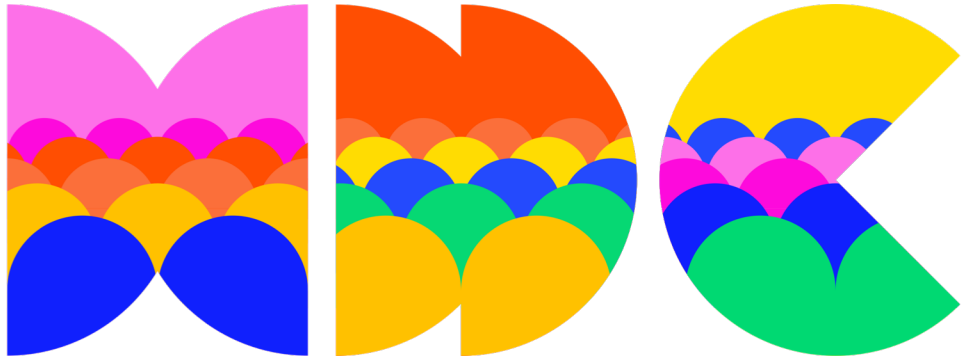
中山大學
SUN YAT-SEN UNIVERSITY

大模型与智能运维的双向奔赴



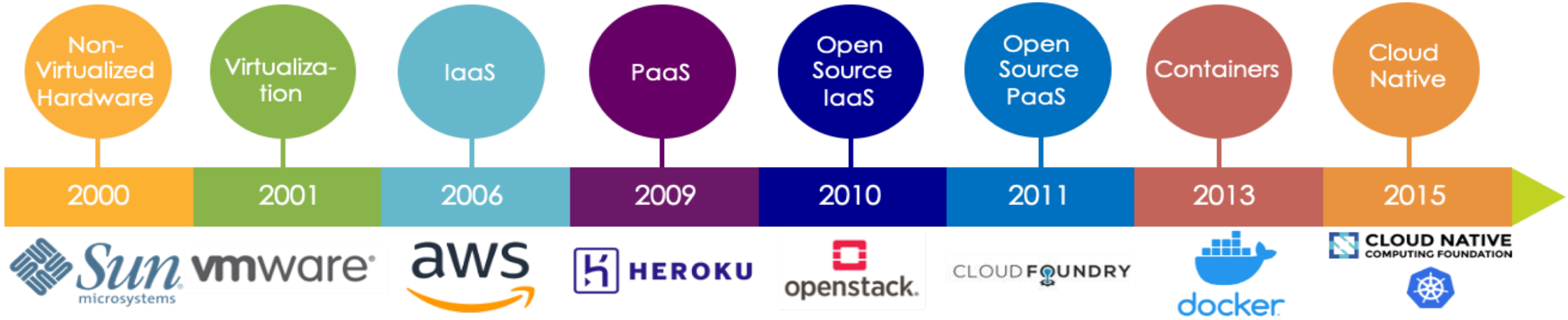
陈鹏飞 中山大学

目录

目录

云原生浪潮

✓ 以容器、微服务等技术为核心的云原生系统成为云计算主流趋势;



2004~2007

Google 大规模使用
容器（cgroups）技术

2008.01

cgroups 合并进入
Linux 内核主干

2013.03

Docker 项目正式发布

2014.06

Kubernetes 项目发布

2015.07

CNCF （云原生基金会）成立
22 个创始成员
Kubernetes 成为第一个 CNCF 项目

2017.07

CNCF 成立2周年
170个成员
14个基金会项目

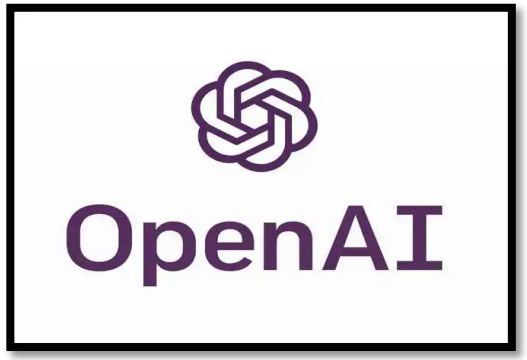
2018.07

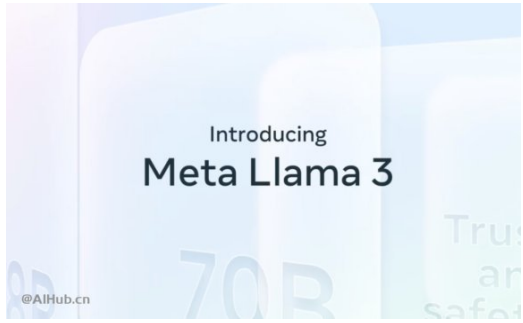
CNCF 成立3周年
195个成员
19个基金会项目
11个孵化项目

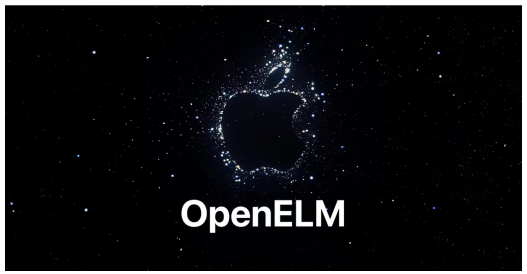
大模型浪潮

✓ 自2022年以来，涌现出了以ChatGPT为代表的成千上万的大模型;

✓ 以智能体为代表的大模型应用如雨后春笋；

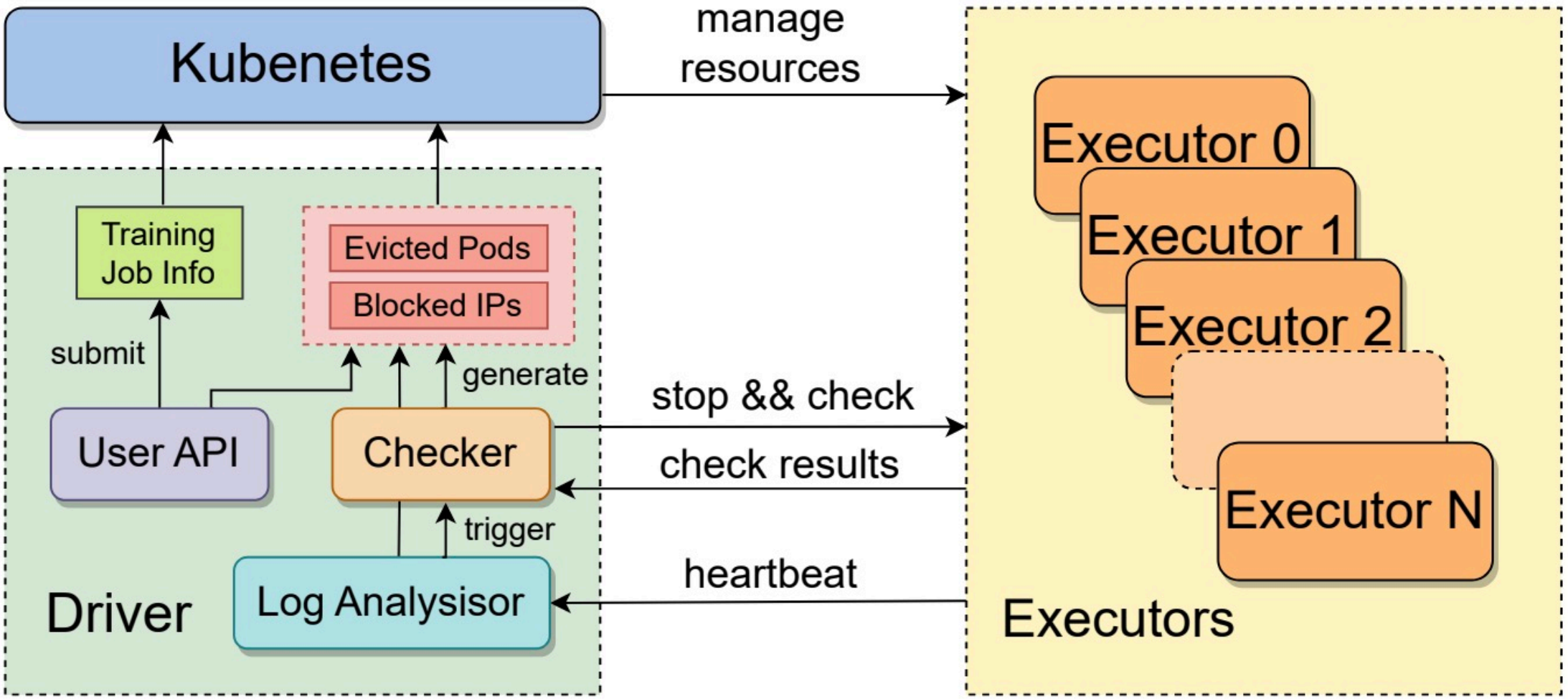






基于云原生的大模型系统

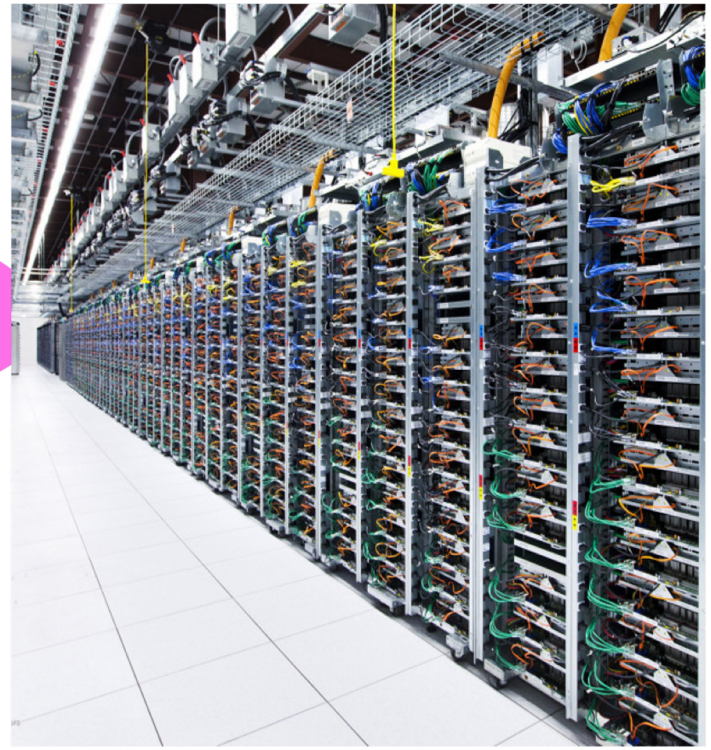
✓ 大模型训练和推理系统普遍采用**云原生**方式部署；

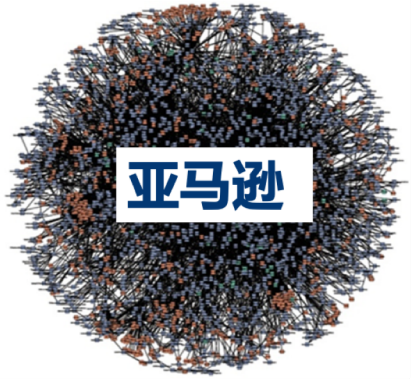


字节跳动，MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs

云原生系统中的故障

硬件（上万台机器）⟹ 软件（上百万实例）⟹ 故障（上亿元损失）





时间	企业	故障现象	故障影响
3 月 29 日	唯品会	宕机 12 小时	业绩损失超亿元
3 月 29 日	腾讯	故障 14 小时	一级事故
7 月 6 日	boss 直聘	短时间服务异常	-
7 月 26 日	小红书	APP 大规模闪退	用户无法登录
10 月 23 日	语雀	服务中断 7 小时	用户数据无法访问
11 月 12 日	阿里云	基础服务故障 3h	控制台访问及 API 调用出现异常
11 月 27 日	滴滴	服务宕机 36 小时	业绩损失超 4 亿元
12 月 3 日	腾讯视频	VIP 服务失效	用户无法使用 VIP 权益
12 月 19 日	喜马拉雅	无法正常使用	-

