What Do We Need for Industrial Machine Learning Systems? Bernoulli, A Streaming System with Structured Designs

Qiang Luo, Xiang-Rong Sheng, Jun Jiang, Chi Ma
Shuguang Wang, Haiyang He, Pengtao Yi, Guowang Zhang
Yue Song, Guorui Zhou, Hongbo Deng, Xiaoqiang Zhu
sherlock.lq,xiangrong.sxr,chuncao.jj,beiji.mc
yuanchu.wsg,haiyangg.hhy,youdan.ypt,guowang.zgw
yue.song,guorui.xgr,dhb167148,xiaoqiang.zxq@alibaba-inc.com
Alibaba Group
Beijing, China

ABSTRACT

Machine learning has empowered Internet businesses, such as search engines, recommender systems, and online advertising. However, far less attention focuses on developing efficient industrial machine learning systems for these services. Industrial business services have the characteristics of structured data that dynamically change and come as a stream, strong demand for algorithm iteration, and limited resources budget for many business scenarios. Directly applying off-the-shelf systems for these services causes issues such as delayed model updates, slow experiment cycles, and inefficient resource usage. The reason is that the general-purpose design of current mainstream systems doesn't take the characteristic of these industrial machine learning tasks into consideration.

In this paper, we present Bernoulli, a streaming system developed for industrial machine learning tasks. Bernoulli builds on online streaming data, where machine learning models are directly experimented with and deployed on the streaming data. In contrast to the conventional manner that keeps separate offline iteration and online deployment pipelines, Bernoulli unifies these two pipelines, enabling algorithm iteration on online streaming data and allowing the model to refresh in real-time. Models can then be seamlessly deployed into production, saving resources and human cost. The core design principle of Bernoulli is building the system in a structured manner throughout the entire flow by utilizing the structured character of industrial data. We will describe the functionality of each component behind the design. Facilitated with Bernoulli, the engineering teams can directly experiment with new approaches on online streaming data, manipulate the training data easily for more types of algorithm trials, reduce resource consumption, and obtain high-quality fresh models for online serving. At present, hundreds of Alibaba services have used Bernoulli in production, showing compelling performance.

KEYWORDS

Machine learning system; continuous learning; neural networks

1 INTRODUCTION

Machine learning has driven advances in many application domains, ranging from computer vision [9, 25], natural language processing [6, 10, 23] to reinforcement learning [29]. One key element of the advances is the rapid development of the general machine learning systems [1–4, 20], on which researchers can explore new ideas conveniently.

Recently, there is a wide-spread trend of utilizing machine learning to empower online businesses, such as search engine [8], recommender system [5, 24, 30] and online advertising [31, 32]. These industrial applications have the following characteristics: (a) structured data that dynamically change and come as a stream. These make these industrial applications different from applications with static data such as computer vision. Figure 1 shows the typical data units in industrial applications. (b) A large number of business scenarios with only limited resources. (c) Strong demand for algorithm iteration to improve user satisfaction and business revenue. Here, the term "algorithm iteration" refers to experiment with new approaches. which includes two aspects in industrial applications: (1). **Data-level iteration**. Data-level iteration further includes feature iteration and sample iteration. Feature iteration means changing the feature description of samples, e.g., adding or deleting features. On the other hand, sample iteration operates data on the sample level. Sample iteration includes but is not limited to altering the labels of samples, augmenting or downsampling samples, and data fusion from multiple sources and tasks. (2) **Model-level iteration**. Model-level iteration refers to experimenting with novel model architectures, optimization algorithms, loss functions, and other advanced techniques at the model level.

In contrast to the rapid development of algorithms, far less attention focuses on the equally important problem of developing powerful machine learning systems for industrial applications. Directly applying off-the-shelf systems neglecting the characteristic of industrial machine learning tasks, causing issues such as delayed model updates, slow experiment cycles, and inefficient resource usage. To better utilize the characteristic of industrial machine learning tasks, we argue that a good industrial system design should adopt the following principles:

(1) Data in industrial applications are structured, dynamically changing, and come as a continuous stream. The models should update continuously to keep fresh. A good system design should enable models to **refresh fast** on streaming data.
(2) To pursue performance improvement for online businesses, the design choice should **accelerate algorithm iteration**.
(3) Industrial applications often have to deal with various business scenarios simultaneously with limited resources. These

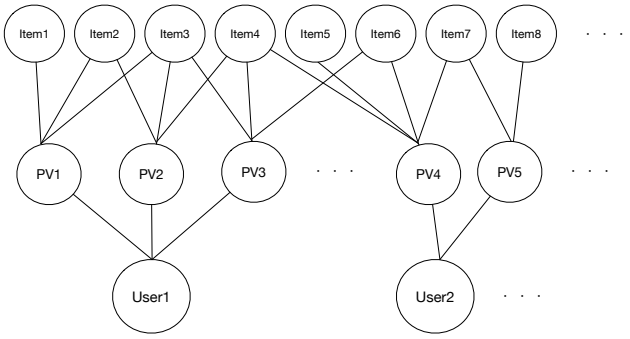

Figure 1: An example of structured data units in industrial applications, where basic data units are connected with each others by various relations. For each page view (PV) from an user, it displays a set of items. Each <*user, PV, item*> triplet constitute a new data unit. The data unit of *user* contains information regarding the user and the data unit of *item* contains information about the item.

tasks have commonalities. It is desirable for the system to **reuse resources** effectively.

