Walle: An End-to-End, General-Purpose, and Large-Scale Production System for Device-Cloud Collaborative Machine Learning

Chengfei Lv (ZJU & Alibaba); Chaoyue Niu (SJTU & Alibaba); Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, Tao Huang, Hui Shu, Jinde Song, Bin Zou, Peng Lan, Guohuan Xu (Alibaba); Fei Wu (ZJU); Shaojie Tang (UT Dallas); Fan Wu, Guihai Chen (SJTU)
1

Background & Motivation
Proliferation of Mobile Intelligent Services

Livestreaming

Speech Recognition

Recommendation
Bottlenecks of Cloud-Based ML Framework

Cloud takes all the load!

- **High Latency**
  - Device-cloud interaction
  - Process requests from millions or billions of users

- **High Cost & Heavy Load**
  - Communication & Storage
  - Process data with complex ML algorithms

- **High Privacy Risk**
  - Upload sensitive raw data
  - Store and process raw data on the cloud

Mobile devices function only as user interfaces!
Mobile devices and the cloud jointly accomplish ML tasks.

### Need for Device-Cloud Collaborative ML

**Overcome Cloud-Side Bottlenecks**
- Reduce latency & communication cost
- Mitigate cloud-side load
- Keep private data on local devices

**Natural Device-Side Advantages**
- Close to users
- At data sources
# Our Unique System-Level Consideration

<table>
<thead>
<tr>
<th>Application Layer</th>
<th>Algorithm Layer</th>
<th>System Layer</th>
<th>Hardware Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Analytics (e.g., FilterForward in MLSys’19, Reducto &amp; DDS in SIGCOMM’20), <strong>Text Processing</strong> (e.g., Gboard in MLSys’19), <strong>Recommend</strong> (e.g., DDCL in KDD’21, MPDA in KDD’22)</td>
<td><strong>Device-Cloud Task Splitting Strategy</strong> (e.g., cloud training-device inference, Neurosurgeon in ASPLOS’17, federated learning in AISTATS’17), <strong>Interaction Paradigm</strong> (e.g., single device-cloud, multiple devices-cloud), <strong>Collaboration Mechanism</strong> (e.g., through exchanging data or model)</td>
<td><strong>How to build a general-purpose system that can put device-cloud collaborative ML in large-scale production?</strong></td>
<td><strong>Hardware Layer</strong> (Mobile Devices &amp; Cloud Servers)</td>
</tr>
</tbody>
</table>

Existing work was at the algorithm layer, normally for ML inference or training in a specific application.
Overall Goal & Architecture
Walle – Overall Goal

End-to-End

- Develop, deploy, runtime
- All three phases of ML task
- Both sides of device and cloud

Hundreds of CV, NLP, recommendation tasks in large-scale production

Walle

General-Purpose

Heterogeneous hardware & software of mobile devices & cloud servers
Walle – Overall Architecture

Oriented by ML task

- **Scripts** (e.g., Python codes for three phases of ML task)
- **Resources** (e.g., data, models, dependent libraries)
- **Configurations** (e.g., trigger conditions)

---

**Deployment Platform**

- Task Management
- Task Release & Deployment

---

**Compute Container**

- Standard APIs
- Python Thread-Level VM
- Data & Model Related Libraries
- Tensor Compute Engine
- Backends (Device & Cloud)

**Data Pipeline**

- Device-Cloud Tunnel
- On-Device Stream Processing Framework

---

ML task execution

- **ML task management & deployment**
- **ML task input preparation**
Walle – Compute Container
## Practical Challenges

### Development
- Monthly/Weekly APP update
- vs. Daily ML task iteration

### Testing
- Integration
- Release in Batches
- APP Store Review

### Integration

### Heterogeneous hardware & software
- of mobile devices & cloud servers
  - 300+ device types
  - 60+ brands
  - 200+ OS with different versions

### Diverse CV, NLP, and recommendation tasks
- Image, text, numerical processing methods
- CNN, RNN, Transformer, GAN, DIN

### Resource limitation of a certain mobile APP
- Each mobile APP runs as one process.
- Mobile Taobao
  - 200MB RAM
  - 100MB Android package
  - 220MB iOS package
### Architecture

<table>
<thead>
<tr>
<th>Standard APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python Thread-Level VM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Processing &amp; Model Execution Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNN-Matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tensor Compute Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric Computing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backends</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM v7/v8/v8.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS &amp; Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android/iOS</td>
</tr>
<tr>
<td>CPU/GPU/NPU</td>
</tr>
</tbody>
</table>

**First!**
- Python dynamically-typed
- widely used

**C/C++**
- cross-platform
- high-performance

**Brand New Design!**

**Integrated Design**
- Expose **high performance** of tensor compute engine
- **Reduce** the **workload** of optimizing each library for heterogeneous backends
- Support the **whole cycle** of ML tasks
- **Keep package small**