大模型系统中的故障

✓ 大模型训练推理过程中在硬件、平台、框架等多个层次产生故障；

Table 1. Failure Analysis in AI Service

Group	Failure	Description	Paper
Data	Data Quality	Issues related to the format, type, and noise of data.	[161]
	Data Drift	A model trained on a specific distribution of data but then encountering a different distribution in practice.	[175]
	Concept Drift	The relationship between features and labels becomes invalid over time.	[175, 207]
Development	Defective Code	Logical errors in code, and poor software design.	[45, 61, 88]
	Service API Fault	API Incompatibility, API Change, and API Misuse.	[14, 185]
Network Fault	Network Transmission Fault	Network congestion, bandwidth limitations, or failures in network hardware.	[85]
External Attack	Network Attack	Lead to temporary service interruptions and data leakage or corruption.	[110, 145]
	Adversarial Attack	Deceive the AI into making incorrect actions through malicious inputs.	[2, 3, 141]

Table 4. Failure Analysis in AI Model

Group	Failure	Description	Paper
Data	Data Quality	Low-quality data leads to poor model performance.	[31, 93, 189]
	Data Preprocessing Fault	The inadequate handling of data noise, damage, loss, and inconsistency.	[14, 32, 107]
Model Hyperparameter	Inappropriate Layer and Neuron Quantity	Incorrectly setting the number of layers and neurons can affect the model's parameter count and performance.	[8, 14, 53, 113, 184]
	Inappropriate Learning Rate, Epochs, and Batch Size	Influence the training speed and model performance (overfitting and underfitting).	[56, 163]
Model Structure and Algorithm	Misuse Neural Networks	Inappropriate types of neural networks.	[167]
	Misuse Activation Function	Introduce non-linearity to enhance model fitting ability.	[39]
	Misuse Regularization	Inappropriate regularization lead to overfitting.	[177]
	Misuse Optimizer	Influence the training speed and model performance.	[54]
	Misuse Loss Function	Affect the speed and degree of convergence in training.	[186]
	Dataset Partitioning Fault	Insufficient data for training and validation.	[120]

Table 7. Failure analysis in AI Framework

Group	Failure	Description	Paper
Data	Tensor Alignment Fault	Tensors do not align as expected, leading to shape mismatches.	[65, 205]
	Input Format Fault	The shape or type of the input data mismatches the expected format.	[65, 67]
	Dataloader Killed Fault	Dataloader is killed due to memory leak in multipel worker.	[62]
API	API Usage Fault	API is used in a way that does not conform to the correct logic.	[65]
	API Compatibility Fault	Different APIs are not compatible with each other.	[65, 67, 205]
	API Versioning Fault	The version of API is incompatible with the code or dependencies.	[65, 67, 205]
Configuration	Framework Config. Fault	Incorrect configuration when using a framework.	[142, 194, 205]
	Device Config. Fault	Inability to leverage computing devices for optimal performance.	[65, 194]
	Environment Config. Fault	Environmental configuration faults when developing and deploying an AI framework.	[21, 92, 203]

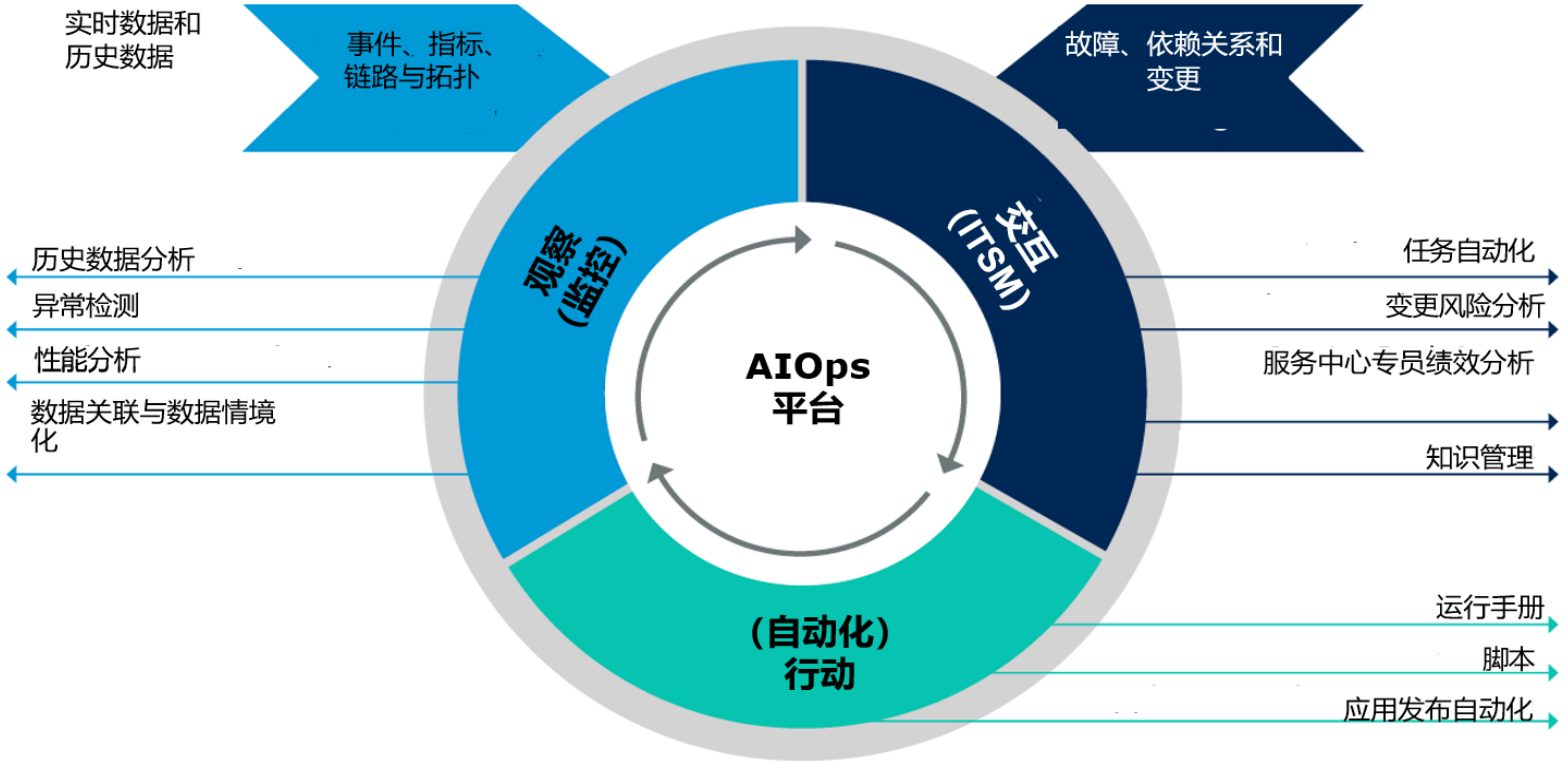
Table 10. Failure Analysis of AI Toolkit

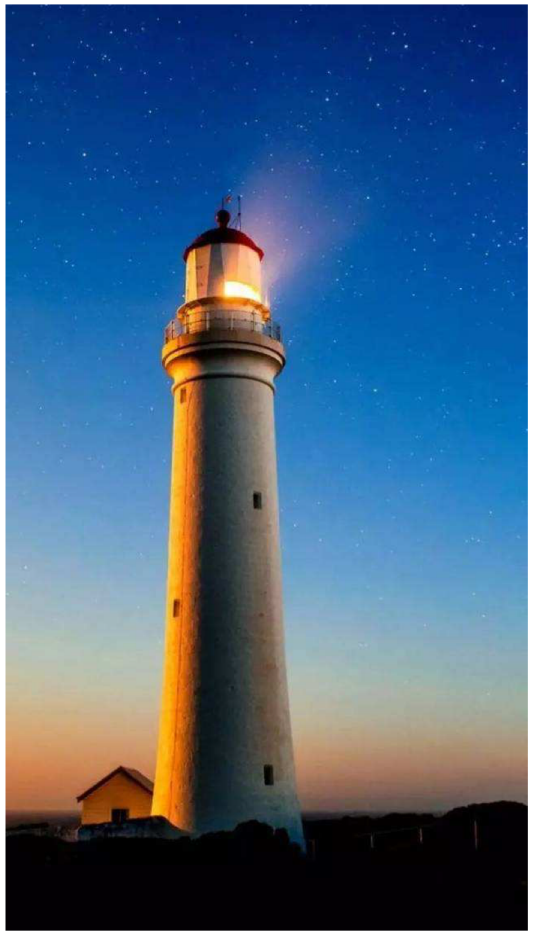
Group	Failure	Description	Paper
Synchronization	Data Race	Inability to determine the order of "read&write" and "write&write" actions among multiple threads.	[68, 191]
	Barrier Divergence	Threads within the same block fail to reach a barrier due to variations in their execution flow.	[25, 191]
	Redundant Barrier Func.	Unnecessary synchronization operations.	[25, 191]
Memory Safety	Out-of-bounds Access	Access buffers beyond boundaries in global memory or shared memory.	[170, 209]
	Temporal Safety Fault	Access GPU memory that has already been freed or has not been properly allocated or initialized.	[170, 209]
	Failed Free Operation	Double free and invalid free operations.	[170, 209]
Dependency	Intra-dependency Fault	Incorrect versioning or unsuccessful installation of a toolkit.	[14, 63]
	Inter-dependency Fault	Mismatch of software and hardware.	[63]
Communication	NCCL Fault	Possibly due to a network error or a remote process exiting prematurely.	[44, 62, 76]
	NVLink Fault	Caused by the hardware failures like GPU overheating.	[44, 62]
	MPI Fault	A failure of network connection to peer MPI process or an internal failure of the MPI daemon itself.	[71]

智能运维

利用大数据、机器学习和其他分析技术，通过分析自动发现IT系统中的问题，并定位根因，自动恢复系统，增强IT系统稳定性、性能和可用性，为IT运维提供“端到端”的解决方法。

