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

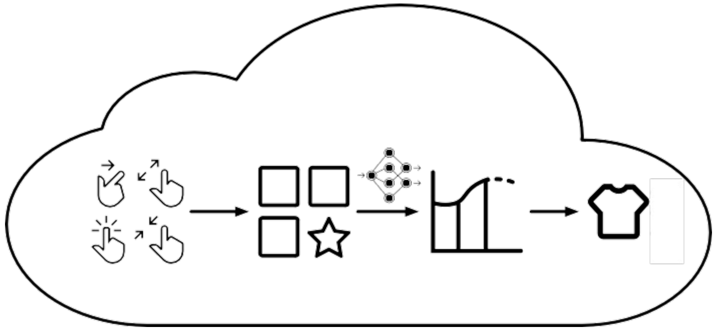
Chengfei Lv (ZJU & Alibaba); Chaoyue Niu (SJTU & Alibaba); Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, Tao Huang, Hui Shu, Jinde Song, Bin Zou, Peng Lan, Guohuan Xu (Alibaba); Fei Wu (ZJU); Shaojie Tang (UT Dallas); Fan Wu, Guihai Chen (SJTU)

1

Background & Motivation

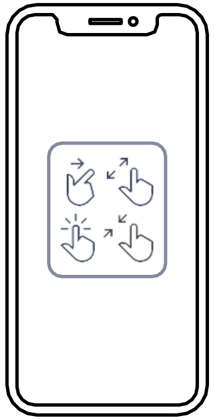
Bottlenecks of Cloud-Based ML Framework

Cloud takes all the load!

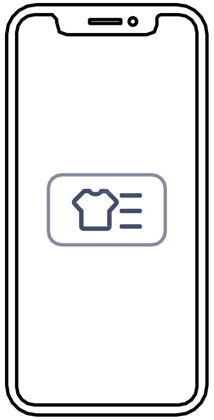


Upload Raw User Data

Return Results



Mobile devices function only as user interfaces!



High Latency

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

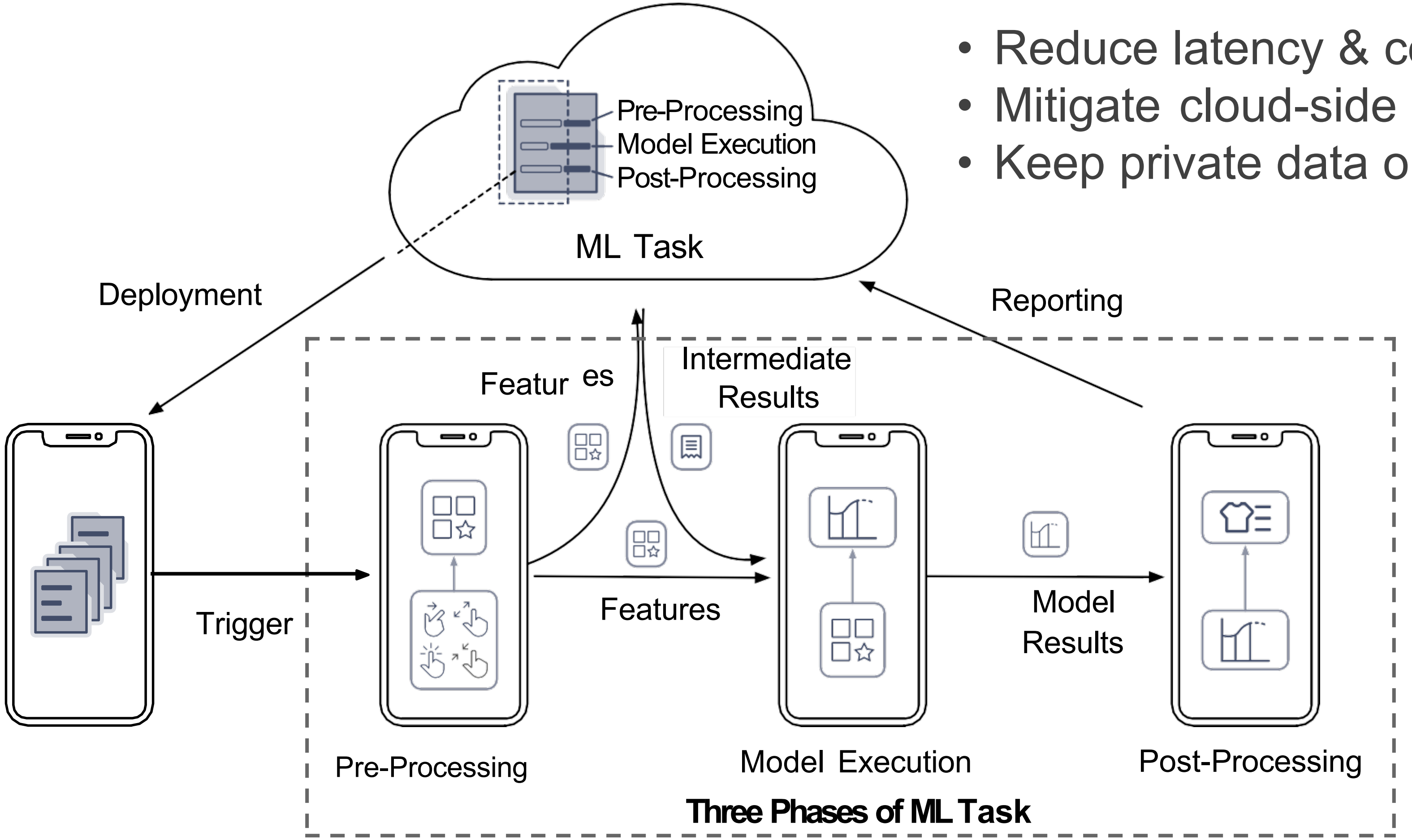
High Cost & Heavy Load

- Communication & Storage
- Process data with complex ML algorithms

High Privacy Risk

- Upload sensitive raw data
- Store and process raw data on the cloud

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

Mobile devices and the cloud jointly accomplish ML tasks.

Our Unique System-Level Consideration

Alibaba Group

Application Layer

Video Analytics (e.g., FilterForward in MLSys'19, Reducto & DDS in SIGCOMM'20), **Text Processing** (e.g., Gboard in MLSys'19), **Recommend** (e.g., DDCL in KDD'21, MPDA in KDD'22)

Existing work was at the algorithm layer, normally for ML inference or training in a specific application.

Algorithm Layer

Device-Cloud Task Splitting Strategy (e.g., cloud training-device inference, Neurosurgeon in ASPLOS'17, federated learning in AISTATS'17), **Interaction Paradigm** (e.g., single device-cloud, multiple devices-cloud), **Collaboration Mechanism** (e.g., through exchanging data or model)

System Layer

How to build a general-purpose system that can put device-cloud collaborative ML in large-scale production?

