STAR: Semantic Targeting Matching Model for Exact/Phrase Expansion

1. Summary

Sponsored Brands team plans to adopt semantic matching for exact and phrase keyword matching, empowering advertisers to reach more shoppers with fewer keywords (PRFAQ). Prior to this update, exact and phrase keywords require strict string match to the search queries and fail to match search queries with different phrasing that convey the same meaning. By adopting semantic matching, the ad will be targeted to the search queries that either have the same meaning with the exact keywords (e.g., query: running shoes vs. exact keyword: running sneakers), or include the meaning of the phrase keywords (e.g., query: nike shoes vs. phrase keyword: sneaker). We experimented with both lexical and semantic baselines for keyword matching, and developed a new Semantic TARgeting matching model, STAR model, that significantly outperforms the baselines. Below are the summary of the doc.

- We experimented with both lexical and embedding baseline method, but found neither has sufficient semantic understanding capability to predict the query-keyword relevance, achieving up to 0.73 AUC only in the benchmark evaluation. We built transformer based STAR model with cross attention applied to both query and keyword, achieving up to 0.92 AUC in the benchmark evaluation.
- We designed three-step training approach, starting from masked language pretraining followed by pre-finetuning and finally finetuning, that help incorporate both language understanding capability and task knowledge into the STAR model. The experiment shows both masked language pretraining and pre-finetuning significantly boost the model performance with 0.17 AUC lift by masked language pretraining and additional 0.05 AUC lift by pre-finetuning.
- We simplified the models and used optimization techniques, such as ONNX optimizer and half-precision training, to make STAR model inference achieve 100K+ TPS on single p3.2xlarge instance, meeting the production needs of handling billions of query-keyword pairs every day.

2. Model Design

2.1 BASELINES

We started our experiments with simple lexical and semantic baselines. The current solr implementation for exact and phrase match is a string matching based method. It requires exact string match for exact keywords and exact substring matching for phrase keywords with minor string normalization (e.g., lowercase and removing stop words). To normalize the string further, we experimented with string stemming [1] that removes morphological affixes from words and keeps the word stem only before matching query and keyword. As shown in Table 1. Illustrative Examples, it can help match query “nose bleed plug” with keyword “nose bleeding plug” since they all become “nose bleed plug” after stemming. However, stemming still falls short on semantic matching, for example, not being able to match “running shoes” with “running sneakers”.

We have also tested embedding model as the semantic baseline. We used the embedding model to generate query and keyword embedding separately, and used cosine similarity to measure the query-keyword relevance. However, due to the nature of two tower structure, the embedding based method lacks the token-by-token comparisons across query and keyword, so it can’t capture the subtle difference between query and keyword which changes their semantic relationship (e.g., query: shark nv750w parts vs. keyword: shark navigator model nv351 replacement parts in Table 1. Illustrative Examples). In addition, cosine similarity based method is particularly problematic for phrase match. Phrase match requires the keyword to be more general than the query, so swapping query and keyword flips the positive/negative label. Cosine similarity, on the other hand, is a symmetric measure that generates the same score regardless of the ordering of query and keyword. A straightforward improvement is to replace cosine similarity with an MLP layer but it only improves the performance marginally. As shown in in the evaluation result Table 2. Benchmark evaluation, the embedding based methods are better than lexical matching methods but still sub-optimal.

