JiZhi: A Fast and Cost-Effective Model-As-A-Service System for Web-Scale Online Inference at Baidu

Hao Liu[†], Qian Gao[†], Jiang Li, Xiaochao Liao, Hao Xiong, Guangxing Chen, Wenlin Wang,
Guobao Yang, Zhiwei Zha, Daxiang Dong, Dejing Dou, Haoyi Xiong[*]
Baidu Inc., Beijing, China
{liuhao30, gaoqian05, lijiang01, liaoxiaochao, xionghao02, chenguangxing, wangwenlin,
yangguobao, zhazhiwei, dongdaxiang, doudejing, xionghaoyi}@baidu.com

ABSTRACT

In modern internet industries, deep learning based recommender systems have became an indispensable building block for a wide spectrum of applications, such as search engine, news feed, and short video clips. However, it remains challenging to carry the well-trained deep models for online real-time inference serving, with respect to the time-varying web-scale traffics from billions of users, in a cost-effective manner. In this work, we present JiZhi— a Model-as-a-Service system — that per second handles hundreds of millions of online inference requests to huge deep models with more than trillions of sparse parameters, for over twenty real-time recommendation services at Baidu, Inc.

In JiZhi, the inference workflow of every recommendation request is transformed to a Staged Event-Driven Pipeline (SEDP), where each node in the pipeline refers to a staged computation or I/O intensive task processor. With traffics of real-time inference requests arrived, each modularized processor can be run in a fully asynchronized way and managed separately. Besides, JiZhi introduces the heterogeneous and hierarchical storage to further accelerate the online inference process by reducing unnecessary computations and potential data access latency induced by ultra-sparse model parameters. Moreover, an intelligent resource manager has been deployed to maximize the throughput of JiZhi over the shared infrastructure by searching the optimal resource allocation plan from historical logs and fine-tuning the load shedding policies over intermediate system feedback. Extensive experiments have been done to demonstrate the advantages of JiZhi from the perspectives of end-to-end service latency, system-wide throughput, and resource consumption. Since launched in July 2019, JiZhi has helped Baidu saved more than ten million US dollars in hardware and utility costs per year while handling 200% more traffics without sacrificing the inference efficiency.

[†] Equal contribution.
[*] Corresponding author.
“JiZhi” means making smart decisions extremely fast in Chinese. An open-sourced version of JiZhi is available at https://github.com/PaddlePaddle/Serving .

KDD ’21, August 14–18, 2021, Virtual Event, Singapore
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467146

CCS CONCEPTS

• **Information systems → Content ranking**; • **Computer systems organization → Cloud computing**.

KEYWORDS

Online inference; recommendation system; intelligent resource management; MLOps

ACM Reference Format:
Hao Liu, Qian Gao, Jiang Li, Xiaochao Liao, Hao Xiong, Guangxing Chen, Wenlin Wang, Guobao Yang, Zhiwei Zha, Daxiang Dong, Dejing Dou, Haoyi Xiong. 2021. JiZhi: A Fast and Cost-Effective Model-As-A-Service System for Web-Scale Online Inference at Baidu. In *Proceedings of the 27th International ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD’21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467146

1 INTRODUCTION

Deep neural networks (DNNs) [19] have recently been used as the standard workhorses for predicting users’ online behaviors, such as click-through rate [12, 36], browsing interests [33], and buying intentions [13], due to its superiority of representation modeling and reasoning for a wide spectrum of data modalities [24]. As early as in the early 2010s, leading internet companies such as Baidu and Google have successfully integrated DNNs into their core products, ranging from sponsored search [9, 26] to voice assistant [23]. Figure 1 depicts several real-world products at Baidu that leveraging DNNs for online recommendations.

While DNNs have demonstrated its effectiveness in various internet application domains, the cost of using DNNs for web-scale real-time online inference becomes the major burden for most companies to adopt the techniques [11, 17] On the one hand, the time consumption (*e.g.*, latency) of the online service is critical for user experience [5] and can influence the long term retention rate [4]. On the other hand, the resource consumption (*e.g.*, hardware and energy usages) of supporting DNNs would request significant serving infrastructure investment (*e.g.*, high-performance clusters) with higher power consumption and sometimes makes the systems design, implementation and operation over-budget [29].

To guarantee the utility of online DNN serving, some efforts have been done from algorithmic and system perspectives. From the algorithmic perspective, model compression [8]can significantly reduce the computational overhead by trade-off the model size and recommendation effectiveness. From the system perspective, dedicated acceleration hardware such as Graphics Processing Unit

曾经遍布大街小巷，卫星锅为何突然消失？国家都是为了我们好

中华青年正能量 13评论

无人驾驶离我们有多远

根据百度提供的数据，仅仅在10月12日一天，单站点自动驾驶出租车的约车累计就达到2608单，最高峰值达到600单。

等6436人参与讨论

把粉笔、减压球、气球放在车轮碾压，勿模仿

2.5万粉丝 | 菏泽卓韩文化...

#31省区市新增确诊11例#
2月6日0—24时，31个省（自治区、直辖市）...
热度 190W

#贴吧整活课堂开课啦#
年关将至，吧友不离不弃，整活永远相随。第...
热度 172.9W

#台湾地震#
中国地震台网正式测定：02月07日01时36分...
热度 158.5W

#贴吧与你一起过年#
今年春节，你是回乡过年还是就地过年？无论...
热度 115.6W

#尤文战胜罗马#
0-21赛季意甲第21轮，尤文图斯主场迎战罗马...
热度 109.7W

#奥运会拳击冠军斯宾克斯去世#
据《华盛顿邮报》等多家外媒报道，奥运会拳...
热度 107.6W

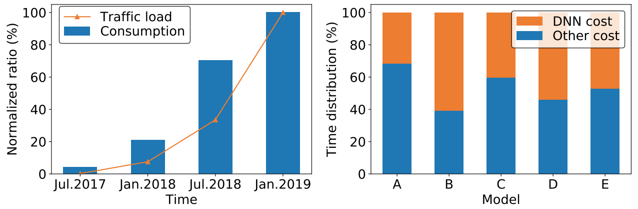
(a) Baidu news feed. (b) Haokan video. (c) Baidu tieba.

Figure 1: Screenshots of several DNN empowered products in Baidu based on JiZhi.

(GPU) [28] and software extensions such as TensorRT [32] have been devised to reduce the DNN computation latency. However, serving DNNs for online recommendations is more than executing the feed-forward neural network. The aforementioned techniques are mainly optimized for single DNN training and inference, but overlook the complicated data and computation dependency of online inference under time-varying internet traffics.