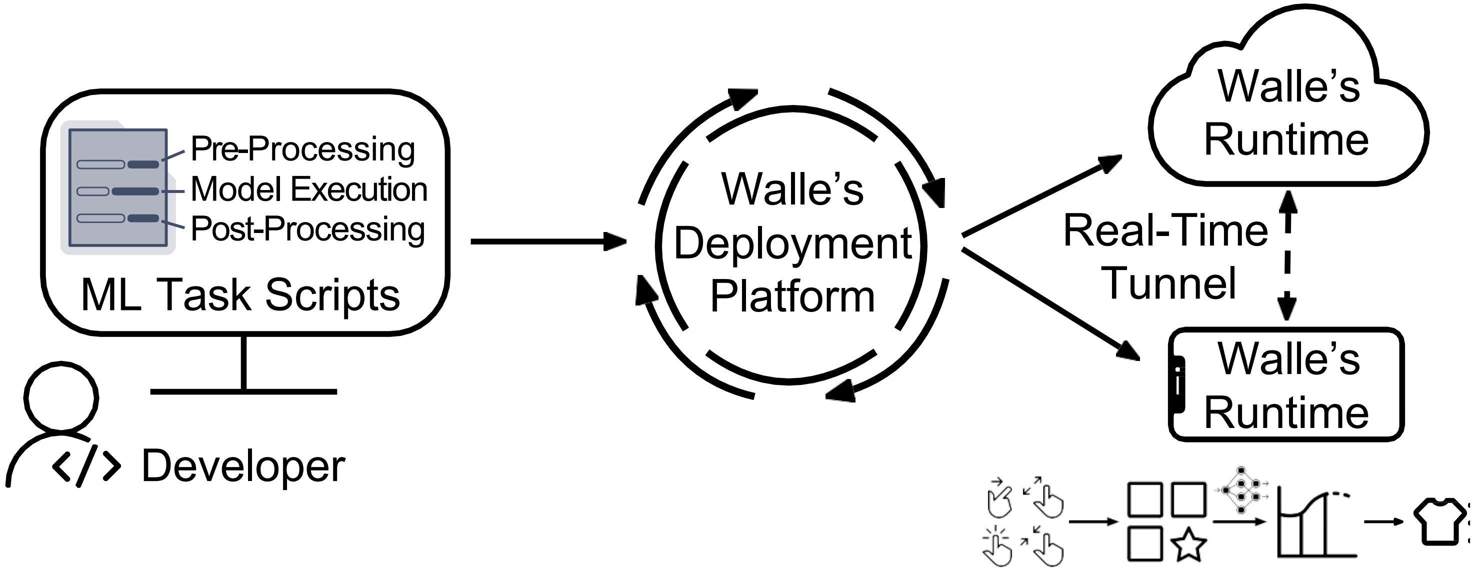
Hardware Layer (Mobile Devices & Cloud Servers)

General System Support

2

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

Pratical Challenges

Execution Challenges

- Long Iteration Cycle: most APPs is updated weekly or monthly, but ML tasks require frequent experiments/deployments.
- Heterogeneous Backends: The cloud servers and mobile devices significantly differ in hardware (e.g., CPU, GPU, NPU, instruction set architecture (ISA), and memory) and OS (e.g., Android, iOS, Windows, and Linux).
- Diverse ML Tasks: Industrial applications involve many kinds of ML tasks, requiring diverse model structures (e.g., CNN, RNN, GAN).
- Limited Device Resources: Each mobile APP has only one process. For Mobile Taobao, the maximum RAM is only 200MB, and the package size cannot exceed 300MB.

Practical Challenges

Input Preparation Challenges

- Atypical User Behavior Data: User’s diverse behaviors in time and page series during interacting with a mobile APP cannot be pre-processed by standard libraries.

- Diverse Trigger Conditions: ML tasks tend to need many features. How to efficiently manage multiple trigger conditions for concurrent task triggering is non-trivial.

Practical Challenges

Deployment Challenges

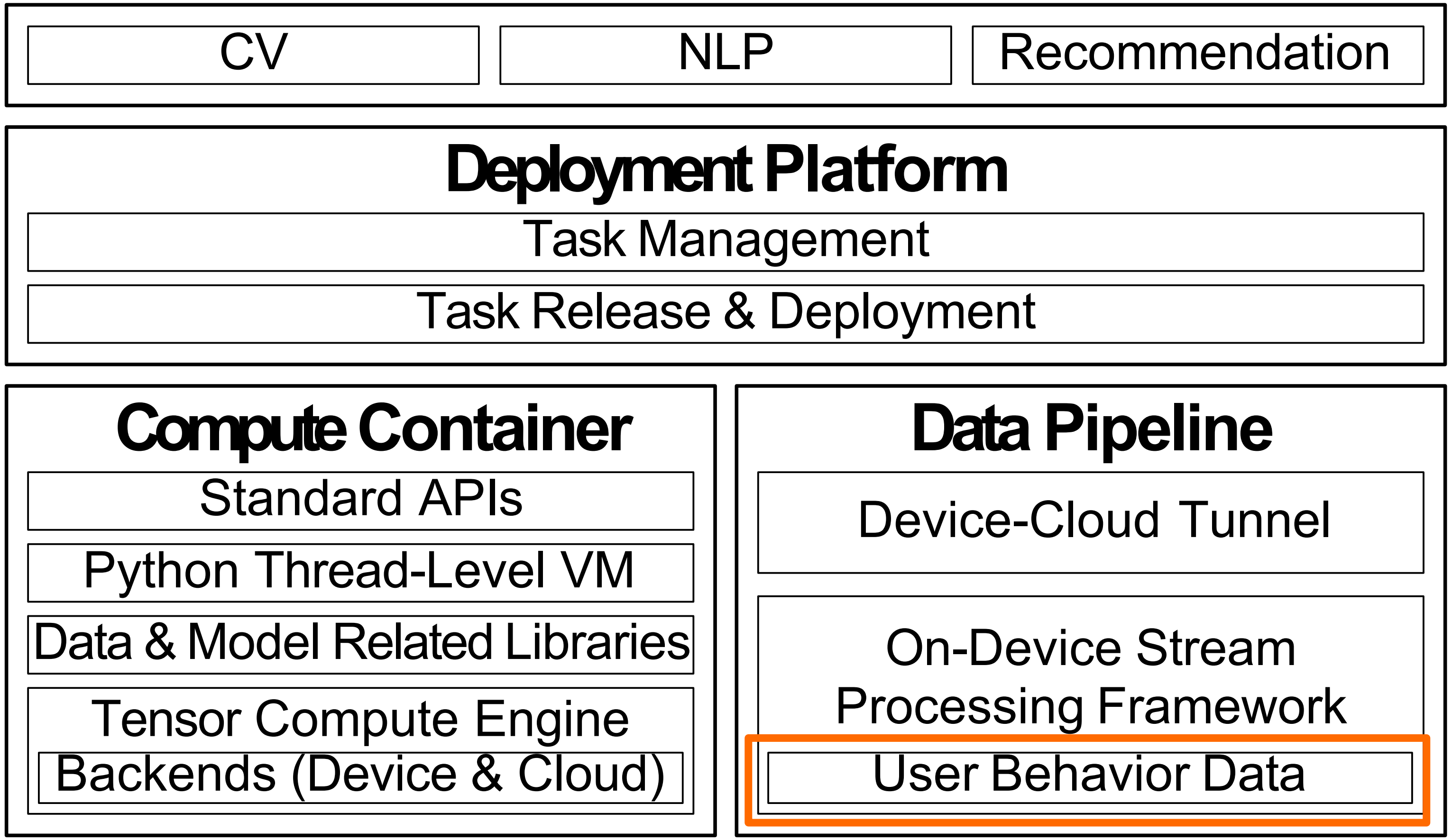
- Massive Task Deployment Requirements
- Intermittent Device Availability: Mobile devices are with unstable wireless connections and allow only one APP to run on the foreground.
- Potential Task Failure: A mobile APP runs as a single process. The failure of any task will lead to the crash of the whole APP, seriously impacting user experience.

