# A Flexible Large-Scale Similar Product Identification System in E-commerce

Zhen Zuo, Lixi Wang, Michinari Momma, Wenbo Wang, Yikai Ni, Jianfeng Lin, Yi Sun
{zhenzuo,lixiwang,michi,wenbowan,yika,ljianfen,yisun}@amazon.com
Amazon
Seattle, WA

## ABSTRACT

Identifying similar products is a common pain-point in the world of E-commerce search and discovery. The key challenges lie in two aspects: 1) The definition of *similarity* varies across different applications, such as near identical products sold by different vendors, products that are substitutable to each other for customers with common interests, personalized products visually similar in terms of design pattern, style or color. It is difficult to build a general solution for all scenarios. 2) Computing pairwise similarities among billions of products are resource demanding which makes it challenging in large-scale data processing. To provide a flexible and consolidated solution at scale, this paper presents an all-in-one system, Product Similarity Service (PSS), which leverages the state-of-the-art Deep Neural Networks and distributed computing technology to serve diverse Amazon-scale product similarity computations. The experimental results show that PSS can return highly relevant similar products in both verification and ranking tasks, and has good computing efficiency and system scalability on large data volume.

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Data mining*; *Computing platforms*;

## KEYWORDS

Similar product search systems, recommendation systems, deep learning, metric learning, multi-task learning, embedding.

**ACM Reference Format:**
Zhen Zuo, Lixi Wang, Michinari Momma, Wenbo Wang, Yikai Ni, Jianfeng Lin, Yi Sun. 2020. A Flexible Large-Scale Similar Product Identification System in E-commerce. In *Proceedings of The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'20 IRS Workshop).* ACM, New York, NY, USA, 9 pages.

## 1 INTRODUCTION

E-commerce reshapes the new era of customer shopping experience with more and more customers choosing online shopping. Meanwhile, a dramatic rise is observed in the number of products sold on e-commerce websites. Identifying similarity among a tremendous number of products becomes a critical problem in a large variety of online business scenarios. For example, Alibaba built a visual search system to find similar products given customer's image queries [28]; Pinterest applied image search in their content-based product recommendations [27]. In Amazon, there is also a trend to apply similarity identification to diverse popular emerging applications. Fig. 1 shows three applications, where (a) recommends substitute products for out-of-stock items based on their images, titles and descriptions with matched key attributes (e.g. the same phone type), (b) personalizes fashion products which look similar but vary in some aspects (e.g. different designs), and (c) searches near identical items sold by different vendors. All of them need to explore product similarities among Amazon's increasingly growing repository but with distinct business goals and requirements.

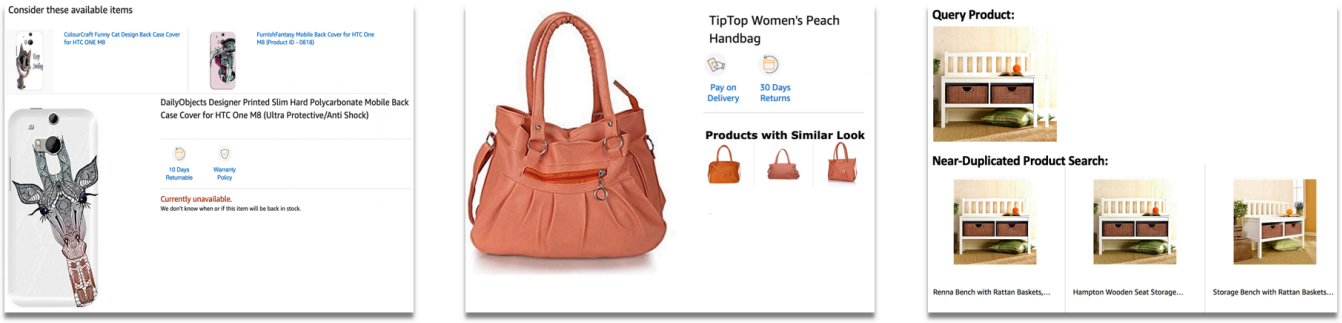
As each business application has its own definition of *similarity*, most of the similarity computing systems in the industry are tailored with their focus on a certain type of application domain. It remains challenging to consolidate them into a generic solution which is flexible enough to serve diverse similarity-based applications, while at the same time, salable enough to process data in large volume. Specifically, Amazon applications launched with similarity features usually need to deal with billions of products from various categories such as fashion, electronic devices, and furniture, etc. Building independent computing systems for different applications is inefficient from both model development and resource utilization perspectives. Instead, we present a general and flexible all-in-one similarity product identification platform, Product Similarity Service (PSS), to compute the similarity as a service for a variety of customized similarity use cases at Amazon scale. The following key challenges are addressed in our system.

**Flexible Similarity Definition for Various Applications**.
*Variant Input Product Information*: For different applications, the system needs to provide the flexibility for input product information selection. For example, for visual similarity recommendations, product image is the key information to use, while for substitute product identifications, image, title, and description could all be important clues for similar product identification.

*Different Optimization Goals*: The system needs to be flexible enough to optimize the output goals of different applications. For example, in visual search, the goal is to return products which have similar visual patterns with the query image in a ranking order. While in substitute products recommendation, the goal is to identify products which can be replaceable by each other when customers make purchase decisions.


*KDD'20 IRS Workshop, August 23–27, 2020, San Diego, CA, USA*

ACM ISBN 978-1-4503-6201-6/20/8.




