Lion: A GPU-Accelerated Online Serving System for Web-Scale Recommendation at Baidu

Hao Liu
liuh@ust.hk
AIT, HKUST(GZ)
CSE, HKUST

Qian Gao
gaoqian05@baidu.com
Baidu, Inc.

Xiaochao Liao
liaoxiaochao@baidu.com
Baidu, Inc.

Guangxing Chen
chenguangxing@baidu.com
Baidu, Inc.

Hao Xiong
xionghao02@baidu.com
Baidu, Inc.

Silin Ren
rensilin@baidu.com
Baidu, Inc.

Guobao Yang
yangguobao@baidu.com
Baidu, Inc.

Zhiwei Zha
zhazhiwei@baidu.com
Baidu, Inc.

ABSTRACT

Deep Neural Network (DNN) based recommendation systems are widely used in the modern internet industry for a variety of services. However, the rapid expansion of application scenarios and the explosive global internet traffic growth have caused the industry to face increasing challenges to serve the complicated recommendation workflow regarding online recommendation efficiency and compute resource overhead. In this paper, we present a GPU-accelerated online serving system, namely *Lion*, which consists of the staged event-driven heterogeneous pipeline, unified memory manager, and automatic execution optimizer to handle web-scale traffic in a real-time and cost-effective way. Moreover, Lion provides a heterogeneous template library to enable fast development and migration for diverse in-house web-scale recommendation systems without requiring knowledge of heterogeneous programming. The system is currently deployed at Baidu, supporting over twenty recommendation services, including news feed, short video clips, and the search engine. Extensive experimental studies on five real-world deployed online recommendation services demonstrate the superiority of the proposed GPU-accelerated online serving system. Since launched in early 2020, Lion has answered billions of recommendation requests per day, and has helped Baidu successfully save millions of U.S. dollars in hardware and utility costs per year.

CCS CONCEPTS

• **Information systems** → **Computing platforms**; **World Wide Web**; • **Computer systems organization** → **Distributed architectures**.

*Qian Gao is the Corresponding author.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539058

KEYWORDS

online serving, recommendation system, heterogeneous computing

ACM Reference Format:
Hao Liu, Qian Gao, Xiaochao Liao, Guangxing Chen, Hao Xiong, Silin Ren, Guobao Yang, and Zhiwei Zha. 2022. Lion: A GPU-Accelerated Online Serving System for Web-Scale Recommendation at Baidu. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539058

1 INTRODUCTION

Deep Neural Network (DNN) based recommendation system has been emerging as a powerful tool to helping users find interesting items from an overwhelming number of candidates [6, 33]. As a leading internet service provider in China, Baidu has been devoted to exploiting DNNs for recommendations since the early 2010s [32]. Figure 1 reports the brief evolution history of the recommendation model and the corresponding serving system at Baidu. Nowadays, Baidu's DNN empowered recommendation systems per second handle billions of recommendation requests, responsible for over 99% content distribution, and directly driving over one billion users' experience as well as the company revenue [7, 32].

Although DNN substantially improves the recommendation effectiveness, the diverse recommendation scenarios and explosive growth of internet traffic present considerable challenges to serve the DNN based recommendation systems. Each year, Baidu spends substantial budget on the online serving infrastructure, and the expense is still quickly growing [21]. Compared with the prosperity of research and practice on training large-scale DNNs [20, 32], optimizing online inference for DNN based recommendation systems attracted relatively little attention. To name a few, TensorRT [26] is an optimized GPU inference library to speed up general DNN inference tasks. To deal with the execution characteristics of DNNs for recommendation, Facebook proposed a hill-climbing based scheduler [10] for scalable online DNN inference on a single machine. In 2019, Baidu launched JiZhi [21], a CPU-based distributed serving system for web-scale online recommendation services, which yield remarkable performance and resource consumption gain in production. In summary, existing approaches either focus on single node serving optimization or serving the recommendation system

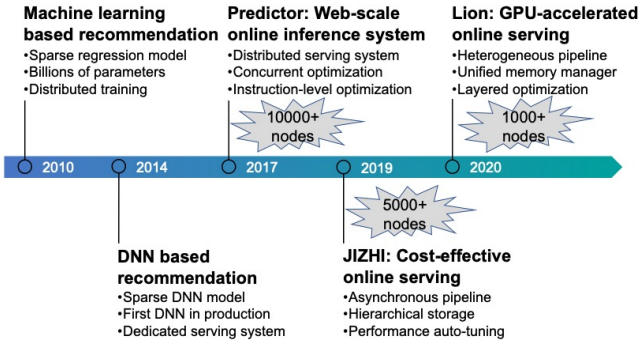


Figure 1: The ten-year evolution history of the web-scale online recommendation system at Baidu. Baidu deployed the first DNN based recommendation system in 2013 and constructed the dedicated online serving system in 2017.

on CPU clusters. In contrast to the proliferation of GPU-based DNN training systems [16, 32], how to exploit GPU clusters for more efficient and cost-effective online serving for web-scale online recommendation remains an under-explored problem.

However, serving web-scale online recommendations on GPU cluster is a non-trivial task, because of the following three major challenges. (1) *Complicated inference workflow.* Serving DNN based recommendation system involves complicated end-to-end inference workflow, where the DNN feed-forward computation only comprises a small portion of the whole inference cycles [21]. Directly migrating the online serving workflow on GPUs not only induce extensive and fragmented kernel launch overhead [25], but also result in sub-optimal resource utilization. How to design tailored serving system and collectively optimize the execution pipeline is the first challenge to fully take the computational advantage of the GPU-equipped cluster. (2) *Irregular parameter access.* Different from DNNs designed for computer vision [27] and nature language processing tasks [30], the DNN for web-scale recommendation usually contains billions of sparse parameters and only a small portion (*e.g.*, hundreds) of them will be accessed in each inference task [12, 32]. Such huge model parameters cannot directly fit into on-chip GPU memory and the randomized cross-device sparse parameter access will significantly degrade the online serving performance. How to optimize the sparse DNN parameter management to avoid irregular cross-device parameter access is another challenge. (3) *Expensive development and migration cost.* Due to the inherent architecture difference between pure CPU cluster and GPU-equipped cluster (*e.g.*, parallel execution, memory management, data transfer), the existing CPU-optimized online serving systems cannot be directly migrated to GPU clusters. However, the diverse recommendation workflow connecting the online service to heterogeneous computing and storage infrastructures make the serving system development even more complex, requiring expert knowledge on GPU programming and performance tuning. The third challenge is how to release engineers and data scientists from the cumbersome development and migration workload of the online serving system to let them focus more on model exploration.