智能运维（AIOps）平台可实现对整个IT运营管理（ITOM）的持续洞察



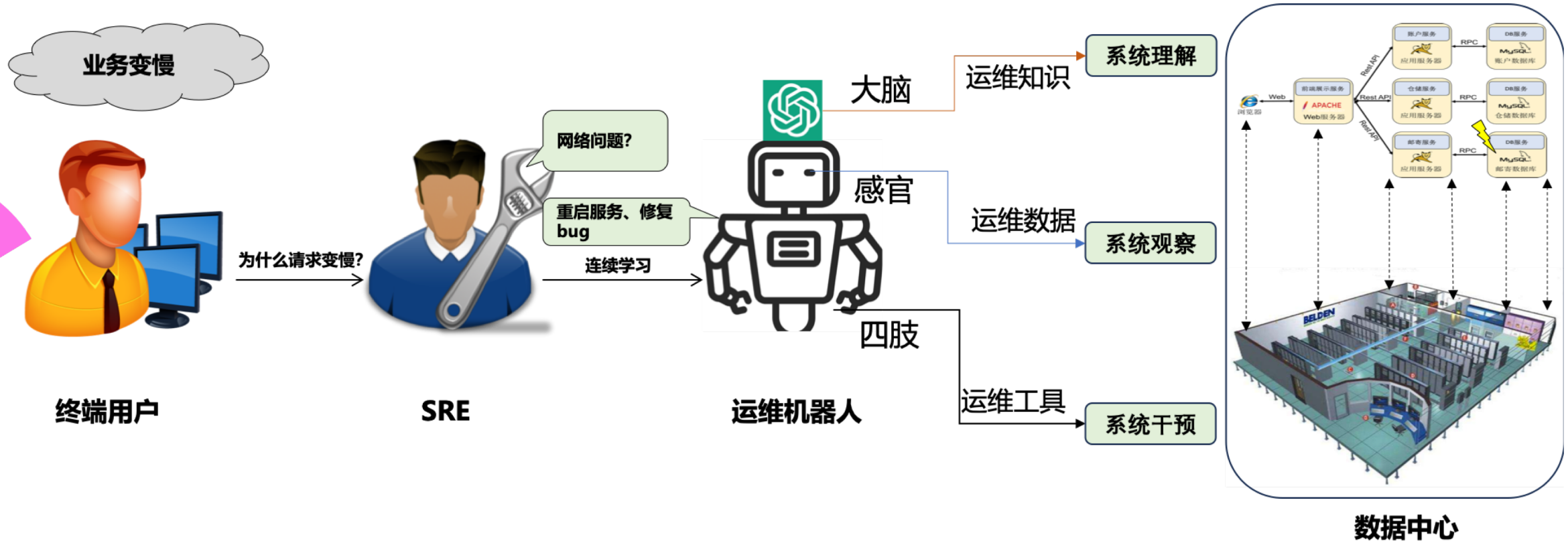


来源：Gartner: Market Guide for AIOps Platforms

目录

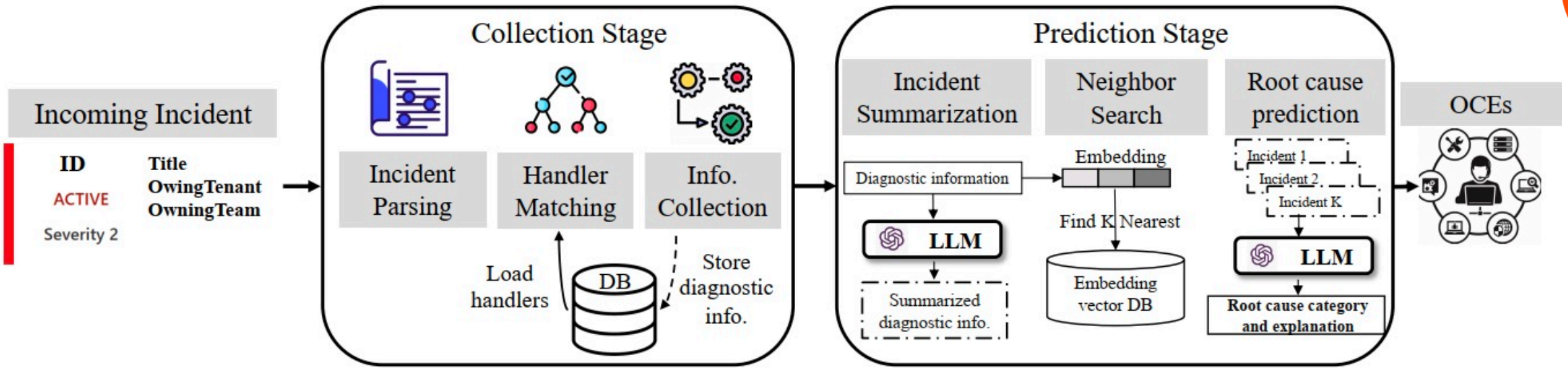
大模型赋能智能运维

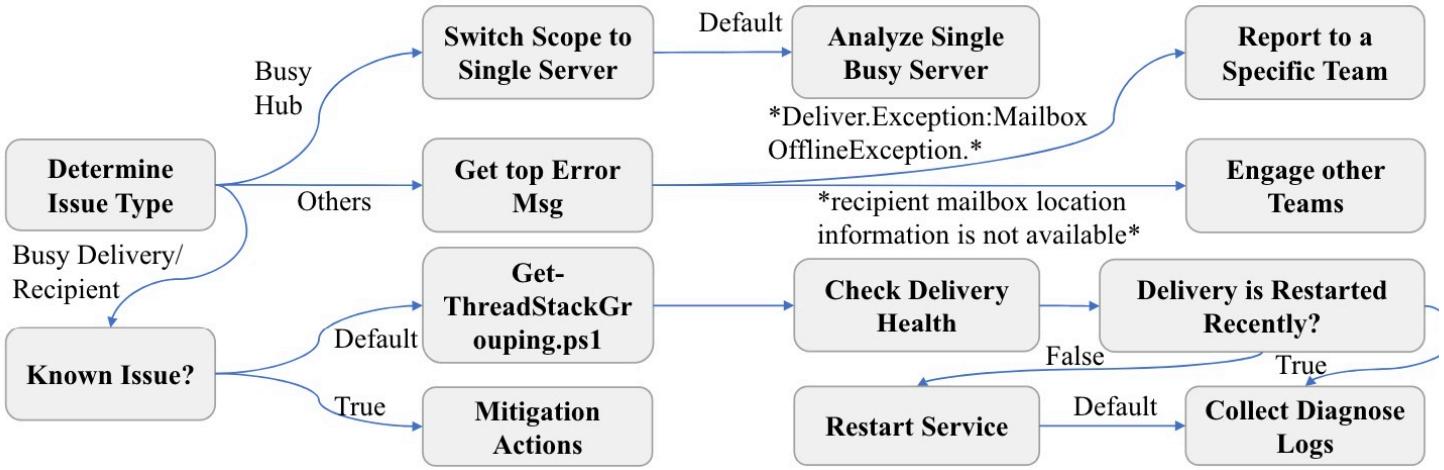
✓ 大模型成为智能运维的指挥大脑，与其他智能体协同工作；



大模型赋能智能运维

✓ 基于大模型的云故障根因定位分析；



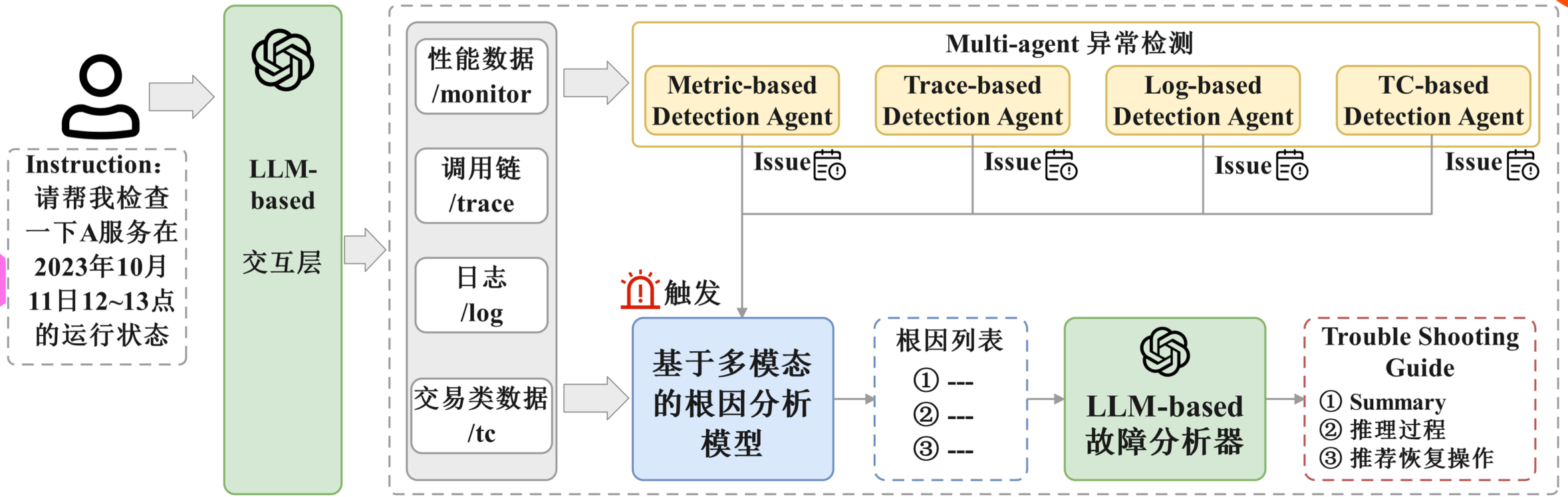


Method	F1-score		Avg. Time (s)	
	Micro	Macro	Train.	Infer.
FastText [61]	0.076	0.004	10.592	0.524
XGBoost [3]	0.022	0.009	11.581	1.211
Fine-tune GPT [1]	0.103	0.144	3192	4.262
GPT-4 Prompt	0.026	0.004	–	3.251
GPT-4 Embed.	0.257	0.122	1925	3.522
RCACopilot (GPT-3.5)	0.761	0.505	10.562	4.221
RCACopilot (GPT-4)	**0.766**	**0.533**	10.562	4.205

Microsoft, Automatic Root Cause Analysis via Large Languag Models for Cloud Incidents, Eurosys'24

大模型赋能智能运维

✓ 基于多智能体的多模态数据融合根因定位方法；



大模型赋能智能运维

✓ 基于多智能体的多模态数据融合根因定位方法；

AI Agents 自动运维报告单

故障问题清单

- **故障报告 ID**: 5230c984-7e16-11ee-a836-965a262bc0e6
- **提交时间**: 2023-11-18 15:45:39.627260
- **问题描述**: 对以下故障根因进行分析并给出恢复建议

疑似根因 1

- **故障问题**: 接口/coupon/txCtrl 的 rpc 方法 rpc 调用耗时加剧
- **排序**: 1
- **trace_id**: trace_3a41df507be7a6157905a500b4532745
- **问题说明**: 调用耗时 971ms 超过正常上限 561.41ms

疑似根因 2

- **故障问题**: 调用链出现结构异常，或出现新的调用链模式
- **排序**: 2
- **trace_id**: trace_6feb7529b23bbee0f535a2be5ad55894
- **问题说明**: 新增服务调用 span('Weblogic_19,AppId008,Sys001', 'Weblogic_19,AppId008,Sys001')

疑似根因 3

- **故障问题**: 调用链出现结构异常，或出现新的调用链模式
- **排序**: 3
- **trace_id**: trace_06f9aa5101e0a2657fadcd9117e44af9
- **问题说明**: 缺失服务调用 span('Weblogic_21,AppId008,Sys001', 'Weblogic_21,AppId008,Sys001')

