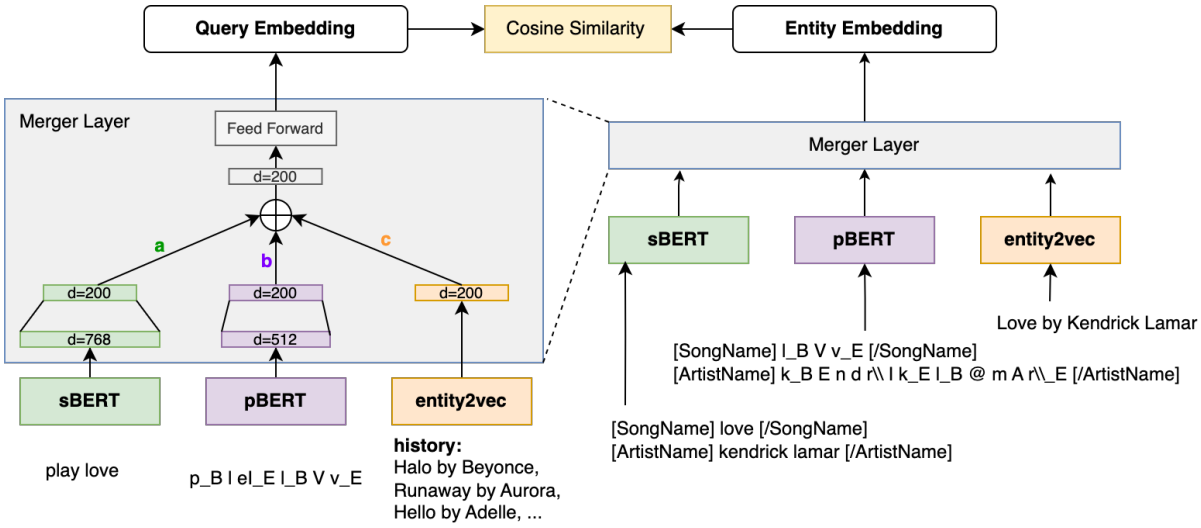


Figure 4: Dual encoder setup with shared phonetic, semantic and entity encoders. Semantic, phonetic and domain signals are embedded with pre-trained encoders (SBERT, PBERT and entity2vec). On the query side (left) we embed the query text, query phonemes and up to 50 recent entities that user interacted with. On the entity side (right), we embed the entity tokens and the entity itself. The component embeddings are combined in the merger layer to form the final representation of query and entity. Model weights are optimized to maximize cosine similarity between groundtruth *(query,entity)* pairs.

request	rephrase	ground truth entity	correction type
play the standard time	play the stains of time	Stains of Time by Kit Walters	ASR
play the gummy bear song	play i am a gummy bear	I Am a Gummy Bear by Gummibar	ER alias
play low	play low by sza	Low by Sza	ER disambiguation

Table 7: Sample user rephrases from our dataset. Rephrases in production traffic illustrate 3 error modes: ASR defect, ER alias, and ER disambiguation.

Method	r@1	r@5	r@10
end-to-end	-8.91%	+53.58%	+81.44%
pre-tune	**+91.09%**	**+162.38%**	**+176.20%**

Table 8: Fine-tuning SBERT encoder before tuning the combined model results in better performance. All metrics reported relative to personalized baseline (our implementation of (Fan et al., 2021)).

F Examples from Rephrase Dataset

We present sample rephrases in our rephrase dataset in Table 7. The dataset extraction method is detailed in Section 4.1. For readability, we omit the user history field which contains up to 50 entities from user's recent interaction history.

Encoder	Inference time	r@1	r@5	r@10
SBERT	45 ms/query	+91.09%	+162.38%	+176.20%
MiniLM	14 ms/query	+71.35%	+133.83%	+145.84%

Table 9: Runtime vs recall comparison of different semantic encoders. Inference time is evaluated as the mean over 1000 inference runs with batch size 1. Metrics reported relative to personalized baseline (our implementation of (Fan et al., 2021)).

Search Method	build speed	ms/query	r@1
Exhaustive	696ms	239	+91.09%
+ IVF	3.9s	23	+89.22%
+ Quantization	151s	6	+79.37%

Table 10: Vector search optimization for online inference. Metrics reported relative to personalized baseline.

train date	test date	r@1	r@5	r@10
	2/26/2023	0%	0%	0%
	2/27/2023	-0.14%	-1.05%	-1.24%
	2/28/2023	-0.16%	-0.95%	-1.23%
2/26/2023	3/19/2023	-7.00%	-6.16%	-6.31%
	3/22/2023	-16.48%	-7.60%	-7.32%
	5/10/2023	-16.48%	-15.07%	-14.25%
	5/14/2023	-16.76%	-15.08%	-14.50%

Table 11: Model performance over time. Model is trained on 1 week of rephrase data ending on **train date**. Model is tested on 1 day of rephrase data from **test date**. Recall values reported relative to a **2/26/2023** test date baseline.