In this paper, we present Bernoulli, a streaming system with structured designs adopting these principles. Bernoulli builds on online streaming data, where machine learning models are directly experimented with and deployed on the streaming data. The core design principle of Bernoulli is implementing the system in a structured manner throughout the entire flow by utilizing the structured character of industrial data. To this end, Bernoulli integrates structured data processing pipeline, structured samples, structured model training, and structured online serving into one system. Facilitated with Bernoulli, the engineering teams can directly experiment with new approaches on online streaming data, manipulate the training data easily for more types of algorithm trials, reduce resource consumption, and obtain high-quality fresh models for online serving.

Up till now, Bernoulli has been deployed in production at the display advertising system of Alibaba, serving the core commercial advertising services. We describe the functionality of each component behind the design. We also present the case study of one deployment of Bernoulli, where Bernoulli helps us fuse the training data easily for multiple scenario CTR prediction [28]. At present, hundreds of Alibaba services have used Bernoulli in production, showing compelling performance. We believe the structured design philosophy, techniques, and lessons learned will drive forward the development of industrial machine learning systems.

2 RELATED WORK

Machine learning systems have been continuously contributing to the successful application of machine learning techniques [1, 14–18, 26, 27]. In academic research, most of the focus is placed on exploring new ideas, ranging from novel model architectures [9], optimization algorithms [13], to other machine learning techniques [11]. The goal of academic experimentation is to validate the efficacy of the proposed approaches. Therefore, the common workflow is

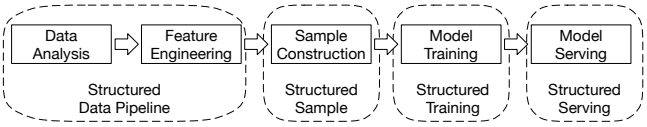

Figure 2: A high-level component overview of Bernoulli.

relatively simple: researchers first prepare the data by themselves and train a model based on an open-source machine learning library such as PyTorch [20] and TensorFlow [1]. The model is then evaluated on the test data. However, the workflow becomes way more complicated when facing Internet online services, where data are streaming, dynamically changing, large-scale, and often high-dimensional and extremely sparse [12]. In concrete, the workflow of deploying machine learning models on industrial serving includes data processing pipeline, storage of samples, model training, and online serving.

Most current machine systems implement a subset of the aforementioned components of the workflow. Some systems mainly focus on analyzing and processing data at large scales, such as Apache Flink [7]. Some systems are developed to ease the training of ML algorithms. The representative systems include TensorFlow [1], MXNet [4], PyTorch [20] and XDL [12]. There are also some serving systems, like TensorFlow-serving [19], designed for production environments. However, these systems all focus on a subset of challenges for deploying machine learning models on production.

As mentioned above, industrial machine learning systems should support fast algorithm iterations on online streaming data, reducing resource consumption, and continuously producing fresh models. Besides, it should ease the difficulty of experimenting with multiple models, serving multiple models at production simultaneously for industrial applications. Different from the aforementioned systems, Bernoulli provides a structured design system that incorporates data processing pipeline, storage of training samples, model training, and online serving into one system. By this means, Bernoulli enables experiment new approaches on online streaming data quickly with relative ease while increasing the scalability and stability of online serving.

The work that is most similar to Bernoulli is TensorFlow Extended (TFX) [2], a TensorFlow-based machine learning platform implemented at Google. TFX also supports continuous training and serving with production-level reliability. The main difference is that TFX is designed for general-purpose machine learning. In contrast, Bernoulli focuses on industrial machine learning services, which lays great emphasis on model freshness, algorithm iteration on online streaming data, utilization of industrial structured data to reuse resources. To this end, Bernoulli adopts the structured design philosophy throughout the entire workflow, including structured data processing pipeline, structured samples, structured model training, and structured online serving. Our current experience with Bernoulli is encouraging, hundreds of online commercial services at Alibaba have been deployed on Bernoulli and the online serving shows compelling performance.

3 OVERVIEW OF BERNOULLI

In industrial applications, there are lots of machine learning tasks running simultaneously on the system with changing and structured data. Figure 2 shows a schematic workflow overview of deploying machine learning models into production, which consists of data analysis, feature engineering, sample construction, model training, and model serving. Instead of building each component in isolation and simply stitching them together, Bernoulli unifies the structured design philosophy throughout the entire workflow. At a high level, Bernoulli instantiates a pipeline that involves structured data pipeline, structured samples, structured training, and structured serving:

- **Structured data pipeline**. Current machine learning systems often treat each learning problem independently and construct training data separately. However, industrial data from different tasks and experiments often share most of the information. For example, when feature iteration needs to add a new feature to current samples, it only differs from the previous data by a single feature. To allow efficient reuse of the data, Bernoulli adopts the structured data pipeline. The structured data pipeline extracts common information from different tasks. This inter-task level information reuse is achieved by abstracting dataflow at the basic data unit level. The task-specific dataflow is assembled by the time the dataflow is consumed. The structured data pipeline saves storage and computation costs significantly. By this means, Bernoulli supports direct algorithm iteration on online streaming data, where each task can construct its own specific online dataflow and complete training and inference. The structured data pipeline accelerates the velocity of algorithm iteration while reducing the development cost.
- **Structured samples**. Basic data units in the industry contain heterogeneous information and connect with each other by various relations. For example, a data unit of a user might contain the behavior data and other user information. The conventional way to solve a machine learning task is to concatenate all the data units the task needed to obtain the most fine-grained data units as input features. We will refer to the fined-grained data units as *plain samples*. Plain samples will be fed as the input of the task. However, there is a lot of redundancy in plain samples. For example, a user's information might occur multiple times in different samples. In this situation, the data of this user will be stored repeatedly by multiple times, causing redundancy. To address this issue, Bernoulli adopts the structured design philosophy, which maintains the structured information. This is achieved by the design of structured samples. Structured samples reuse common information and compress plain samples to a large extent. The structured information is maintained until the data is consumed.
- **Structured training and serving**. Data from the industrial applications are high dimensional and extremely sparse, which makes it different from applications with dense and static data such as image classification. The common way to model the high dimensional sparse data is to transform it into low dimensional embeddings firstly and then treat it like dense data. Keeping separate embeddings per model brings serious storage and computational challenges to the system, especially when multiple services with hundreds of serving and experimenting models are running simultaneously. In Bernoulli, we adopt the principle of structured training and serving, which is achieved by *structured embedding*. Structured embedding means that different tasks share the same embeddings. The use of structured embedding is motivated by the observation that embeddings contains some extent of task-agnostic semantic information and can be reused. Therefore, Bernoulli let different tasks share most of the embeddings while also keeping a small set of private embeddings. By this means, Bernoulli takes a structured training manner, facilitating algorithm iteration. For instance, when experimenting with new model structures, one can easily use the shared embedding as initialization instead of training it from scratch, which accelerates the training process.
  The structured embeddings also allow structured serving, which relieves the storage and computational burden while enhancing the stability and scalability of online serving. At the same time, the structured serving decouples the update of embeddings and other parts of the model. Since embeddings account for most of the parameters, only refreshing other parts of the model enables more frequent model updates, achieving better freshness.

Overall, Bernoulli builds these components to adhere to the aforementioned design principles and seamlessly integrating these components into a single system. The structured design allows engineering teams to make fast algorithm iterations and improve the performance of online serving. In what follows, we will describe these components in detail and explain how these components contribute to the fast algorithm iteration and improve online services at Alibaba.

4 STRUCTURED DATA PIPELINE

Data in industrial applications are organized in a structured manner naturally, where basic data units are connected with each other by various relations. Take click-through rate (CTR) prediction as an example, for each page view (PV) from a user, it displays a set of items. Each <*user, PV, item*> triplet constitute a new data unit. The data unit of *users* contains information regarding user behavior and user profile and the data unit of *item* contains information about the item like the price. *users* are connected with *items* through the page views. To utilize the data characteristic, Bernoulli employs a structured data pipeline.

In this section, we first introduce *sample skeleton* in Bernoulli, which is defined as the reusable units in the structured data pipeline. Then we introduce our implementation of *sample factory* that abstracts dataflow in the basic data unit level. The sample factory takes the responsibility of combining the sample skeleton with other data units to form the input data for each task. Finally, we show how the structured data pipeline enables efficient algorithm iteration on online streaming data, reduces resource consumption, and saves development cost.

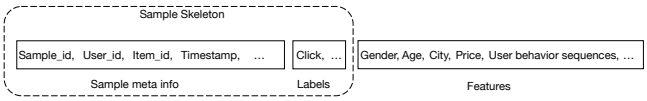

Figure 3: Sample skeleton for CTR prediction. The reusable data unit is defined as the sample skeleton. In CTR prediction, the sample skeleton consists of Sample_id that is the unique ID for each sample, User ID, Item ID, Timestamp, Click label, and other basic features that can be used to join other features like gender and age. Different tasks can combine sample skeleton with other data to assemble its input data format.

4.1 Sample Skeleton

The common way to organize input data for machine learning tasks is to abstract data at the task-level. The task-level abstraction treats each task independently and organizes task-specific data for each task. In this situation, each experiment for a particular task requires constructing a unique procedure of data processing pipeline. Although the task-level abstraction supports a variety of data types, this general-purpose design is not particularly suitable for industrial data, where data are highly structured. In concrete, different tasks often share the vast majority of the data and only differ by a small portion. Task-level abstraction can cause large redundancy and additional resource consumption. For example, if engineering teams wants to experiment with new approaches, e.g., adding new features to current samples, combining data from different sources, they need to re-develop a new dataflow to construct its input data format. The re-development is both inefficient and time-consuming, which is due to the lack of flexibility and information reuse. As a consequence, this conventional way of task-level abstraction is not beneficial to algorithm iteration.