Indeed, serving real-time DNN inference requests for web-scale recommendations is non-trivial for any major internet player with hundreds of millions of Daily Active Users (DAUs), because of the following three major technical challenges. (1) *Huge and sparse DNN models.* To improve the recommendation performance, the industrial DNNs are usually trained with massive data samples, where each data sample is typically extremely high-dimensional and sparse (*i.e.*, hundreds billions of features with only hundreds of non-zero values). For example, the CTR model in Baidu sponsored search and Ads retrieval are over 10TB large [35] and are still quickly growing. As a result, these huge DNN models with trillions of sparse parameters require an efficient way to store, retrieve, transfer, and compute. (2) *Time-varying web-scale traffics.* The serving system of such deep learning based recommenders is responding to hundreds of millions concurrent inference requests per second. More critically, the online inference service may experience 100-fold traffic increase in peak hour and fast traffic load growth, as depicted in Figure 2(a). Since it is unrealistic to increase the budget for every single recommendation service unlimitedly, it is challenging to build an efficient and cost-effective online inference system for such explosive and temporally imbalanced traffic loads. (3) *Diverse recommendation scenarios.* Figure 2(b) shows the diversified data-processing and inference time distribution of five real-world deep learning based recommendation services at Baidu. Depending on different application scenarios, the recommendation strategy and inference pipeline may vary. For instance, news recommendation usually handles both text and image related features and may involve multiple DNNs (up to 20 DNNs in single recommendation service in Baidu), while search engine pays more attention to user preference modeling [33]. It is challenging to build a unified online serving architecture that can support various recommendation scenarios with different online inference workflow.

In this work, we present JiZhi, the Model-as-a-Service system for web-scale online inference at Baidu. Specifically, JiZhi first transforms the workflow of deep learning based recommendation inference into the *Staged Event-Driven Pipeline* (SEDP), where each node refers to a staged computation task and the edge between two nodes refers to a possible transition from one task to another. By modularizing the complex and application specific inference pipeline into individual staged processors, SEDP supports flexible customization, performance tuning, and can be well adapted for diverse application scenarios. Moreover, a tailor-designed *Heterogeneous and Hierarchical Storage* (HHS) module is introduced for the huge and ultra-sparse DNN model management. By eliminating redundant computations and parameter access costs, HHS significantly improves the online inference speed and throughput in a cost-effective way. Besides, to improve the computational resource utilization, we propose the *Intelligent Resource Manager* (IRM) module for automatic system performance tuning. By formulating autotuning tasks as constrained optimization problems, IRM searches the optimal resource allocation plan for each stage processor in the offline and adaptively controls the online load shedding policy based on real-time system feedback, without sacrificing the recommendation effectiveness.



(a) Traffic load and budget explosion. (b) Online inference time breakdown.

Figure 2: Characteristics of serving online recommendations at Baidu.

To the best of our knowledge, this work is the first to study the system design, implementation, and integration for web-scale DNN inference from an industrial practitioner’s perspective, by addressing the goals of minimizing the overall resource consumption under latency constraints. Our major contributions are summarized as follows. (1) We propose an asynchronous design of online inference for deep learning based recommendation, which exposes the opportunity of fine-grained resource management and flexible workflow customization. (2) We introduce a cost-effective data placement strategy for huge and ultra-sparse DNN models, which significantly improves the online inference system throughput with slight resource overhead. (3) We propose a novel intelligent resource management module to optimize the system resource utilization in both online and offline fashion. (4) We conduct extensive experiments to demonstrate the advantages of JiZhi by using real-world web-scale traffics from various products in Baidu. Moreover, we share our hands on experiences of implementing and deploying the proposed system, which we envision would be helpful to the community. We have released an open-source version of JiZhi on PaddlePaddle (https://github.com/PaddlePaddle/Serving).

2 BACKGROUND AND RELATED WORK

This section overviews the common routine of industrial deep learning based online recommendation and online inference speedup techniques, including the legacy online serving design at Baidu.

Deep learning based online recommendation. Given a set of items (*e.g.*, contents and goods), the recommendation system aims to return personalized probability score for each item to achieve one or more targets, such as maximizing the CTR [36], improving the dwell time [18], and increase the overall profitability of the product [33]. In the past years, DNN has been widely adopted to serve as the core model in online recommendation [16, 21, 22]. Based on the application scenario, different DNN architectures have been adopted for recommendation. For instance, neural collaborative filtering for user-item interaction modeling [14], recurrent neural network for sequential user behavior modeling [15, 30], and deep reinforcement learning for long-term recommendation optimization [37]. In Baidu, DNN based recommendation service has been successfully integrated into over twenty online internet products (*e.g.*, news feed, short video clips, search engine, and online advertising), serving over one hundred billion online recommendation requests made by hundreds of millions users per day.

Serving deep learning based online inference. To guarantee the user experience, the online inference process is usually constrained to be finished in an ultra low latency, *i.e.*, dozens of milliseconds. However, the majority of research efforts on DNN speedup are focused on model training [7, 34], while limited efforts have been made for online inference. On the one hand, various model compression techniques [8] have been proposed to reduce the model size by sacrificing the recommendation effectiveness. On the other hand, dedicated processors (*e.g.*, GPU [28], TPU [17], Kunlun [27]) and software extensions (*e.g.*, AVX[10], SSE [1], TensorRT [32], and TensorFlow-Serving [25]) have been developed to accelerate online inference computation. However, the above approaches are mainly optimized for general DNN acceleration, but overlook the unique challenge of diversified data dependency and model sparsity in the online recommendation.

The previous system design for online inference at Baidu. Since 2013, Baidu has proposed dedicated hardware and software platforms to accelerate online data processing and DNN inference [27, 35]. When real-time recommendation requests arrive, the system first retrieves relevant candidate items (usually millions) and then apply multi-phase inference for user-item pairs [9] to reduce the candidate size from millions to thousands and finally return a dozen results to the user. In each phase, by following the data parallel paradigm, the inference tasks (user-item pairs) are packaged into multiple batches, and different batches are then processed in parallel. For each inference task, the online inference system requires to fetch corresponding raw data from remotely distributed storage, transform input to proper features, and execute the feed-forward inference computation to output the relevance score. For different recommendation applications, the online inference service needs to be developed from scratch and tuning based on expert experience. More critically, the system suffers from pipeline stall because of the computation and access latency of long-tail candidates, and quickly meets the resource bottleneck under the explosive recommendation traffic demand.