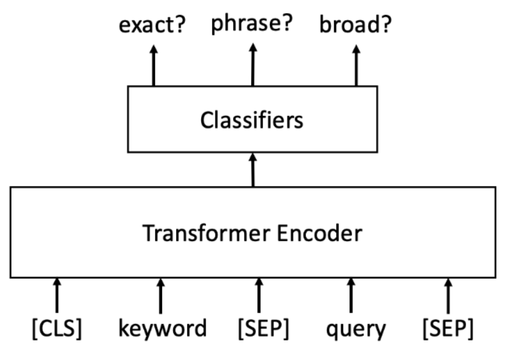
Table 1. Illustrative Examples

	A	B	C	D	E	F
1	**query**	**keyword**	**ground truth label**	**prediction by string match with stemming**	**prediction by embedding with cosine similarity**	**STAR prediction**
2	nose bleed plug	nose bleeding plug	exact match	exact match	match	exact match
3	outdoor tree decor	outdoor tree decorations	exact match	exact match	match	exact match
4	portable charger	power bank	exact match	not match	match	exact match
5	running shoes	running sneakers	exact match	not match	match	exact match
6	shark nv750w parts	shark navigator model nv351 replacement parts	not match	not match	match	not match
7	adidas shoes	sneaker	phrase match	not match	match	phrase match
8	sneaker	adidas shoes	broad match	not match	match	broad match
9	canadian boots for women	wide calf women boots	broad match	not match	match	broad match

2.2 STAR: SEMANTIC TARGETING MATCHING MODEL

We developed STAR model that takes the concatenation of keyword and query as input and predicts three scores for exact/phrase/broad match. Figure 1. STAR: Semantic Targeting Matching Model shows the model structure. The main block of the model is a transformer encoder layer with cross attention on the input tokens, which enables the token-by-token comparison across query and keywords. The transformer encoder extracts the semantic features on top of which classification heads are applied to make the predictions. We will define three score thresholds online for exact/phrase/broad match respectively. A query and an exact match keyword is classified as a match if the exact match prediction score is above the threshold for exact match. Similar logic applies to the phrase match.

Figure 1. STAR: Semantic Targeting Matching Model



The model training has three steps, masked language pre-training, pre-finetuning and finetuning.

2.2.1 Mask Language Pre-training

Mask language pre-training (MLM) is the key to the success of the well known transformer encoder models, e.g., BERT, as it brings the foundational language understanding capability to the model. In the experiment, we found that STAR model performs poorly without any pre-training. Initializing the model parameters with the public pre-trained model can boost the performance significantly, but it lacks the flexibility to control the model complexity, e.g., control the hidden size, which is important due to the high TPS requirement. So we experimented with our own pre-training on Amazon data. Our pre-training corpus consists of over 300MM query-ASIN pairs collected from Tommy ASIN data and each pair is represented as the concatenation of query, item name, brand and PT. We randomly masked 15% tokens and the transformer encoder is trained to predict the tokens given the context. Compared with the public pre-trained model, our own pre-trained model has better language understanding capability in product domain due to the domain of the pre-training corpus.

Figure 2. MLM Pre-training

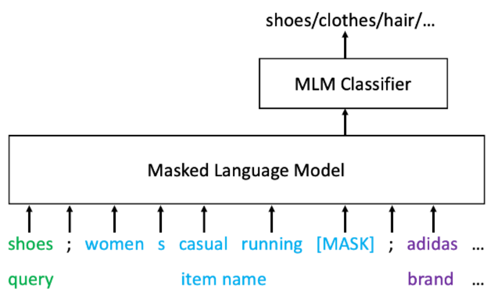
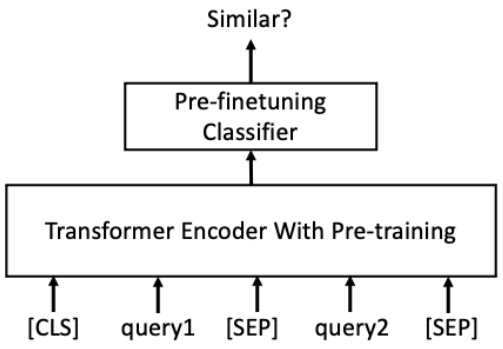


Figure 3. Pre-finetuning


Similar?
Pre-finetuning Classifier
Transformer Encoder With Pre-training
[CLS]
query1
[SEP]
query2
[SEP]

2.2.2 Pre-finetuning

Although mask language pre-training can establish the language understanding capability, the model still lacks the task specific knowledge, which usually relies on the finetuning step to address. However, finetuning data has a limited size so we added a new step, called pre-finetuning, that uses label data from similar tasks and pseudo label data to help incorporate task knowledge into the model. More specifically, we get the pre-finetuning data from two different kinds of sources, public paraphrase detection datasets and similar query pairs from Nota Query Graph [2]. Paraphrase detection is a similar task to the semantic matching task. We tried multiple paraphrase detection datasets, including quora question pair dataset [3] which is found to be useful to improve the performance. Nota Query Graph incorporates the query-ASIN semantic and behavior information and we leveraged it to generate large amount of similar/irrelevant query pairs. Eventually, we combined the pre-finetuning data from both sources and use cross entropy loss to train the transformer encoder. Note that, pre-finetuning data isn't appropriate to be used for finetuning directly since their label is either not strictly match to our task definition or noisy without human verification. The pre-finetuning classification head is dropped after this step.