Different from the task-level abstraction, Bernoulli abstracts machine learning tasks from the basic data unit level. The motivation of the data-level abstraction is to utilize the structured format of industrial data to reuse resources. In particular, Bernoulli treats different data units as independent dataflows. For each task, Bernoulli defines the reusable data unit as *sample skeleton*. Take CTR prediction as an example, the goal of CTR prediction is to predict the probability of the *user* to click the *item*. As shown in Figure 3, we let the sample skeleton consist of Sample_id that is unique for each sample, User ID, Item ID, Timestamp, Click label, and other basic features that can be used to join other features like gender and age. Other tasks can define its sample skeleton, and access other dataflows to assemble its final data format. Then developers can use the assembled final data to finish training and inference. The data-level abstraction of Bernoulli allows efficient data iteration. Different data iteration tasks can reuse the sample skeleton and combine it with other dataflows to constitute the input data. The structured data pipeline avoids the waste of resources of building a whole new dataflow for each task.

4.2 Sample Factory

The structured data pipeline in Bernoulli is implemented by **sample factory**, which consists of the log parser, feature center, feature joiner, sample pool, and sample builder. Figure 4 shows the structure of the sample factory. The sample factory takes the raw dataflows back flowed from online service as the input and output of the target dataflow. The functionality of each component is given as follows:

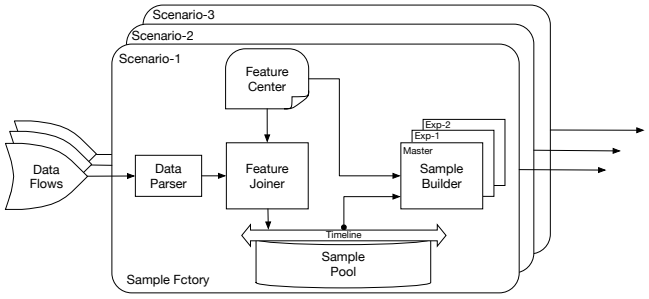

Figure 4: The functionality of the structured data pipeline is achieved by the sample factory. Sample factory abstracts dataflow in the minimal basic data unit level, the input dataflows are assembled by the time the dataflow is consumed.

- The raw **data flows** include sample skeleton, real-time features, and other data units and are fed as input to the sample factory.
- **Log parse** is the module that parses input dataflows.
- **Feature center** stores the feature that are less time-sensitive, which we refer to these features as daily features.
- **Feature joiner** is the module that joins features from the feature center and real-time dataflow to construct real-time samples.
- **Sample pool** stores samples obtained from the feature joiner in chronicle order. For each business scenario, the samples that contain the set of features used in the serving model are stored. The stored data can be used for the following tasks, e.g., feature iteration. The samples will be stored in the sample pool for tens of days.
- **Sample builder** further processes the sample from the sample pool with features from feature centers to obtain the final training samples for each task. For example, the sample builder can sub-sample data or combining multiple dataflows for sample iteration.

Facilitated with the sample factory, when building a new dataflow for iteration, we can create a new sample builder. This sample builder loads data from the sample pool and feature centers and combines them to output the desired dataflow. For example, when the engineering team needs feature iteration or sample iteration, they can create a new sample builder to combine the samples from the sample pool and feature center to generate the task-specific samples. Overall, the sample factory reuses the basic information, saving the resource consumption of building a new dataflow. This implementation enables efficient data-level iteration, including feature iteration and sample iteration.

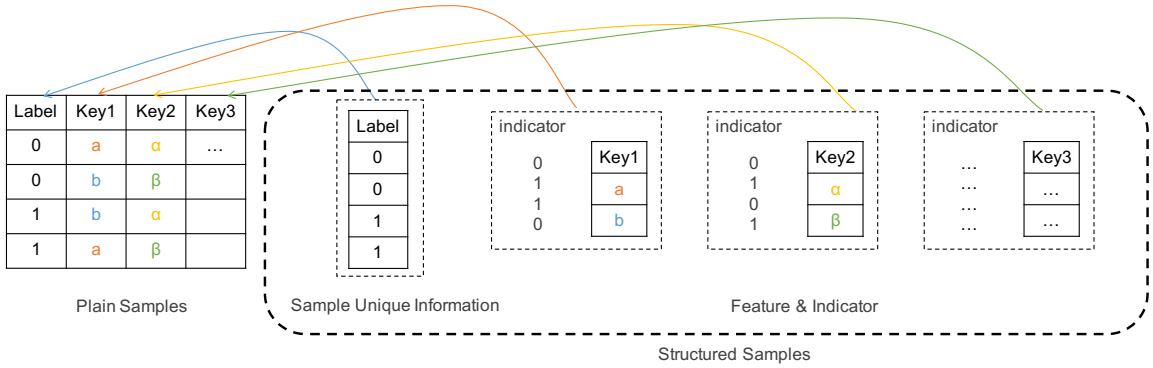

Figure 5: An example of the construction of structured samples. Structured samples reuse common data units for each column. Instead of storing all the data units, structured samples extract reusable data units and keep them as the dictionary. The indicator indicates the position of the data unit in the dictionary. By this means, structured samples compress the plain samples effectively.