3 SYSTEM OVERVIEW

Figure 3 shows the architecture of JiZhi, which consists of three major components, the *Staged event-driven pipeline* (SEDP), the *Heterogeneous and hierarchical storage*, and the *Intelligent resource manager*. Given an online inference task, the *Staged event-driven pipeline* first compiles the inference workflow into a directed acyclic graph and then execute each independent node in the graph in a full asynchronous way. By modularizing the inference task into a set of sequence or parallel stages, SEDP can significantly improve the resource utilization of the infrastructure, enable workflow customization, and expose opportunities for more intelligent dynamic resource allocation. The *Heterogeneous and Hierarchical Storage* (HHS) is a key-value based data placement system including a distributed sparse parameter cube and a heterogeneous and hierarchical cache. By eliminating unnecessary computation and various I/O costs, the storage system further reduces the online inference latency and improves the system throughput in a cost-effective way. Based on SEDP, the *Intelligent Resource Manager* (IRM) achieves quasi-optimal resource utilization by monitoring and automatically tuning the computation resource allocation of the serving pipeline in both offline and online. In particular, the offline resource management component searches the optimal resource allocation plan for each stage in the SEDP and HHS to maximize the system throughput. The online resource management module adaptively fine-tunes the load shedding policies based on the intermediate system feedback to guarantee the serving latency when the traffic is overloaded, without sacrificing the recommendation effectiveness.

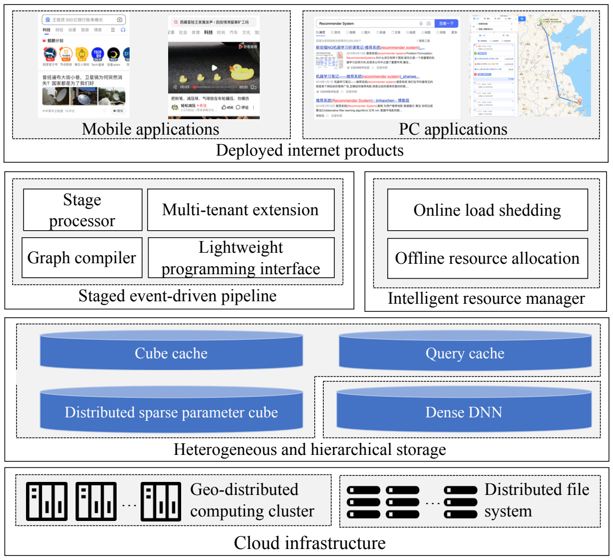

Mobile applications
PC applications
Deployed internet products
Stage processor
Multi-tenant extension
Graph compiler
Lightweight programming interface
Staged event-driven pipeline
Online load shedding
Offline resource allocation
Intelligent resource manager
Cube cache
Query cache
Distributed sparse parameter cube
Dense DNN
Heterogeneous and hierarchical storage
Geo-distributed computing cluster
Distributed file system
Cloud infrastructure

Figure 3: Layered architecture of JiZhi.

4 STAGED EVENT-DRIVEN PIPELINE

The primary design objective of JiZhi is to provide customizable and tunable online inference service for diverse online recommendation scenarios, which is achieved by the asynchronous *Staged Event-Driven Pipeline*. We begin with the construction of stage processor.

Definition 1. ***Stage processor***. *Given an online inference workflow, a stage processor p is defined as a tuple $p = \langle op, c \rangle$, where op is the operator representing a unit primitive that encapsulate the functionality for a particular execution stage, and c is the channel that queuing events from upstream processors via a shared data structure.*

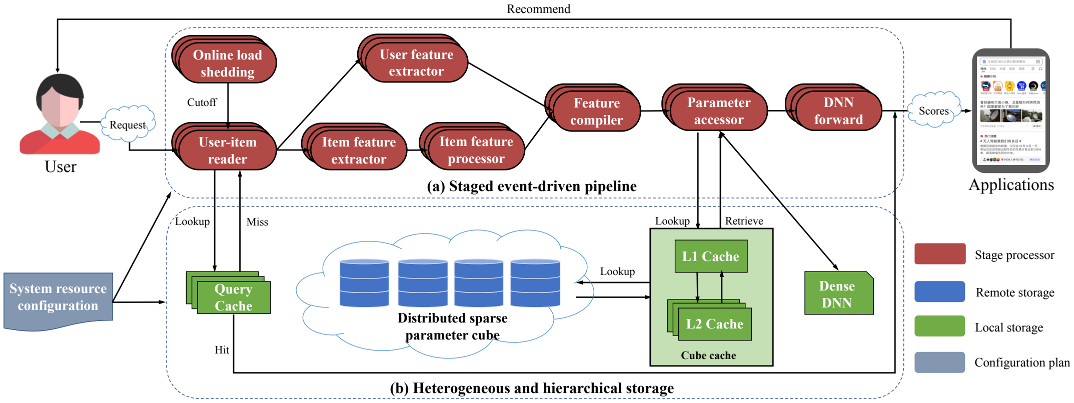


Figure 4: An illustrative online user-item inference workflow based on JiZhi.

Note an execution stage can be any steps in the online inference workflow, such as feature extraction, model access, and DNN feed-forward propagation. In this way, an online inference workflow is modularized into a set of stage processors $P = \{p_1, p_2, \ldots\}$. A stage processor can be understood as a micro-service or a thread, which is a reusable and self-contained component that may appear one or multiple times in a workflow depend on the online inference logic. In each stage processor, the operator processes incoming events concurrently, and passes the finished events to the next channel queue attached to subsequent stage processors.

Definition 2. ***Staged Event-Driven Pipeline (SEDP)****. A staged event-driven pipeline is defined as a directed acyclic graph $G = (P, E)$, where $p_i \in P$ is a stage operator, and $e_{ij} \in E$ is an edge connecting two stage processors p_i and p_j. Note that in a SEDP, all edges point to processors of a same stage share a same channel, to provide complex event processing capability, such as aggregation, join, and ordering.*

For a predefined SEDP, JiZhi can automatically analyze the dependency between each stage processor, construct shared channels, and compile the fully asynchronous execution plan. In particular, each arrived online inference request (*i.e.*, user-item pair) is regarded as an event and buffered in a queue of the next stage processor. Based on the graph dependency, each stage processor in a SEDP can be executed in sequence or parallel.

Figure 4 (a) depicts an example online workflow of user-item inference task by using SEDP. The user and item information is first collected and dispatched to a user-specific processor and item-specific processors. After internal processing stages such as feature filtering, ranking, and wrangling, all raw features of the request are recombined to retrieve the corresponding feature parameters (*i.e.*, pre-computed embeddings and sparse vectors). Finally, the retrieved sparse parameters are fed to the DNN network to derive the item relevance score for personalized recommendation.

By asynchronously executing stage processors in an event-driven way, SEDP successfully improves the overall throughput of the online inference system by reducing potential pipeline stalls of long-tail items. Note that the capacity (*e.g.*, parallelism, memory) of each stage processor can be tuned separately to achieve a higher resource utilization rate of the shared infrastructure. Moreover, the modularized design exposes the customizable online inference workflow interface to engineers and data scientists to support fast service workflow and model iteration.