To this end, in this paper, we present Lion, a GPU-accelerated end-to-end online serving system for the web-scale recommendation. The key design objective of Lion is moving online serving from CPUs to GPU-equipped clusters, so that to handle hundred thousands of concurrent online recommendation requests in a more efficient and cost-effective way. Specifically, we first propose a *Staged Event-driven Heterogeneous Pipeline* (SEHP) to compile the complicated online recommendation workflow to a Directed Acyclic Graph (DAG). The SEHP not only enables mixed and asynchronous execution of modularized online recommendation operators on CPUs and GPUs, but also exposes the opportunity of SEHP customization for diverse recommendation scenarios. Moreover, we construct a *Unified Memory Manager* (UMM) to provide dynamic memory resource management for on-chip GPU memory. The UMM dynamically allocates resources for different operators on GPUs without requiring repetitive kernel launch, thus significantly reducing the resource management overhead for online serving on GPU-equipped clusters. A distributed and heterogeneous cache component is also proposed in UMM to further reduce random parameter access and cross-device transfer costs. Besides, we propose an *Automatic Execution Optimizer* (AEO) to generate the optimal execution and resource allocation plan. By automatically packing dependency-free operators into shared execution kernels and adaptively tuning global resource assignments, AEO significantly reduces the unnecessary thread instantiation overhead and improves the online serving throughput under latency constraints. Finally, a *Heterogeneous Template Library* (HTL) is provided to ease the development and migration of diverse online recommendation services. By providing a seamless programming interface and optimized primitives, engineers and researchers are free to explore new recommendation workflow without requiring expert knowledge on heterogeneous programming and the underlying hardware.

To the best of our knowledge, this is the first production-level GPU-accelerated serving system for web-scale online recommendation. The contributions of this paper are summarized as follows. (1) We discuss challenges and requirements for building a GPU-accelerated online serving system for web-scale recommendation. (2) We present a GPU-accelerated online serving system, namely Lion, to handle web-scale recommendation traffic efficiently and cost-effectively. We share our practical experiences on system design, integration, implementation, and deployment to benefit the community for the system development. (3) We have deployed the proposed system at Baidu for over twenty DNN based online recommendation services and conducted extensive performance study under real-world traffics. The results demonstrate Lion outperforms CPU-based online serving system in terms of latency, throughput, and resource consumption.

2 BACKGROUND AND RELATED WORK

2.1 DNN based Web-Scale Recommendation

Web-scale online recommendation aims to suggest new items (*e.g.*, news, videos, products) from millions of candidates, with the objective of improving personalized user satisfaction such as Click Through Ratio (CTR) and long-term engagement [33, 34]. Due to the superiority of high-order feature extraction and complex interaction modeling, DNN has become an essential module in modern recommendation system [14, 33]. Figure 2 shows an illustrative workflow of DNN based personalized recommendation. Compared with deep learning based computer vision [27] and natural language

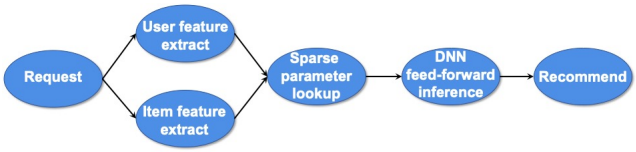


Figure 2: An illustrative workflow of online serving.

processing tasks [30], the DNN based recommendation systems have several unique characteristics in industrial practice.

First, the DNN for web-scale recommendation is learned from extremely high-dimensional and sparse categorical features (*i.e.*, hundreds of non-zero values out of hundreds billions of features), resulting in super large sparse network and moderate sized dense network [12]. For instance, the raw DNN model used for Baidu news feed contains of over 10^{11} sparse parameters and is over 10 TB large. Even after compression, the model is over 500 GB large in production. Such larges-scale sparse network requires dedicated compute and storage system design for online serving. Second, in industrial applications, the recommendation workflow usually follows the funnel-shaped structure, where multiple independent DNNs are involved to give the final recommendation suggestion [7]. To reduce the computational complexity, the recommendation system first retrieve and filter candidates from millions to thousands based on lightweight features and DNN models (even statistical models in the very first stage), then ranking the relevance between user-item pair based on more complicated and time-consuming DNN models. Such multi-pass paradigm induce complicated recommendation workflow and require intensive computing resources for scalable online serving. Third, serving DNN based web-scale online recommendation is more than executing the DNN feed-forward computation. After performing large-scale online recommendation traffic analysis, we find the DNN feed-forward computation for online recommendation only comprises 10% to 40% of the whole online serving cycles in most of deployed online recommendation services. The construction of sophisticated features based on real-time user behaviors and the sparse parameter lookup dominate the online serving pipeline. Please refer Appendix A for more detailed analysis. Therefore, the complicated feature construction and parameter access steps are critical to reduce the online serving latency and improve the system throughput.

2.2 Serve DNN based Online Recommendation

Each year, Baidu spends considerable budget on serving DNN based recommendation systems, including both the high performance infrastructure and operational costs. In fact, the cost of serving DNN based recommendation system becomes the major burden for many companies to adopt the techniques [9]. Facing hundred of thousands of concurrent recommendation requests, serving web-scale online recommendation can be formulated as a latency-bounded throughput maximization problem [12]. And some efforts have been made from both the industrial and academia.

2.2.1 CPU-based online serving system. Given a recommendation request, the DNN is required to estimate the relevance score between the user and millions of candidate items. Therefore, the parallelism degree becomes critical for the online serving system's

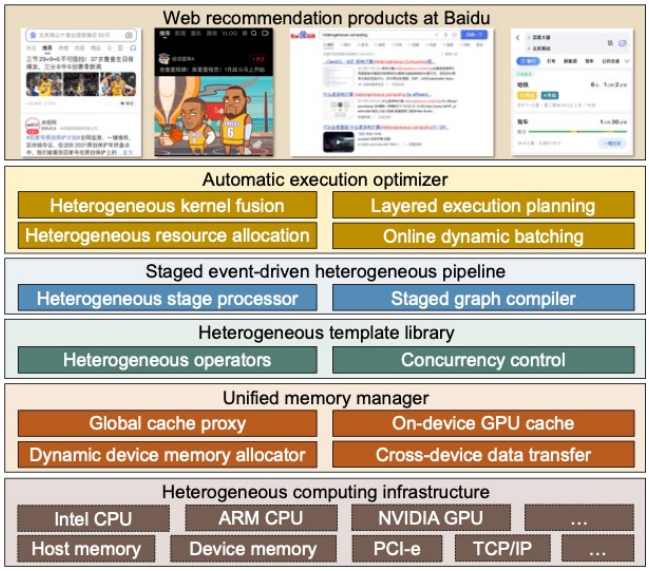


Figure 3: Framework overview of Lion.

performance. On the one hand, leveraging multi-threading and SIMD instruction sets (*e.g.*, AVX-512 [8], SSE [2]) can considerably improve the CPU utilization from the data-parallel perspective. On the other hand, scheduling the online recommendation tasks execution and reducing irregular memory access can reduce the CPU stall and improve overall system throughput. For example, Facebook proposed a mini-batched scheduler [10] and a tailored memory system [19] to improve the serving throughput on a single machine. Baidu proposed JiZhi [21] for cost-effective DNN inference, which optimized the online serving system at the data center level. By decomposing the online recommendation workflow to staged operators, JiZhi can significantly reduce resource consumption without sacrificing the serving latency.