4.3 Algorithm Iteration on Online Streaming Data

The design of the sample pool that stores history samples in chronicle order enables fast algorithm iteration on online streaming data. Concretely, to experiment with new approaches, the conventional paradigm consists of three steps: 1. Generate data of the latest N days. 2. Train a base model using the generated offline data. 3. Fine-tune the base model using online streaming data. The reason why this paradigm can not directly experiment with online streaming data is that the model needs a large amount of training data to converge. Since the conventional paradigm cannot cache historical samples, the model needs to wait for real-time data during training, which is time-consuming. To accelerate the algorithm iteration, it is a waste of development cost to generate offline data from petabytes (PB) of raw log data. Different from this paradigm, Bernoulli supports direct algorithm iteration on online streaming data, which is achieved by the sample pool. Since the sample pool caches data in chronological order, new experiments can use the data in the sample pool for continuous training on fresh data. By this means, the sample factory allows direct algorithm iteration on online streaming data, saving resource consumption of building offline training stage while supporting continuous training with streaming arriving data.

5 STRUCTURED SAMPLES

As mentioned above, the conventional machine learning problem measures each task as the basic entity, where data units are assembled and consumed individually. Beyond this inter-task redundancy, there also exists inner-task redundancy for industrial data. Take the click-through rate prediction as the example, the conventional way of constructing samples is collecting data units for each sample individually. To predict a user's preference toward an item, the users' features are gathered firstly and then concatenated with features of the item to be predicted to construct samples. Here, we refer to samples constructed in this manner as *plain samples*. In the format of plain samples, if we want to predict the user's preferences towards N items, the user information is repeated by N times. In other words, for a specific user, the redundancy grows linearly with respect to the number of items to be predicted. The reason for this redundancy is that plain samples neglect the structured characteristic existed in data, resulting in a waste of storage cost and high pressure on bandwidth.

To make full use of the reusable structure in industrial data, Bernoulli organizes data units in the form of structured samples as input instead of plain samples. Compared with plain samples, structured samples reuse common information and compressing data in a structured way. Figure 5 gives an illustration of how structured samples are constructed. Given a batch of plain samples, Bernoulli first extracts common data units for each column. The reusable data units are then taken as the dictionary. With the help of the dictionary, Bernoulli only needs to store the indicators of each sample but not the data units. The design of structured samples reduces duplicated features computation and drastically reduces the storage space and the communication cost.

At Alibaba, there are billions of features and hundreds of billions of samples. A large number of plain samples place great press on the system. As a solution, the design choice of structured samples reuses common information and compresses data to a large extent, largely relieving the burden for the system. In practice, we find the use of structured samples can compress plain samples significantly, saving tremendous storage costs, reducing communication costs while improving computational efficiency.

6 STRUCTURED TRAINING

Industrial data are streaming and the distribution is shifting dynamically. Therefore online serving models need to update their parameters continuously with real-time data to keep fresh. In this section, we introduce the structured training design of Bernoulli. The main advantages of structured training are the ability to continuously refresh the model and the support of fast algorithm iteration, including data-level iteration and model-level iteration. Besides, the structured training paradigm reduces resource consumption significantly.

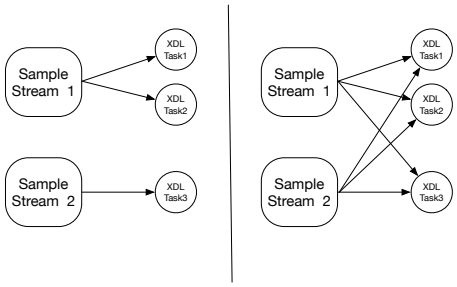

Figure 6: Schematic data-level iteration of Bernoulli.

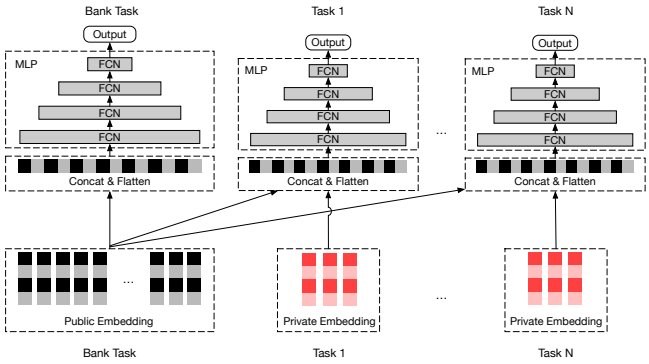

Figure 7: The structured embedding design of model-bank. Different tasks share the same public embeddings while keeping a small portion of private embeddings.

6.1 Freshness of Model

To capture the dynamic change of data in real-time, it is important to use the real-time samples to update the model. As mentioned above, the structured data pipeline utilizes the structure of industrial data and views the basic data unit as the basic entity. Real-time features, e.g., the latest clicked behaviors of users, are also abstracted as the input dataflow. The sample factory takes the responsibility to assemble real-time features and other features to obtain real-time samples. The real-time samples are then used to refresh the model.