2.2.3 Finetuning

We obtained human labelled query-keyword pairs data from SP Search Sourcing team [4] annotated by 2 to 7 reviewers. Each pair was labeled into 5 classes: “strictly relevant”, “somewhat relevant - generalized keyword”, “somewhat relevant - generalized query”, “not relevant”, and “cannot be judged”. After a team review, we found “strictly relevant” is a good match to our new exact match definition and “somewhat relevant - generalized keyword” is a good match to our new phrase match definition. For each query-keyword pair, we took the majority vote of the annotations from reviewers as the final label. Cases that cannot break the tie or “cannot be judged” being the majority vote were dropped. We ended up with 18656 query-keyword pairs annotated into 4 classes with the following distribution: “strictly relevant” 55.5%, “somewhat relevant - generalized keyword” 10.9%, “somewhat relevant - generalized query” 28.3% and “not relevant” 5.3%.

We applied train/test split on the 18,656 query-keyword pairs. 10,000 pairs were reserved for evaluation and the rest 8,656 pairs were for model training/finetuning. We applied tricks that swap query and keyword to augment the training data. For “strictly relevant” pairs, swapping query and keyword should still be “strictly relevant” pairs. For “somewhat relevant - generalized keyword” pairs, swapping query and keyword makes the pairs become “somewhat relevant - generalized query” based on the definition. Similarly, “somewhat relevant - generalized query” pairs would become “somewhat relevant - generalized keyword” pairs after swapping query and keyword. Finally, swapping query and keyword for “not relevant” pairs generates new “not relevant” pairs. By these tricks, we doubled the size of training data and got 17,312 pairs for model training.

2.2.4 Speeding Up Inference

In production, STAR model is expected to process tens of billions of query-keyword pairs every day so we control the number of transformer layers and hidden size to make model small and efficient. In addition, we have applied the following techniques to speed up the model inference. We first converted the pytorch model to the ONNX model, which is known to be faster than the native pytorch model. On top of it, we applied transformer ONNX optimizer [5, 6] that does further optimization on the ONNX computation graph according to the characteristics of transformer structure. Moreover, we also found using 16-bit float to store the model parameters has a better tradeoff between accuracy and efficiency than the default 32-bit float. Hence we turned on half precision training along all model training steps, i.e., MLM pre-training, pre-finetuning and finetuning, so that the final model artifact can use 16-bit float for both storing the model parameters and computation.