**Open Source**
- https://github.com/alibaba/MNN
- https://www.mnn.zone/

- 6.8k stars
- 1.4k forks
Tensor Compute Engine – Design Principle

**Manual Operator Optimization**

- Composite Ops (16)
- Control-Flow Ops (2)
- Atomic Ops (61)
- Transform Ops (45)
- Atomic Raster Op (1)
- Hardware Backends (16)

**Graph-Level Runtime Optimization**

- a series of operators: \( op_1 \rightarrow op_2 \rightarrow \ldots \rightarrow op_n \)
- available backends: \( ba_1, ba_2, ba_3 \)

**Backends**

<table>
<thead>
<tr>
<th>Backends</th>
<th>Algorithm</th>
<th>SIMD</th>
<th>Memory</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM (Device)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPU (Device)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>x86 (Server)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CUDA (Server)</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

reduce roughly 46% workload

quickly find min-cost backend
Python Virtual Machine (VM) – Refining CPython

Package Tailoring for APP Need

Functionality Tailoring
- Keep only interpreter for mobile devices

Library & Module Tailoring
- Keep only 36 necessary libraries (e.g., abc, type, re, functools, etc)
- Keep only 32 necessary modules (e.g., zipimport, sys, exceptions, gc, etc)

10MB+ to 1.3MB (ARM64-based iOS)

First in industry to be ported to mobile devices!

Task-Level Multi-Threading

Motivations
- The global interpreter lock (GIL) & Single process of mobile APP → parallel X
- Practical characteristics of ML tasks
  - Concurrent triggering of many tasks
  - Independence across different tasks
  - Sequential execution of different phases in each individual task

How?
- Bind each ML task with a thread
- Do thread isolation

Abandon GIL and support multi-threading!
Walle – Data Pipeline
Bottlenecks of Mainstream Data Pipeline

Cloud-Based Stream Processing

Raw Data of Massive Users

Flink

Features

Transfer 500KB Data in 5min (modified http request)

Offline Tunnel

Basic Events

- page enter
- page scroll
- exposure
- click
- page exit

Process User Data Far Away from Source

- Device-cloud communication for redundant raw data
- Cloud-side computation & storage for aggregate data from billions of users

- Time-Consuming
- Resource-Consuming
- Error-Prone
- Privacy-Sensitive
New Data Pipeline – More Natural & Efficient

Enable each mobile device to process only its user’s behavior data at source
Walle – Deployment Platform
Practical Considerations & Challenges

Frequent experiment & deployment for daily ML task iteration

Massive multi-granularity task deployment requirements

Intermittent device availability

Potential task failure

- Unstable wireless network
- Frequent APP switch & Only one APP on the foreground
- Each mobile APP has one process. Failure of any task leads to APP crash.
- Extensive real-device testing is impractical.

- Hundreds
- Billion
- • APP version
- • Device-side differentiation
- • User-side differentiation
Deployment Strategy
- Uniform Deployment
- Coarse-Grained Grouping
  - Mobile App Version
  - Shared Resources
- Customized Deployment
  - Fine-Grained Grouping
  - Device-Side Information
  - User-Side Information
  - Exclusive Resources

Release
- Test
  - Rollback
  - Exception Statistics
- Beta
  - Gray Release
- Monitor

Real-Time Reach
- CDN
- Push Service
- Pull Service

Device
- Decode
- Query
- Store
- Execute

Cloud-based simulators with compute container

Existing client-side http request for business services
6 Evaluation Results
6.1 Practical Performance in E-Commerce Scenarios
Compute Container in Livestreaming

Cloud-Based Design
- Key bottleneck: **Heavy load** (lots of streamers, long video streams, stringent latency requirement)
- Cover **part** of streamers
- Analyze **part** of video frames

On-Device Recognition with Small Model
- CV Task
- Feature Map
- Highlights
- Matching
- NLP Task
- Feature Map DB
- Item Pool

Device-Cloud Co-Design
- Cloud-side load: **-87%**
- #Covered streamers: **+123%**
- #Daily recognized highlights per unit of cloud cost: **+74%**
- Overall latency per highlight recognition: **< 150ms**

<table>
<thead>
<tr>
<th>Model</th>
<th>Item Detection</th>
<th>Item Recognition</th>
<th>Facial Detection</th>
<th>Voice Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huawei P50 Pro</td>
<td>56.92ms</td>
<td>25.68ms</td>
<td>41.42ms</td>
<td>0.07ms</td>
</tr>
<tr>
<td>iPhone 11</td>
<td>33.71ms</td>
<td>29.74ms</td>
<td>22.58ms</td>
<td>0.01ms</td>
</tr>
</tbody>
</table>