Walle – Overall Architecture

Alibaba Group

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)



ML task management & deployment

ML task input preparation

ML task execution

3

Walle – Compute Container

Execution Challenges

Development → Testing → Integration → APP Store Review → Release in Batches

Monthly/Weekly APP update
vs. **Daily ML task iteration**

vs.

CPU
GPU
NPU

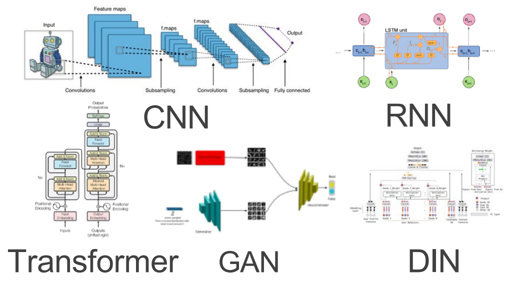
android
iOS
Windows
Linux

vs.

Mobile Taobao

- **300+** device types
- **60+** brands
- **200+** OS with different versions

Heterogeneous hardware & software of mobile devices & cloud servers



Image, text, numerical processing methods

Diverse CV, NLP, and recommendation tasks

Each **mobile APP** runs as **one process**.

Mobile Taobao

- **200MB** RAM
- **100MB** Android package
  220MB iOS package

Resource limitation of a certain mobile APP

Architecture

Python
dynamically-typed
widely used

First!

C/C++
cross-platform
high-performance

Brand New Design!

ML Task Scripts
decouple daily task iteration from weekly/monthly mobile APP update

Standard APIs

Python Thread-Level VM

Data Processing & Model Execution Libraries
MNN-Matrix | MNN-CV | MNN-Inference | MNN-Training

Tensor Compute Engine
Geometric Computing | Semi-Auto Search

Backends
ARM v7/v8/v8.2 | OpenCL/Vulkan/Metal | x86 AVX/AVX-512 | CUDA

OS & Hardware
Android/iOS | Linux/Windows/MacOS/Docker
CPU/GPU/NPU

MNN
Mobile Neural Network

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

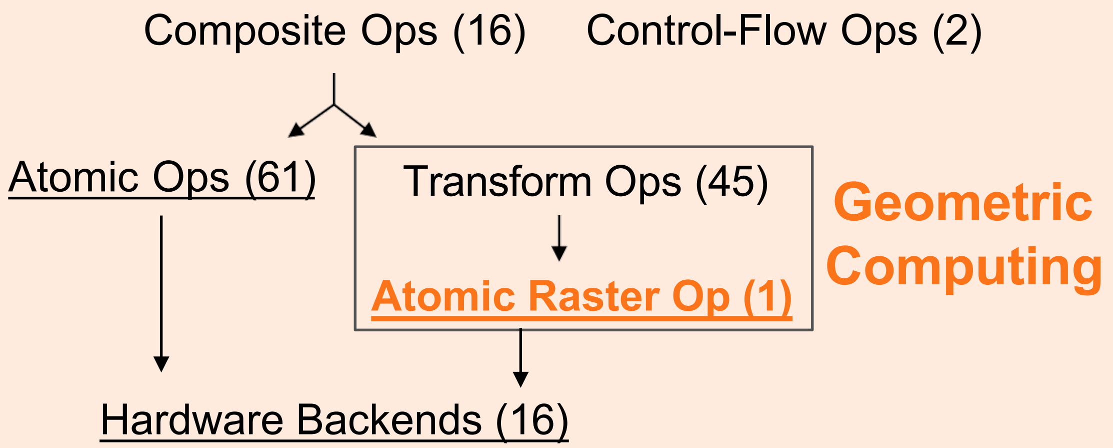
https://github.com/alibaba/MNN
https://www.mnn.zone/

6.8k stars 1.4k forks

Tensor Compute Engine – Design Principle

Alibaba Group

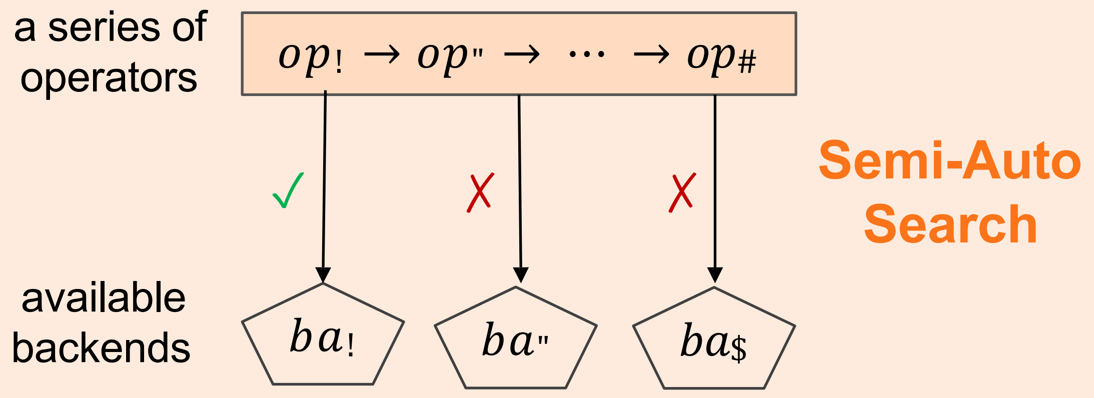
Manual Operator Optimization



Backends	Algorithm	SIMD	Memory	Assembly
ARM (Device)	✓	✓	✓	✓
GPU (Device)	✓	✓	✓	✗
x86 (Server)	✓	✓	✓	✓
CUDA (Server)	✗	✓	✓	✗

reduce roughly 46% workload

Graph-Level Runtime Optimization



Search Strategies	Dynamic Deployment	Light Workload	Manual Experience	OPT
Manual	✓	✗	✓	✗
Auto (TVM)	✗	✓	✗	✓
Semi-Auto	✓	✓	✓	✓

quickly find min-cost backend

Tensor Compute Engine – Geometric Computing

Manual Operator Optimization

Composite Ops (16) Control-Flow Ops (2)

Atomic Ops (61)

Transform Ops (45)

Atomic Raster Op (1)

Geometric Computing

Hardware Backends (16)

$$O((N_{aop} + N_{top} + N_{cop}) \times N_{ba} + N_{fop} = 1954)$$

$$O((N_{aop} + 1) \times N_{ba} + N_{top} + N_{cop} + N_{fop} = 1055)$$

Tensor Compute Engine – Geometric Computing

- The basic functionality of the transform operators is to move an element from a memory address to another memory address, or from geometry, is to transform the coordinate of the element to another coordinate.

- Given a certain transform operator, the formula of coordinate transformation can be determined.

- With the coordinate of an element in the input or output tensor, the original memory address and the memory address after movement can also be determined.