2.2.2 GPU-based online serving system. Along with the development of deep learning, heterogeneous processors such as GPU have been widely adopted for accelerating DNN training [16, 31, 32]. Another promising way of optimizing the online serving system is to offload part of the computation task to heterogeneous processors. For example, TensorRT is an optimized GPU library for accelerating DNN inference by leveraging techniques such as tensor fusion and quantization [26], DART [29] and RecPipe [11] respectively proposed pipeline-based scheduler to support real-time DNN inference on CPU-GPU mixed single server. However, the above GPU-based serving systems are mainly designed for serving general-purpose DNNs but are limited to serve complicated online recommendation workflow under time-varying web-scale traffic. In this work, we investigate the GPU-accelerated serving system to optimize the efficiency and resource consumption for web-scale online recommendation. Note many other heterogeneous processors such as Tensor Processing Unit (TPU) [5] and FPGA [28] may also be adopted to accelerate online serving, which we left as future work.

3 SYSTEM OVERVIEW

Figure 3 illustrates the overall architecture of Lion, which is comprised of four major modules, *Staged Event-driven Heterogeneous*

Pipeline (SEHP), *Unified Memory Manager* (UMM), *Automatic Execution Optimizer* (AEO), and *Heterogeneous Template Library* (HTL). Given a recommendation service, the SEHP first compiles the online serving workflow to a Directed Acyclic Graph (DAG) with execution context (*i.e.*, GPU or CPU) specification for each node in the DAG. As hundred of thousands of recommendation requests concurrently arrive, the SEHP partitions inference tasks into different batches and estimates the recommendation score in parallel. Moreover, the UMM dynamically manages on-device memory to eliminate expensive on-device resource allocation overhead, along with a distributed and heterogeneous cache component to reduce irregular parameter access and cross-device data transfer. Besides, the AEO intelligently searches the layered execution and resource allocation plan in offline and adaptively tunes the online batching strategy to improve the overall serving throughput with latency constraint. Finally, the HTL is built to provide a seamless programming interface to reduce the migration and development workload for the practical use of the GPU-accelerated online serving system.

4 STAGED EVENT-DRIVEN HETEROGENEOUS PIPELINE

The key objective of SEHP is to enable online serving on heterogeneous infrastructures and to provide pipeline customization for diverse online recommendation workflows. We begin with the definition of heterogeneous stage processor.

Definition 1. ***Heterogeneous stage processor*** *(HSP). A heterogeneous stage processor is defined as a tuple* $\langle op, p \rangle$*, where* op *is the operator that handles a unit event (*e.g.*, feature aggregation, parameter retrieve) in the recommendation workflow, and* $p \in \{CPU, GPU\}$ *indicates the underlying execution context of the HSP.*

Once an HSP is implemented and registered in the system, it can be reused for different recommendation tasks. Currently, we have implemented over 160 commonly used HSPs (*e.g.*, feature extractors, cross-feature mapping), and all HSPs are optimized for both CPU and GPU execution to fully utilize the underlying heterogeneous architecture. Note that the operator op is also customizable to flexibly support different feature construction processes and new emerging DNN architectures. p is a generalized execution context which can be extended to support various heterogeneous processors, *e.g.*, TPU, FPGA. Based on HSP, we can construct the SEHP.

Definition 2. ***Staged event-driven heterogeneous pipeline*** *(SEHP). A SEHP is a Directed Acyclic Graph (DAG)* $G = (V, E)$*, where each node* $v_i \in V$ *is a heterogeneous stage processor, and each edge* $e_{ij} \in E$ *is an event channel that automatically queuing and passing processed data to a HSP from the previous one it depends on.*

Based on the execution context of connected HSPs v_i and v_j, the event channel e_{ij} passing accumulated data in batch based on the underlying network protocol (*e.g.*, RMDA for cross-node data transfer, PCI-e for CPU-GPU communication). In practice, there are three phases to deploy a SEHP, *i.e.*, *define*, *compile* and *execution*. After configuring the functionality and dependency of HSPs in the SEHP in *define* phase, the *compile* phase checks the SEHP validity and generates the execution plan of the SEHP on GPU-equipped clusters. Once the SEHP is compiled, the pipeline can be executed and reused online without requiring extra run-time analysis.

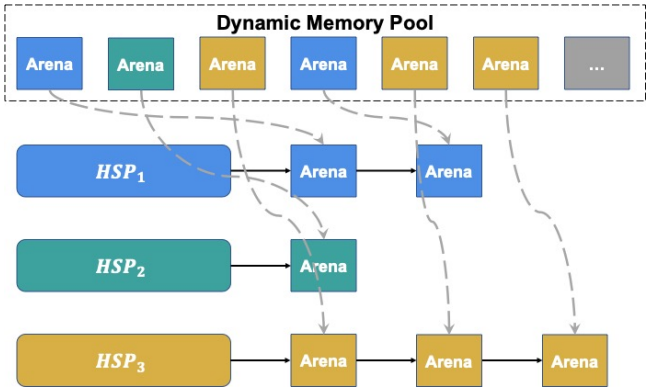


Figure 4: Dynamic memory management on GPU. The dynamic memory pool assign and recycle the on-device memory for different HSPs without requiring the kernel launch overhead of memory allocation.

The advantages of SEHP are at least threefold. First, by decomposing the online recommendation workflow into connected stage processors, different HSPs can be executed asynchronously. Such a design can better exploit the task-level and data-level parallelism of heterogeneous processors to improve the system throughput. Second, the modularized design provides the flexibility of scheduling different HSPs on distributed and heterogeneous clusters to improve the utilization of CPUs and GPUs [4]. Such design also opens the opportunity of performance auto-tuning, which we detail in Section 6. Third, the abstraction of SEHP hides lower-level implementation and optimization details of the heterogeneous infrastructure. The machine learning engineer and data scientist can focus more on algorithm-aspect online recommendation optimization.

5 UNIFIED MEMORY MANAGER

This section presents the *Unified Memory Manager* (UMM), including the dynamic GPU memory management and the cache system.

5.1 Dynamic GPU Memory Management

During the execution of SEHP, the complicated stage processors usually require dynamic memory resources (*e.g.*, the number of retrieved items may vary), which is unknown before run-time. This is challenging in on-device memory management, as the GPU is inefficient on logic control but the run-time resource allocation brings repetitive kernel launch steps [17]. Take the recommendation system used for Baidu news feed as an example. Without any optimization, the kernel launch cost for resource allocation comprises over 50% of the online inference cycles. Other than run-time allocation, straightforward alternative approaches such as pre-allocating larger on-device memory blocks will induce significant resource-wasting, which will decrease the overall serving throughput and is unpreferred in production.