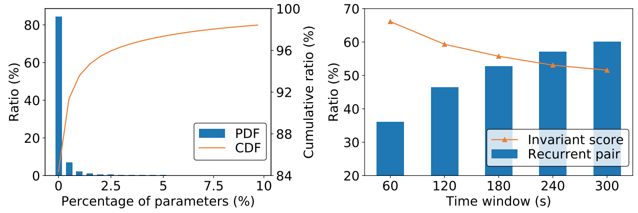
Multi-tenant extension. SEDP also offers the flexibility to simultaneously access multiple DNNs in a single pipeline, *e.g.*, multi-phase inference where each phase corresponds to a DNN, and multi-objective inference accesses multiple DNNs in parallel. The multi-tenant extension not only enables more complex recommendation strategies in a unified service, but also benefits the recommendation system iteration via the natural support of online A/B testing. Specifically, the access of multiple DNNs usually involves tremendous repetitive data processing and sparse parameter accessing operations, which can be significantly reduced in a single service. Moreover, in industrial product iteration, data scientists and researchers often conduct tremendous A/B testing daily (usually tens of test groups in concurrent). A common approach is to conduct A/B testing between the current recommendation system and a new test group to determine the satisfaction of the new model/strategy. In the previous online inference system, the system administrator needs to deploy new online services for each new test group, which requires manually splitting the traffic load and assigning computational resources for different variants. However, such an approach is cumbersome and error-prone. In JiZhi, the A/B testing is well-supported by hosting multiple different test groups in a multi-tenant way. By devising a stage processor to dispatch traffic loads to different test groups, each variant is an independent branch in the SEDP and shares the same computing infrastructure.

5 HETEROGENEOUS AND HIERARCHICAL STORAGE

We further present the cost-effective data placement design, including the distributed sparse parameter cube and the heterogeneous and hierarchical cache. Both components are tailor-designed for deep learning based recommendation in a data-driven approach and tunable to handle different recommendation scenarios.

5.1 The Distributed Sparse Parameter Cube

In online recommendation, the DNN usually consists of a huge and sparse sub-network (up to Terabytes large) and a moderate-sized dense network (usually hundreds of Megabytes) [35]. We first introduce the distributed sparse parameter cube (*a.k.a.* the



(a) Access distribution of sparse parameters.

(b) Recurrence distribution and variance of user-item pairs.

Figure 5: Data access patterns in online inference service.

cube) to manage the extremely high-dimensional parameters of the sparse sub-network. Specifically, the cube is a read-only distributed key-value store, where the key is defined as a compact feature signature (a universally unique identifier of a feature), and the value is defined as model weights and statistics of user feedback of the corresponding sparse feature. To guarantee the distributed consistency, the feature signature is derived via universal hash function [6]. The online inference tasks can look up the extremely sparse non-zero values in constant time complexity by accessing the cube. To hide the latency induced by random access of the hash table, all keys are stored purely in memory, and values are grouped into one Gigabyte-sized block that can be either placed in memory or disk (*i.e.*, SSD). The placement of keys and values is a trade-off between the online inference latency and the hardware consumption. In this way, the cube operates on heterogeneous hardware can support ten-thousand level concurrent parameter lookup with less than ten milliseconds latency within a data center. The cube is also fault-tolerant by replicating data blocks in the distributed cluster.

5.2 Heterogeneous and Hierarchical Cache

Although the distributed sparse parameter cube successfully reduces the feature access latency to several milliseconds, the online inference process is still communication and computational extensive. We further introduce the heterogeneous and hierarchical cache to reduce both the I/O and computation costs, including the cube cache for the distributed sparse parameter cube access and the query cache for the inference pipeline computation. Figure 4 (b) shows two types of caches and their interaction with the SEDP.

Cube cache. The design of the cube cache is motivated by the observation that the feature access to the cube is heavy-tailed, where only a small portion of hot features are frequently accessed and updated. Figure 5(a) plots the distribution of 100 million consecutive cube accesses of a deployed online recommendation service in Baidu, where over 80% lookup are focused on 1% values in the cube. Therefore, it is reasonable to cache such hot features locally to reduce expensive network I/O costs. Specifically, the cube cache is a two-layer local storage that persistent a small portion of frequently accessed key-value pairs in the cube. We adopt the Least Frequently Used (LFU) [20] policy to replace key-value pairs in the cache. There are two levels of the cube cache, *i.e.*, the memory-level and the disk-level. The disk-level cache is used to hide the expensive network I/O cost of accessing the remote cube, which stores about 1% frequently accessed key-value pairs to local SSDs. On top of the disk cache, a memory cache is further introduced to avoid disk I/O for the top 0.1% hottest key-value pairs. Thanks to the cube cache, the SEDP can avoid up to 90% remote accesses, which reduces 10% latency on average.

Query cache. We further introduce the query cache to eliminate unnecessary computations in online inference. Remember the general target of the online recommendation service is to derive the user-item relevance score. The key design insight of the query cache is that the user-item score is stable in a short time period (*e.g.*, several minutes) under certain conditions. Figure 5(b) depicts the timeliness of the calculated user-item score. As can be seen, by bounding the time window to two minutes, over 60% scores are invariant. Therefore, for certain recently processed user-item pairs, the computation can be safely eliminated by reusing the previously calculated score, with little influence on the overall recommendation performance. The query cache is designed as purely in-memory storage, which persists a set of recently calculated user-item scores that fulfill certain conditions (*e.g.*, high relevance items). To guarantee the recommendation effectiveness, the cached scores will expire after a tunable time window or any user feedback (*e.g.*, click, unlike), which indicates the intermediate change of user preference. Unlike the cube cache, the query cache employs the Least Recently Used (LRU) policy. Query cache helps JiZhi reduces over 20% computations, which significantly improves the cost-effectiveness of the online inference service.

6 INTELLIGENT RESOURCE MANAGER

In this section, we describes the *Intelligent Resource Manager* (IRM) in detail, including both offline and online auto-tuning components. By automatically searching and learning the resource allocation and the load shedding policies for SEDP and HHS, IRM shields engineers and data scientists from the complexity of performance tuning, and achieves even better performance than expert prior.

6.1 Offline Auto-Tuning for Quasi-Optimal Resource Allocation

Beyond tuning every single stage processors and caches independently, JiZhi optimizes the parameters of the system on a shared computing infrastructure in a global manner. Specifically, the goal of offline optimization is to minimize the resource consumption under the latency constraint. We first categorize the tunable system parameters into two classes. (1) *System level* parameters that influence the global system. (2) *Stage level* parameters that work on dedicated stage processors. Please refer to Appendix A for more detailed offline optimization parameters.