根因摘要

- **解析结果**: 根据历史故障信息，这些故障同时发生时最大可能的根因是
  1. 接口/coupon/txCtrl 的 rpc 方法 rpc 调用耗时加剧，可能是由于服务端处理能力不足或者网络延迟导致的。这可能导致调用链中的其他服务受到影响，从而出现结构异常或新的调用链模式。
  2. 调用链出现结构异常或新的调用链模式，可能是由于服务间的依赖关系发生变化或者某个服务出现了问题。新增的服务调用 span('Weblogic_19,AppId008,Sys001', 'Weblogic_19,AppId008,Sys001')可能是为了解决某个问题而引入的新的逻辑。
  3. 调用链出现结构异常或新的调用链模式，缺失服务调用 span('Weblogic_21,AppId008,Sys001', 'Weblogic_21,AppId008,Sys001')可能是由

恢复方案

- **故障恢复清单**: 每个故障根因的恢复建议如下

疑似根因 1

- **故障问题**: 调用链出现结构异常，或出现新的调用链模式
- **恢复方案**:
  1. 检查服务端的代码，确保逻辑正确；
  2. 检查服务的配置文件，确保配置正确；
  3. 如果可能的话，可以考虑使用调试工具或者日志来定位问题所在。

疑似根因 2

- **故障问题**: 接口/coupon/txCtrl 的 rpc 方法 rpc 调用耗时加剧
- **恢复方案**:
  1. 检查网络连接，确保客户端和服务端之间的通信畅通；
  2. 优化服务端的代码，提高处理速度；
  3. 如果可能的话，可以考虑使用负载均衡或者分布式架构来分散请求压力。

疑似根因 3

- **故障问题**: 调用链出现结构异常，或出现新的调用链模式
- **恢复方案**:
  1. 检查服务端的代码，确保逻辑正确；
  2. 检查服务的配置文件，确保配置正确；

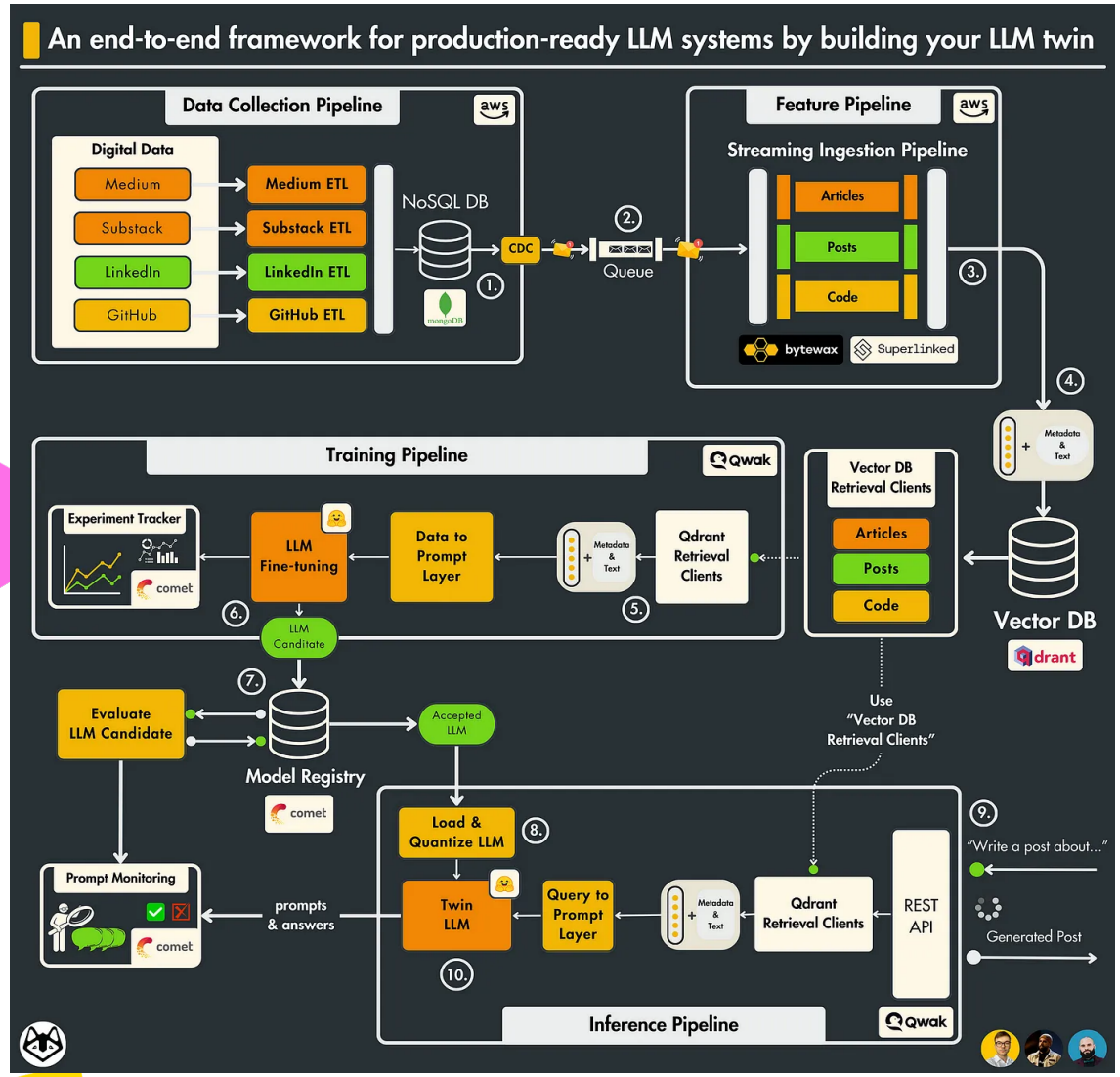
2023-11-19 14:41:23 Sunday November

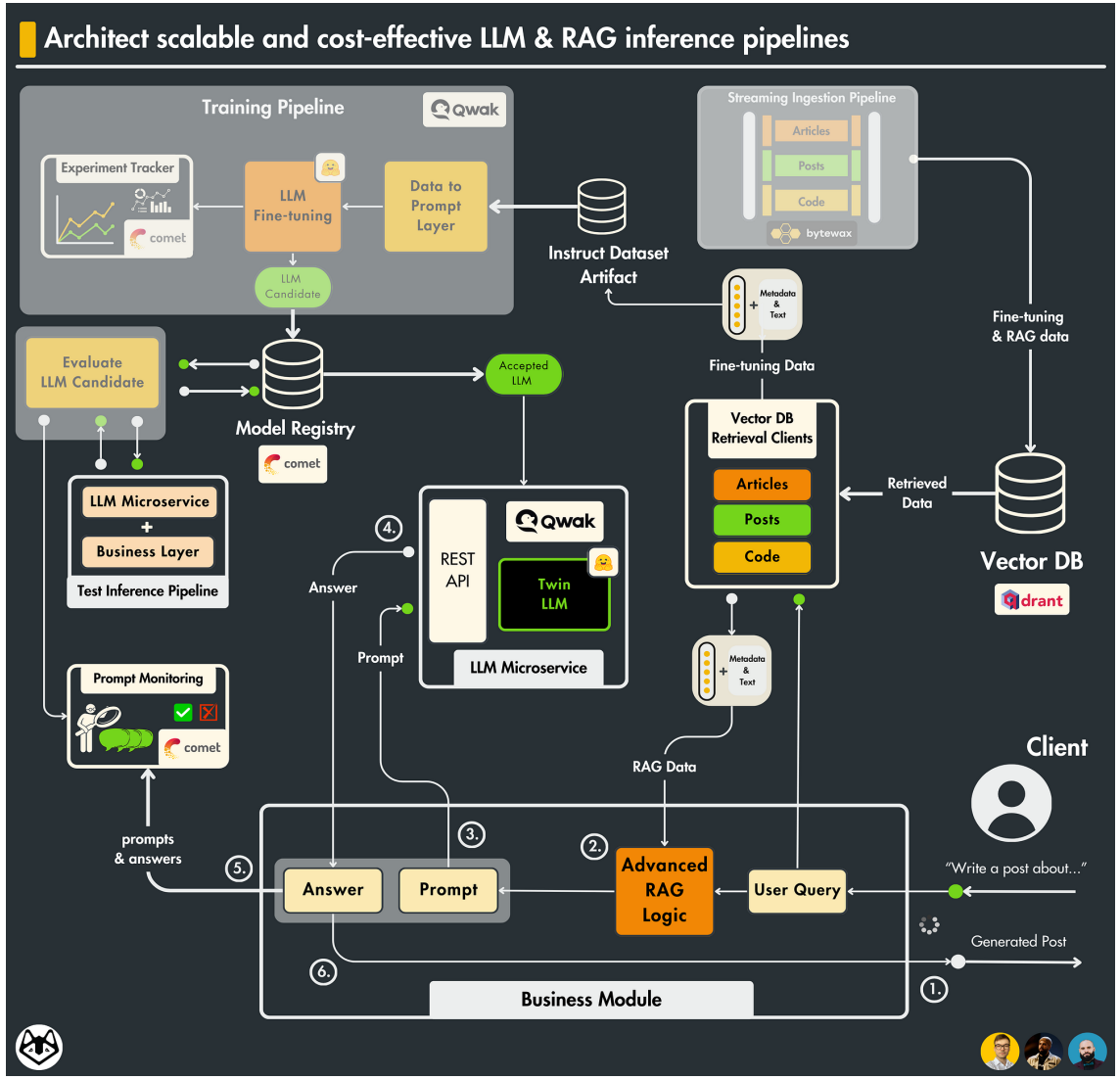
目录

三 智能运维赋能大模型

大模型可观测性

✓ 大模型训练和推理具有复杂的软件栈，亟需可观测性；





https://medium.com/decodingml/architect-scalable-and-cost-effective-llm-rag-inference-pipelines-73b94ef82a99