In the UMM, we construct a dynamic memory manager to provide unified on-device memory management for efficient SEHP execution on GPU. Figure 4 gives an illustrative example of dynamic memory management for SEHP execution on GPUs. In the

SEHP compile phase, the dynamic memory manager first initialize a shared on-device dynamic memory pool for all HSPs in the SEHP. The dynamic memory pool consists of a set of memory arenas (*i.e.*, contiguous memory chunks) and is allocated with one kernel launch. During the execution phase, each HSP is assigned by the required number of arenas on-demand without requiring run-time kernel launch. When an HSP is done, the allocated arenas will be released back to the memory pool and can be re-assigned for subsequent HSPs. Therefore, the GPU memory can be dynamically allocated and repetitively used for various HSPs with only one kernel launch overhead in advance. Note one exception is that when the memory pool is empty or not enough for the new HSP execution, the dynamic memory manager will expand the memory pool with an extra kernel launch overhead. Besides, the dynamic memory manager maintains a compact memory layout at run-time so that fragmented data can be merged to reduce the required number of cross-device data transfers, which further reduces the kernel launch overhead induced by the computing context switch.

5.2 Distributed and Heterogeneous Cache

As discussed in Section 2, the web-scale recommendation model usually comprises a huge sparse network, where the sparse parameter access in GPU-based online serving can induce highly irregular memory access and fragmented cross-device data transfer. In UMM, we further optimize the memory access of large-scale sparse parameters from both local and global perspectives.

5.2.1 Local GPU cache. From the local perspective, we construct an on-device GPU cache to reduce fragmented and frequent cross-device parameter access and transfer. It has been proven that the sparse parameter access follows a long tail distribution, where over 80% lookup are for less than 1% parameters in the DNN [21]. To this end, we devise a hash map based cache component in the global memory of GPU to persist a small fraction of hot parameters. When a new sparse parameter is retrieved and transferred to GPU, the on-device cache inserts the new parameter to the corresponding bucket based on Hopscotch hashing [15]. If there is a collision, the GPU cache will re-order parameters in the bucket based on the parameter access frequency, which is associated with each parameter during model construction. Note the parameter frequency is time-sensitive and decays over time. The GPU cache periodically removes infrequent parameters to include more hot parameters. In practice, the GPU cache achieves approximately 70% hit ratio by persisting top 0.01% hot parameters, and therefore significantly reducing the cross-device parameter access and transfer overhead.

5.2.2 Distributed cache proxy. In practice, the sparse network learned for online recommendation is also too large to fit into host memory and duplicate in every node. From the global perspective, we further construct a distributed parameter cache component to decouple the sparse parameter lookup with the online inference pipeline. Specifically, we construct a distributed cache for the sparse network in a separate memory-intensive node aside from the GPU-equipped cluster. Once an inference task arrives, SEHP sends a sparse parameter lookup request to the distributed cache proxy instead of retrieving the sparse parameter locally. In such a case, the sparse parameter lookup operation is transformed to a specific HSP in the SEHP, which can be designed and optimized separately. By leveraging the distributed cache component, the sparse parameter lookup operation can be executed asynchronously to hide the parameter access latency, and the system achieves better scalability.

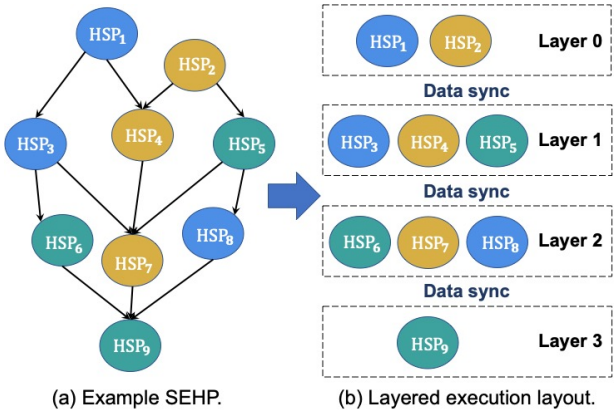


Figure 5: Example of layered execution layout generation.

6 AUTOMATIC EXECUTION OPTIMIZER

This section introduces the *Automatic execution optimizer* (AEO), which optimizes the serving performance from two aspects: (1) offline layered execution planning, and (2) online dynamic batching.

6.1 Offline Layered Execution Planning

Inference latency and throughput are two major concerns of online serving systems, where the first one is critical to user experience [22, 23] and the second one directly influences the budget expense and company profit. In this work, we formulate the performance optimization of SEHP execution as a latency-bounded throughput maximization (LBTM) problem [12]. The offline optimization solves the LBTM problem by searching the execution organization and placement plan of the SEHP, including (1) the layered execution plan generation to reduce the GPU kernel launch cost, and (2) the layer-based heterogeneous resource allocation to improve the global resource utilization.

6.1.1 Layered execution plan generation. Despite the UMM providing highly optimized on-device memory management to reduce kernel launch for resource allocation, the frequent kernel launch induced by fragmented HSP execution is still one major burden for GPU-based online serving. Take the recommendation service used for Baidu news feed again as an example. The online serving system is required to process over 400 HSPs in every inference task, where each HSP execution requires an individual kernel launch. In fact, without any optimization, the overall execution time of SEHP on GPUs can be even longer than on CPU. To this end, we propose the layered SEHP execution planning further to reduce the kernel launch overhead on GPU.

Modern GPU follows the Single-Instruction-Multiple-Thread (SIMT) paradigm [13], where multiple processors can be assigned to handle different instructions in parallel. Such architecture provides us an opportunity to collectively initialize multiple independent HSPs in the SEHP and then simultaneously execute them in parallel. By encapsulating multiple HSPs into one kernel, the