(a) Substitutable Product Recommendation (b) Visually Similar Product Recommendation (c) Near-identical Product Search

**Figure 1: Similar product identification applications at Amazon: for a query product, PSS is able to return similar products with different application definitions.**

*Key Attributes Matching*: The system needs to have the flexibility to remove unqualified *similar* items using post filters. For example, for substitute phone case recommendations, if a customer is exploring iPhone X case, the post filter should be able to remove phone cases that are not compatible to iPhone X. Otherwise, it will cause a very disappointing customer experience.

**Scalable and Efficient Similarity Computing**.

High scalability and efficiency are needed for a similarity computing system to handle Amazon-scale data. Firstly, the system needs to be able to scale to process enormous and continuously growing product sets and publish refreshed similarity results with controllable time frames. Secondly, the system needs to manage the computing resource (e.g., CPU/GPU and memory usage) to optimize for cost as it involves image and text embedding computation from deep neural networks and large item indexing for similarity search. Thirdly, the system should be capable of identifying and caching the pre-computed result to make it reusable across different applications without repetitive computations.

In PSS, we utilized a hybrid modeling method with both product contextual and customer behavior information and made it configurable for application-specific goals. We leveraged the state-of-the-art cloud computing technologies to design a distributed similarity computing solution for scaling. We evaluated our system on comprehensive modeling and system experiments across multiple applications. We demonstrated that PSS is able to support product-based similarity identification tasks with high quality model and system performance at the Amazon scale.

In the rest of the paper, Section 2 compares the related work to our system, Section 3 describes our similarity framework and approach, Section 4 demonstrates the system architecture in details, Section 5 evaluates the system performance through experiments, and Section 6 gives the conclusions and future works.

## 2 RELATED WORKS

### 2.1 Similarity Identification Systems

Most of the industry similarity identification systems focus on providing highly specialized solutions optimized for very specific application scenarios. They are not flexible enough to support multiple applications. In general, for any application, there are mainly two types of signals for similarity computing: product contextual information, and customer behavioral information.

Content based similarity identification systems have been adopted throughout industry, they aim to return items with highly similar contextual information (e.g. image, video, description, attribute, etc.) to the query item. Among which, visual search is one prominent application trend: e.g. Alibaba Pailitao Visual Search [28], Pinterest Visual Search [27], Microsoft's Visual Search [9], and they all focus on learning image representations optimizing for image retrieval problem. Another highly developed direction is content based recommendation: e.g. [5], which leverages video contextual information to recommend customers videos with similar category/search tokens, etc.

Behavior based recommendation systems are also widely applied, They aim to recommend items viewed/purchased by users with similar preferences. Youtube [13] learn from customers' co-watched behavior for video recommendation. Amzon [2] learn from re-purchase behavior for *Buy It Again* recommendation.

PSS supports the hybrid of contextual and behavior based signals in similarity modeling, and provides enough flexibility to choose any combination for different applications (e.g. product image, purchase behavior, combination of image & title & behavior, etc.). Moreover, PSS also allows customer to give their own definition of similarity for different applications, e.g. identical products searching, visually similar products recommendation, etc.

### 2.2 Deep Metric Learning

Another related research domain is deep metric learning, which has been shown to achieve impressive performance for identifying similar items at multiple application areas: visual search [27, 28], recommendation [5, 8, 22, 23], face verification / recognition [17, 20, 21], etc. Based on the relations of items, there are mainly two types of supervision signals for similarity: pairs of items marked as positive or negative [1, 20], and triplets of items (anchor item, positive item, negative item) [7, 14, 18, 19, 21]. The pairwise supervision can be trained via deep neural networks like SiameseNets [3, 4, 11], which is usually relatively easy to train, and pairwise training data is also easier to collect at industry. In contrast, the triplets supervision can be trained by Tripletnet [7, 17, 25], which has stronger capability to preserve ranking information, but it

usually needs more careful training strategies (e.g. triplet selection) to get good results[17]. Both pair-wise and triplet supervisions are used to optimize for the item embedding, where similar items are close together, while dissimilar items are pushed far away from each other in the embedding space.

In this work, we leverage multi-task SiameseNet to train item embedding with pairwise datasets, which provides flexibility to support application requirements.

## 3 PSS FRAMEWORK

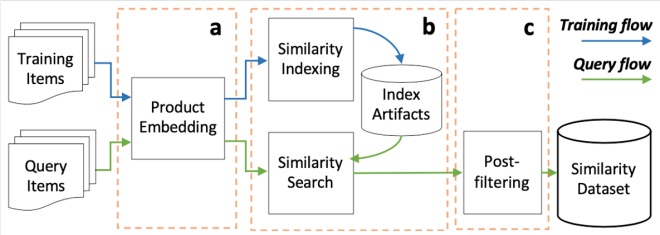



**Figure 2: PSS overall framework.**

As shown in Fig. 2, PSS similarity identification framework has three main components: a) *Product Embedding*: At the training stage, a product embedding learning model is optimized to represent each product with a feature vector. The embedding should be encoded with product's contextual & customer behavior information, and be optimized for specific application based *similarity* goals. At the query stage, each query item is passed through the trained embedding extraction model, and represented as an embedding vector. b) *Similarity Indexing & Search*: At the training stage, all product embeddings are built into index artifacts based on their similarity correlations. Then at the query stage, multiple queries are sent in parallel to retrieve top$K$ ranked similar items efficiently. c) *Post-filtering*: For any application that require further refinement on similarity results, PSS provides a general attribute-based post-filtering module, which only keeps the items whose attribute values match to those in the query item. Finally, the product similarity datasets are uploaded to the data warehouse, from where downstream applications (e.g. search, recommendation) can further consume.