Tensor Compute Engine – Geometric Computing

Use slicing as an example

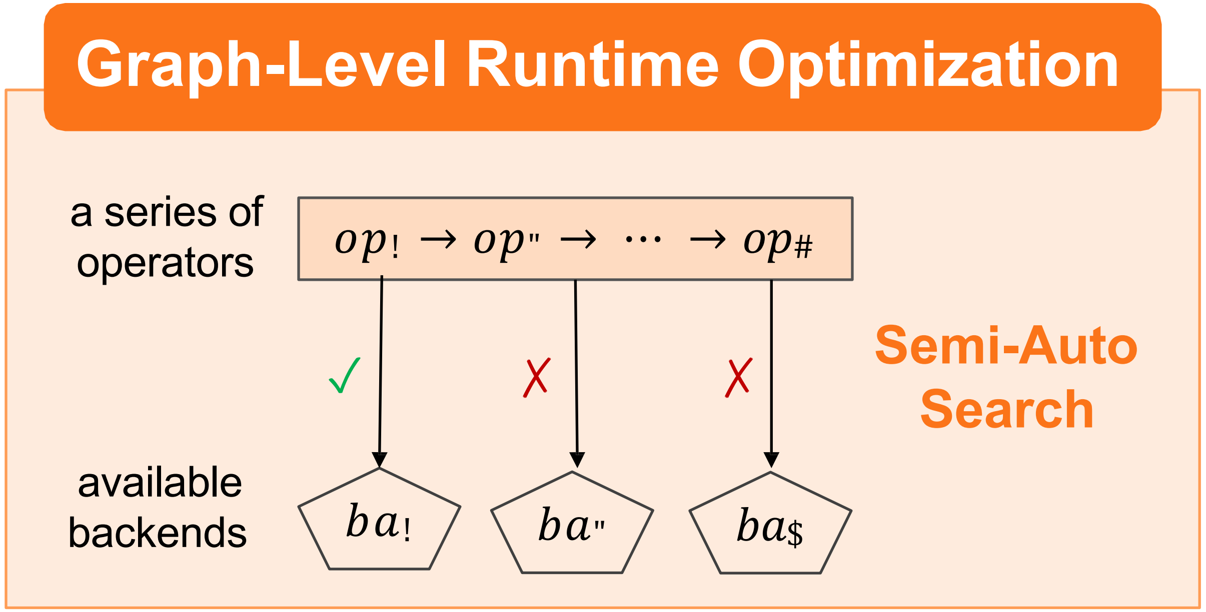
- A is a 2 × 4 matrix, the second row is sliced as B, which is a 1 × 4 matrix.
- For an element $B_{i, j}$, its relative memory identifier is $i \times 4 + j$
- $B_{i, j} = A_{i+1, j}$
- The coordinate of the corresponding element A_{i+1}, j in A is $(i + 1, j)$, and the relative memory identifier is $(i+1)\times 4 + j = 4i + j + 4$
- The raster operator can realize the functionality of slicing by iterating the coordinates(i,j) and moving each $A_{i+1, j}$ to $B_{i, j}$ using their memory addresses

Tensor Compute Engine – Geometric Computing

Atomic Operator Optimization

- The algorithm-level optimization: more efficient algorithms, including Winograd and Strassen algorithms.
- The ISA-level optimization: leverage single instruction multiple data (SIMD), such as ARM Neon and x86 AVX512, for speedup
- The memory-level optimization: reduce the number of read and write as well as improving the contiguity of memory allocation.
- The assembly-based optimization

Tensor Compute Engine – Semi-Auto Search



quickly find min-cost backend

Tensor Compute Engine – Semi-Auto Search

- the series of n operators for execution:

$$op_1 \rightarrow op_2 \rightarrow \ldots \rightarrow op_n$$

- the cost of a backend ba ∈ BA:

$$C_{ba} = \sum_{i=1}^{n} C_{op_i,ba}$$

- The goal of semi- auto search is to find the backend with the minimum cost:

$$\arg\min_{ba \in BA} C_{ba}$$

- the problem is how to compute:

$$C_{op_i,ba} = \min_{alg \in algs(op_i,ba)} \frac{Q_{alg}}{P_{ba}} + S_{alg,ba}$$

Tensor Compute Engine – Semi-Auto Search

- The objective is to minimize the computation or memory cost
- The constraints mainly include the packing size in SIMD, the tile size in matrix multiplication, the block unit in the Winograd algorithm, and the reduction of the elementary calculations using the Strassen algorithm.
- let *A* denote an *a* × *e* matrix, let *B* denote an *e* × *b* matrix, let *t_e, t_b* denote the tile, and let *N_r* denote the number of registers

$$
\begin{aligned}
\min_{t_e, t_b} \quad & \frac{e}{t_e} \times \frac{b}{t_b} \times (a \times t_e + a \times t_b + t_e \times t_b), \\
\text{s.t.} \quad & t_e \times t_b + t_e + t_b \leq N_r,
\end{aligned}
$$

Data and Model Related Libraries

Scientific Computing & Image Processing

- array creation and manipulation routines, binary operations, linear algebra, logic functions, padding arrays, random sampling, mathematical functions, etc
- image filtering, geometric and miscellaneous image transformations, drawing functions, color space conversions, etc

Model Inference & Model Training

Python Virtual Machine (VM) – Refining CPython

Alibaba Group

We choose the official and the most widely-used Python compiler and interpreter, called CPython

Problems of CPython

- The first problem is that the size of the package is large. For example, CPython 2.7.15 contains 500+ scripts in C and 1,600+ libraries

- The second problem is that CPython cannot support multi-threading to improve efficiency. GIL allows only one thread to be processed at one time within a process.

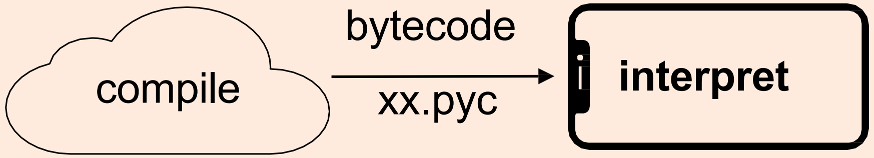
Python Virtual Machine (VM) – Refining CPython

Alibaba Group

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, execeptions, gc, etc)

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

First in industry to be ported to mobile devices!

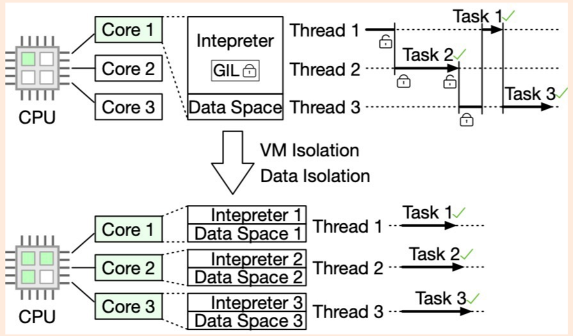
Task-Level Multi-Threading

Motivations

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

How?