Roughly 12% of highlights recognized with low confidences on mobile devices need to be processed by cloud-based big model.
Data Pipeline in Recommendation

Cloud-Based Data Pipeline
- **Time-Consuming:** 33.73s per IPV feature generation (using Alibaba’s Blink)
- **Resource-Consuming:** 253.25 CU (1 CU denotes 1 CPU core + 4GB memory)
- **Error-Prone:** 0.7% error rate

Walle’s New Data Pipeline
- **Lower On-Device Latency:** 44.16ms per IPV feature generation
- **Lower Communication & Storage Cost**

<table>
<thead>
<tr>
<th></th>
<th>Raw Events</th>
<th>IPV Feature</th>
<th>IPV Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td>21.2KB</td>
<td>1.3KB</td>
<td>128B</td>
</tr>
<tr>
<td><strong>Reduction</strong></td>
<td>/</td>
<td><strong>93.9%</strong></td>
<td><strong>99.4%</strong></td>
</tr>
</tbody>
</table>

- **No** Feature Error

---

**On-Device Stream Processing**
(aggregate a user’s behaviors in the detailed page of an item)

**Item Page-View (IPV) Feature**

**For Use**

RNN Encoding with MNN

**IPV Encoding**

**Raw Event Stream**
- **Page Switch**
  - enter
  - leave
- **Click**
  - view pic
  - add favorite
  - add cart
  - purchase
- **Exposure**
  - item 0
  - item 1
  - item 2
Large-Scale Production Use

- As part of Alibaba’s ML backbone infrastructure
- Put in use since 2017 & already run for roughly \textbf{1,500 days}
- Invoked \textbf{153 billion+} times per day
- Deployed \textbf{1,000+} kinds of ML tasks in total, each with \textbf{7.2} versions on average
- Supporting \textbf{30+} mobile APPs
- Supporting \textbf{300+} kinds of active ML tasks for \textbf{0.3 billion} daily active users with mobile devices

Cover all 7 million online devices in 7min and all the target 22 million devices in 19min
6.2 Extensive Micro-Benchmark Testing Results
MNN vs. TensorFlow (Lite) & PyTorch (Mobile)

MNN outperforms other frameworks in almost all the test cases and is more full-featured on the side of mobile devices.
MNN vs. TVM

- **TVM** (autotuning + compiling) roughly costs thousands of seconds. **MNN’s semi-auto search** for runtime optimization costs roughly hundreds of milliseconds.
- **MNN** can support the industrial scenarios that involve numerous heterogeneous devices and require frequent and quick task iteration, whereas **TVM cannot**.

MNN outperforms TVM due to manual operator optimization.
Python Thread-Level VM vs. CPython with GIL (analyzed over 30 million online ML task executions)

![Performance Improvement Chart]

<table>
<thead>
<tr>
<th>Task Category</th>
<th>Performance Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light-Weight [0, 100) ms</td>
<td>52.11%</td>
</tr>
<tr>
<td>Middle-Weight [100, 500) ms</td>
<td>144.36%</td>
</tr>
<tr>
<td>Heavy-Weight [500, 1200) ms</td>
<td>25.70%</td>
</tr>
</tbody>
</table>

Task-level multi-threading without GIL is the key of performance boosting.

Practical Delay of Real-Time Tunnel with Varying Size of Data Upload (analyzed over 364 million uploads)

![Delay vs. Data Size Graph]

Delay (ms)

Data Size (KB)

Average Delay

Median Delay

Count

0 50 100 150 200 250 300 350 400 450 500

0 1.2 × 10^7 2.4 × 10^7 3.6 × 10^7 4.8 × 10^7 6.0 × 10^7 7.2 × 10^7 8.4 × 10^7 9.6 × 10^7 1.08 × 10^8 1.2 × 10^8 1.6 × 10^8 2.0 × 10^8

Count of Uploads

90% uploads < 3KB, 250ms
0.1% uploads = 30KB, 450ms
Summary
• Design and build the first end-to-end, general-purpose, and large-scale production system, called Walle, for device-cloud collaborative ML.
• Compute container comprises MNN, which introduces geometric computing to sharply reduce the workload of manual operator optimization, and semi-auto search to identify the best backend with runtime optimization; and a Python VM, which abandons GIL and supports task-level multi-threading, and also is the first to be ported to mobile devices.
• Data pipeline introduces on-device stream processing with trie-based concurrent task triggering to enable processing user behavior data at source.
• Deployment platform supports fine-grained task release and deployment to billion-scale devices with strong timeliness and robustness.
• Evaluation in practical e-commerce scenarios and extensive micro-benchmarks have demonstrated the superiority of Walle.
• Walle has been in large-scale production use in Alibaba, while MNN has been open source with a broad impact in the community.
Thanks for listening!
Comments & Questions?