### 3.1 Product Embedding Learning

The module (a) in Fig. 2 is the first step in the workflow and generates a vectorized embedding for a product, which should concisely encode key product information required for similarity identification. However, the definition of *similarity* at different applications may vary a lot. For example, similar shoes usually have similar heel type, but this similarity criteria is not suitable for the other products (e.g. electronic devices, furniture). One common solution is to carefully select suitable input signals and similarity criteria for each application. However, this is neither scalable nor efficient. To build a flexible product embedding learning framework, we use common product information applicable for multiple product types (e.g. image, title, customer behavior responses, etc.) as input signals. We leverage SiameseNet to learn from the pairwise (i.e., similar / dissimilar) relationship between products. Meanwhile, we leverage multi-task learning to cover various similarity tasks (e.g., human labeled similarity data, click-through similarity data). Then at the query stage, the product embeddings can be extracted from the last layer of SiameseNets.

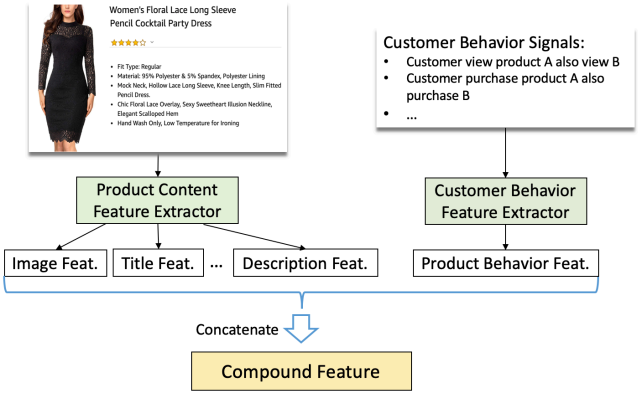



**Figure 3: Compound feature generation using both product content features and customer behavior features.**

*3.1.1 Feature Extraction from Multiple Contexts.* Fig. 3 shows the process of extracting initial product compound feature from multiple sources of raw product information. There are mainly two types of information: product content (e.g. image, title) and customer behavior responses (e.g. co-purchase, co-view).

As shown on the left branch of Fig. 3, for product content information, contextual feature for each input content domain is extracted. For example: image feature (e.g. extract via AlexNet [12]), title / description feature (e.g. extract via Word2Vec [16]), etc. Each feature extraction model is trained to focus on a specific content domain. E.g. AlexNet image embedding, which is trained on ImageNet [6] for image recognition; Word2Vec embedding, which is trained on Google News corpus for reconstructing word linguistic contexts. As shown on the right branch of Fig. 3, for product level customer behavior information extraction, an Amazon internal service is leveraged to extract product-level behavior embedding, which is a graph based solution.

As shown at the bottom of Fig. 3, after extracting features from both product content and customer behavior responses, each feature is normalized and concatenated together to get the product *Compound Embedding*.

*3.1.2 Multi-Task SiameseNets for Product Embedding Learning.* The initial product compound feature has a good coverage of all types of product contextual information, however, it's not directly optimized for similarity identification tasks. Thus, as shown in Fig. 4, we build a SiameseNet [3, 11] on top of the compound feature to learn a more discriminate product embedding from the pairwise similarity dataset. Furthermore, to enable the flexibility of optimizing for multiple similarity training goals (for example, customer behavior similarity goal, human labeled dataset similarity goal), we leverage multi-task learning.

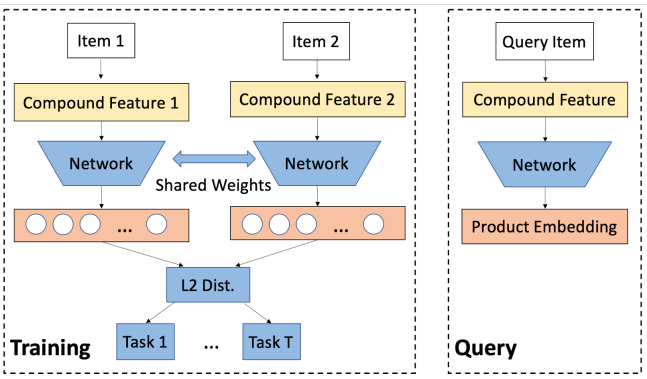



**Figure 4: SiameseNets product embedding learning.**

More specifically, in the training stage (Fig. 4 left), pairs of similarity training data are forwarded in two branches of networks with shared weights. Then at the network's last hidden layer, the outputs from the two branches are merged and their L2 distance is computed for similarity prediction. Finally, the pairwise loss (e.g. contrastive loss) of each optimization task is computed, and aggregation of weighted losses for all the tasks is the final loss. We use back-propagation to optimize the network. In the query stage (Fig. 4 right), we directly consume the learned network to extract the product embedding for each item.

Mathematically, for an item $i$, its compound feature $f_i$ passes the $L$-layer fully connected (FC) networks $FFN(.)$, and each FC layer is then followed with a batch normalization [10] layer which makes the optimization more robust. Finally, the product embedding can be represented as the last hidden layer of the network:

$$\begin{aligned} h_i^L &= FFN(f_i) \\ &= \sigma\left(\sigma\left(f_i W^{(1)} + b^{(1)}\right)\cdots\right) W^{(L)} + b^{(L)} \end{aligned} \tag{1}$$

where $W^{(1)}, W^{(2)}, \cdots, W^{(L)}$ are weight matrices and $b^{(1)}, \cdots, b^{(L)}$ are biased terms. $\sigma(.)$ is the non-linear activation function (in this paper, we use Relu [12]),