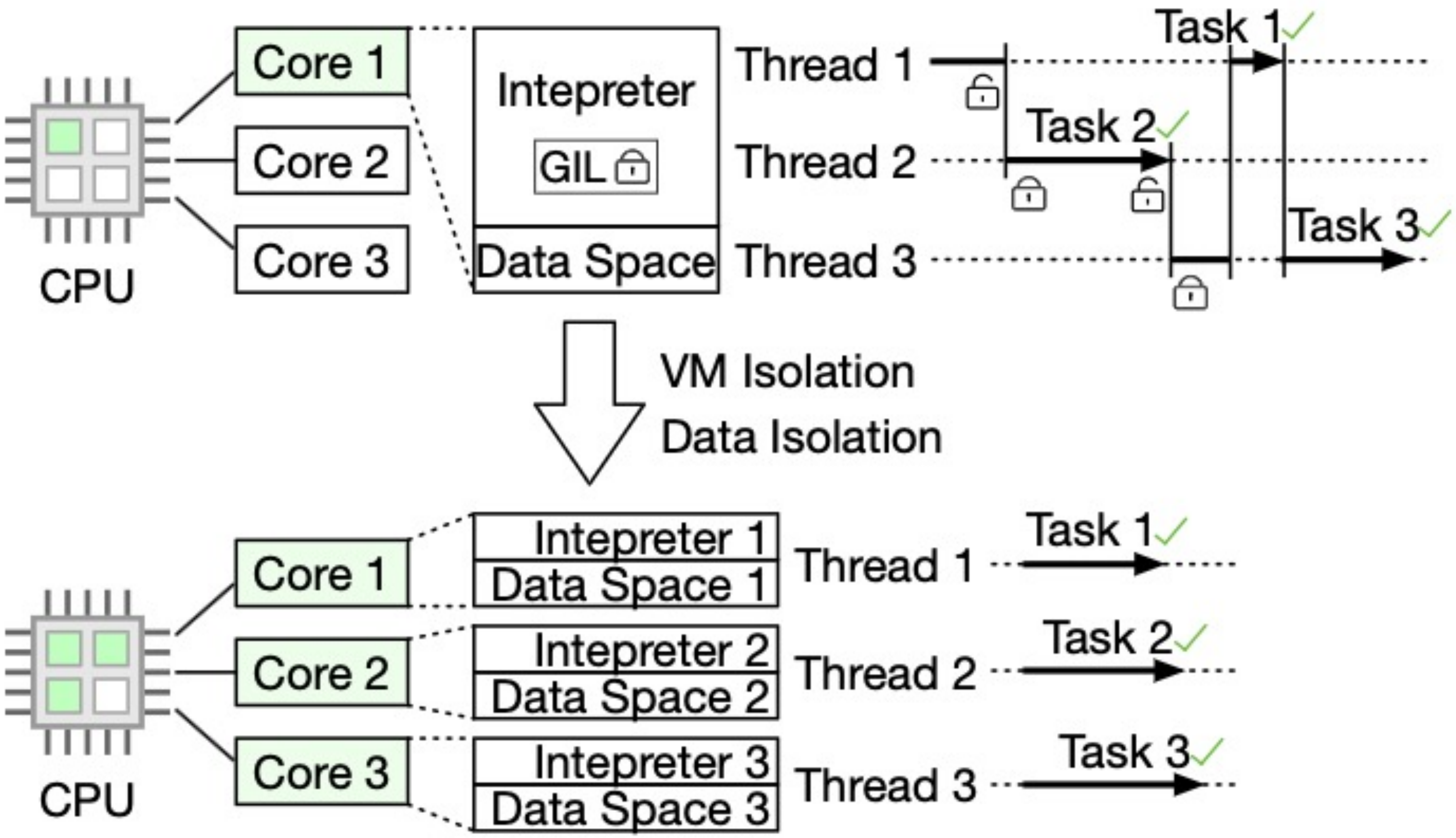
- Bind each ML task with a thread
- Do **thread isolation**



Abandon GIL and support multi-threading!

Python Virtual Machine (VM) – Refining CPython

Multi-threading



Python Virtual Machine (VM) – Refining CPython

VM Isolation

- Modify the creation of VM instances such that a process can hold multiple thread-level VMs, each VM having its independent lifecycle
- Modify the initialization of CPython, particularly creating and initializing a PyInterpreterState instance for each thread

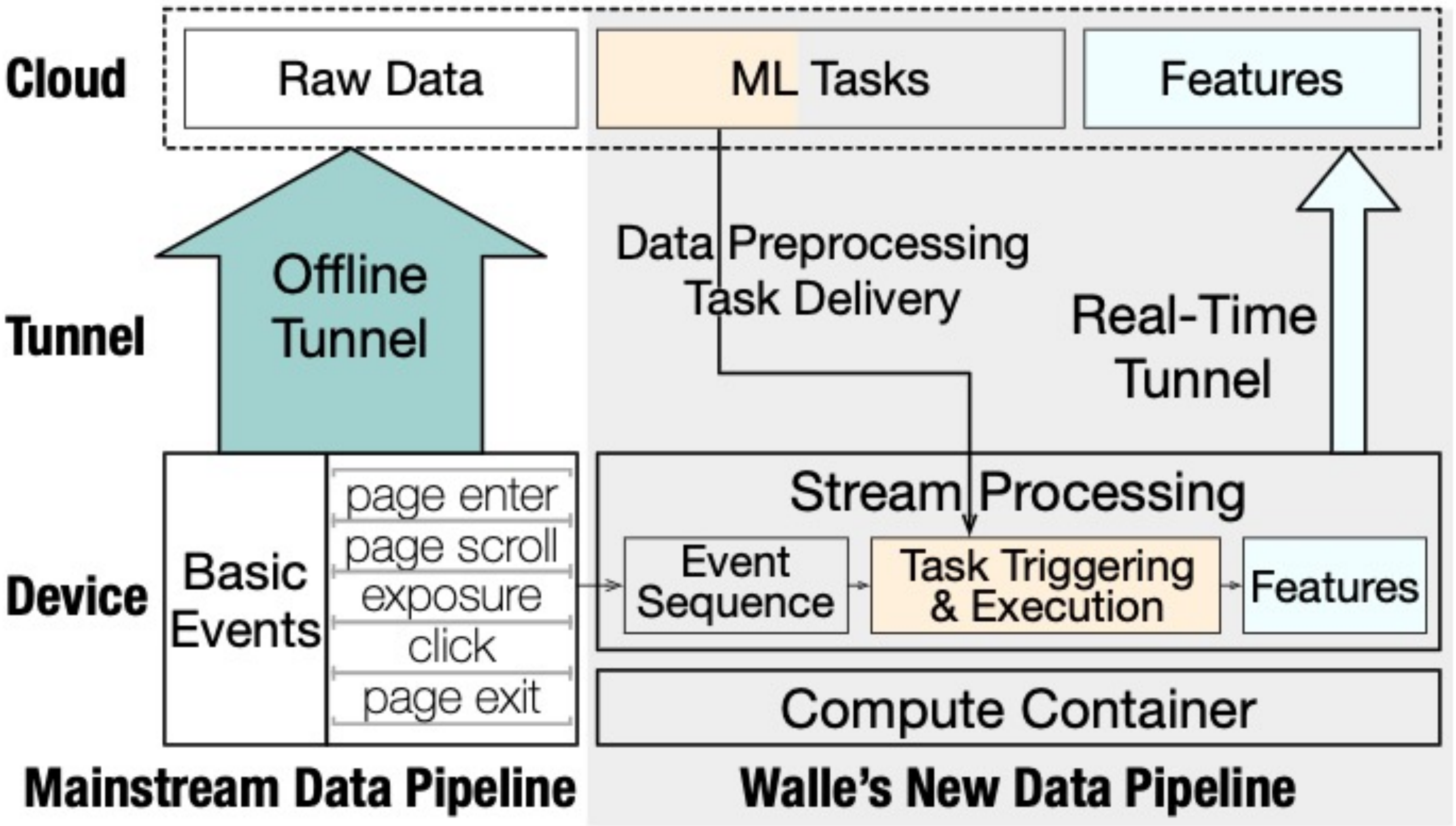
Data Isolation

- Adopt the thread-specific data (TSD) technique for data isolation, such that each thread has its own data space

4

Walle – Data Pipeline

New Data Pipeline – More Natural & Efficient



Enable each mobile device to process only its user's behavior data at source

New Data Pipeline – More Natural & Efficient

Event Sequence Creation

- There are five major kinds of basic events: page enter, page scroll, exposure, click, and page exit.
- Each kind of event is recorded with a unique event id, a page id, a timestamp, and event contents.
- The page-level event sequence is created by aggregating the events between the enter and exit events of the same pages.