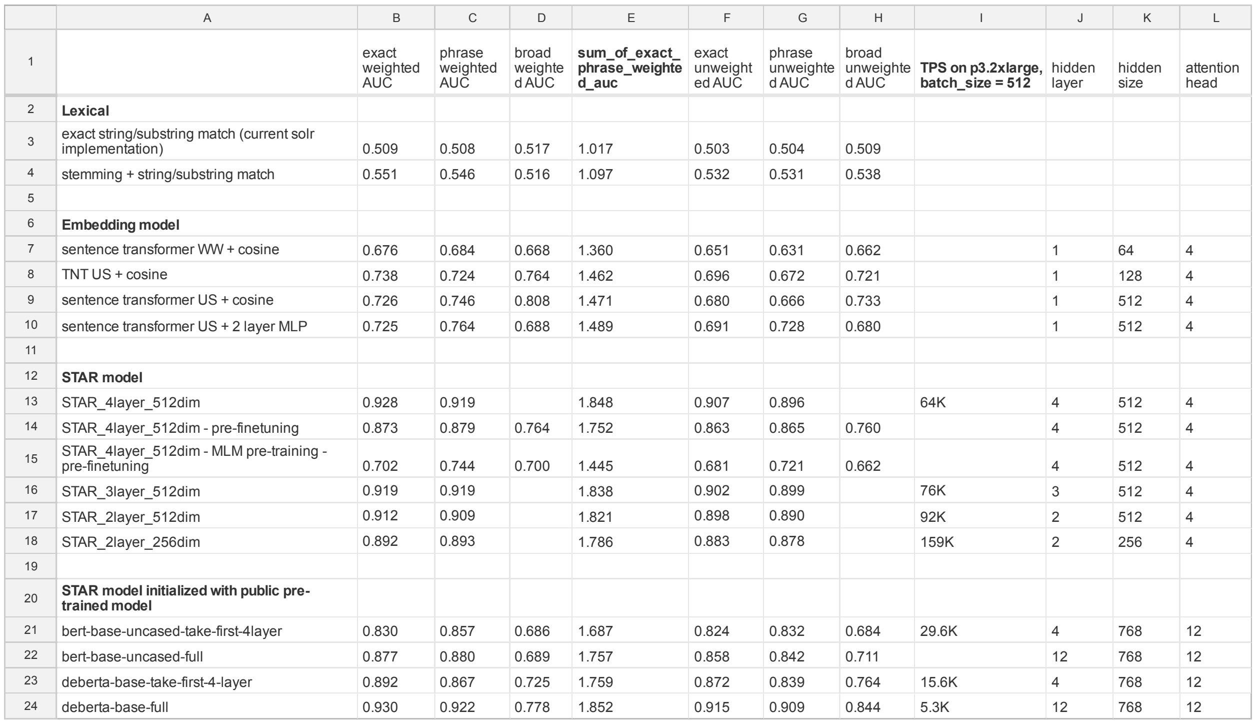
3. Offline Evaluation

3.1 BENCHMARK EVALUATION

We used the reserved 10K pairs to evaluate both the baseline models and the proposed semantic targeting matching models. AUC and recall@90% precision are used as the evaluation metrics. We run both unweighted evaluation and weighted evaluation where each pair is assigned with a weight based on the number of SB clicks of the query collected in a one-year period. Table 2. Benchmark evaluation shows the AUC result and Table 3. Recall@90% precision shows the results of recall@90% precision. Below are the summary of the observations.

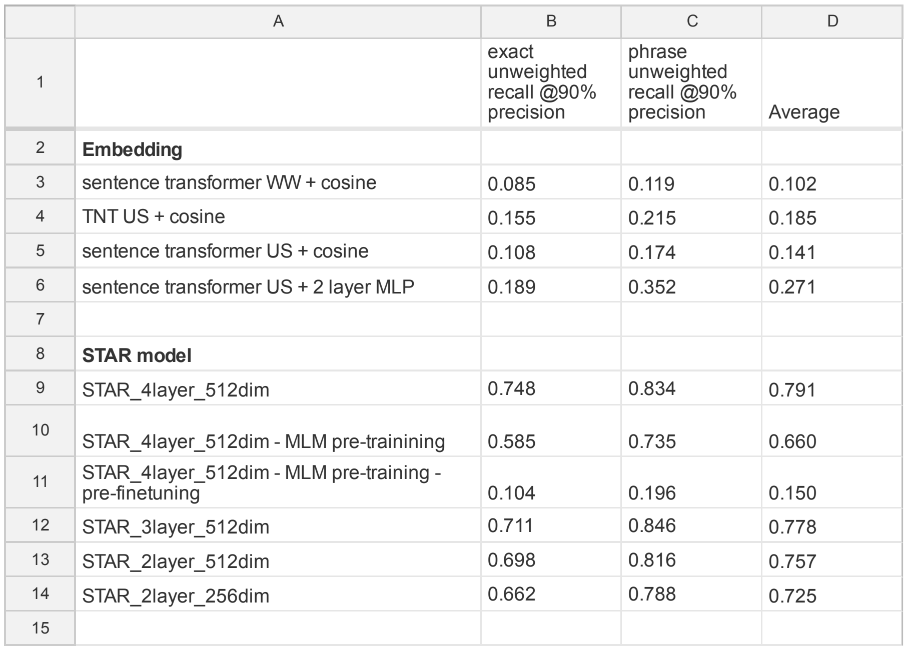
- **Lexical (row 2-4):** Exact string/substring match has high precision and very low recall given its constraint on strict string match, resulting a low AUC (~0.50). String stemming normalizes the string and improve the recall a bit, but the performance is still low.
- **Embedding model (row 6-10):** Embedding methods are significantly better than lexical methods achieving 0.72+ AUC. Replacing cosine similarity with 2 layer MLP as the classification layer improves the performance, especially for phrase match.
- **STAR model (row 12-18):** For the sake of TPS, we limited the number of transformer layers up to 4 and the dimension size up to 512 for STAR model. STAR model significantly outperforms the embedding model by around 0.2 AUC. We did the ablation study to study the impact of pre-training and pre-finetuning. Comparing row 15 with row 13, out-of-box STAR model without any pre-training and pre-finetuning performs poorly with only around 0.7 AUC. With MLM-pretraining, the model performance is boosted by language understanding capability, achieving up to 0.17 AUC lift. By doing additional pre-finetuning, the model learns the task related knowledge and the performance is further increased by 0.05 AUC. We also studied the impact of the model complexity. As expected, model has better performance with more transformer layers and larger hidden size.
- **STAR model initialized with public pre-trained model (row 20-24):** We also experimented with STAR models initialized with public pre-trained models. We finetuned the models directly without further pre-training or pre-finetuning. deberta performs slightly better than our own pre-trained models at the expense of significant lower TPS.
- **Recall@90% precision:** Table 3. Recall@90% precision shows the recall at 90% precision metric. The comparisons among different methods are consistent with the findings on AUC. The weighted results are skipped since the weighted PR-curves fluctuate a lot due to the limited size of label dataset and the query weight with large variance.