Specifically, based on historical logs of every stage processor with the varying parameters, JiZhi collectively searches the global optimal resource allocation plan. The optimization objective is

$$\begin{aligned} O_{\text{offline}} = \underset{(\Theta,\theta_1,\theta_2,\ldots,\theta_N)\in\Gamma}{\operatorname{argmin}} \sum_{j=1}^{N} \mathcal{F}_j^R(\Theta;\theta_j), \\ s.t.\ \mathcal{F}_j^L(\Theta,\theta_j) \le \mathcal{F}_j^L(\bar{\Theta},\bar{\theta}_j),\ \forall 1 \le j \le N, \end{aligned} \tag{1}$$

where N refers to the number of stage processors in the system, Γ denotes the set of all possible parameters, Θ refers to the system level parameters, θ_j for $1 \le j \le N$ refers to the stage level parameters for the j^{th} stage processor, $\bar{\Theta}$ and $\bar{\theta}_j$ are the default settings of the parameters. The model $\mathcal{F}_j^R(\Theta; \theta_j)$ predicts the resource consumption of the j^{th} stage processor based on the parameters of Θ and θ_j, while the model $\mathcal{F}_j^L(\Theta; \theta_j)$ predicts the end-to-end latency of the j^{th} stage processor based on the parameters of Θ and θ_j. This optimization problem is challenging as (1) the feasible parameters should be able to bound the end-to-end latency of every stage processor (i.e., N constraints), (2) $\mathcal{F}_j^R(\Theta; \theta_j)$ and $\mathcal{F}_j^L(\Theta; \theta_j)$ are built using ensemble of practical regression models, which are not naturally non-differentiable over the parameter spaces, and (3) the prediction results of $\mathcal{F}_j^R(\Theta; \theta_j)$ and $\mathcal{F}_j^L(\Theta; \theta_j)$ are noisy and probably biased compared to the ground truth. Thus, in addition to common gradient-based approaches, global optimization is required for parameter search. JiZhi devises the Covariance Matrix Adaptation – Evolution Strategy (CMA-ES) with constraints [2] for optimal plan search. The objective and constraints are all encoded as black-box functions, while CMA-ES with constraints handles these functions naturally for global optimization in a sampling manner. Note that, in addition to using the minimizers obtained by CMA-ES, JiZhi restores the solution paths of CMA-ES (*i.e.*, the parameters having been ever reached), pickup constraint-satisfied results with the minimal objectives from solution paths, and evaluate these results in the realistic system deployment (*i.e.*, using 5% of overall online requests) to find the global optimal parameters.

6.2 Online Resource Management for Adaptive Load Shedding

Different with the offline auto-tuning, the online resource management aims to handle the time-varying traffic load by swiftly deriving fine-grained computation policy for each online inference task based on real-time system feed-backs. The online resource management is more tightly coupled with the recommendation service logic and usually requires task-specific optimization. In this paper, we showcase a commonly used online load shedding module integrated in various online inference services in Baidu, to demonstrate the cost-effectiveness of online resource management.

As aforementioned, most industrial recommendation services adopt a funnel-shaped pipeline [9] by first retrieving a subset of potential candidates based on lightweight features (*a.k.a.* the recall phase) and then scoring each candidate more accurately based on more computation expensive features (*a.k.a.* re-ranking phase). In most cases, only a dozen items will eventually be recommended to users, but over three orders of magnitude candidates will be passed to the computational expensive re-ranking stage processors. The key insight of online load shedding is to adaptively prune low-quality candidates when the traffic load exceed the system capacity, with the minimum recommendation effectiveness degradation constraint. Formally, given a set of selected candidate items $D^{inf} = \{d_1, d_2, \ldots, d_n\}$ from the recall phase, the objective of the online load shedding component is

$$O_{\text{online}} = \arg\max_{q} \frac{1}{K} \sum_{i=1}^{K} C(p_i(q_i \circ D_i^{inf})), \qquad (2)$$
$$s.t. \; |\mathcal{L}^* - \hat{\mathcal{L}}| \leq \epsilon,$$

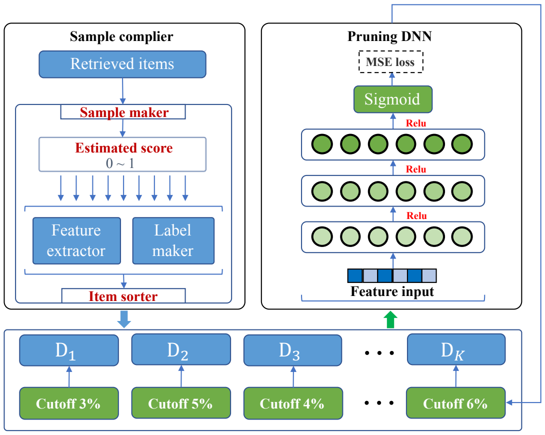


Figure 6: Pruning DNN based online load shedding.

where K is the total number of recommendation requests, C is the computational cost function, p_i is the corresponding stage processor, q_i is the pruning function conduct element-wise pruning on D_i^{inf}, $\mathcal{L}^*$ and $\hat{\mathcal{L}}$ are respectively the accuracy before and after online load shedding, ϵ is a small value bounds the recommendation effectiveness degradation.

To find the optimal pruning policy in an ultra-efficiency way, we transform the constrained optimization problem to a regression problem and employ a lightweight DNN (*a.k.a.* the pruning DNN) to decide the number of candidates to cutoff. Specifically, we first sort D_i^{inf} by leveraging the estimated score from the recall phase. Then we apply the well-trained DNN to decide the cutoff line for each D_i^{inf}, where the candidates behind the cutoff line will be directly dropped without passing to the subsequent re-ranking stage processors. Figure 6 depicts the workflow of the online load shedding process. Based on extracted features from the upstream stage processors and context features such as the intermediate system latency, the pruning DNN is ultra-lightweight and can prune low-quality candidates in dozens of microseconds. Please refer to Appendix A for detailed feature list. Different with general load shedding on data streams [3, 31], the online load shedding in JiZhi adaptively decides the request-wise pruning ratio by taking the global recommendation effectiveness into consideration.

7 DEPLOYMENT

JiZhi has been deployed in the production environment to support a variety of real-world products in Baidu. In this section, we discuss the deployment details. Please refer to Appendix B for details of implementation.

Online deployment. JiZhi answers remote service access via BRPC (https://github.com/brpc/brpc), a scalable Remote Procedure Call framework used throughout Baidu. The online inference service is duplicated in data centers distributed across China to reduce potential network latency. Note in JiZhi, the response latency and resource consumption is a design trade-off, where allocate more sources can improve the system throughput and reduce latency, and developers can reduce the hardware consumption by relaxing the latency in a reasonable range.

Model management and hot-loading. In JiZhi, the sparse part of the DNN is managed by the distributed sparse parameter

Table 1: Statistics of online inference services.