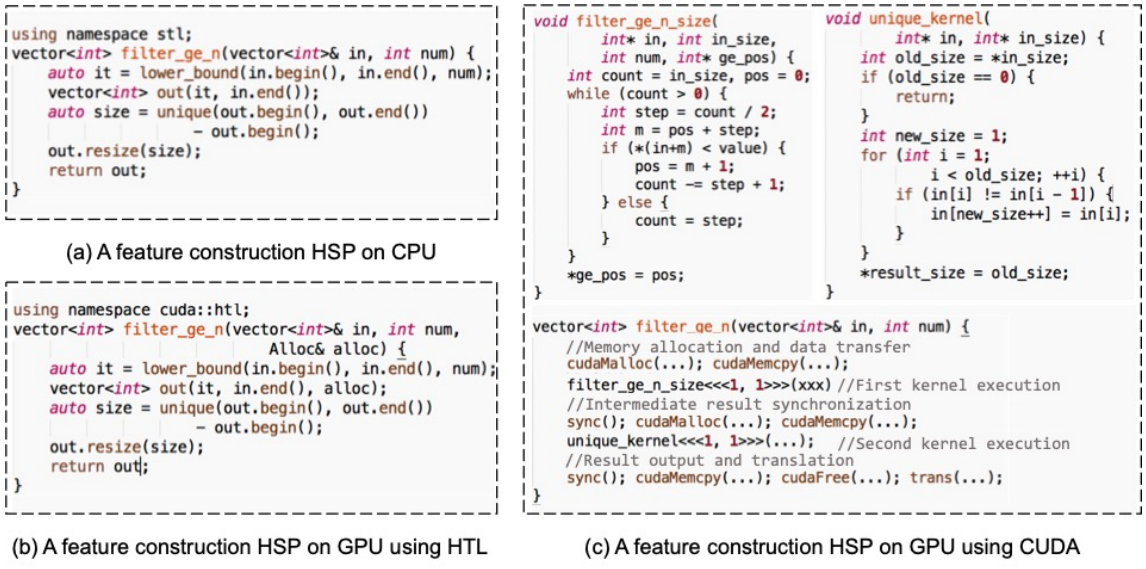


Figure 6: An illustrative example of HTL.

launch overhead can be paid in once to reduce the overall online inference latency. Specifically, given a SEHP, we first apply the topological sort to decompose the directed acyclic graph to linear ordered HSP subsets. As all HSPs in the same subsets are independent to each other, we can safely wrap them into a shared kernel for collective launch and parallel execution without conflict. Take the illustrative SHEP in Figure 5 (a) for example, by applying the *TopologicalDecompose*() function, we derive four HSP subsets $\{HSP_1, HSP_2\}$, $\{HSP_3, HSP_4, HSP_5\}$, $\{HSP_6, HSP_7, HSP_8\}$, $\{HSP_39\}$. Each HSP subset is packed into a shared kernel, *i.e.*, dashed rectangle in Figure 5 (b). HSPs in the same layer can be executed in parallel. By leveraging the layered execution plan, the required number of kernel launch is reduced from 9 to 4. Please refer Appendix B for detailed algorithm description. Note that each layer can be run on CPU or GPU to better utilize the heterogeneous resource. Assuming there are n HSPs and m edges in the SEHP, the time complexity of layered plan generation is $\log(n+m)$. In practice, the layered execution plan can reduce the required number of kernel launch to 1/80, significantly reducing the serving overhead.

6.1.2 Layer based heterogeneous resource allocation. Based on the generated layered execution topology, we further optimize the resource allocation plan to maximize the resource utilization of the heterogeneous cluster. On the one hand, the resource allocation for the whole system (*e.g.*, cache ratio) and for each execution layer (*e.g.*, HSP batch size, memory page size) is vital for the overall system performance. On the other hand, the placement (*e.g.*, CPU or GPU) of different execution layers may also result in diverged performance, depending on the characteristics (*e.g.*, compute-intensive or memory-intensive) of packed HSPs. Formally, AEO searches the optimal source allocation plan to address the LBTM problem by minimizing the following objective function,

$$\begin{aligned} O = \underset{\Theta, theta_i \in \Gamma, p_i \in P}{\operatorname{argmin}} & \sum_{i=1}^{N} \mathcal{F}_i^R(\Theta, \theta_i) + \sum_{i=0}^{N+1} \mathcal{F}_{i,i+1}^C(p_i, p_{i+1}), \\ s.t.\ & \mathcal{F}_i^L(\Theta, \theta_i) \le \mathcal{F}_i^L(\bar{\Theta}, \bar{\theta}_i),\ \forall 1 \le i \le N, \end{aligned} \tag{1}$$

where N refers to the number of layers in the execution plan, Θ denotes the system level parameters, $\theta_i \in \Gamma$ is the set of possible system parameters in layer $1 \le i \le N$, $p_i \in P$ refers to the placement of each execution layer, *e.g.*, CPU and GPU. $\mathcal{F}_i^R(\cdot,\cdot)$ is the estimation function of resource consumption of the i-th execution layer based on parameter θ_i. Moreover, $\mathcal{F}_{i,i+1}^C(\cdot,\cdot)$ is the pair-wise communication overhead estimation function of two consecutive execution layers based on their placement p_i and p_{i+1}. $\mathcal{F}_i^L(\cdot,\cdot)$ is the latency estimation function for layer i, $\bar{\Theta}$ and $\bar{\theta}_i$ are parameters based on expert knowledge. Note $\mathcal{F}_i^R(\cdot,\cdot)$, $\mathcal{F}_{i,i+1}^C(\cdot,\cdot)$, and $\mathcal{F}_i^L(\cdot,\cdot)$ are all derived based on historical parameters extracted from real-world logs for better cost approximation. Finally, we extend the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [3] with constraints to search the optimal plan in a heuristic way.

6.2 Online Dynamic Batching

To handle the time-varying recommendation traffic, we further propose the asynchronous dynamic batching strategy to improve the online serving performance. For a SEHP, increasing the batch size can reduce the number of kernel launches and improve the system throughput but increase the averaged serving latency. Therefore, AEO devised an online batch manager to adaptively tune the batch size for each execution layer based on real-time traffic situations. Specifically, when the number of remaining online recommendation requests increases in the event channel, the online batching manager will adaptively increase the batch size to improve the system throughput by sacrificing the average serving latency. In contrast, when the traffic load decrease, the online batching manager reduce the batch size to avoid task waiting to achieve lower latency. Thanks to the dynamic memory manager introduced in Section 5, the batch size of each execution layer can be directly tuned without requiring further resource application. In such a way, we can tune the batch size at run-time to trade-off the system throughput and latency to improve online serving performance.

Table 1: Summary of five online serving services.

	Model size	# of features	Traffic load
Service A	611 GB	1.03×10^9	$1.87 \times 10^7/s$
Service B	611 GB	1.03×10^9	$5.2 \times 10^6/s$
Service C	795 GB	1×10^9	$5.2 \times 10^6/s$
Service D	471 GB	5.57×10^9	$7.2 \times 10^6/s$
Service E	887 GB	1.31×10^9	$1.6 \times 10^7/s$

7 HETEROGENEOUS TEMPLATE LIBRARY

This section describes the *Heterogeneous Template Library* (HTL), a standard library for GPU-accelerated online serving development.