大模型可观测性

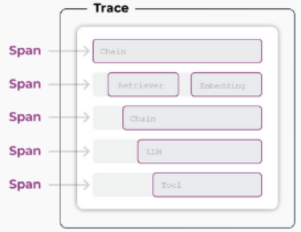
✓ 评价、追踪、提示工程、搜索和查询、微调实现大模型可观测性

Evaluation

Evaluations of LLM outputs by using a separate evaluation LLM

Traces & Spans

Visibility into where the agentic workflow broke

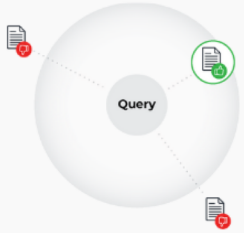


Prompt Engineering

Iterating on prompt templates for improved results

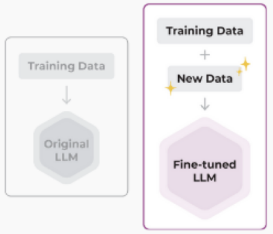
Search & Retrieval

Locate and improve retrieved context



Fine-tuning

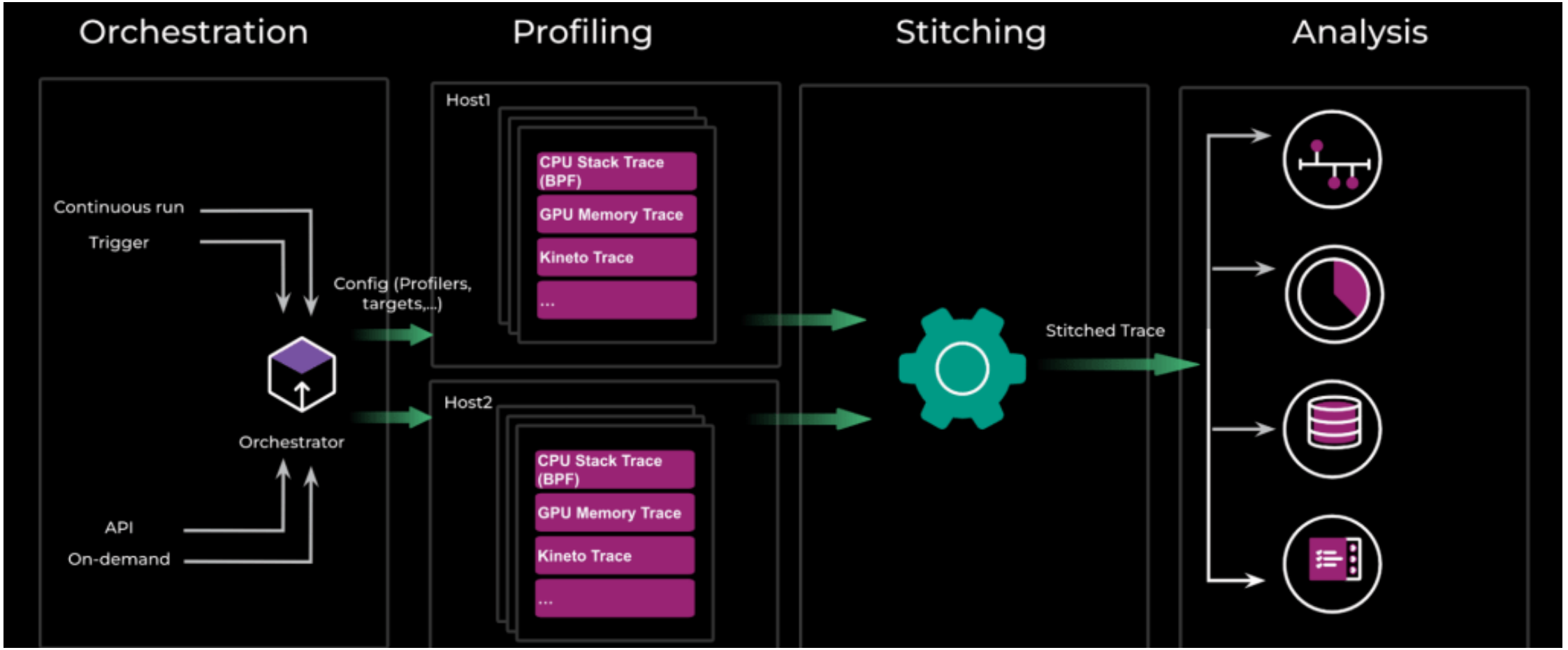
Re-train LLM on use case / company data



Arize,https://arize.com/blog-course/large-language-model-monitoring-observability/