	Model size	# of feature groups	Traffic load
Service A	430 GB	379	$4.58 \times 10^8/s$
Service B	500 GB	430	$4.21 \times 10^8/s$
Service C	285 GB	270	$3.67 \times 10^7/s$
Service D	210 GB	106	$7.15 \times 10^7/s$

cube and the dense part is directly stored in memory. In real-world recommendation applications, the model is updated frequently (*e.g.*, per hour), and it is undesired to update the model by frequently restart the service. To guarantee the availability of the online service, JiZhi supports model hot loading without service interruption. Specifically, we build a monitoring module to continuously track model update state in the remote address (*i.e.*, the model training cluster). Once a new model is ready (identified by model generation timestamp), the new model will be pulled to local and loaded to serve new coming recommendation requests via double buffering.

8 EXPERIMENTAL EVALUATION

8.1 Experimental Setup

In this section, we conduct extensive experiments to evaluate the effectiveness of the proposed system. We report evaluations on four different JiZhi empowered real-world online inference services deployed in Baidu. All results are collected from the online living system. Specifically, *Service A* is for a multi-modal recommendation model on Baidu news feed, *Service B* and *Service C* are video recommendation services used in different product scenarios, *Service D* is a multi-objective recommendation service fulfilling multiple targets. Table 1 summarizes model statistics of five recommendation services. We use the previous online inference system (Legacy) as described in Section 2 as the compared baseline.

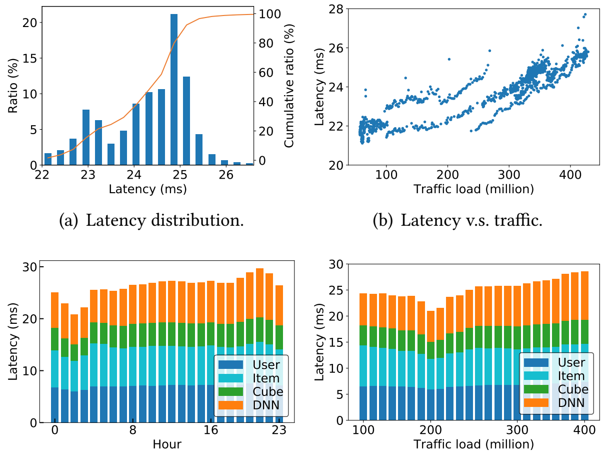
In the evaluation, we focus on system performance metrics. We use latency, throughput, and the number of required instances (*i.e.*, the computational resource) to evaluate the efficiency and cost-effectiveness of the system. Specifically, the latency is computed as the averaged time per task (user-item pair) in millisecond (ms), the throughput is defined as the number of tasks processed per second. An instance refers to a virtual machine on the cloud, which is commonly equipped with four CPU cores, 20 GB memory, and shares 8 TB disk in a physical server (4TB SSD and 4TB SATA).

8.2 Overall Production Experience

Table 2 reports the overall performance of JiZhi compared with the legacy online inference system on four different services with respect to three metrics. As can be seen, JiZhi outperforms Legacy on all services in terms of all metrics. Specifically, JiZhi achieves (23.33%, 17.24%, 2.5%, 18.18%) latency improvement and (188.37%, 178.13%, 86.16%, 133.41%) throughput improvement on four services, respectively. Benefit from such improvement, JiZhi reduces (65.33%, 62.56%, 46.3%, 57.17%) instances in the online serving infrastructure without sacrificing system efficiency, which helps Baidu save over ten million US dollars' computational resource budgets (hardware, energy consumption, labor cost, *etc.*) per year. And we foresee this economic gain will improve with the future traffic load growth of each recommendation service. Further look into the performance of each service, we notice the performance gain of Service C is relatively lower than others, which mainly because the corresponding product is resource constrained (*i.e.*, less budget than others), and the system is tuned to minimize the resource consumption.

Table 2: Overall performance.

		Latency	Throughput	# of instance
Service A	Legacy	30 ms	1.53×10^6	11,450
	JiZhi	23 ms	4.42×10^6	3,970
Service B	Legacy	29 ms	1.63×10^6	12,750
	JiZhi	24 ms	4.36×10^6	4,773
Service C	Legacy	41 ms	2.8×10^6	2,067
	JiZhi	40 ms	5.21×10^6	1,110
Service D	Legacy	22 ms	3.53×10^6	4280
	JiZhi	18 ms	8.24×10^6	1833



(a) Latency distribution.

(b) Latency v.s. traffic.

(c) Latency breakdown by hour.

(d) Latency breakdown by traffic.

Figure 7: Effect of SEDP on latency.

8.3 Effect of SEDP

Then we look into the inference time distribution to study the effect of SEDP. Due to page limit, we only report the results on Service A, the results on other services are similar.

We first study the overall latency distribution. Figure 7(a) and Figure 7(b) report the overall online inference latency distribution and the latency distribution with respect to the varying traffic load, respectively. As can be seen, the overall latency follows a bimodal distribution. This makes sense as the first peak corresponds to requests hit two caches, and the second peak corresponds to majority requests in the peak hour. Moreover, we observe the latency increases sub-linearly with respect to the traffic load growth, which verifies the robustness of the system under time-varying traffics.

Figure 7(c) and Figure 7(d) breakdown the latency into four major stages, *i.e.*, user related processing, item related processing, cube related processing, and DNN forward computation. Overall, we observe the ratio of latency in each stage is relatively stable at different hours and traffic loads. Moreover, the overall latency slightly increases in the evening peak hour. Besides, with the traffic load growth, we observe the averaged latency first decrease and then increase. This is related to the cache hit ratio, which first increases when the ratio of frequent requests increase and then decreases when more long-tailed items flush two caches.

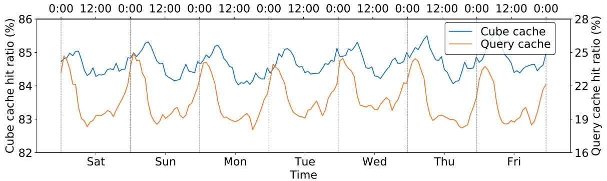


Figure 8: Hit ratio of the cube cache and query cache.

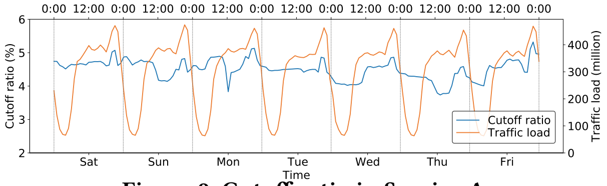


Figure 9: Cutoff ratio in Service A.