One key design objective of HTL is to provide an easy-to-use interface and be compatible with a vast amount of existing CPU-nested online recommendation services. Although the SEHP and UMM provide scalable and efficient online serving capability, implementing task-specific online serving pipeline for GPU-equipped infrastructure is a non-trivial task. To fully exploit the parallelism of heterogeneous processors, it does not only require the algorithm engineer and data scientist to have a deep understanding of the system architecture but also familiar with GPU programming (*e.g.*, CUDA [1]). Therefore, HTL encapsulates a set of commonly used online serving system functionalities, such as $MemoryAllocator()$, $Container()$, $Iterator()$, $DataMapping()$, $ThreadSync()$, to hidden low-level parallel execution and resource management details for developers. HTL also provides various mutating or non-mutating stage operators (*e.g.*, searching, sorting, aggregation algorithms) that can run on both CPU and GPU. Figure 6 showcases example code snippets of a staged operator. Specifically, Figure 6 (a) depicts the implementation of "deduplicate elements greater than a threshold" on CPU. Figure 6 (b) and Figure 6 (c) respectively depict the corresponding implementation of the same function with and without using HTL on GPU. As can be seen, the developers can implement the same function on GPU in a consistent way as on CPU by using HTL, which is much more convenient than directly implementing HSPs from scratch. Benefiting from HTL, the engineer and data scientist can easily customize HSPs for GPU as easily as on CPUs and migrate existing CPU-based online recommendation services to GPU clusters with the least labor effort.

8 EXPERIMENT

8.1 Experimental Setup

We evaluate Lion by using five real-world deployed DNN based online recommendation systems. Table 1 reports the model and service statistics, including the DNN model size, the number of features, and the averaged traffic load per second. In particular, *Service A* and *Service B* are respectively multi-objective re-ranking and sparse user interaction modeling service used for Baidu news feed. *Service C* is used for long-term user engagement optimization. *Service D* and *Service E* are video-oriented and multi-modal recommendation service, respectively.

In the evaluation, we compare Lion with the pure CPU-based online serving system (JiZhi) [21], which is previously used at Baidu. We compare the system performance by using latency, throughput, and the resource consumption. Specifically, the latency measures the end-to-end online inference time per candidate in millisecond (ms), the throughput is computed as the number of processed user-item pair inference tasks per second on single instance. The resource consumption is defined as the normalized unit price for serving the recommendation system. The resource consumption includes both hardware costs (*i.e.*, GPU, CPU, memory expenses), as well as operational and energy costs (*i.e.*, maintenance fee and regular charge for electricity). Note that all services are deployed on the cloud, and the resource allocation of each service after offline optimization varies, which we report in Appendix D.

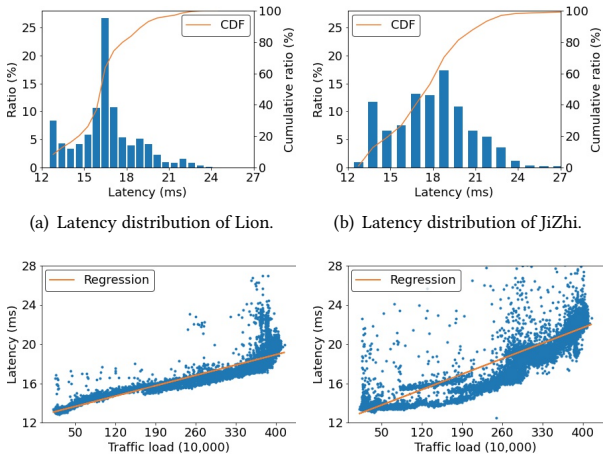
Table 2: Overall performance comparison.

		Latency	Throughput	Resource consumption
Service A	JiZhi	22 ms	10.91×10^3	1,714
	Lion	19 ms	43.8×10^3	612
Service B	JiZhi	18 ms	7.74×10^3	672
	Lion	21 ms	46.02×10^3	244
Service C	JiZhi	18 ms	14.17×10^3	367
	Lion	18 ms	70.9×10^3	122
Service D	JiZhi	49 ms	7.76×10^3	928
	Lion	31 ms	71.29×10^3	309
Service E	JiZhi	23 ms	4.29×10^3	3,733
	Lion	19 ms	56.14×10^3	1,697

8.2 Evaluation Results

8.2.1 Overall performance. Table 2 reports the overall performance of Lion compared with the CPU-based online serving system on five different recommendation services with respect to three metrics. In summary, Lion significantly outperforms the legacy system on five services in terms of throughput and resource consumption with similar latency performance. Specifically, Lion achieves (487.53%, 437.49%, 370.5%, 818.69%, 1208.62%) throughput improvement on five services compared with JIZHI. Note the latency can be further reduced by sacrificing the system throughput. In production, we choose to maximize the system throughput under a latency constraint, which explains the fact that the latency of Lion on *ServiceB* is slightly higher than JiZhi. One important fact is that the throughput improvement of Lion is partially induced by additional resource allocation (*i.e.*, more CPU cores and GPU as reported in Appendix D) after offline optimization. Therefore, we further report the overall system resource consumption to demonstrate the cost-effectiveness of Lion. In particular, Lion outperforms JIZHI by using only (35.71%, 33.34%, 33.24%, 33.3%, 45.46%) resources. More importantly, the resource gain will continue to increase as time goes on, because of the sustainable operational expense reduction and the future traffic load growth of each recommendation service.

8.2.2 Effectiveness of SEHP. Figure 7 reports the detailed latency distribution of Lion and JiZhi. Due to space limitations, here we report the results on *ServiceA*. The results on other recommendation services are similar. Specifically, Figure 7(a) and Figure 7(b) depict the latency distribution of the GPU-equipped serving system (Lion) and the CPU-based online serving system (JiZhi). Compared with the CPU-based online serving system, the latency distribution of Lion is more concentrated. As can be seen, over 25% recommendation requests are responded in 15 ms, which is significantly higher than the peak of CPU-based online serving system that 17% requests responded in 19 ms. Such concentrated latency distribution is more



(a) Latency distribution of Lion. (b) Latency distribution of JiZhi.

(c) Latency v.s. traffic of Lion. (d) Latency v.s. traffic of JiZhi.

Figure 7: Detailed latency distribution of Lion and JiZhi.

stable and can provide more consistent serving quality. Figure 7(c) and Figure 7(d) further illustrate the latency variation of Lion and JiZhi under different traffic conditions. As can be seen, with the number of traffic load increase, the averaged latency of Lion increases much slower than JiZhi. Besides, the latency distribution of Lion fall in a small margin compared to JiZhi, further validating the serving stability of Lion. One interesting observation is that the latency of Lion increases dramatically when the traffic load approach 4 million. This is because the performance of GPU decreases significantly when overload. Such observation also indicates the future optimization direction of Lion, which we left as future work.