Table 2. Benchmark evaluation



	A	B	C	D	E	F	G	H	I	J	K	L
1		exact weighted AUC	phrase weighted AUC	broad weighte d AUC	**sum_of_exact_ phrase_weighte d_auc**	exact unweight ed AUC	phrase unweighte d AUC	broad unweighte d AUC	**TPS on p3.2xlarge, batch_size = 512**	hidden layer	hidden size	attention head
2	**Lexical**											
3	exact string/substring match (current solr implementation)	0.509	0.508	0.517	1.017	0.503	0.504	0.509				
4	stemming + string/substring match	0.551	0.546	0.516	1.097	0.532	0.531	0.538				
5												
6	**Embedding model**											
7	sentence transformer WW + cosine	0.676	0.684	0.668	1.360	0.651	0.631	0.662		1	64	4
8	TNT US + cosine	0.738	0.724	0.764	1.462	0.696	0.672	0.721		1	128	4
9	sentence transformer US + cosine	0.726	0.746	0.808	1.471	0.680	0.666	0.733		1	512	4
10	sentence transformer US + 2 layer MLP	0.725	0.764	0.688	1.489	0.691	0.728	0.680		1	512	4
11												
12	**STAR model**											
13	STAR_4layer_512dim	0.928	0.919		1.848	0.907	0.896		64K	4	512	4
14	STAR_4layer_512dim - pre-finetuning	0.873	0.879	0.764	1.752	0.863	0.865	0.760		4	512	4
15	STAR_4layer_512dim - MLM pre-training - pre-finetuning	0.702	0.744	0.700	1.445	0.681	0.721	0.662		4	512	4
16	STAR_3layer_512dim	0.919	0.919		1.838	0.902	0.899		76K	3	512	4
17	STAR_2layer_512dim	0.912	0.909		1.821	0.898	0.890		92K	2	512	4
18	STAR_2layer_256dim	0.892	0.893		1.786	0.883	0.878		159K	2	256	4
19												
20	**STAR model initialized with public pre-trained model**											
21	bert-base-uncased-take-first-4layer	0.830	0.857	0.686	1.687	0.824	0.832	0.684	29.6K	4	768	12
22	bert-base-uncased-full	0.877	0.880	0.689	1.757	0.858	0.842	0.711		12	768	12
23	deberta-base-take-first-4-layer	0.892	0.867	0.725	1.759	0.872	0.839	0.764	15.6K	4	768	12
24	deberta-base-full	0.930	0.922	0.778	1.852	0.915	0.909	0.844	5.3K	12	768	12