Trigger Management

- Leverage the data structure of prefix tree, called a trie, for effi- cient trigger management.

New Data Pipeline – More Natural & Efficient

Task Execution

- Standard data processing and mode execution APIs.
- KeyBy, which returns the events matched with a given key.
- TimeWindow, which returns the events in a given time window.
- Filter, which returns the events filtered by a defined rule.
- Map, which processes the event contents with a defined function.

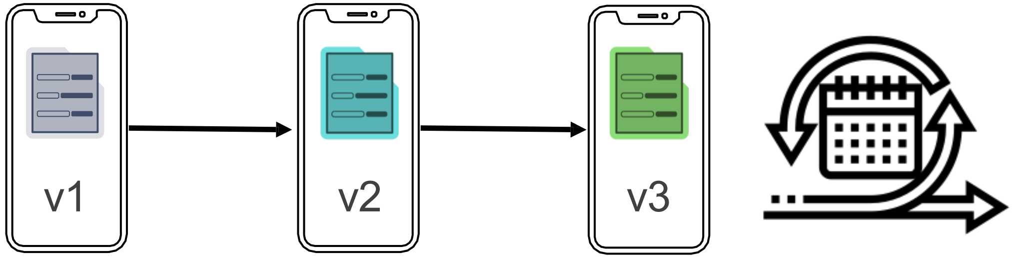
Collective Storage

- Each stream processing task saves its output features as a table using SQLite.
- A collective data storage API is encapsulated over SQLite to reduce the number of write, thereby improving performance.

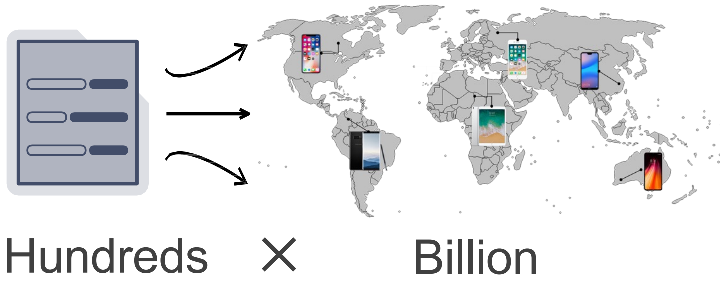
5

Walle – Deployment Platform

Practical Considerations & Challenges



Frequent experiment & deployment for daily ML task iteration



- APP version
- Device-side differentiation
- User-side differentiation

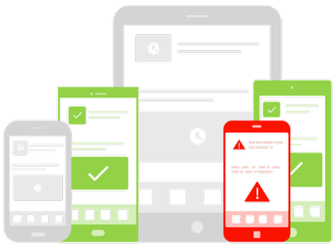
Massive multi-granularity task deployment requirements

Unstable wireless network

Frequent APP switch & **Only one APP on the foreground**

Intermittent device availability

Each mobile APP has one process. **Failure of any task** leads to **APP crash**.

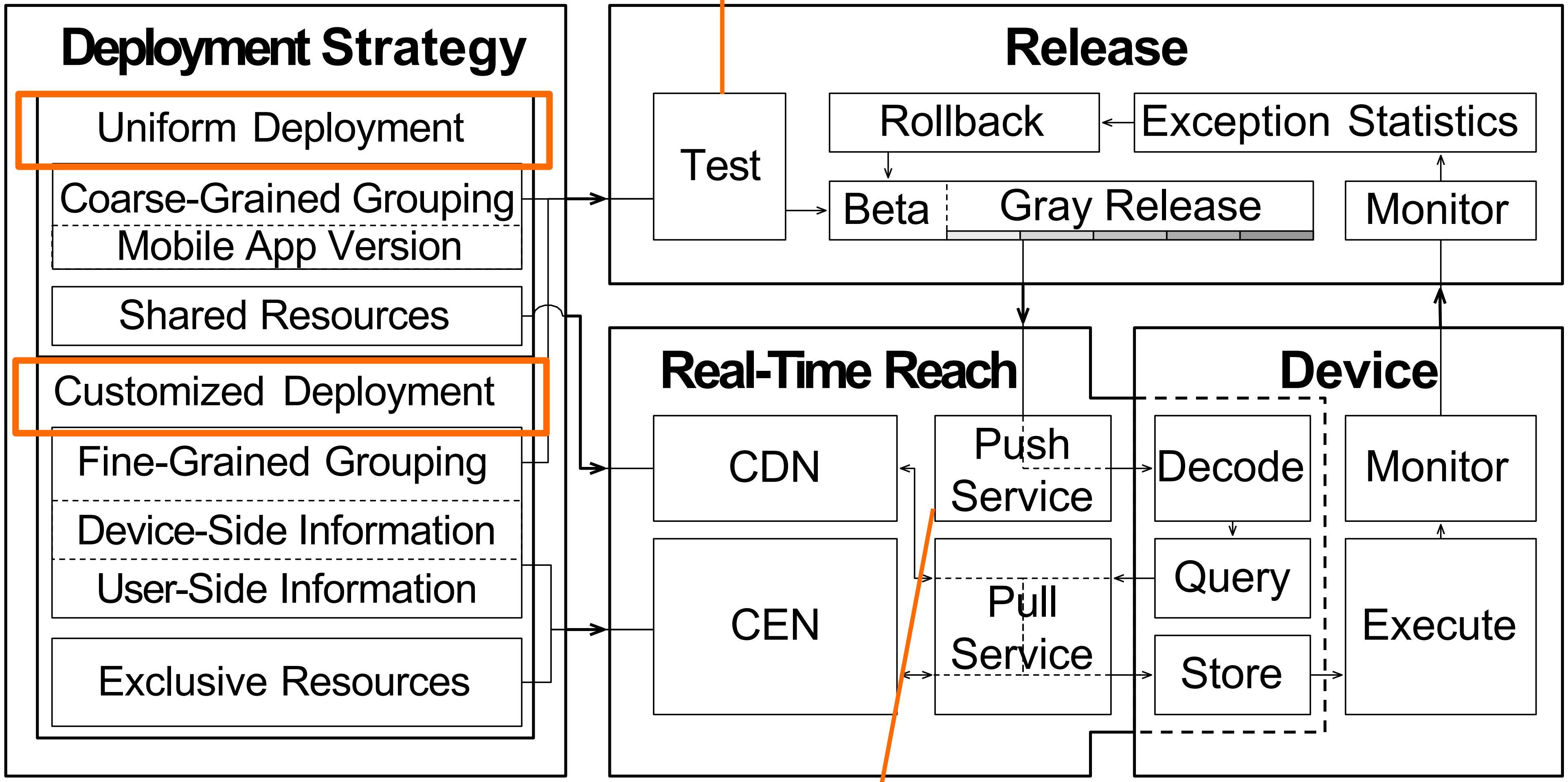


Extensive real-device testing is impractical.