8.2.3 Effectiveness of the cache system. Figure 8 further reports the one-week performance of the distributed and heterogeneous cache system on *ServiceA*. Specifically, we report the time-varying hit ratio of the GPU cache and the distributed cache along with the corresponding traffic loads. As can be seen, the GPU cache successfully hit 67.56% to 71.53% sparse parameters and the distributed cache successfully hit 57.65% to 66.25% sparse parameters. Moreover, we observe the hit ratio of the cache system is regularly varying in a small range with the periodical traffic load variation. For example, the cache achieves the highest hit ratio under the bottom traffic load at midnight. This makes sense as fewer sparse parameters are accessed in this duration. Besides, the hit ratio significantly decreases when the traffic load quickly increases in the morning (6:00 a.m. to 12 p.m.) because the growth of traffic load induces more severe irregular sparse parameter access. In summary, the cache system successfully preserves top parameters with reasonable performance variation, and significantly reduce the irregular parameter access and cross-device data transfer overhead.

8.2.4 Effectiveness of the AEO. We further justify the effectiveness of AEO, including offline planning and online dynamic batching. Figure 9 depicts the performance variation of Lion with and without offline optimization on *ServiceA*. As can be seen, offline optimization consistently reduces the recommendation latency in a whole week under different traffic loads. It is worth mentioning that with offline optimization, Lion achieves higher performance gain at the daytime under a higher traffic load. By fusing independent HSPs

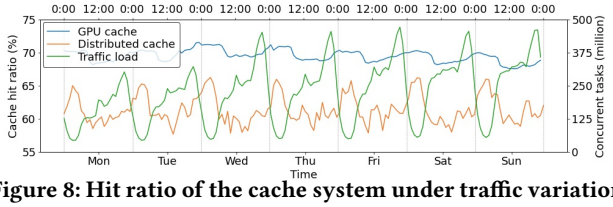


Figure 8: Hit ratio of the cache system under traffic variation.

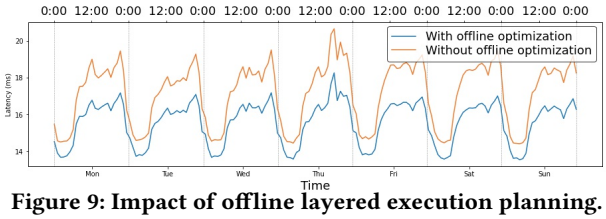


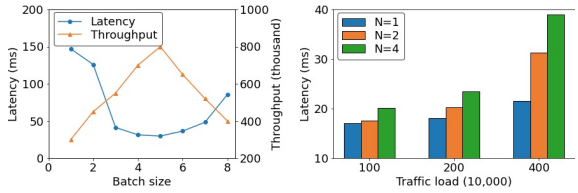
Figure 9: Impact of offline layered execution planning.

into shared execution kernels and assigning proper resources, the serving system achieves up to 13.2% latency reduction at traffic peak hour. This is important as the latency reduction at traffic peak hours directly optimizes the system scalability and delivers a better user experience. Figure 10(a) shows the latency and throughput variation under different batch size setting on a single node. When increase the batch size, the system latency first decrease then increase, whereas the system throughput first increase then decrease. This is because properly batching the inference tasks can better exploit the parallelism of heterogeneous processors. However, too large batch size will induce inference task wait for these earlier arrived recommendation requests and block the result until all tasks in the same batch are finished, therefore inducing extra idle of CPUs and GPUs. In practice, we prefer to turn down the batch size under lower traffic load to avoid unnecessary response delay. Such results illustrate the usefulness of dynamic batching to trade-off the latency and system throughput under different traffic loads. Overall, both offline planning and online dynamic batching improve the system's scalability and robustness under time-varying traffic loads.

8.2.5 Effectiveness of Lion for multi-tenant serving. Finally, we investigate the effectiveness of Lion for multi-tenant serving. As introduced in Section 2, the recommendation service may involve multiple DNNs, and there are diverse recommendation services co-locate in the data center. Serving multiple recommendation systems on the same cluster can further improve resource utilization. Figure 10(b) reports the performance of Lion on serving multiple online recommendation systems on the same cluster, where N denote the number of co-locating recommendation services. As can be seen, increasing the number of concurrent serving recommendation services increases the average latency under different traffic loads. This is because co-locating multiple services on the same cluster will aggravate the irregular data access, resulting in fragmented memory reading and cache performance degradation. To reduce the service interference and guarantee service utility, we restrict $N \leq 2$ in practice.

9 CONCLUSION

In this work, we presented Lion, a GPU-accelerated online serving system for the efficient and cost-effective web-scale recommendation. Specifically, we proposed the staged event-driven heterogeneous pipeline, unified memory manager, and automatic execution



(a) Dynamic batching. (b) Multi-tenant serving.

Figure 10: Impact of online tuning and multi-tenant serving.

optimizer to move online serving execution from pure CPU clusters to GPU-equipped clusters. Besides, we also provide a heterogeneous template library to ease the development and migration of the online serving pipeline for diverse recommendation scenarios. Extensive experiments on real-world applications demonstrate the superiority of Lion compared with the previous pure CPU-based online serving system. Since early 2020, Lion has been deployed in the production environment at Baidu, serving over twenty web-scale online recommendation services and has helped the company save millions U.S. dollars resource consumption every year. In the future, we will extend Lion to support more heterogeneous processors and web-scale recommendation scenarios.

ACKNOWLEDGMENTS

Hao Liu is supported by the National Natural Science Foundation of China under Grant No. 62102110 and Foshan HKUST Projects FSUST21-FYTRI02A.