Table 3. Recall@90% precision



	A	B	C	D
1		exact unweighted recall @90% precision	phrase unweighted recall @90% precision	Average
2	**Embedding**			
3	sentence transformer WW + cosine	0.085	0.119	0.102
4	TNT US + cosine	0.155	0.215	0.185
5	sentence transformer US + cosine	0.108	0.174	0.141
6	sentence transformer US + 2 layer MLP	0.189	0.352	0.271
7				
8	**STAR model**			
9	STAR_4layer_512dim	0.748	0.834	0.791
10	STAR_4layer_512dim - MLM pre-trainining	0.585	0.735	0.660
11	STAR_4layer_512dim - MLM pre-training - pre-finetuning	0.104	0.196	0.150
12	STAR_3layer_512dim	0.711	0.846	0.778
13	STAR_2layer_512dim	0.698	0.816	0.757
14	STAR_2layer_256dim	0.662	0.788	0.725
15				

16	**STAR model initialized with public pre-trained model**			
17	bert-base-uncased-take-first-4layer	0.366	0.560	0.463
18	bert-base-uncased-full	0.513	0.614	0.564
19	deberta-base-take-first-4-layer	0.627	0.629	0.628
20	deberta-base-full	0.774	0.865	0.820

3.2 FEW SHOT LEARNING EXPERIMENT

We used STAR_3layer_512dim model and run the few shot learning study. In this experiment, we only changed the number of training data used in the finetuning step while keeping the original MLM pre-training step and the pre-finetuning step. In addition, we used the original 8K human label data without any swapping based data augmentation trick in this study. As shown in Table 4, increasing training data does help improve the performance and the gain is diminishing when the training data is larger than 5K.

Table 4. Few shot learning study

training data size	exact weighted AUC	phrase weighted AUC	sum of exact phrase weighted auc	exact unweighted AUC	phrase unweighted AUC	exact unweighted recall @90% precision	phrase unweighted recall @90% precision
1k	0.867	0.880	1.746	0.860	0.863	0.572	0.695
2k	0.878	0.883	1.761	0.866	0.868	0.594	0.704
3k	0.890	0.898	1.788	0.878	0.883	0.629	0.778
4k	0.895	0.902	1.798	0.885	0.887	0.669	0.785
5k	0.899	0.910	1.809	0.886	0.891	0.671	0.809
6k	0.902	0.908	1.810	0.890	0.890	0.675	0.785
7k	0.910	0.914	1.824	0.895	0.896	0.693	0.819
8k	0.916	0.919	1.835	0.894	0.897	0.699	0.832
full 17K after swapping	0.919	0.919	1.838	0.902	0.899	0.711	0.846

3.3 FURTHER IMPROVEMENT

Color/gender/size are the key product attributes and we made STAR model finetuning on them to prevent the potential risks. A color/gender/size patching dataset of size 108/108/122 is added to the finetuning of the STAR model. After the addition of the patching dataset, a significant negative shift in the phrase score can be observed when tested on datasets sampled from our offline data as shown in Figure 4.

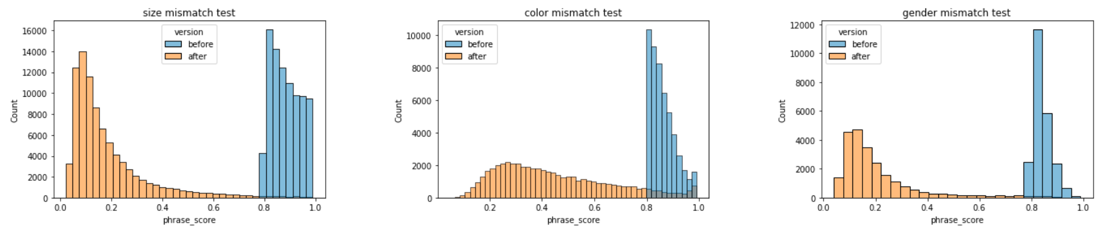
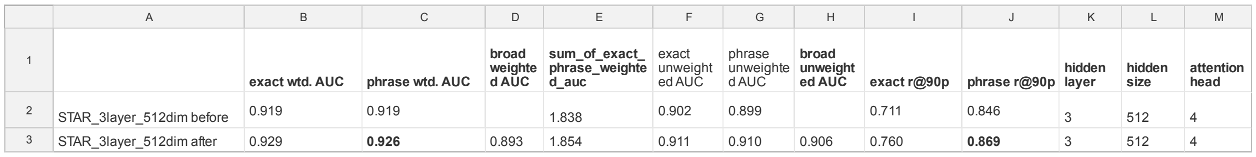