A high quality product embedding is able to pull similar items closer while push dissimilar items away from each other in the embedding space. Thus, the pairwise objective function for a task $t$ is:

$$\begin{aligned} L^t &= \frac{1}{|\mathcal{M}|} \sum_{(i,j)\in\mathcal{M}} L_{ij}^t\left(y_{ij}, h_i, h_j\right) \\ L_{all} &= \sum_{(t)\in\mathcal{T}} \alpha_t * L^t \end{aligned} \tag{2}$$

where $L_{ij}^t$ is the loss between SiameseNet left branch output $h_i$ and right branch output $h_j$ for task $t$. $y_{ij}$ is the pairwise ground truth label (similar/dissimilar), $\mathcal{M}$ is the set of training pairs, and $|\mathcal{M}|$ is the training dataset size. $\mathcal{T}$ is the set of all tasks. In this paper, we use contrastive loss for each task, and $L_{all}$ is the weighted aggregated loss for all tasks. Note $FFN(.)$ is optimized with the learning of (2) and once model training is done, $FFN(.)$ can be used to generate product embeddings.

*3.1.3 Pairwise Similarity Data Mining.* The major challenge of training a high quality product embedding from SiameseNet is to find hard positive/negative training pairs [24]. A naive way is to randomly select products from the same category as positive pairs, and products from different categories as negative pairs. However, there are lots of false positives which do not meet the application requirements (e.g. pairing a sport style t-shirt and a hip-hop style t-shirt as positive might not be appropriate), meanwhile the cross-category negatives are very easy to train, and the convergence is too quick without contributing to the learning.

In order to make the model training efficient meanwhile effective for any give task, firstly, we leverage general similarity customer behavior data sources (e.g. view-to-purchase) to mine positive and hard negative training pairs. Secondly, we leverage application focused data sources (e.g. human labeled similar fashion items) to min positive/negative training pairs. For any given similarity data source, we further conduct cleaning steps to collect high quality positive & negative training samples. More specifically, for positive pair candidates, we remove the pairs which don't have high similarity (e.g. cosine similarity < 0.6) on either image / title / product behavior embeddings. For negative training candidates, we remove the pairs which have very high similarity (e.g. cosine similarity > 0.8) on all embeddings.

## 3.2 Similar Products Indexing & Search

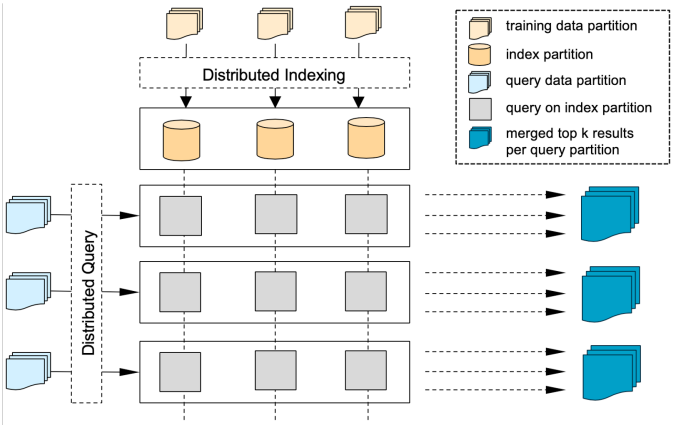



**Figure 5: Distributed similarity indexing and query**

After each product is represented with embedding, similarity indexes (Fig. 2 module b) is built, which enables similarity search in an accurate and scalable manner. k-Nearest Neighbor (kNN) is a natural and popular way to do similarity indexing and querying. The similarity indexing step targets on organizing and recording all the items in the search space (the set of products from which we want to find similarities) with a indexing structure, which is called *Index Artifacts*, and it can support fast search afterwards. Popular indexing algorithms include HNSW [15], K-D tree [26]. In PSS, we select HNSW, since it provides a good balance between accurate similarity indexing and fast searching. The querying step targets on retrieving similar items for each query from the built index artifacts. The main challenges are: 1) How to efficiently build


**Figure 6: Effects of configuring post-filter refinement logic**

index for a large-scale search space; 2) How to do similarity search at scale.

To enable a Amazon scale similar product computing, we developed a distributed KNN solution illustrated in Fig. 5. To build the index given very large dataset which cannot fit into a single machine, we divide the dataset into multiple partitions. Each partition is small enough, so it can be loaded into the memory of an index node and used to build a partitioned index. A set of the indices built from the partitioned dataset is considered as an index collection. It is referenced together to represent the entire search space for given query. Similarly, query data is also partitioned and distributed to a cluster of query nodes. Each query node searches in a partition index for the query partition and return the sorted top K similar items for each query in the partition. Finally, all partition results from the same index collection are pushed to a merge node where it heap sorts final top$k$ most similar items from all candidate lists and upload them to destination.

## 3.3 Post-filtering Refinement

As shown in Fig. 2 module (c), the last process is post-filtering refinement. For some business use cases, additional constraints are required to make the similarity identification valid. For example, two T-shirts with identical image but different sizes are for completely different customers. In order to further enhance performance, PSS provides the flexibility of adding application specific constraints through configuring a list of key attributes for similarity identification refinement. Any similarity candidates with unmatched attribute values will be removed in the post-filtering stage. Fig. 6 shows an example of attribute based post-filtering: (a) shows similar items without any filtering; (b) shows similar items with the same color as the query item; (c) shows similar items with the same dress sewing style as the query item. In production, different application have enough flexibility to choose its own list of attributes.