6.2 Efficient Algorithm Iteration

Bernoulli supports fast data-level iteration, including feature iteration and sample iteration. For each task, new features can be easily combined with the sample skeleton for feature iteration. Besides, different tasks can assemble the sample skeleton and relevant dataflows to finish sample iteration. This is particularly useful for data fusion from different business scenarios, as shown in Figure 6, the following task is trained on the XDL [12] platform that is an industrial deep learning framework for high-dimensional sparse data. Thus, we can utilize the data from data-rich business scenarios to facilitate the learning of data-poor business scenarios [28].

As mentioned above, the sample pool of sample factory caches data for each task. The design of the sample factory enables efficient algorithm iteration on online streaming data, however, training a substantially deep model from scratch is still time-consuming. This issue is even more serious when multiple algorithm iterations are progressing simultaneously, which is common in industrial business services. A simple solution is to early stop the training stage, but the performance of early stopping is not able to catch the performance of training with more mini-batches. To accelerate the model-level iteration, Bernoulli implements the structured training by the design of *model-bank* framework, as will be described in the following.

6.3 Model-Bank: Structured Embedding

Industrial online businesses often have to deal with high-dimensional and sparse data. Most industrial machine learning models with high-dimensional sparse input follows the Embedding & Multi-Layer Perceptron (MLP) paradigm [5, 12, 22, 32]. In particular, the embedding module first transforms each discrete feature into a low dimensional vector, i.e., embedding. The MLP module then aggregates embeddings by various means, e.g., sum pooling, to obtain a fixed-length vector. The fixed-length vector is fed as input into the following fully-connected layers for final prediction.

Model-bank is motivated by the following observations: (1). The vast majority of storage costs are consumed by the embedding modules while the MLP modules only account for a small portion. Take the online model at Alibaba as an example, the number of parameters of embeddings exceeds 10 billion while the number of parameters of MLP module is less than 10 million. (2). The embeddings have some degree of generality across different tasks while the MLP module is more task-specific. Motivated by these observations, model-bank aims to utilize *structured embeddings* to improve training efficiency. To be more concrete, structured embeddings means decoupling the embedding module from the MLP module and let different models share the same embeddings, which we will refer to as *public embeddings*. The intuition behind the structured embedding is that the embedding modules contain lots of semantic information about the categorical feature, thus less relevant to a specific task. Besides, we also observe that some features are less general and contain task-specific information. To reduce the performance gap, each task will also keep a small portion of *private embeddings*. If a feature has private embedding, the final embedding will be the private embedding, otherwise, the public embedding will be used. Since most of the model parameters are occupied by embeddings, the efficient reuse of public embeddings reduces resource consumption largely.

With the help of structured embeddings, when experimenting with new model designs, one can easily use the public embedding as initialization to warm up the training stage. The other parts of the model are randomly initialized. In contrast to training the whole model from scratch, the structured training strategy accelerates model training significantly. Note that the model is trained on XDL [12], but the structured system design of Bernoulli is not limited to this specific framework. Besides, the structured design of the model bank could also relieve the cold-start dilemma. Given a new task, the online model gets poor performance until collecting enough training data for training. As a remedy, the structured embeddings can warm up the cold-start learning stages and enable fast learning.

7 STRUCTURED SERVING

At Alibaba, there are hundreds of services with thousands of models running simultaneously on the online system. The large memory footprint and complicated computational logic bring serious challenges to real-time prediction. The model-bank framework of Bernoulli, inheriting the structured design principle, optimizes resource consumption while enables better model freshness.

7.1 Lower Resource Consumption

With the development of modern deep learning techniques, the storage space of a model easily reaches TBs or even higher in industrial systems. The storage pressure is even more serious considering that each business scenario also maintains multiple versions of the serving models and multiple experimenting models. Without the structured reuse across different models, the resource consumption could easily exceed the resource limit, thus restricting the quality of online serving and the velocity of algorithm iteration. Facilitated with the structured embedding, now different production and iteration experiments can efficiently reuse public embeddings to relieve the storage pressure and accelerate the model training. Since most of the storage costs are caused by embeddings, the structured embedding design can reduce the resource consumption significantly, which strengthens the stability and scalability of online serving.

7.2 Freshness via Asynchronous Update

At Alibaba, online serving is especially time-sensitive due to the dynamic change of users' behavior and interest [21, 31, 32]. This requires the model to update with real-time data frequently, however, update the whole model is time-consuming and also brings serious computational pressure. To solve this issue, Bernoulli updates the embeddings module and MLP module asynchronously. The asynchronous update is motivated by the observation that embeddings contain some degree of semantic information about the features and are less time-sensitive. On the other hand, MLP modules are more task-specific and require frequent updates. Motivated by these facts, we let the embedding module and MLP module update asynchronously in contrast to the common end-to-end learning paradigm. Embeddings are updated less frequently than the MLP module. Since the amount of parameters of MLP is much smaller than embeddings, only refreshing the light-weight MLP module reduces the computational cost and enable frequent model update, giving better freshness. In other words, in addition to efficiency, the framework of model-bank also enables the online model for the update in real-time.

To be more concrete, Bernoulli updates the public embedding offline every few days, while update the MLP module and private embeddings in real-time. This trade-off between model freshness and resource consumption allows our serving system to maximize its serving quality. Overall, the structured serving relieves resource consumption burden while obtaining high-quality fresh models for online serving.