大模型可观测性

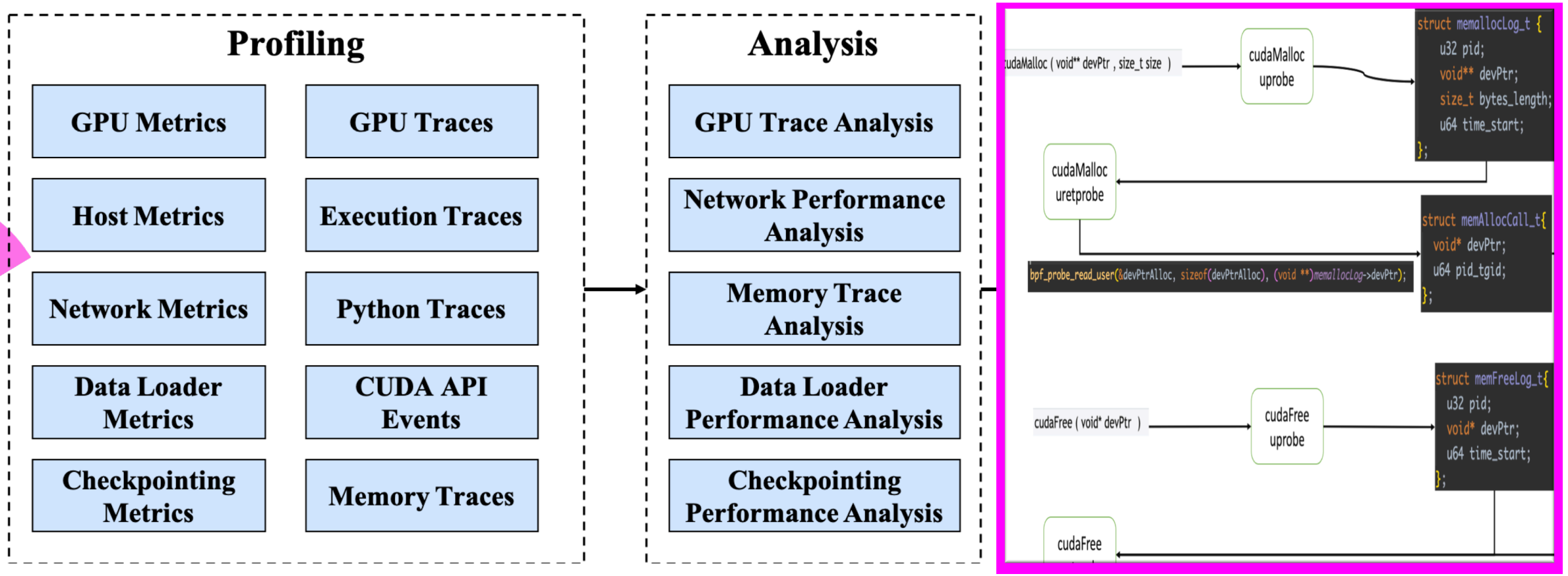
✓ Meta提出了基于profiling的多模态数据关联方法；



Meta, https://atscaleconference.com/systemscale-ai-observability/

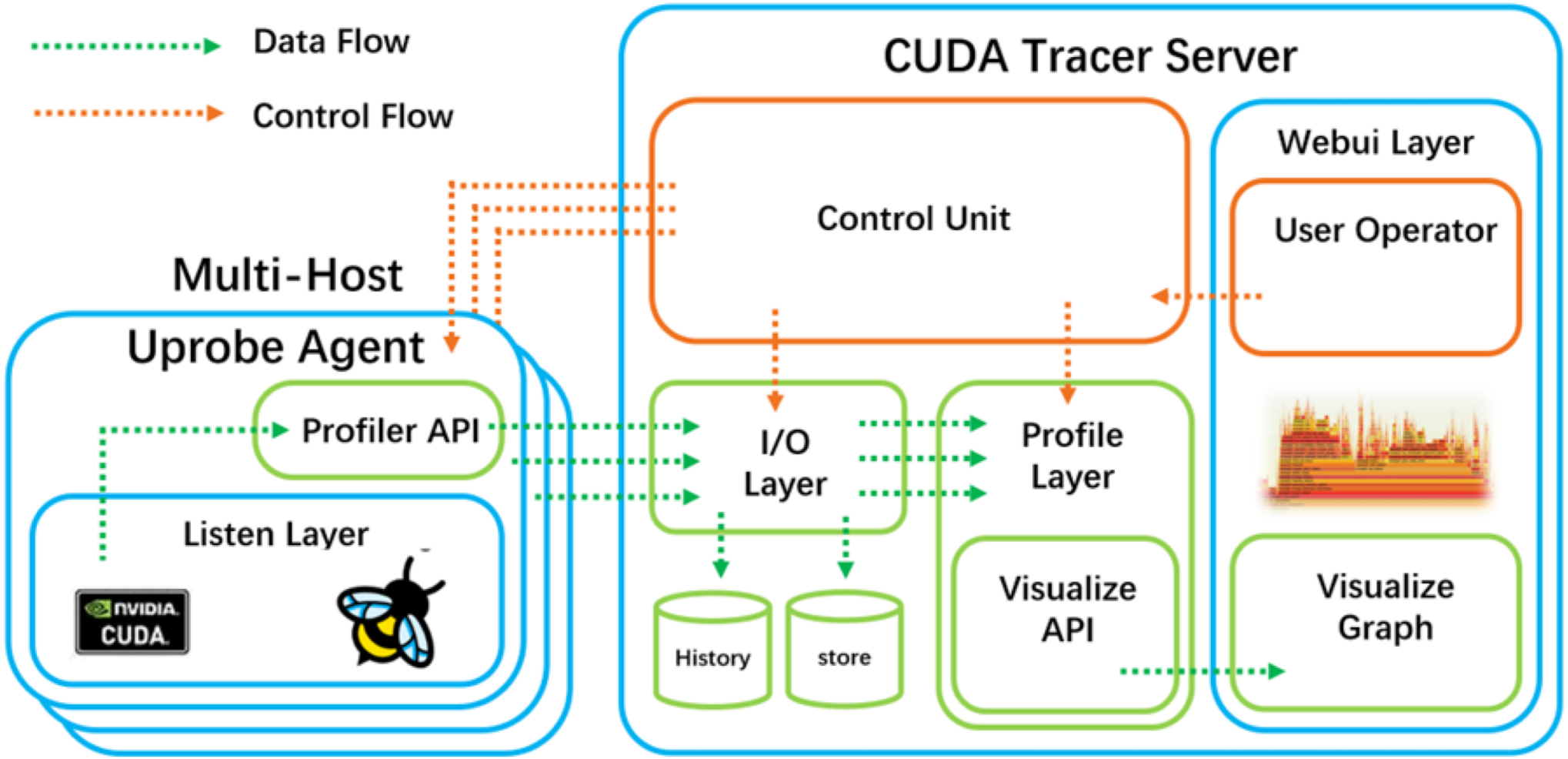
大模型可观测性

✓ 基于eBPF的跨层次、跨节点大模型请求追踪；



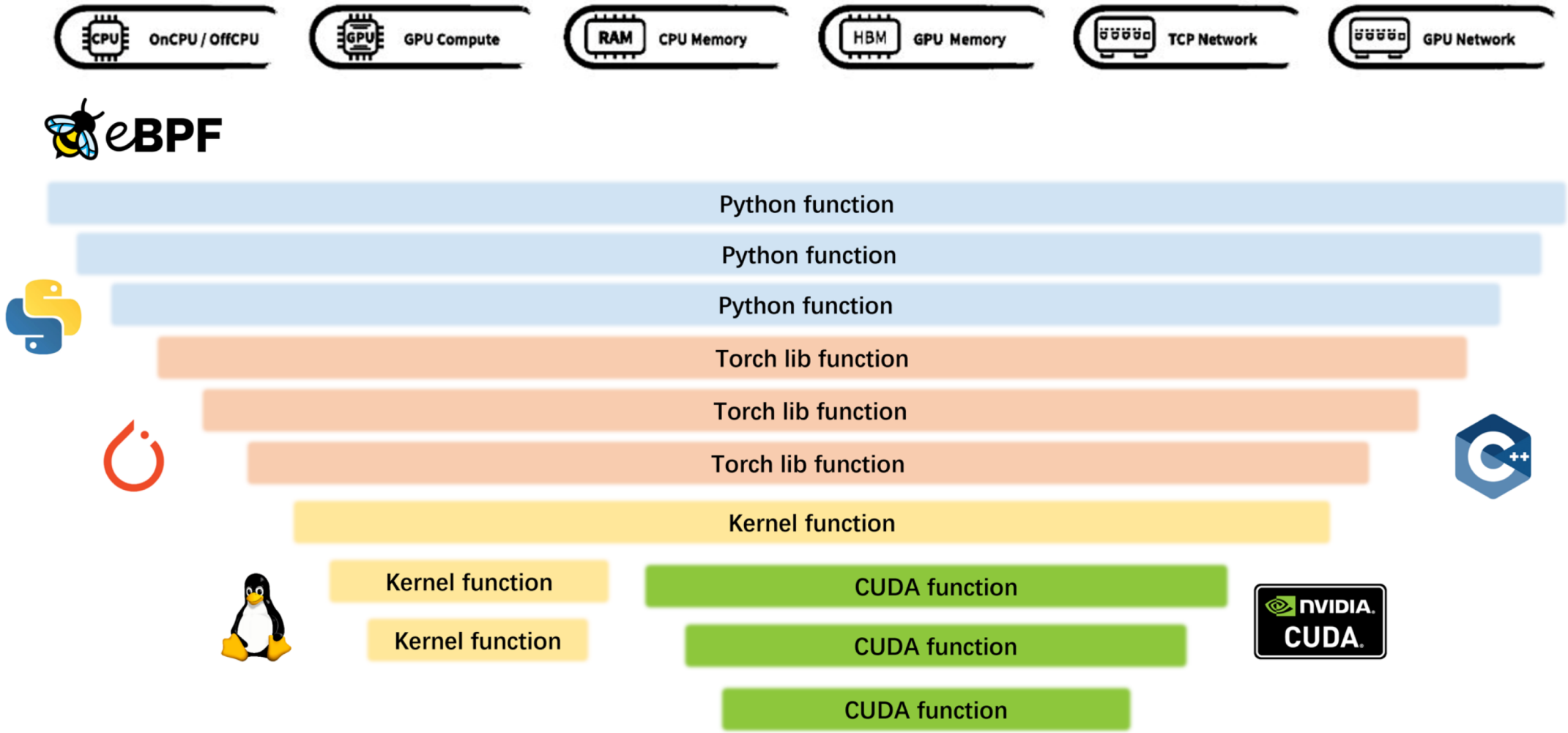
基于eBPF uprobe的CUDA监控

大模型可观测性



监控框架

大模型可观测性



数据全景图

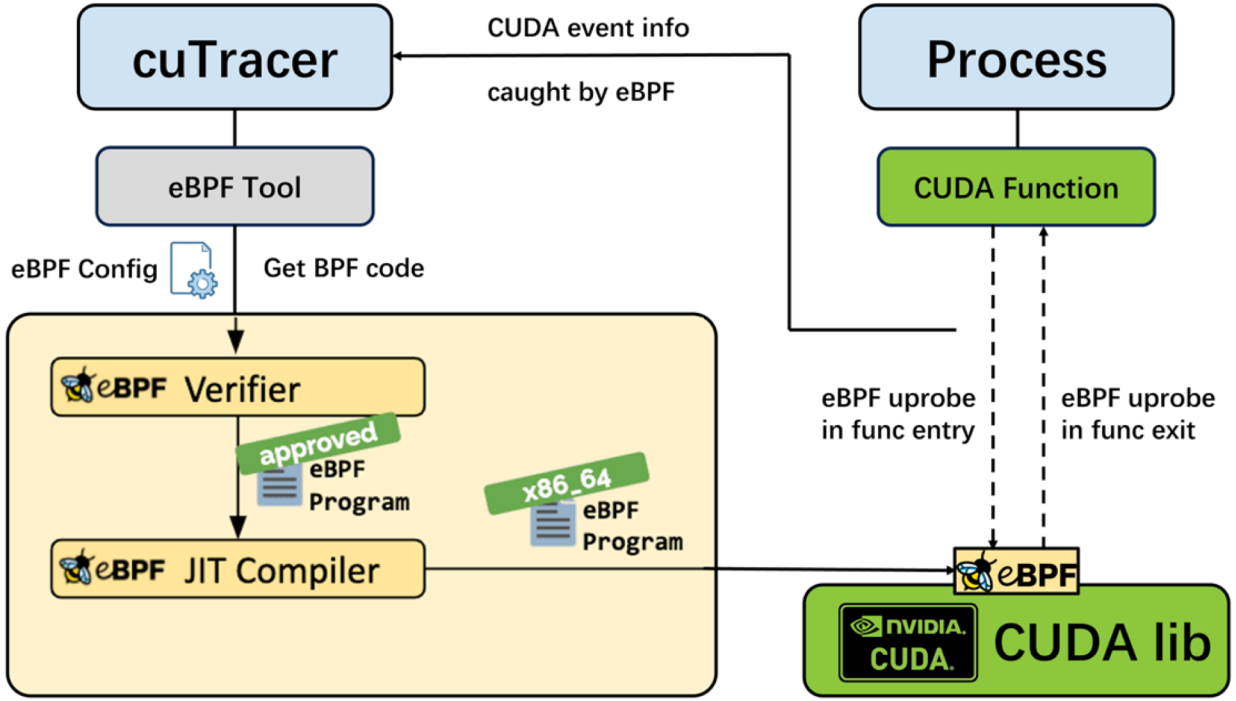
大模型可观测性

3.2 基于 eBPF 的 CUDA 事件追踪

01 定位 Pytorch 的 CUDA 运行库

02 定位 C-CUDA 运行库

03 使用 eBPF 的 uprobe 注入



大模型可观测性

3.2 基于 eBPF 的 CUDA 事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 定位 Pytorch 的 CUDA 运行库

PyTorch 的 CUDA 运行库通常以 pip 包的形式存在。可以通过查看 PyTorch 源码中的 _load_global_deps() 函数来确定这些库的位置。

02 定位 C-CUDA 运行库

03 使用 eBPF 的 uprobe 注入

Code 3.2 PyTorch 源码 _load_global_deps() 函数

```
# See Note [Global dependencies]
def _load_global_deps() -> None:
    ...

    try:
        ctypes.CDLL(lib_path, mode=ctypes.RTLD_GLOBAL)
    except OSError as err:
        # Can only happen for wheel with cuda libs as PYPI deps
        # As PyTorch is not purelib, but nvidia-*-cu12 is
        cuda_libs: Dict[str, str] = {
            'cublas': 'libcublas.so.*[0-9]',
            'cudnn': 'libcudnn.so.*[0-9]',
            'cuda_nvrtc': 'libnvrtc.so.*[0-9]',
            'cuda_runtime': 'libcudart.so.*[0-9]',
            'cuda_cupti': 'libcupti.so.*[0-9]',
            'cufft': 'libcufft.so.*[0-9]',
            'curand': 'libcurand.so.*[0-9]',
            'cusolver': 'libcusolver.so.*[0-9]',
            'cusparse': 'libcusparse.so.*[0-9]',
            'nccl': 'libnccl.so.*[0-9]',
            'nvtx': 'libnvToolsExt.so.*[0-9]',
        }
```

大模型可观测性

3.2 基于 eBPF 的 CUDA 事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 定位 Pytorch 的 CUDA 运行库

PyTorch 的 CUDA 运行库通常以 pip 包的形式存在。可以通过查看 PyTorch 源码中的 _load_global_deps() 函数来确定这些库的位置。

02 定位 C-CUDA 运行库

c-cuda 编程运行库通常位于全局 CUDA 安装目录 /usr/local/cuda/lib64/，包括常见的 CUDA 运行时库和工具库。

03 使用 eBPF 的 uprobe 注入

```
(base) msc@Tokisakix-SCC:~$ ls -lah /usr/local/cuda-12.5/targets/x86_64-linux/lib
总计 4.2G
drwxr-xr-x 4 root root 4.0K  6月  7 15:54 .
drwxr-xr-x 4 root root 4.0K  6月  7 15:06 ..
drwxr-xr-x 6 root root 4.0K  6月  7 15:06 cmake
lrwxrwxrwx 1 root root   19  6月  7 15:54 libaccinj64.so -> libaccinj64.so.12.5
lrwxrwxrwx 1 root root   22  6月  7 15:54 libaccinj64.so.12.5 -> libaccinj64.so.12.5.39
-rw-r--r-- 1 root root 2.4M  4月 16 13:16 libaccinj64.so.12.5.39
-rw-r--r-- 1 root root 1.5M  4月 16 13:11 libcheckpoint.so
lrwxrwxrwx 1 root root   17  6月  7 15:54 libcublasLt.so -> libcublasLt.so.12
lrwxrwxrwx 1 root root   26  6月  7 15:54 libcublasLt.so.12 -> ./libcublasLt.so.12.5.2.13
-rw-r--r-- 1 root root 426M  4月 16 10:52 libcublasLt.so.12.5.2.13
-rw-r--r-- 1 root root 684M  4月 16 10:52 libcublasLt_static.a
lrwxrwxrwx 1 root root   15  6月  7 15:54 libcublas.so -> libcublas.so.12
lrwxrwxrwx 1 root root   24  6月  7 15:54 libcublas.so.12 -> ./libcublas.so.12.5.2.13
-rw-r--r-- 1 root root 100M  4月 16 10:52 libcublas.so.12.5.2.13
-rw-r--r-- 1 root root 167M  4月 16 10:52 libcublas_static.a
-rw-r--r-- 1 root root 1.6M  4月 16 10:49 libcudadevrt.a
lrwxrwxrwx 1 root root   15  6月  7 15:54 libcudart.so -> libcudart.so.12
lrwxrwxrwx 1 root root   20  6月  7 15:54 libcudart.so.12 -> libcudart.so.12.5.39
-rw-r--r-- 1 root root 696K  4月 16 10:49 libcudart.so.12.5.39
-rw-r--r-- 1 root root 1.4M  4月 16 10:49 libcudart_static.a
lrwxrwxrwx 1 root root   14  6月  7 15:54 libcufft.so -> libcufft.so.11
lrwxrwxrwx 1 root root   21  6月  7 15:54 libcufft.so.11 -> libcufft.so.11.2.3.18
-rw-r--r-- 1 root root 264M  4月 16 10:58 libcufft.so.11.2.3.18
-rw-r--r-- 1 root root 300M  4月 16 10:58 libcufft_static.a
-rw-r--r-- 1 root root 299M  4月 16 10:58 libcufft_static_nocallback.a
lrwxrwxrwx 1 root root   15  6月  7 15:54 libcufftw.so -> libcufftw.so.11
lrwxrwxrwx 1 root root   22  6月  7 15:54 libcufftw.so.11 -> libcufftw.so.11.2.3.18
-rw-r--r-- 1 root root 953K  4月 16 10:58 libcufftw.so.11.2.3.18
-rw-r--r-- 1 root root  91K  4月 16 10:58 libcufftw_static.a
```