# 4 SYSTEM ARCHITECTURE

This section demonstrates the system architecture of PSS. As shown in Figure 7, PSS allows the users to invoke a step-based workflow for similarity computing through a service call so that both similarity training and query flows can be instantiated with

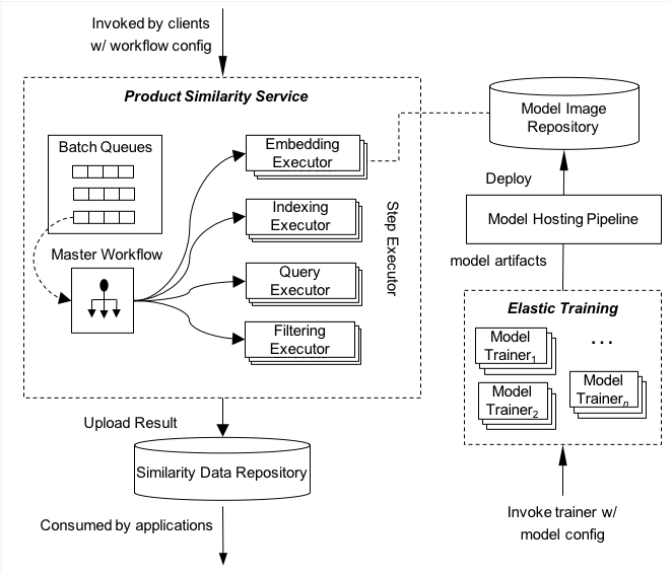



**Figure 7: Similarity computing as a service**

user requests to deliver application specific similarity results. To process data at large scale, distributed executors are orchestrated hierarchically at each step to perform the computations on partitioned data concurrently. To flexibly integrate different product embedding models to the internal search engine, a dedicated model pipeline is maintained to automate the model learning and the hosting process.

## 4.1 Cache-based Product Embedding

As mentioned in Section 3, *Product Embedding* is served as a preprocess step of similarity computing to represent each product as a vector using a pre-trained product embedding model as specified in the workflow configs. For example, the user can choose to represent a product with an image embedding, a compound embedding with both the image and title information, or an application specific embedding trained with SiameseNet.

Currently, a variety of heterogeneous Deep Learning (*DL*)-based models are hosted in our system, implemented on a common model framework with their own training and inference logics. A light-weighted elastic training solution is implemented based on *AWS Sagemaker*[1] to launch the training on demand. When the training is done, a model artifact will be pushed to a separate model hosting pipeline where the model is containerized into a *docker* image [2] and published to an embedding model repository. Later the search engine launches the model instance from its docker image to generate the embedding for each product. With such a model pipeline, it enables the model to be continuously deployed and flexibly integrated to the similarity service. When code changes related to the model packages are pushed, the images with newer version will be deployed without any changes on the search engine side.

[1]https://aws.amazon.com/sagemaker
[2]https://docs.docker.com/

For efficiency consideration, a cache mechanism is leveraged to avoid duplicate calculations in the embedding extraction which is expensive especially for the *DL*-based models. More specifically, a cache table is associated with model identifier with its embedding stored as a key-value pair. A cache handler is implanted in the base model layer to apply to all models. At embedding extraction step, it first attempts to read the embeddings from the cache table and returns immediately if hits the cache, otherwise it performs the actual embedding extraction and writes the new embedding to the cache table. Note that it is assumed that the product embedding always remains invariant for the same model. However, the only exception is when the model is updated and the inference functionality changes which is not very common in the production stage. In that scenario, we will clean up the corresponding cache table.

## 4.2 Distributed Similarity Computing

*4.2.1 Batch Framework.* The similarity search engine is built on a distributed computing framework integrated with the *AWS Batch*[3] infrastructure which enables efficient and dynamic resources provisioning and job scheduling for large scale batch processing. Under this computing framework, the resource pool is divided by batch queues configured with virtual computing environments such as number of vCPUs, scaling strategies, etc.; a *Batch* scheduler dispatches the job executions across queues in priority order on optimal computing resources. As the service is invoked, a master job is submitted to the batch queue. When the requested computing resource is provisioned, a step-based workflow is initiated based on the user's request, following a predefined workflow graph as shown in Fig. 7.

*4.2.2 Distributed Executors.* Each step in the master workflow is scaled by step executors. A primary executor is initiated to index the source partitions in a *manifest* file and then distribute the partitions to an array of secondary executors for parallel processing. Each secondary executor is assigned a set of partitions according to the *manifest* and is dedicated to process them locally. Both the *manifest* and the partition size can be tuned for an appropriate concurrency level to achieve the overall scaling performance. Multi-level *manifests* are used for extreme large-scale processing.

This distribution solution applies to all the major steps. As mentioned above, embedding executors compute the product vectors on the partitioned product set for similarity computing; filter executors refine the similarity candidate partitions. The similarity training and query are implemented in a similar way. In training, data is split to fit into a single indexing executor node. In querying, the search executors first perform the top$K$ query on partition indexes distributively and then a merger collects and merges the all the partition results from the same query for a final top$K$.

# 5 EXPERIMENTS & ANALYSIS

In this section we analyze experimental results from three aspects: 1) Offline-evaluation: we describe offline experiment setups, then test two tasks, pairwise product similarity verification, and similarity ranking. 2) Online-evaluation: we show the similarity identification performance in online A/B test on Amazon products. 3) System performance: we test system scalability on large-scale similarity computing tasks. At last, we visualize the similarity identification results deployed for multiple productions.

[3]https://aws.amazon.com/batch

## 5.1 Offline Evaluation Setups

*5.1.1 Input Contextual Information.* We mainly experiment on the *Compound* feature (Section 3.1.1), which is the integration of three input contextual information: 1) *Image*: we use AlexNet[12] to extract initial image embedding, 2) *Title*: we use Word2Vec [16] to extract word-level embedding, then pool all word embeddings to get a title-level embedding, 3) *ProdBehavior*: we use an internal service to learn product embedding based on customer behavior (e.g. co-view, co-purchase). We further reduce each feature's dimension to 128-dim for computation efficiency. In Section 5.2 & 5.3, unless explicitly written, the default SiameseNet input is the Compound feature that integrates image, title, and ProdBehavior features.