Figure 4. Phrase score distribution before and after the addition of the color/gender/size patching dataset on coarsely filtered mismatch dataset.

Recently, we are able to get additional 27K human annotation data from the human annotation workflow. The STAR model is then finetuned with these additional data. Table 5 compares the STAR model before and after the addition of extra finetuning data. With the additional training data, the model performance can be further improved, which aligns with our finding in the few shot learning experiments.

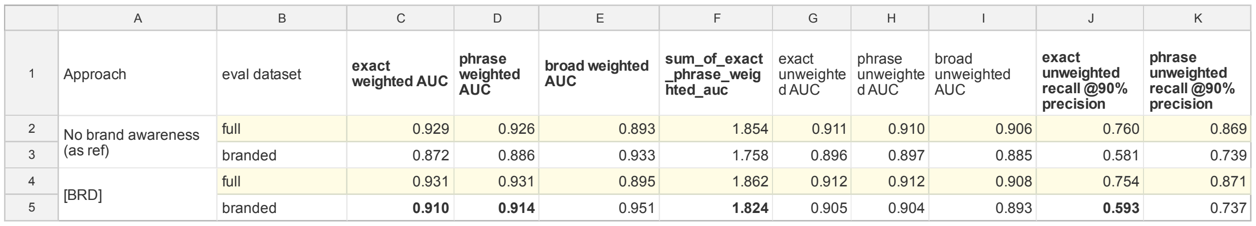
Table 5. Comparison of STAR model performance before and after extra finetuning data



	A	B	C	D	E	F	G	H	I	J	K	L	M
1		**exact wtd. AUC**	**phrase wtd. AUC**	**broad weighted AUC**	**sum_of_exact_phrase_weighted_auc**	exact unweighted AUC	phrase unweighted AUC	**broad unweighted AUC**	**exact r@90p**	**phrase r@90p**	**hidden layer**	**hidden size**	**attention head**
2	STAR_3layer_512dim before	0.919	0.919		1.838	0.902	0.899		0.711	0.846	3	512	4
3	STAR_3layer_512dim after	0.929	**0.926**	0.893	1.854	0.911	0.910	0.906	0.760	**0.869**	3	512	4

We also test incorporating brand awareness to the STAR model with the brand signal from Nota QSL to avoid brand mismatching predictions. Experiments show that appending a special token [BRD] to queries and keywords considered branded by Nota QSL significantly improves the model performance on the 1,447 branded subset of the 10k benchmark. For example, input “[CLS] adidas shoes [SEP] sneakers [SEP]” now becomes “[CLS] adidas shoes [BRD] [SEP] sneakers [SEP]”.

Table 6. Comparison of STAR model performance on benchmark and its branded subset with/without brand awareness



	A	B	C	D	E	F	G	H	I	J	K
1	Approach	eval dataset	**exact weighted AUC**	**phrase weighted AUC**	**broad weighted AUC**	**sum_of_exact_phrase_weighted_auc**	exact unweighted AUC	phrase unweighted AUC	broad unweighted AUC	**exact unweighted recall @90% precision**	**phrase unweighted recall @90% precision**
2	No brand awareness (as ref)	full	0.929	0.926	0.893	1.854	0.911	0.910	0.906	0.760	0.869
3		branded	0.872	0.886	0.933	1.758	0.896	0.897	0.885	0.581	0.739
4	[BRD]	full	0.931	0.931	0.895	1.862	0.912	0.912	0.908	0.754	0.871
5		branded	**0.910**	**0.914**	0.951	**1.824**	0.905	0.904	0.893	**0.593**	0.737