大模型可观测性

3.2 基于 eBPF 的 CUDA 事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 定位 Pytorch 的 CUDA 运行库

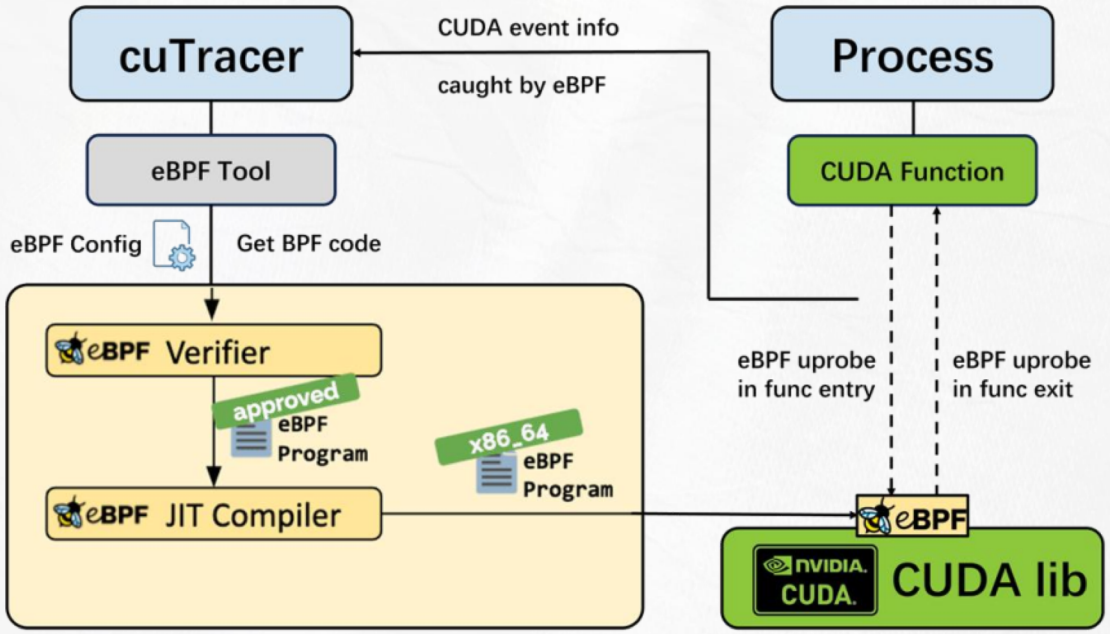
PyTorch 的 CUDA 运行库通常以 pip 包的形式存在。可以通过查看 PyTorch 源码中的 _load_global_deps() 函数来确定这些库的位置。

02 定位 C-CUDA 运行库

c-cuda 编程运行库通常位于全局 CUDA 安装目录 /usr/local/cuda/lib64/，包括常见的 CUDA 运行时库和工具库。

03 使用 eBPF 的 uprobe 注入

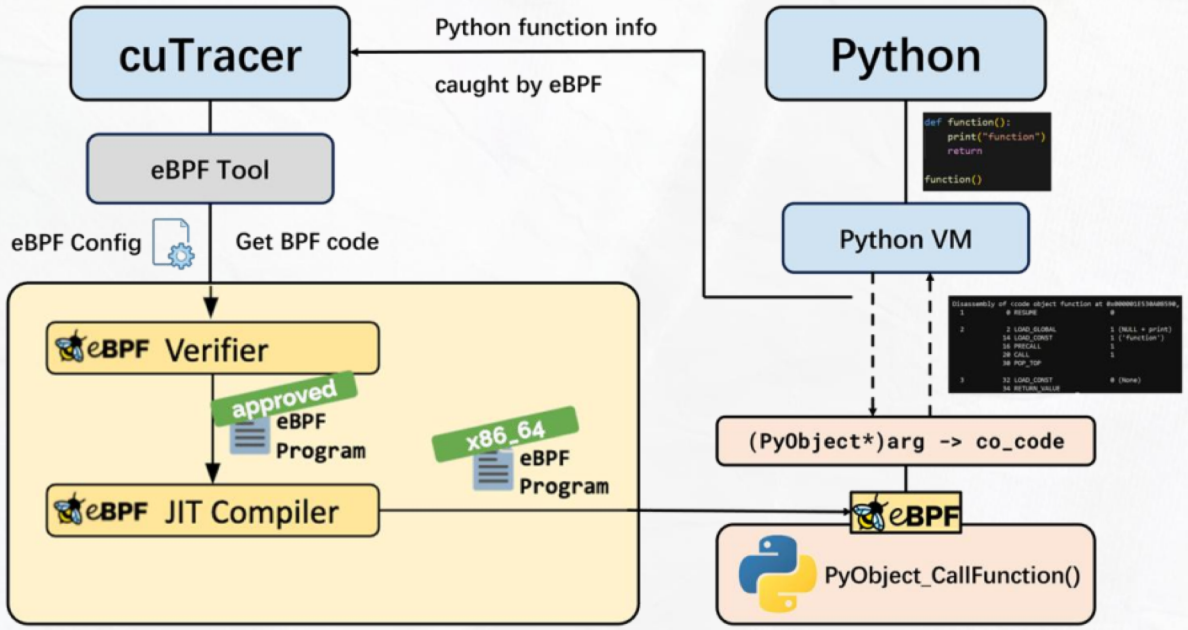
在确定了动态库的位置后，只需要在 CUDA 库函数的入口或出口插入探测点，就可以捕获函数调用的相关信息，实时监测 CUDA 运行库中的关键函数调用。



大模型可观测性

3.3 基于 eBPF 的 Python 层事件追踪

Implementation



Python虚拟机原理

Python 是一个基于栈式虚拟机的解释器,简单来讲，Python 的虚拟机通过解析和执行字节码来运行 Python 代码。

通过查阅 Python 官方文档，我们发现 Python 虚拟机有一个名为 PyObject_CallFunction 的函数，该函数用于处理 Python 字节码中的函数调用。

我们进一步了解到， PyObject_CallFunction 函数在处理函数调用时接受多个参数，其中第一个参数是一个 PyObject* 类型的指针，指向被调用的函数对象。通过分析这个参数，我们可以获取被调用函数的名称及其他相关信息。

大模型可观测性

3.3 基于 eBPF 的 Python 层事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

使用 eBPF 进行函数调用追踪

eBPF 提供了一个 PT_REGS_PARM 宏，用于获取函数调用的参数。我们可以利用这个功能，在 PyObject_CallFunction 调用时获取第一个参数，并解析出函数的名称。

右侧是具体实现的代码。

通过右边的 eBPF 程序，我们可以在函数调用的入口和退出时分别记录调用的函数名及其对应的线程 ID，从而实现对 Python 函数调用情况的监控。

Code 3.3 使用 eBPF 进行函数调用追踪

```
1.  #include <python.h>
2.
3.  // 函数调用入口追踪
4.  int PyCallFuncEntry(struct pt_regs *ctx, PyObject* arg1) {
5.      // 获取当前线程 ID
6.      u64 tid = bpf_get_current_pid_tgid();
7.      // 从参数 arg1 中解析出被调用函数的名称
8.      const char* func_name = PyUnicode_AsUTF8(((PyFunctionObject*)arg1)->func_nam
    e);
9.      // 记录函数调用的开始时间和函数名称
10.     bpf_trace_printk("%d B %s\\n", tid, func_name);
11.     return 0;
12. }
13.
14. // 函数调用退出追踪
15. int PyCallFuncExit(struct pt_regs *ctx, PyObject* arg1) {
16.     // 获取当前线程 ID
17.     u64 tid = bpf_get_current_pid_tgid();
18.     // 从参数 arg1 中解析出被调用函数的名称
19.     const char* func_name = PyUnicode_AsUTF8(((PyFunctionObject*)arg1)->func_nam
    e);
20.     // 记录函数调用的结束时间和函数名称
21.     bpf_trace_printk("%d E %s\\n", tid, func_name);
22.     return 0;
23. }
```

大模型可观测性

3.4 基于 eBPF 的 Torch 算子层事件追踪

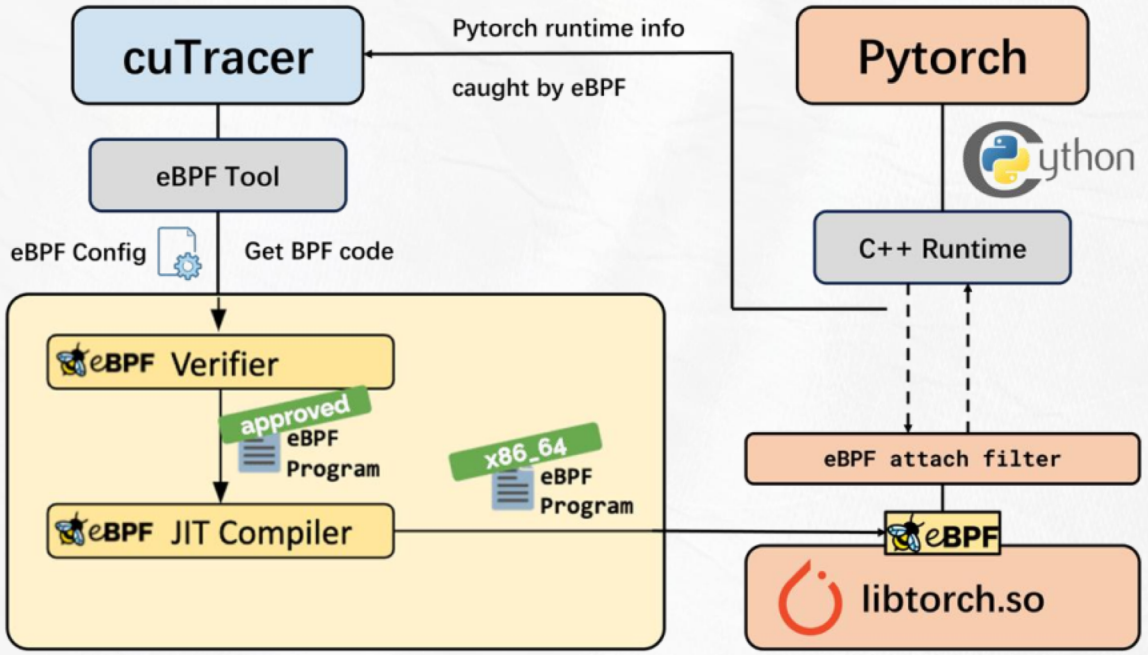
Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 查找并逆向运行库

02 学习 C++ 和 Torch 的混合开发

03 使用 eBPF 进行事件追踪



大模型可观测性

3.4 基于 eBPF 的 Torch 算子层事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 查找并逆向运行库