*5.1.2 Data Sources for Model Training.* We introduce two resources for training: customer view-to-purchase (V2P) data, and human expertise labeled (Labeled) data. *V2P Data*: It has huge volume of candidates, but the data quality is relatively noisy. For positive product pairs, we select the most recent viewed products, and the purchased products right afterwards. For hard negative pairs, we select the product viewed N (e.g. N=20) slots before the purchased product. For both positive/negative pairs, we further do data cleaning (refer to Section 3.1.3) to remove behavior data irrelevant to similarity. In this experiment, we randomly select 100K positive pairs, and 500K negative pairs for V2P-based similarity task training. *Labeled Data*: it's an accurate data source, but usually only contains limited number of data since it's labor intensive to collect. For positive pairs, we collect domain experts labeled substitute products (e.g. substitutable apparel products should have similar image, title, color, material, etc.). For negative pairs, we randomly sampled product pairs with 5x volume of positive pairs. For model training, we split 4K positive human Labeled pairs with 20K negative pairs for human label based similarity task. The remaining human labeled pairs are for model evaluation, which will be explained further in Section 5.2.

## 5.2 Product Similarity Verification

To evaluate the learned product embedding quality (Section 3.1), we conduct experiments on pairwise similarity verification tasks: model performance on predict similar/dissimilar labels for pairs of products. For evaluation, we select 5K positive pairs labeled by human expertise (Section 5.1.2), and randomly select 25K negative product pairs. The evaluation metrics are recall at precision 80%, 90%, 95%.

*5.2.1 Effects of using different input contextual information.* To quantify the effects of introducing multiple product context information (Section 3.1.1), we test different combinations of input contextual information. Comparing on R@P80% metric, compound feature (concatenating image, title and ProdBehavior) gets 43.32%, while image only gets 29.61%, ProdBehavior only is 32.44%, and Title only is 7.36%. Thus, different input signals are complementary to each other, and aggregating all contextual input information can bring to the best performance on this similarity verification task.

*5.2.2 Effects of Multi-Task Learning.* To understand the effects of supporting configurable multiple sources of similarity training signals, all the numbers in Table 1 are tested on different combination of multi-task learning (Section 3.1.2) with the same SiameseNets structure: four layer fully connected with hidden nodes number: $256+256+256+256$. As shown in Table 1 second line, if we only leverage the most general product similarity signal V2P, we're able to return some high precision substitute products with reasonable recall, which can be a good option when no extra application specific training signal is available. As shown in line 3, when only training with a small set of human expertise labeled data (e.g. 4k human labeled positive pairs), we can get slightly better results with V2P. As shown in line 4, when integrating multiple optimization signals with multi-task learning, the optimal results can be achieved via learning complementary information.

**Table 1: Comparison of different optimization tasks**

| Data Augmentation | R%@P80% | R%@P90% | R%@P95% |
|---|---|---|---|
| Compound Feature | 43.32 | 32.31 | 22.99 |
| View-to-Purchase(V2P) | 48.53 | 18.06 | 8.62 |
| Labeled 4K | 52.87 | 19.92 | 7.93 |
| V2P + Labeled 4K | **90.46** | **78.78** | **68.59** |

*5.2.3 Effects of Different SiameseNet Structures.* We conduct this experiment to understand the effect of using different SiameseNet structures (Section 3.1.2). All the models tested in Table 2 are trained with two tasks: V2P + Labeled 4K. From the fist two lines of Table 2, we can clearly see the product embedding learned from SiameseNet is much better than the original compound feature (refer to Section 3.1.1). Comparing line 2-5, we can clearly see adding more layers will progressively lead to higher performance than shallower one when there is enough training data, but the extra gain might be relatively small when the network is almost saturated with the given training data (line 4 vs 5). Comparing the line 4 & 6, or line 5th & 7th, we can observe deeper network performs slightly better than wider net.

**Table 2: Comparison of different SiameseNet structures**

| SiameseNet Struct. | R%@P80% | R%@P90% | R%@P95% |
|---|---|---|---|
| Compound Feature | 43.32 | 32.31 | 22.99 |
| 256 | 79.33 | 53.00 | 33.33 |
| 256 + 256 | 83.27 | 64.11 | 44.04 |
| 256 + 256 + 256 | 88.95 | 74.42 | 61.39 |
| 256 + 256 + 256 + 256 | **90.46** | **78.78** | **68.59** |
| 512 + 256 | 85.53 | 66.88 | 50.39 |
| 512 + 256 + 256 | 89.18 | 77.03 | 64.31 |

## 5.3 Similarity Ranking

To evaluate the performance on ranking task, we test on a ranking dataset collected from click-through data in substitute recommendation. It has 15K query products, and in-total 150K products in searching scope. For ranking evaluation, we generate top$K$ substitutes for each query product. Similar to [13], We use Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) for performance evaluation. *NDCG* measures the quality of ranking. Mathematically, the formula of DCG at rank position k is defined as: $DCG_k = \sum_{i=1}^{p} \frac{2^{rel_i}-1}{\log_2(i+1)}$, where $i$ is the position in the recommended list and $rel_i \in \{0,1\}$ stands for whether the $i$th recommendation is labeled positive. NDCG is the ratio between DCG and the largest DCG possible. *MAP* measures the average accuracy of a ranking algorithm. A query product's AP is the area under the Precision-Recall curve of that query: $AP = \int_0^1 P(r)dr \approx \sum_{k=1}^{K} P(r)(r(k) - r(k-1))$, where $P(r)$ is the precision at recall level r and $r(k)$ stands for the recall at top $k$ recommendations with $r(0) = 0$. In practice, AP is approximated by a finite sum. We choose $K = 30$ in our calculation. MAP is the mean of AP over all the queries. For the SiameseNet model, we use $256 + 256 + 256 + 256$ structure.