8 EVALUATION AT ALIBABA

Up to now, hundreds of commercial services at Alibaba have used Bernoulli for production and algorithm iteration. In this section, we first report the performance improvement brought by Bernoulli. Then we will present a case study in detail.

Table 1: Improvement of iteration efficiency brought by sample factory.

Method	Time of iteration
Offline training followed by online fine-tuning	Days
Direct iteration on online streaming data	Hours

Table 2: Compression ratio of structured samples compared with plain samples.

	Streaming data	Daily data
Compression ratio	$\frac{1}{4}$	$\frac{1}{12}$

8.1 Results from Production Deployment

The structured pipeline abstracts basic data units as the dataflows, enabling direct algorithm iteration on online streaming data. By this means, the algorithm iteration has been accelerated significantly. Since structured dataflow of reuses common and structured information, we only need little development cost to combine multiple dataflows to obtain the specific dataflow for each task. The effort of building offline data pipelines can be saved. As shown in Table 1, the previous paradigm that is offline training followed by online fine-tuning needs a few days to finish algorithm iteration. In contrast, direct algorithm iteration on online streaming data only needs a few hours. This structured data pipeline reduces the development and time cost, which is beneficial to algorithm iteration. This paradigm is especially suitable for starting new business services. With the help of Bernoulli, we start our short video business from scratch quickly. The result is promising, the view count is improved by 5.8% compared with the previous paradigm thanks to the fast iteration.

Compared with plain samples, the structured samples can compress data to a large extent. This structured design saves storage and computation costs significantly, as it reduces duplicated embedding vector computation and storage. In *Guess What You Like*, the scenario in the Taobao App homepage, the structured samples compress streaming data to $\frac{1}{4}$ on our production system. The result is even more encouraging on daily data (data that are produced every day but not in real-time). In contrast to streaming data that are sorted in chronological order, daily data can be shuffled for a larger compression ratio. In Bernoulli, the daily data can be compressed to $\frac{1}{12}$ thanks to the structured samples.

Compared with keeping separate and private models for each task and experiment, the model-bank framework reuses the structured embedding thus reduces resource consumption. The comparison of model-bank and keeping separate models is shown in Table 3. Suppose there are N serving and experimenting models running concurrently on the system, reusing embeddings could save storage cost to approximately $\frac{1}{N}$. This is due to the fact that public embeddings often reach TBs while other parts of the model are less than 1 GB. On the other hand, only update the light-weight

Table 3: The resource consumption ratio of model-bank compared with the separate models (suppose there are N serving and experimenting models running concurrently on the system).

	Storage cost	Communication cost
Model-bank	$\approx \frac{1}{N}$	$\approx \frac{1}{N}$

Table 4: The frequency of fresh models being deployed into production.

Method	Frequency
End-to-end Update	3 hours
Asynchronous Update	10 minutes

MLP and private embeddings also reduces the communication pressure. The communication cost is also saved to approximately $\frac{1}{N}$. On our online business, N often exceeds a thousand, thus the reduced resource consumption is very impressive.

In the online serving environment, the freshness of the model is important. Updating all models end-to-end every a few minutes is infeasible considering computational capability. As the trade-off, the asynchronous update of the model bank enables continuously push fresh models into the production environment. As shown in Table 4, Bernoulli can produce fresh models at minute-level compared with the hour-level freshness previous system. In practice, we find the model freshness improves the system performance significantly.

Overall, our online businesses have been benefited a lot from the structured design of Bernoulli. Up to now, hundreds of business scenarios have been deployed on Bernoulli, facilitating real-time updates and improving user satisfaction and income.

8.2 Case Study: Data Fusion from Multiple Business Scenarios

One advantage of the structured data pipeline is the ease of data fusion from different business scenarios. At Alibaba, there are a large number of business scenarios, ranging from *Guess What You Like* in Taobao App homepage, *Banner* of Taobao App homepage, to other scenarios. The amount of data from different business scenarios differ by orders of magnitude. How to obtain effective models for data-poor scenarios places a serious challenge.

In practice, we find it is of importance to utilize the data from data-rich business scenarios to facilitate the learning for data-poor business scenarios. The reason is that different business scenarios have overlapping user groups and items, thus exist commonalities. Enabling information transferring from data-rich scenarios to data-poor scenarios is beneficial for model learning. However, conventional machine learning systems are hard to combine dataflows from different scenarios. By contrast, Bernoulli utilizes the structure in data and abstract dataflow at the basic data unit level, which allows engineering teams to add or delete data from a particular scenario easily. Multiple dataflows can be assembled from different business scenarios flexibly.

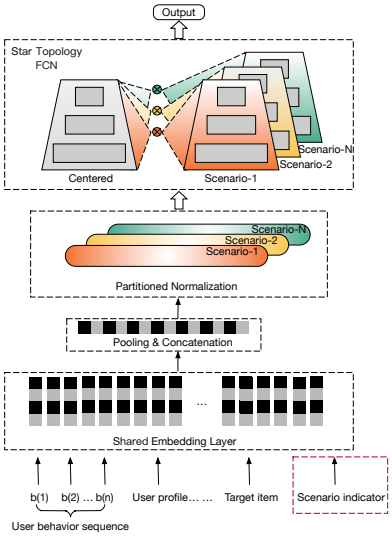

Figure 8: Data fusion assists the learning for data-poor scenarios. . The structured data pipeline eases the difficulty of data fusion from multiple business scenarios.