8.4 Effect of the Storage Module

We report the hit ratio of the cube cache and the query cache at different hours of Service A in a whole week to demonstrate the effect of the heterogeneous and hierarchical storage, as shown in Figure 8. Overall, the hit ratio of two caches are varying by hour and showing strong daily periodicity, which is highly correlated with the traffic load variation. In particular, the cube cache achieves 84.21% hit ratio in average and the daily variation is less than 3.61%, while the query cache yield 19.26% hit ratio in average but with a higher variation range. In summary, the cube cache achieves higher hit ratio and every hit reduces partial data access latency, while the query cache achieves a relative lower hit ratio but per hit eliminates the whole inference task computation. Two caches are robust to reduce the inference latency and computational overhead at different times in the week.

8.5 Effect of the Intelligent Resource Manager

We further justify the performance gain of the offline resource optimization and online load shedding.

Offline resource optimization. Table 3 reports the quantitative results of the resource gain of each service with the offline resource optimization. As can be seen, the offline resource optimization module successfully reduces 8.91% to 16.45% instances without sacrificing system latency. We further conduct the qualitative study by comparing key optimization parameters of Service A before (*noOpt*) and after (*Opt*) applying offline resource optimization, which is shown in Table 4. As can be seen, the resource manager assigns a larger batch size to most stage processors except the cube accessor. This makes sense since over 80% computations in this stage is reduced by the cube cache, as discussed in Section 8.4. Moreover, the size of the cube cache and the expire window of the query cache are increased, and the offline manager derives a more aggressive memory allocation strategy. Such settings can further reduce the data access latency and improve the CPU utilization, which therefore improves the overall resource utilization rate.

Online load shedding. To illustrate the effectiveness of online load shedding, Figure 9 depicts the cutoff ratio of the online load shedding and the corresponding traffic loads over a week. We observe highly synchronized fluctuations of the cutoff ratio and the traffic load. Specifically, in the evening peak hour of traffic load, we observe the peak of cutoff ratio simultaneously. Besides, we observe a relatively higher cutoff ratio at midnight, perhaps because of a lower click ratio in this period, indicating more items can be safely dropped without influencing recommendation effectiveness.

Table 3: Resource gain of offline auto-tuning.

Service	Reduced # of instance	Gain
A	1,636	14.29%
B	1,736	13.62%
C	184	8.91%
D	704	16.45%

Table 4: Offline auto-tuning results in Service A.

Category	Parameter	*noOpt*	*Opt*
Stage	User batch	30	34
	Item extractor batch	4	12
	Item processor batch	6	17
	Cube batch size	10	6
	DNN batch size	15	25
System	Cube cache ratio	1%	1.2%
	Query cache window	120s	143s
	# of arenas	500	549
	Max active extent	6	25
	Huge page	Default	Always

Table 5: Effectiveness of multi-tenant serving.

	Latency	Throughput	# of instance
CTR	50 ms	2.48×10^6	3,617
FR	48 ms	3.42×10^6	2,560
CMT	42 ms	3.4×10^6	1,350
JiZhi	52 ms	4.53×10^6	1,980

Overall, both offline and online resource management components successfully tune the online inference system to achieve better performance automatically.

8.6 Multi-Tenant Serving

Finally, we use *Service E*, a recommendation service including multiple DNNs, to demonstrate the effectiveness of multi-tenant support in JiZhi. Specifically, *Service E* including three different deep learning models, namely *CTR*, *FR*, *CMT*, where each model corresponds to a different recommendation target. The models are $1,743$ GB in total, involve 968 different feature groups, and the service handles 9.19×10^7 traffics per second. By deploying each different model as individual services, Table 5 reports the latency, throughput and number of instances of each individual services compared with the multi-tenant based online inference on JiZhi. Overall, we observe similar system latency by simultaneously executing three different models, indicating a linear speedup ratio of parallelization in JiZhi. Moreover, JiZhi achieves 82.68% throughput improvement comparing with the bottleneck model (*i.e.*, *CTR*). Most importantly, by integrating three individual models into a unified online inference service, JiZhi saves 73.69% instances in the cluster without losing efficiency. JiZhi is even more cost-effective in the multi-tenant scenario than other single-model services (*i.e.*, Service A to Service D). This is because the unified service can reduce repetitive data access cost for multiple individual services (*e.g.*, over 80% feature groups are shared by three models), and largely reuses the intermediate results using the heterogeneous and hierarchical storage.

9 CONCLUSION AND FUTURE WORK

In this work, we presented JiZhi, a fast and cost-effective online inference system at Baidu for the web-scale recommendation. We first proposed SEDP, a directed acyclic graph style asynchronous pipeline for online inference serving. By decoupling the inference process into modularized stage processors, SEDP reduces pipeline stall and enables flexible workflow customization and auto-tuning for a wide spectrum of recommendation scenarios. Then we introduce the heterogeneous and hierarchical storage for cost-effective sparse DNN model management, to simultaneously accelerate the online inference speed and reduce the overall computational overhead. After that, an intelligent resource management module is proposed to optimize the computational resource utilization in both offline and online, under inference latency constraints. JiZhi has been deployed in Baidu, serving over twenty real-world internet products, and helped Baidu saved over ten million US dollars computational resource budget per year while handles hundred millions online inference requests per second without sacrificing the inference latency. We share our design and practice experience on building the web-scale online inference system and hope to provide useful insights to communities in both academia and industry.

REFERENCES

[1] Hossein Amiri and Asadollah Shahbahrami. 2020. SIMD programming using Intel vector extensions. *J. Parallel and Distrib. Comput.* 135 (2020), 83–100.

[2] Dirk V Arnold and Nikolaus Hansen. 2012. A (1+ 1)-CMA-ES for constrained optimisation. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 297–304.

[3] Brian Babcock, Mayur Datar, and Rajeev Motwani. 2007. Load shedding in data stream systems. In *Data Streams*. 127–147.

[4] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 2019. 150 successful machine learning models: 6 lessons learned at booking. com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1743–1751.

[5] Jake Brutlag. 2009. Speed Matters for Google Web Search. https://services.google.com/fh/files/blogs/google_delayexp.pdf

[6] J Lawrence Carter and Mark N Wegman. 1979. Universal classes of hash functions. *Journal of computer and system sciences* 18, 2 (1979), 143–154.

[7] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 609–622.

[8] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).

[9] Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. 2019. MOBIUS: towards the next generation of query-ad matching in baidu's sponsored search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2509–2517.

[10] Nadeem Firasta, Mark Buxton, Paula Jinbo, Kaveh Nasri, and Shihjong Kuo. 2008. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel white paper* 19, 20 (2008).

[11] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14.

[12] Zhoutong Fu, Huiji Gao, Weiwei Guo, Sandeep Kumar Jha, Jun Jia, Xiaowei Liu, Bo Long, Jun Shi, Sida Wang, and Mingzhou Zhou. 2020. Deep Learning for Search and Recommender Systems in Practice. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3515–3516.