*5.3.1 Comparison on Different Product Embedding.* Comparing first 3 rows of Table 3, image feature relatively have high performance as a single contextual information source, since image is usually more easily to be captured by customers. Comparing line 4 with first 3 lines , aggregating more product contextual information (all vs image/ProdBehavior/Word2Vec only) can capture more complementary signals, and get better similarity ranking results. Comparing line 4 & 5, we can observe product embedding learning through SiameseNets can encode extra information for similarity identification, and generate better ranking results.

**Table 3: Ranking results on different product embeddings**

| Prod. Embeddings | NDCG@1 | NDCG@10 | NDCG@20 | MAP |
|---|---|---|---|---|
| Image | 0.3585 | 0.4661 | 0.4778 | 0.4448 |
| ProdBehavior | 0.2712 | 0.3593 | 0.3712 | 0.3433 |
| Title | 0.2974 | 0.3907 | 0.4031 | 0.3729 |
| Compound | 0.3928 | 0.5093 | 0.5255 | 0.4899 |
| SiameseNet (ours) | **0.4143** | **0.5538** | **0.5720** | **0.5267** |

*5.3.2 Performance on Different Product Categories.* As shown in Table 4, for different categories, customers show strong signal preferences, which further justify the importance of providing flexible similarity signals for multiple applications. For example, for fashion categories (e.g. apparel, shoes) and home category, the image is the dominant signal. While for other general categories, e.g. PC, toy, aggregating multiple sources of contextual information can better help customers with multiple preference. We also observe the SiameseNet based product embedding can overall do better similarity ranking across multiple product categories.

## 5.4 Online A/B Test

To evaluate the effectiveness of our models in real world, we conduct the online A/B test for typical similarity applications in E-commerce. To measure the actual customer engagement differences between the control (A) and treatment (B) groups, the lift in Purchased Units are collected for the applications in Amazon.com. Note that we only report the purchased units lift due to the page limits as other important metrics such as products purchased profits lift, etc. are highly aligned.

**Table 4: Average precision for click-through data**

| Category | pc | apparel | jewelry | luggage | home | toy | office product | shoes | kitchen | All |
|---|---|---|---|---|---|---|---|---|---|---|
| Image | 0.3567 | 0.4662 | 0.3223 | 0.4602 | **0.7303** | 0.2692 | 0.3152 | 0.4127 | 0.2905 | 0.4448 |
| ProdBehavior | 0.3168 | 0.1328 | 0.7367 | 0.4271 | 0.2087 | 0.4691 | 0.5807 | 0.1570 | 0.5317 | 0.3433 |
| Title | 0.3514 | 0.1707 | 0.6373 | 0.4444 | 0.2552 | 0.5281 | 0.6435 | 0.2171 | 0.6118 | 0.3729 |
| Compound | **0.4664** | 0.2771 | **0.7624** | **0.5570** | 0.4051 | **0.5982** | **0.6865** | 0.3332 | **0.6798** | 0.4899 |
| SiameseNet (Ours) | 0.4476 | **0.5687** | 0.6006 | 0.5392 | 0.6901 | 0.4659 | 0.4688 | **0.4328** | 0.5080 | **0.5267** |

*5.4.1 Substitutes Recommendation for OOS Products.* This recommendation application displays the top similar substitutes for customers when their selected products are out-of-stock (OOS). We choose this application because it has good coverage on most of the product types (e.g. apparel, furniture, electronic devices, etc.). For the treatment group, the substitute data is learned from product image and behavior information, and constrained on key attributes matching (e.g. material, color, etc.) through PSS. For the control group, the shown recommendation is the voting of best results from all in-production OOS recommendations methods. This experiment results in a significant lift on number of purchased items (+6.58%, p=.03) in the treatment vs. control during four weeks' experiment period.

*5.4.2 Similar Fashion Product Recommendation.* This application shown on personalized Amazon Fashion page is aimed to encourage customer to explore similar new fashion products based on their selections. For the treatment group, the similar fashion items are learned from items with both similar images and title / description through PSS. For the control group, it's the voting of best results from all in-production personalized recommendations for fashion products. This experiment results in a lift on purchased units (+1.02%, p-value = 0.04) in the treatment vs. control during four weeks' experiment period.

## 5.5 Distributed Computing Performance

To evaluate the scalability of our PSS service, we compare the similarity computing performance of our distributed solution with the non-distributed one. To setup the environment, the computing queue is configured with a cluster of virtual computing nodes hosting on an EC2 host pool.

Four groups of tests are prepared with varied input size ranging from 1MM to 100MM of products. Each input dataset is a list of 128-dim image embeddings of products extracted from several related categories and used as both indexing and query input within the test group. For each group, similarity indices are first built using the given dataset and then the top 1000 nearest neighbors are searched from the indices built for each item in the set. This flow is performed in both non-distributed and distributed scenarios. In the non-distributed case, single executor is launched and provisioned with sufficient resources to process the entire dataset, specifically the memory which is considered as the bottleneck in a traditional KNN solution; in the distributed case, we scale up to 500 partitions in query while restrict the concurrency level in indexing to 5 partitions for each dataset. This is to minimize the overhead in merging topK from multiple index models. The amount of resource allocated to a distributed executor is in proportion to the size of the data partition assigned to it.