Potential task failure

Timely, Robust Task Release & Deployment

Cloud-based simulators with compute container



Existing client-side http request for business services

New Data Pipeline – More Natural & Efficient

Task Management

- Git is adopted to achieve the isolation of different tasks and the version control of a certain task.
- the entire task management is regarded as a git group; each business scenario corresponds to a git repo (repository); each task in a business scenario corresponds to a branch; and each version of a task corresponds to a tag.

- The files are divided into two categories: one is the shared files; and the other is the exclusive files.

New Data Pipeline – More Natural & Efficient

Task Deployment

- The uniform policy supports task release grouped by the APP version.

- The customized policy supports grouping by device-side information and user-side information.

Task Release A novel push-then-pull method.

- Add a mobile device’s local task profile into the http header and letting the cloud compare it with the latest task profile.

- If a new task needs to be released and takes the uniform deployment policy, then the cloud responds with the CDN address of the shared task files.

6

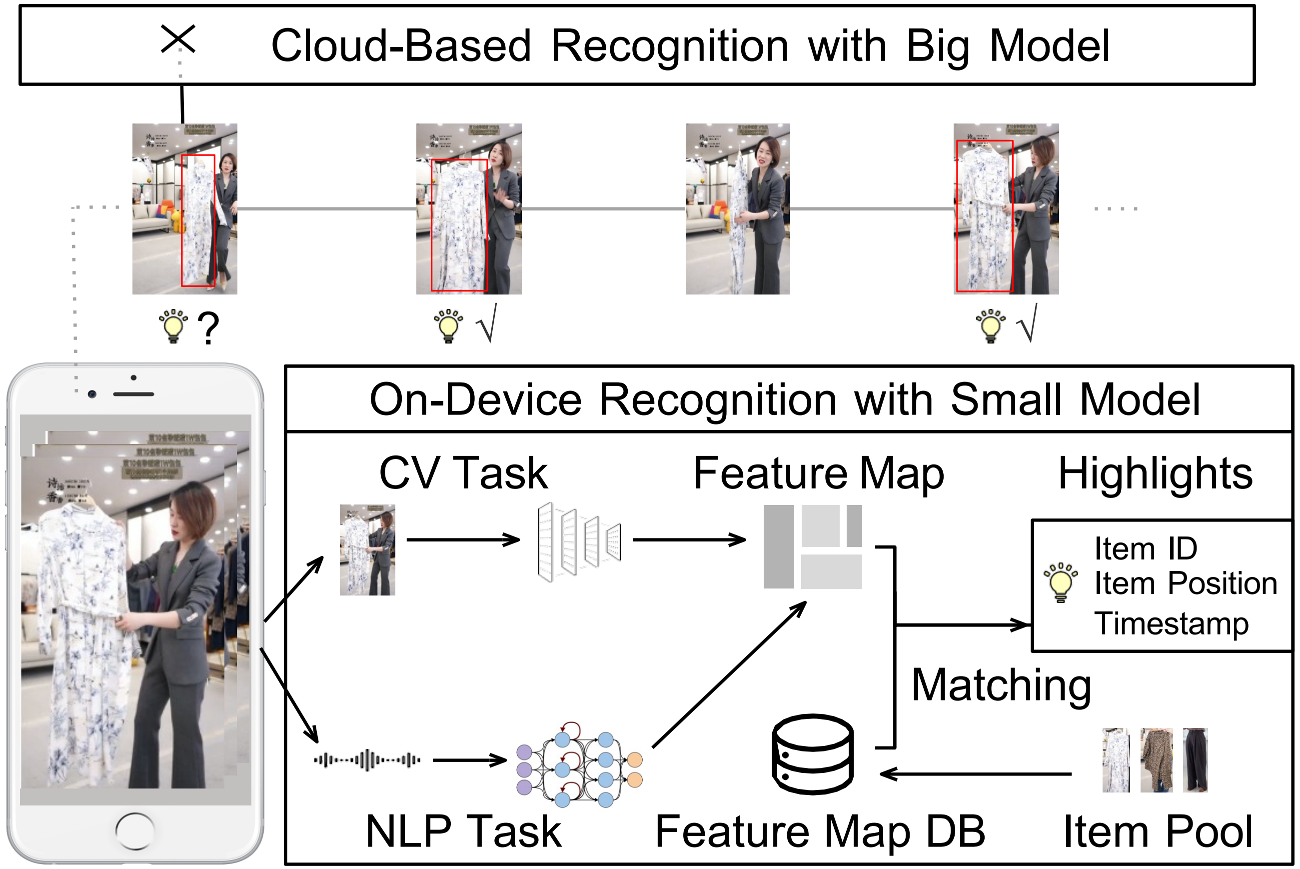
Evaluation Results

6.1

Practical Performance in E-Commerce Scenarios

Compute Container in Livestreaming

Task: Highlight Recognition



Cloud-Based Design

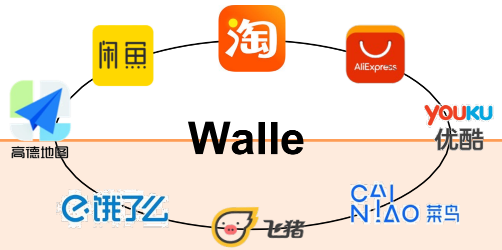
- Key bottleneck: **Heavy load** (lots of streamers, long video streams, stringent latency requirement)
- Cover **part** of streamers
- Analyze **part** of video frames