In production, data fusion from multiple business scenarios alleviates the difficulty of learning for scenarios with little data. In detail, we model the CTR prediction for multiple business scenarios simultaneously and propose Star Topology Adaptive Recommender (STAR) for multi-scenario CTR prediction [28]. In STAR, one model is learned to serve all data-poor scenarios. The model architecture is shown in Figure 8. STAR has the star topology, which consists of the shared centered parameters and scenario-specific parameters. The shared parameters are used to learn commonalities of all scenarios and the scenario-specific parameters capture the distinction for more refined prediction. Overall, Bernoulli helps us fuse the training data conveniently and the efficient reuse of model bank helps save resource consumption largely, enabling stable online serving. The result is promising, our online testing shows that the CTR is improved by 8.0% and RPM (Revenue Per Mille) is improved by 6.0% for all business scenarios.

9 CONCLUSION

We have discussed the desirable features for industrial machine learning systems and presented our structured system design. Bernoulli builds on online streaming data, where machine learning models are directly experimented with and deployed on the streaming data. The discussed desirable features are incorporated into the system. The core principle of the system design is building Bernoulli in a structured manner throughout the entire workflow including structured data pipeline, structured samples, structured model training, and structured online serving. We have demonstrated how the structured design of Bernoulli improves model freshness, facilitates algorithm iteration on online streaming data, reduces system redundancy, and improves online services. Our current experience with Bernoulli is encouraging, hundreds of online commercial services at Alibaba have been deployed on Bernoulli. More importantly,

Bernoulli is continuously helping our engineering teams for experimentation with new approaches, driven advances in our various online services.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*. Savannah, GA, 265–283.

[2] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax, NS, Canada, 1387–1395.

[3] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning. In *Proceedings of Machine Learning and Systems 2019*. Stanford, CA.

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015). arXiv:1512.01274 http://arxiv.org/abs/1512.01274

[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*. Boston, MA, 191–198.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Minneapolis, MN, USA, 4171–4186.

[7] Ellen Friedman and Kostas Tzoumas. 2016. *Introduction to Apache Flink: stream processing for real time and beyond.* " O'Reilly Media, Inc.".

[8] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-Scale Bayesian Click-Through rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th International Conference on Machine Learning*. Haifa, Israel, 13–20.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, 770–778.

[10] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Melbourne, Australia, 328–339.

[11] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37. Lille, France, 448–456.

[12] Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, et al. 2019. XDL: An Industrial Deep Learning Framework for High-Dimensional Sparse Data. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 1–9.

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations*. San Diego, CA, USA.

[14] Tim Kraska, Ameet Talwalkar, John C. Duchi, Rean Griffith, Michael J. Franklin, and Michael I. Jordan. 2013. MLbase: A Distributed Machine-learning System. In *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research*. Asilomar, CA, USA.

[15] Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter, and Tawfiq Hasanin. 2015. A Survey of Open Source Tools for Machine Learning with Big Data in the Hadoop Ecosystem. *Journel of Big Data* 2 (2015), 24.

[16] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*. Broomfield, CO, USA, 583–598.

[17] Jimmy J. Lin and Alek Kolcz. 2012. Large-scale Machine Learning at Twitter. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, Scottsdale, AZ, USA, 793–804.

[18] Xiangrui Meng, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research* 17 (2016), 34:1–34:7.

[19] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Vancouver, BC, Canada, 8024–8035.

[21] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on Long Sequential User Behavior Modeling for Click-Through Rate Prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage, AK, USA, 2671–2679.

[22] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *Proceedings of the16th International Conference on Data Mining*. 1149–1154.

[23] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf* (2018).

[24] Steffen Rendle. 2010. Factorization machines. In *Proceedings of the 10th International Conference on Data Mining*. 995–1000.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[26] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems 28*. Montreal, Quebec, Canada, 2503–2511.

[27] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA, USA, 2135.

[28] Xiang-Rong Sheng, Liqin Zhao, Guorui Zhou, Xinyao Ding, Binding Dai, Qiang Luo, Siran Yang, Jingshan Lv, Chi Zhang, and Xiaoqiang Zhu. 2021. One Model to Serve All: Star Topology Adaptive Recommender for Multi-Domain CTR Prediction. *CoRR* abs/2101.11427 (2021). arXiv:2101.11427 https://arxiv.org/abs/2101.11427

[29] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John P. Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. 2017. StarCraft II: A New Challenge for Reinforcement Learning. *CoRR* abs/1708.04782 (2017). arXiv:1708.04782 http://arxiv.org/abs/1708.04782

[30] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. London, UK, 839–848.

[31] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep Interest Evolution Network for Click-Through Rate Prediction. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*. Honolulu, Hawaii, USA, 5941–5948.

[32] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1059–1068.