[13] Long Guo, Lifeng Hua, Rongfei Jia, Binqiang Zhao, Xiaobo Wang, and Bin Cui. 2019. Buying or browsing?: Predicting real-time purchasing intent using attention-based deep network with multiple behavior. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1984–1992.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[15] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[16] 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 *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 359–367.

[17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[18] Hemank Lamba and Neil Shah. 2019. Modeling dwell time engagement on visual multimedia. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1104–1113.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[20] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *Proceedings of the 1999 ACM SIGMETRICS International conference on Measurement and Modeling of Computer Systems*. 134–143.

[21] Hao Liu, Jindong Han, Yanjie Fu, Jingbo Zhou, Xinjiang Lu, and Hui Xiong. 2021. Multi-Modal Transportation Recommendation with Unified Route Representation Learning. *Proceedings of the VLDB Endowment* 14, 3 (2021), 342–350.

[22] Hao Liu, Yongxin Tong, Jindong Han, Panpan Zhang, Xinjiang Lu, and Hui Xiong. 2020. Incorporating Multi-Source Urban Data for Personalized and Context-Aware Multi-Modal Transportation Recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[23] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. 2011. Acoustic modeling using deep belief networks. *IEEE transactions on audio, speech, and language processing* 20, 1 (2011), 14–22.

[24] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *International Conference on Machine Learning*.

[25] Christopher Olston, Fangwei Li, Jeremiah Harmsen, Jordan Soyke, Kiril Gorovoy, Li Lao, Noah Fiedel, Sukriti Ramesh, and Vinu Rajashekhar. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. In *Workshop on ML Systems at NIPS 2017*.

[26] Jian Ouyang, Shiding Lin, Wei Qi, Yong Wang, Bo Yu, and Song Jiang. 2014. SDA: Software-defined accelerator for large-scale DNN systems. In *2014 IEEE Hot Chips 26 Symposium*. 1–23.

[27] Jian Ouyang, Mijung Noh, Yong Wang, Wei Qi, Yin Ma, Canghai Gu, SoonGon Kim, Ki-il Hong, Wang-Keun Bae, Zhibiao Zhao, et al. 2020. Baidu Kunlun An AI processor for diversified workloads. In *2020 IEEE Hot Chips 32 Symposium*. 1–18.

[28] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. 2008. GPU computing. *Proc. IEEE* 96, 5 (2008), 879–899.

[29] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. 2018. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. *arXiv preprint arXiv:1811.09886* (2018).

[30] Chuan Qin, Hengshu Zhu, Tong Xu, Chen Zhu, Chao Ma, Enhong Chen, and Hui Xiong. 2020. An enhanced neural network approach to person-job fit in talent recruitment. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–33.

[31] Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. 2003. Load shedding in a data stream manager. In *Proceedings of the 29th International Conference on Very Large Data Bases*. 309–320.

[32] Han Vanholder. 2016. Efficient inference with TensorRT.

[33] Xiao Yang, Daren Sun, Ruiwei Zhu, Tao Deng, Zhi Guo, Zongyao Ding, Shouke Qin, and Yanfeng Zhu. 2019. Aiads: Automated and intelligent advertising system for sponsored search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1881–1890.

[34] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu. 2013. Asynchronous stochastic gradient descent for DNN training. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 6660–6663.

[35] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. 2019. AIBox: CTR prediction model training on a single node. In *Proceedings of the 28th ACM International Conference on Information & Knowledge Management*. 319–328.

[36] 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.

[37] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2810–2818.

Table 6: Representative search parameters in offline performance tuning.

Parameter	Type	Range
Stage Level Parameters		
User batch	Integer	[10,45]
Item extractor batch	Integer	[2,45]
Item processor batch	Integer	[2,45]
Cube batch size	Integer	[1,20]
DNN batch size	Integer	[10,45]
System Level Parameters		
Cube cache ratio	continuous	[0.1,5]%
Query cache window	continuous	[60,600]s
# of arenas (jemalloc)	Integer	[350,700]
Max active extent (jemalloc)	Integer	[5,40]
Huge page (jemalloc)	Boolean	{Default, Always}

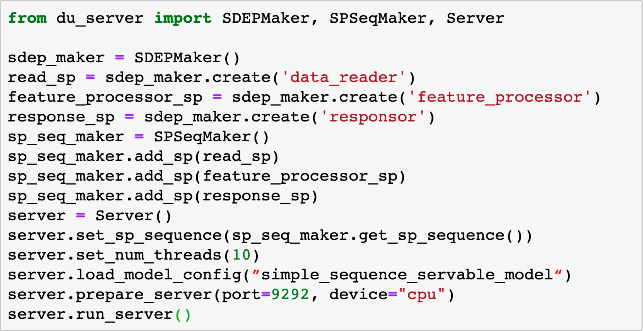
Table 7: Features for online load shedding.

Feature name	Description
quota	Available resource
cutoff ratio	Previous cutoff ratio
qid	Current item queue ID
escore_avg	average of estimated scores
escore_variance	variance of estimated scores
escore_max	max of estimated scores
escore_min	min of estimated scores

A DETAILED PARAMETER DESCRIPTION

Table 6 lists some important parameters and their feasible ranges in offline tuning. Specifically, more memory related parameters can be found in the jemalloc document (http://jemalloc.net/).

Table 7 lists lightweight features used for intelligent online load shedding. All features are either derived from real-time system feedback or upstream processing results.



```python
from du_server import SDEPMaker, SPSeqMaker, Server

sdep_maker = SDEPMaker()
read_sp = sdep_maker.create('data_reader')
feature_processor_sp = sdep_maker.create('feature_processor')
response_sp = sdep_maker.create('responsor')
sp_seq_maker = SPSeqMaker()
sp_seq_maker.add_sp(read_sp)
sp_seq_maker.add_sp(feature_processor_sp)
sp_seq_maker.add_sp(response_sp)
server = Server()
server.set_sp_sequence(sp_seq_maker.get_sp_sequence())
server.set_num_threads(10)
server.load_model_config(”simple_sequence_servable_model“)
server.prepare_server(port=9292, device="cpu")
server.run_server()
```

Figure 10: Constructing customized online inference service by using the Python API in a few lines of code.

B SYSTEM IMPLEMENTATION

The core framework of JiZhi is implemented by C++. Developers can construct customized stage processors by implementing complex data pre-possessing and post-processing logic based on several simple abstractions. Moreover, JiZhi also provide a Python API extension including of a set commonly used stage processors, such as *data_reader*, *feature_processor*, *cube_accessor*, and *model_inference*. Engineers, data scientists and researchers can easily and efficiently construct and customize online inference services for a variety of online recommendation tasks. Figure 10 showcases an example code block based on the Python API on PaddlePaddle, which defines an online inference service in a few lines of code.