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**

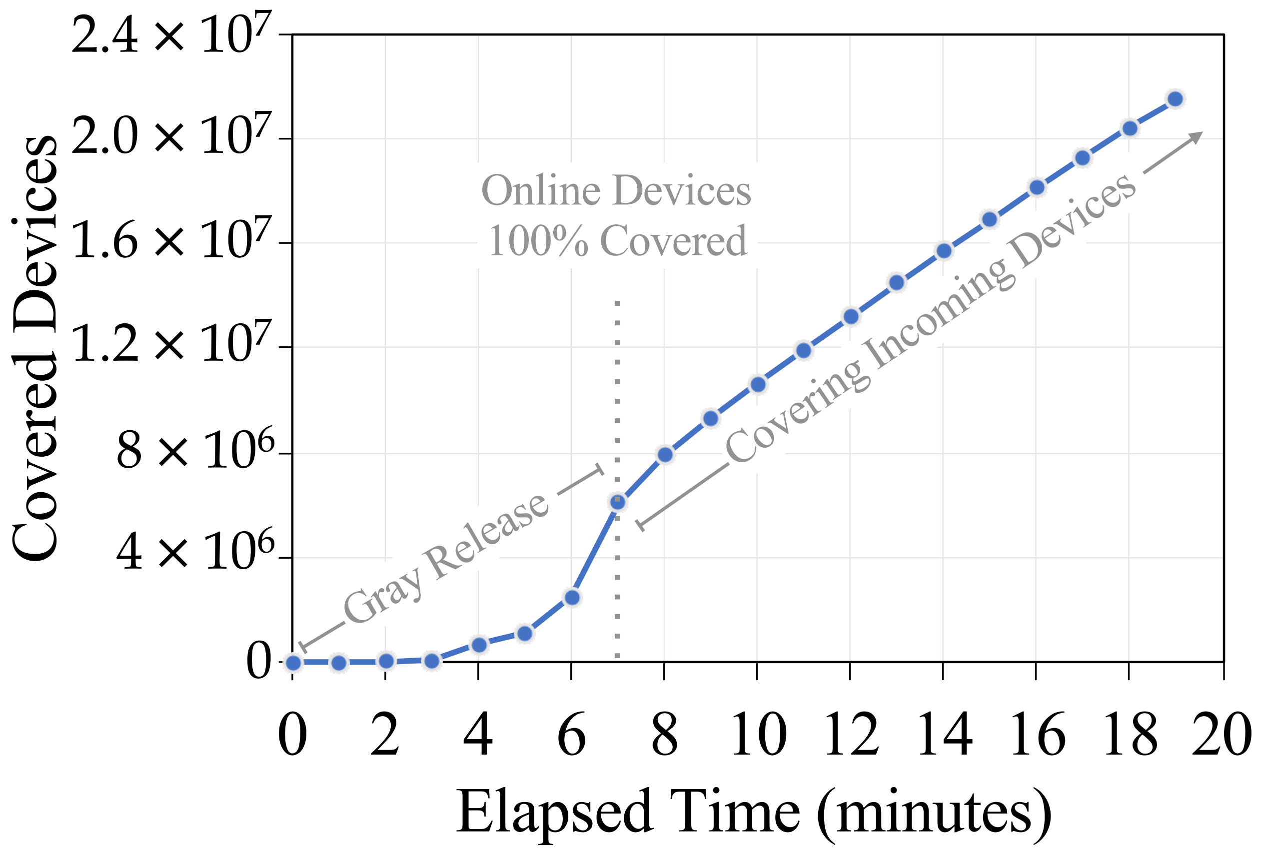
	Item Detection	Item Recognition	Facial Detection	Voice Detection
Model Parameter Size	FCOS [40] 8.15M	MobileNet [25] 10.87M	MobileNet [25] 2.06M	RNN 8K
Huawei P50 Pro iPhone 11	56.92ms 33.71ms	25.68ms 29.74ms	41.42ms 22.58ms	0.07ms 0.01ms

ML Task Deployment Statistics



Large-Scale Production Use

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

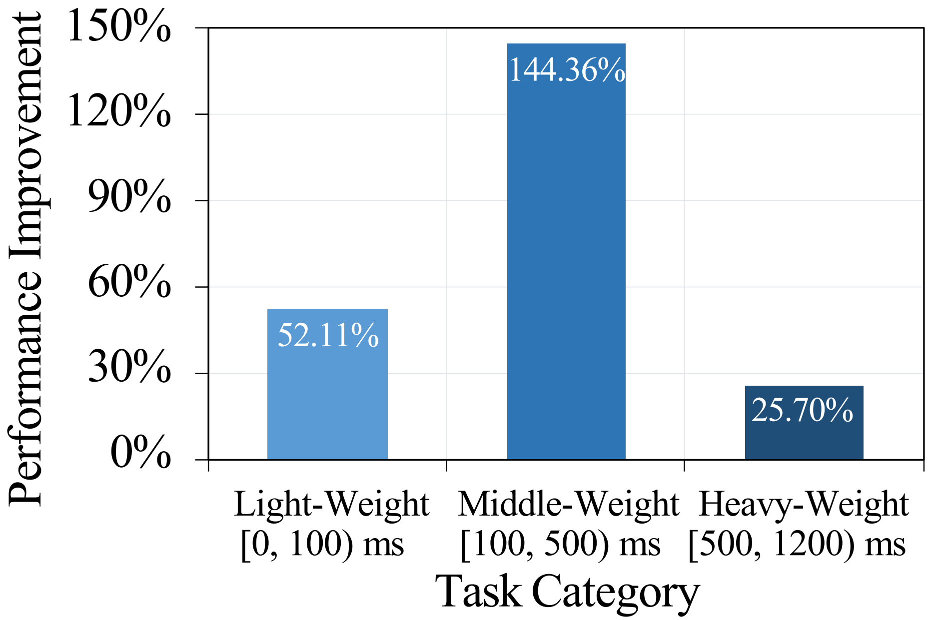
MNN *vs.* TensorFlow (Lite) & PyTorch (Mobile)

Alibaba Group

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

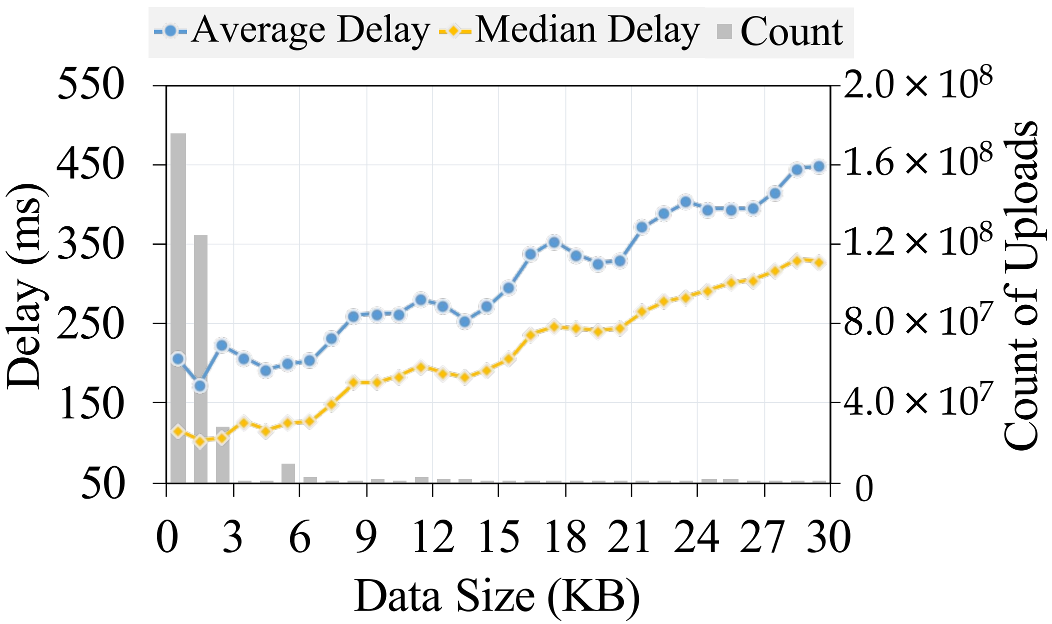
Python Thread-Level VM, Real-Time Tunnel

Python Thread-Level VM *vs.* CPython with GIL (analyzed over 30 million online ML task executions)



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)



90% uploads < 3KB, 250ms
0.1% uploads = 30KB, 450ms

Thanks for listening!
Comments & Questions?