**Table 5: Performance of Distributed Similarity Computing**

| Data Size | Indexing | | Query | |
|---|---|---|---|---|
| | Speedup | Cost Save | Speedup | Cost Save |
| 1MM | 4.65 | 78.48% | 41.63 | 33.14% |
| 10MM | 4.40 | 77.26% | 48.42 | 46.16% |
| 50MM | 5.33 | 81.22% | 70.33 | 61.90% |
| 100MM | 3.52 | 71.60% | 77.82 | 63.92% |

Table 5 demonstrates the performance and efficiency improvement that our distributed similarity computing can achieve when benchmark against a non-distributed solution. From the result, we can see that both distributed indexing and query constantly outperforms the corresponding non-distributed solution. Although distribution introduces overhead from both resource provisioning and partition merging, it is able to speed up 4.5X on average in index-building with 5 partitions on the training dataset, and 59.6X in searching for top 1000 similar items with 500 partitions on the query dataset while still as good as the non-distributed with a 99.5% overlap of results. With current setting, the indexing can achieve its optimal performance when the dataset is divided into 10MM items per partition; in query, the performance improves as the data size becomes larger, distribution on smaller partitions does not gain as much as the bigger ones as the above-mentioned overhead dominates over the actual searching time. We also compare the costs in both solutions based on the total vCPU and memory allocation to the executors. Although more executors are launched in the distributed version, each executor tends to run much faster with less resource which substantially improves the overall resource utilization. As a result, we are able to save as much as 77.14% and 51.28% of resource on average in distributed indexing and query respectively.

## 5.6 Visualizations

As shown in Figure 8, our PSS can support flexible similarity definition for multiple applications via configuring input contextual information, and application-related optimization goals. For example, in the duplicated product identification application (Figure 8 top), image and title information are consumed as inputs, and the optimization goal is to identify products with near-identical product image / title as similar. The visual similar recommendation and substitute product search are the visualization for the productions discussed in Section 5.4.


**Figure 8: Visualizations of different similarity applications.**

## 6 CONCLUSIONS & FUTURE WORKS

This paper presents PSS, a flexible and scalable similarity identification system, which supports different product similarity computation applications. With configurable SiameseNet-based models, PSS is capable of generating various high quality application-specific product embeddings and is scalable of efficient similarity computations at Amazon scale owning to its cloud-based distributed system framework. In the future, we will extend our solution to identify similarity among entities with richer information such as video or audio. We will support different types of product embedding such as triplet model or graph-based models to fit more production use cases. We are also considering making PSS a public service available for research communities to use.

## REFERENCES


[1] Sean Bell and Kavita Bala. 2015. Learning visual similarity for product design with convolutional neural networks. *ACM TOG* 34, 4 (2015), 1–10.

[2] Rahul Bhagat, Srevatsan Muralidharan, Alex Lobzhanidze, and Shankar Vishwanath. 2018. Buy It Again: Modeling Repeat Purchase Recommendations. In *ACM SIGKDD*. 62–70.

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*. 737–744.

[4] Dahjung Chung, Khalid Tahboub, and Edward J Delp. 2017. A two stream siamese convolutional neural network for person re-identification. In *Proceedings of the IEEE International Conference on Computer Vision*. 1983–1991.

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

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[7] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*. Springer, 84–92.

[8] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of the 26th international conference on world wide web*. 193–201.

[9] Houdong Hu, Yan Wang, Linjun Yang, Pavel Komlev, Li Huang, Xi Chen, Jiapei Huang, Ye Wu, Meenaz Merchant, and Arun Sacheti. 2018. Web-scale responsive visual search at bing. In *ACM SIGKDD*. 359–367.

[10] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[11] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML*.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[13] Joonseok Lee, Sami Abu-El-Haija, Balakrishnan Varadarajan, and Apostol Natsev. 2018. Collaborative deep metric learning for video understanding. In *ACM SIGKDD*. 481–490.

[14] Rui Lu, Kailun Wu, Zhiyao Duan, and Changshui Zhang. 2017. Deep ranking: Triplet MatchNet for music metric learning. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 121–125.

[15] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE TPAMI* (2018).

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

[17] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[18] Devashish Shankar, Sujay Narumanchi, HA Ananya, Pramod Kompalli, and Krishnendu Chaudhury. 2017. Deep learning based large scale visual recommendation and search for e-commerce. *arXiv preprint arXiv:1703.02344* (2017).

[19] Kihyuk Sohn. 2016. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in neural information processing systems*. 1857–1865.

[20] Yi Sun, Xiaogang Wang, and Xiaoou Tang. 2014. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1891–1898.

[21] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1701–1708.

[22] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 2018 World Wide Web Conference*. 729–739.

[23] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*. 2643–2651.

[24] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. 2014. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1386–1393.

[25] Kilian Q Weinberger and Lawrence K Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research* 10, Feb (2009), 207–244.

[26] wikipedia. [n. d.]. k-d tree. https://en.wikipedia.org/wiki/K-d_tree.

[27] Andrew Zhai, Hao-Yu Wu, Eric Tzeng, Dong Huk Park, and Charles Rosenberg. 2019. Learning a Unified Embedding for Visual Search at Pinterest. In *ACM SIGKDD*. 2412–2420.

[28] Yanhao Zhang, Pan Pan, Yun Zheng, Kang Zhao, Yingya Zhang, Xiaofeng Ren, and Rong Jin. 2018. Visual search at alibaba. In *ACM SIGKDD*. 993–1001.