4. Next Steps

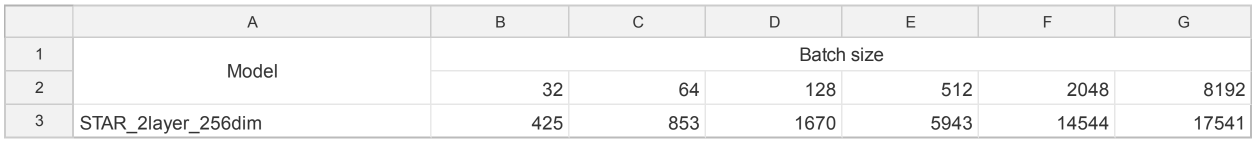
We will continue exploring various techniques to improve the model. Here are some of the ideas.

- Multi-lingual: We will build multi-lingual STAR model for WW expansion. We have done the preliminary work on multi-lingual MLM pre-training with WW Tommy ASIN data. We are working with Brand Understanding team on collecting multi-lingual pre-finetuning data. We will also work with Sherlock team and SP team to collect multi-lingual labeled data to finetune and evaluate the models. In addition, we will explore using tricks, such as machine translation, to augment the training data.
- Enriching training data: We developed Query-Keyword SOP For SB and are working with Sherlock team on curating more query-keyword human label data. We will also incorporate the advertiser escalation case into our finetuning dataset. In addition, we will explore using heuristic rules to generate pseudo label data to augment the training data.
- Knowledge distillation. Our online service has a tight latency budget and applying STAR model as an online filter isn’t feasible yet due to the low TPS on CPU instance. See Appendix A1. TPS testing on CPU instance. One direction is to explore online GPU serving. On the other hand, we can experiment with knowledge distillation technique to further reduce the model complexity and improve the TPS.

Appendix

A1. TPS TESTING ON CPU INSTANCE

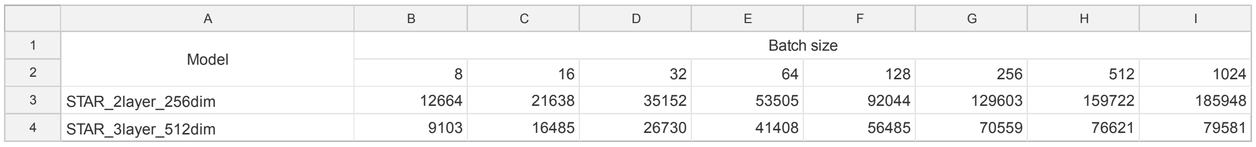
We took the fastest STAR model, STAR_2layer_256dim, and run the TPS testing on r5.12xlarge instance.



	A	B	C	D	E	F	G
1	Model	Batch size					
2		32	64	128	512	2048	8192
3	STAR_2layer_256dim	425	853	1670	5943	14544	17541

A2. TPS TESTING WITH DIFFERENT BATCH SIZES ON GPU INSTANCE

We run TPS testing with different batch sizes on P3.2xlarge.



	A	B	C	D	E	F	G	H	I
1	Model	Batch size							
2		8	16	32	64	128	256	512	1024
3	STAR_2layer_256dim	12664	21638	35152	53505	92044	129603	159722	185948
4	STAR_3layer_512dim	9103	16485	26730	41408	56485	70559	76621	79581

A3. AMLC PAPER

https://drive.corp.amazon.com/documents/bingyinh@/presentation/AMLC2023/AMLC_2023_STAR_camera_ready.pdf

Reference

[1] https://www.nltk.org/api/nltk.stem.html
[2] Fast and Refined Tail-to-Gold Query Mapping with Application to Brand Prediction
[3] https://huggingface.co/datasets/glue/viewer/qqp/train
[4] Query-Keyword Audit SOP
[5] https://github.com/huggingface/transformers/blob/main/src/transformers/convert_graph_to_onnx.py#L257
[6] https://github.com/microsoft/onnxruntime/blob/main/onnxruntime/python/tools/transformers/optimizer.py