REFERENCES

[1] 2021. NVIDIA CUDA Toolkit v11.6.0. https://docs.nvidia.com/cuda/
[2] Hossein Amiri and Asadollah Shahbahrami. 2020. SIMD programming using Intel vector extensions. *J. Parallel and Distrib. Comput.* 135 (2020), 83–100.
[3] 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.
[4] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. 2020. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 499–514.
[5] Stephen Cass. 2019. Taking AI to the edge: Google's TPU now comes in a maker-friendly package. *IEEE Spectrum* 56, 5 (2019), 16–17.
[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.
[7] 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.
[8] 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).
[9] 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.
[10] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 982–995.
[11] Udit Gupta, Samuel Hsia, Jeff Zhang, Mark Wilkening, Javin Pombra, Hsien-Hsin Sean Lee, Gu-Yeon Wei, Carole-Jean Wu, and David Brooks. 2021. RecPipe: Co-designing Models and Hardware to Jointly Optimize Recommendation Quality and Performance. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 870–884.
[12] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. 2020. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 488–501.
[13] Axel Habermaier and Alexander Knapp. 2012. On the correctness of the SIMT execution model of GPUs. In *European Symposium on Programming*. 316–335.
[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] Maurice Herlihy, Nir Shavit, and Moran Tzafrir. 2008. Hopscotch hashing. In *International Symposium on Distributed Computing*. 350–364.
[16] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 463–479.
[17] Shoaib Kamil, Cy Chan, Leonid Oliker, John Shalf, and Samuel Williams. 2010. An auto-tuning framework for parallel multicore stencil computations. In *2010 IEEE international symposium on parallel & distributed processing (IPDPS)*. 1–12.
[18] Sagar Karandikar, Chris Leary, Chris Kennelly, Jerry Zhao, Dinesh Parimi, Borivoje Nikolic, Krste Asanovic, and Parthasarathy Ranganathan. 2021. A Hardware Accelerator for Protocol Buffers. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 462–478.
[19] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S Lee, et al. 2020. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 790–803.
[20] 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 *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 583–598.
[21] Hao Liu, Qian Gao, Jiang Li, Xiaochao Liao, Hao Xiong, Guangxing Chen, Wenlin Wang, Guobao Yang, Zhiwei Zha, Daxiang Dong, et al. 2021. JIZHI: A Fast and Cost-Effective Model-As-A-Service System for Web-Scale Online Inference at Baidu. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3289–3298.
[22] Hao Liu, Ying Li, Yanjie Fu, Huaibo Mei, and Hui Xiong. 2022. Polestar++: An Intelligent Routing Engine for National-Wide Public Transportation. *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–1.
[23] Hao Liu, Yongxin Tong, Jindong Han, Panpan Zhang, Xinjiang Lu, and Hui Xiong. 2022. Incorporating Multi-Source Urban Data for Personalized and Context-Aware Multi-Modal Transportation Recommendation. *IEEE Transactions on Knowledge and Data Engineering* 34, 2 (2022), 723–735.
[24] Ignacio Sanudo Olmedo, Nicola Capodieci, Jorge Luis Martinez, Andrea Marongiu, and Marko Bertogna. 2020. Dissecting the CUDA scheduling hierarchy: a Performance and Predictability Perspective. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 213–225.
[25] Xulong Tang, Ashutosh Pattnaik, Huaipan Jiang, Onur Kayiran, Adwait Jog, Sreepathi Pai, Mohamed Ibrahim, Mahmut T Kandemir, and Chita R Das. 2017. Controlled kernel launch for dynamic parallelism in GPUs. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 649–660.
[26] Han Vanholder. 2016. Efficient inference with tensorrt.
[27] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. 2018. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience* 2018 (2018).
[28] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. 2016. DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 3 (2016), 513–517.
[29] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. 392–405.
[30] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine* 13, 3 (2018), 55–75.
[31] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. 2017. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *2017 USENIX Annual Technical Conference (ATC)*. 181–193.
[32] 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.
[33] 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.
[34] 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.

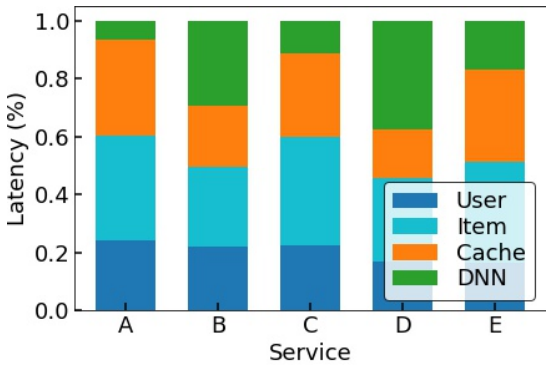


Figure 11: Online serving time breakdown of five deployed online recommendation service.

A ONLINE-SERVING TIME ANALYSIS

Figure 11 depicts the serving time breakdown of five deployed web-scale online recommendation services we evaluated in Section 8. Overall, the DNN feed-forward computation comprises 6.81% to 37.35% of the online inference cycles. The access of sparse parameters (Cache) takes 16.94% to 32.98% online inference cycles. The construction of user and item related features takes the rest of the time. The above results demonstrate the online serving is a complicated pipeline and requires end-to-end optimization to achieve efficiency and cost-effectiveness.

B DETAILED OFFLINE LAYERED EXECUTION PLANNING ALGORITHM

Algorithm 1 describes the layered execution plan generation procedure. We devise a topology sort based strategy to decompose SEHP to multiple dependent subsets, where each HSPs in a subset are dependency-free and can be executed in parallel.

Algorithm 1: Layered execution plan generation

Input: A staged event-driven heterogeneous graph $G = (V, E)$.
Output: $\mathcal{S}$: an ordered set of HSP subsets.

```
Set S ← ∅, counter i ← 0;
while G is not empty do
    S_i ← HSPs in G without predecessor;
    Remove HSPs in S_i from G;
    Remove edges originate from HSPs in S_i;
    S ← S ∪ S_i;
    i ← i + 1;
return S;
```

C IMPLEMENTATION AND DEPLOYMENT

Since early 2020, Lion has been deployed at Baidu as the core online serving system, supporting over twenty DNN based online recommendation services.

Table 3: Per-instance resource allocation of 5 services.

		# of CPU cores	# of GPU	Memory
Service A	JiZhi	4.2	-	20 GB
	Lion	5.6	1 T4	20 GB
Service B	JiZhi	4.2	-	20 GB
	Lion	5.6	1 T4	20 GB
Service C	JiZhi	4.2	-	20 GB
	Lion	5.6	1 T4	20 GB
Service D	JiZhi	5.6	-	20 GB
	Lion	8.4	1 A10	20 GB
Service E	JiZhi	4.2	-	20 GB
	Lion	8.4	1 A10	20 GB

Implementation details. The core online serving system is implemented by C++. Specifically, we leverage the warp-level primitives and scheduling mechanism (*e.g.*, synchronization) [24] to further improve the GPU resource utilization. Moreover, for the HSPs, we optimize operators by exploiting the hierarchical memory architecture of GPU. For instance, the aggregation operation is first performed on each local memory as local variables and then merged in global memory. In the UMM, we have also implemented a series of translators to map data structures on GPU and GPU, *e.g.*, we have developed a dedicated on-GPU data structure for Protocol buffers [18]. Finally, to guarantee the accuracy of online recommendations, the GPU cache is updated with a 5 minutes time window.

Online deployment. Once the SEHP is compiled, the service can be deployed for the online recommendation based on a pre-generated execution plan. At run-time, the recommendation requests are issued from different geographical locations. To minimize the network latency, we duplicate the online serving system in multiple data centers distributed in mainland China and serve recommendation requests based on geographical locality. For DNN storage, we manage the sparse network and dense network separately. The sparse network is stored as embedding tables and managed by in-memory database on dedicated storage node, whereas the dense network is directly duplicated in memory on each node.

D RESOURCE ASSIGNMENT FOR EACH SERVICE

Table 3 reports the per-instance resource allocation of each service in Lion and JiZhi. Note that all services are deployed on the cloud, where CPUs in each physical node is shared virtually by multiple instances. In particular, each node is equipped with four Intel Xeon Gold 5117 CPUs. In Lion, all GPUs are NVIDIA T4 or NVIDIA A10, which are optimized for inference tasks.