首先，我们需要找到 Torch 的运行库，并且进行逆向分析，列出库中包含的符号。得到的符号大部分是 C++ 混淆符号，直接识别目标函数变得困难。

02 学习 C++ 和 Torch 的混合开发

03 使用 eBPF 进行事件追踪

```
(base) msc@Tokisakix-SCC:~/Desktop/project2210132-239561$ ls -lah /ho
总计 1.5G
drwxrwxr-x  2 msc msc 4.0K  6月  7 10:38 .
drwxrwxr-x 60 msc msc 4.0K  7月 15 19:15 ..
-rwxrwxr-x  1 msc msc 641K  6月  7 10:38 libc10_cuda.so
-rwxrwxr-x  1 msc msc 1.3M  6月  7 10:38 libc10.so
-rwxrwxr-x  1 msc msc  23K  6月  7 10:38 libcaffe2_nvrtc.so
-rwxrwxr-x  1 msc msc  42M  6月  7 10:38 libcusparseLt-f80c68d1.so.0
-rwxrwxr-x  1 msc msc 166K  6月  7 10:38 libgomp-a34b3233.so.1
-rwxrwxr-x  1 msc msc  52K  6月  7 10:38 libshm.so
-rwxrwxr-x  1 msc msc 460M  6月  7 10:38 libtorch_cpu.so
-rwxrwxr-x  1 msc msc  82M  6月  7 10:38 libtorch_cuda_linalg.so
-rwxrwxr-x  1 msc msc 841M  6月  7 10:38 libtorch_cuda.so
-rwxrwxr-x  1 msc msc  21K  6月  7 10:38 libtorch_global_deps.so
-rwxrwxr-x  1 msc msc  25M  6月  7 10:38 libtorch_python.so
-rwxrwxr-x  1 msc msc 250K  6月  7 10:38 libtorch.so
(base) msc@Tokisakix-SCC:~/Desktop/project2210132-239561$
```

```
1                  U abort@GLIBC_2.2.5
2                  U access@GLIBC_2.2.5
3                  U acos@GLIBC_2.2.5
4                  U acosf@GLIBC_2.2.5
5                  U acosh@GLIBC_2.2.5
6                  U acoshf@GLIBC_2.2.5
7  000000000367a460 T aoti_torch_create_cuda_guard
8  000000000367a5b0 T aoti_torch_create_cuda_stream_guard
9  0000000001228510 T aoti_torch_cuda_abs_
10 0000000001271ca0 T aoti_torch_cuda__adaptive_avg_pool2d
11 00000000012262e0 T aoti_torch_cuda__adaptive_avg_pool2d_backward
```

大模型可观测性

3.4 基于 eBPF 的 Torch 算子层事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 查找并逆向运行库

首先，我们需要找到 Torch 的运行库，并且进行逆向分析，列出库中包含的符号。得到的符号大部分是 C++ 混淆符号，直接识别目标函数变得困难。

02 学习 C++ 和 Torch 的混合开发

为了找到 Torch 算子调用的函数，我们使用 C++ 编写了一些程序，并对编译后的 libtorch 程序进行了逆向分析，最后定位到 Torch 算子的符号和运行库位置。

03 使用 eBPF 进行事件追踪

Code 3.4 C++ 和 PyTorch 混合开发示例程序

```cpp
#include <torch/torch.h>
#include <iostream>

struct Net : torch::nn::Module {
    Net() {
        conv1 = register_module("conv1", torch::nn::Conv2d(1, 32, 5));
        pool = register_module("pool", torch::nn::MaxPool2d(2));
        conv2 = register_module("conv2", torch::nn::Conv2d(32, 64, 5));
        fc1 = register_module("fc1", torch::nn::Linear(64 * 5 * 5, 256));
        fc2 = register_module("fc2", torch::nn::Linear(256, 10));
    }

    torch::Tensor forward(torch::Tensor x) {
        x = pool(torch::relu(conv1(x)));
        x = pool(torch::relu(conv2(x)));
        x = x.view({-1, 64 * 5 * 5});
        x = torch::relu(fc1(x));
        x = fc2(x);
        return torch::log_softmax(x, 1);
    }

    torch::nn::Conv2d conv1{nullptr}, conv2{nullptr};
    torch::nn::MaxPool2d pool{nullptr};
    torch::nn::Linear fc1{nullptr}, fc2{nullptr};
};
```

大模型可观测性

3.4 基于 eBPF 的 Torch 算子层事件追踪

Implementation

中山大學 SUN YAT-SEN UNIVERSITY

01 查找并逆向运行库

首先，我们需要找到 Torch 的运行库，并且进行逆向分析，列出库中包含的符号。得到的符号大部分是 C++ 混淆符号，直接识别目标函数变得困难。

02 学习 C++ 和 Torch 的混合开发

为了找到 Torch 算子调用的函数，我们使用 C++ 编写了一些程序，并对编译后的 libtorch 程序进行了逆向分析，最后定位到 Torch 算子的符号和运行库位置。

03 使用 eBPF 进行事件追踪

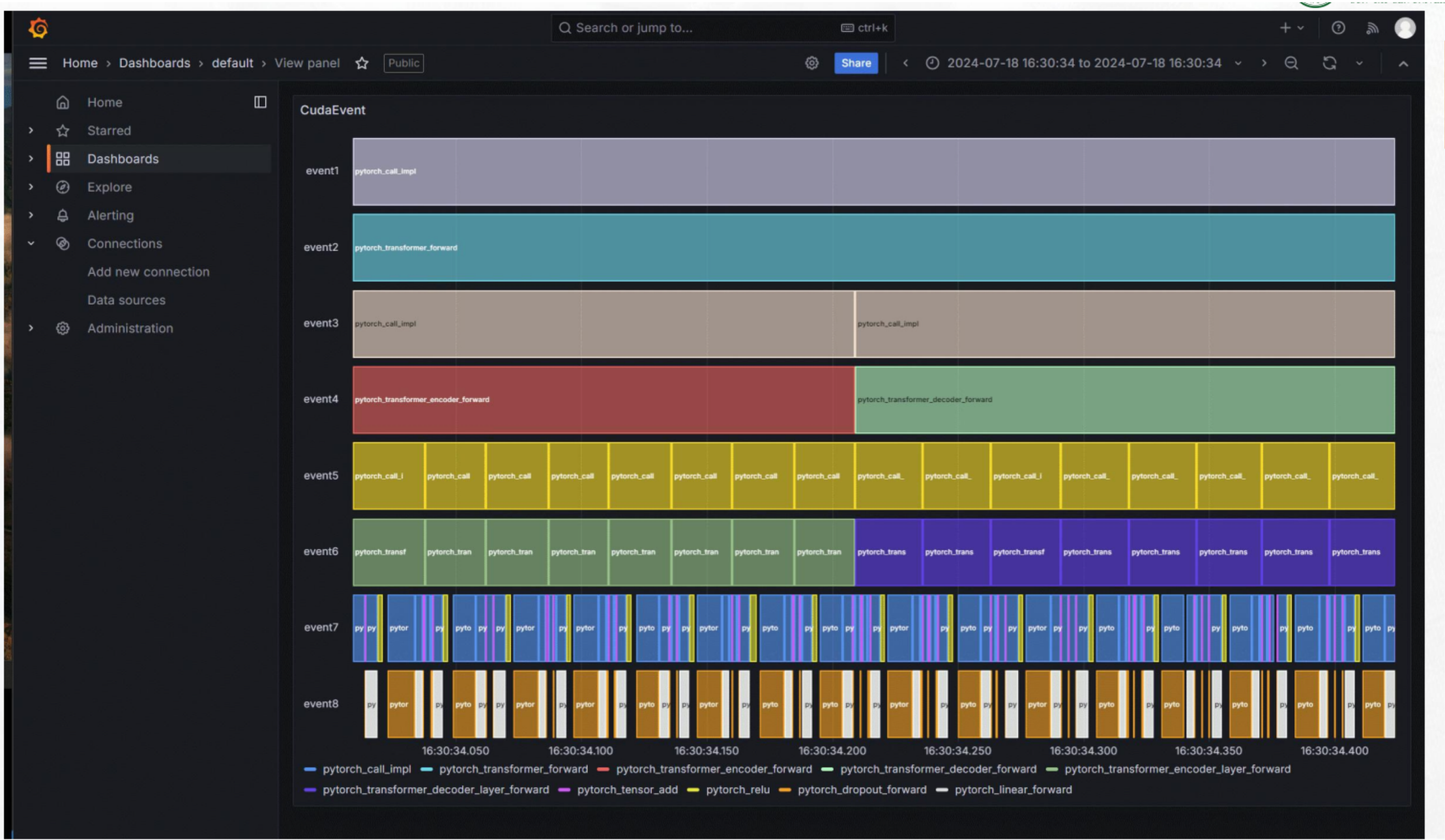
首先通过逆向分析，确定需要追踪的 Torch 算子函数符号。

然后编写 eBPF 程序，利用 eBPF 的 `uprobe` 功能挂钩到目标函数入口和出口，记录函数调用信息。

Code 3.5 使用 eBPF 进行事件追踪

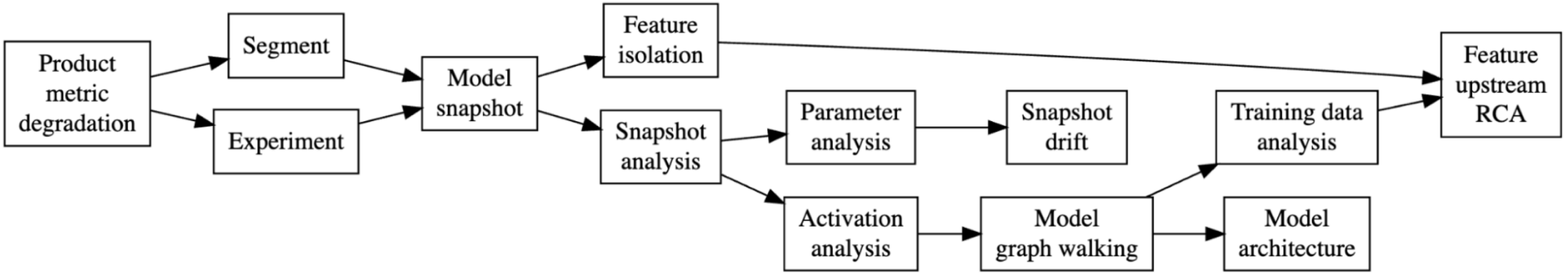
```
// 定义数据结构
struct torch_op_log {
    u64 pid_tgid;
    u64 timestamp;
    char func_name[128];
};
// 定义 eBPF 哈希表
BPF_HASH(op_logs, u64, struct torch_op_log);
// 函数调用入口追踪
int torch_op_entry(struct pt_regs *ctx) {
    u64 pid_tgid = bpf_get_current_pid_tgid();
    u64 ts = bpf_ktime_get_ns();

    struct torch_op_log log = {};
    log.pid_tgid = pid_tgid;
    log.timestamp = ts;
    // 获取函数名称（假设函数名称在符号表中）
    bpf_probe_read_user_str(&log.func_name, sizeof(log.func_name), (void *)PT_REGS_PARM1(ctx));

    op_logs.update(&pid_tgid, &log);

    bpf_trace_printk("Torch op start: %s\\n", log.func_name);
    return 0;
}
// 函数调用退出追踪
int torch_op_exit(struct pt_regs *ctx) {
    u64 pid_tgid = bpf_get_current_pid_tgid();
    struct torch_op_log *log = op_logs.lookup(&pid_tgid);

    if (log) {
        u64 ts = bpf_ktime_get_ns();
        bpf_trace_printk("Torch op end: %s, duration: %llu\\n", log->func_name, ts - log->timestamp);
        op_logs.delete(&pid_tgid);
    }
```

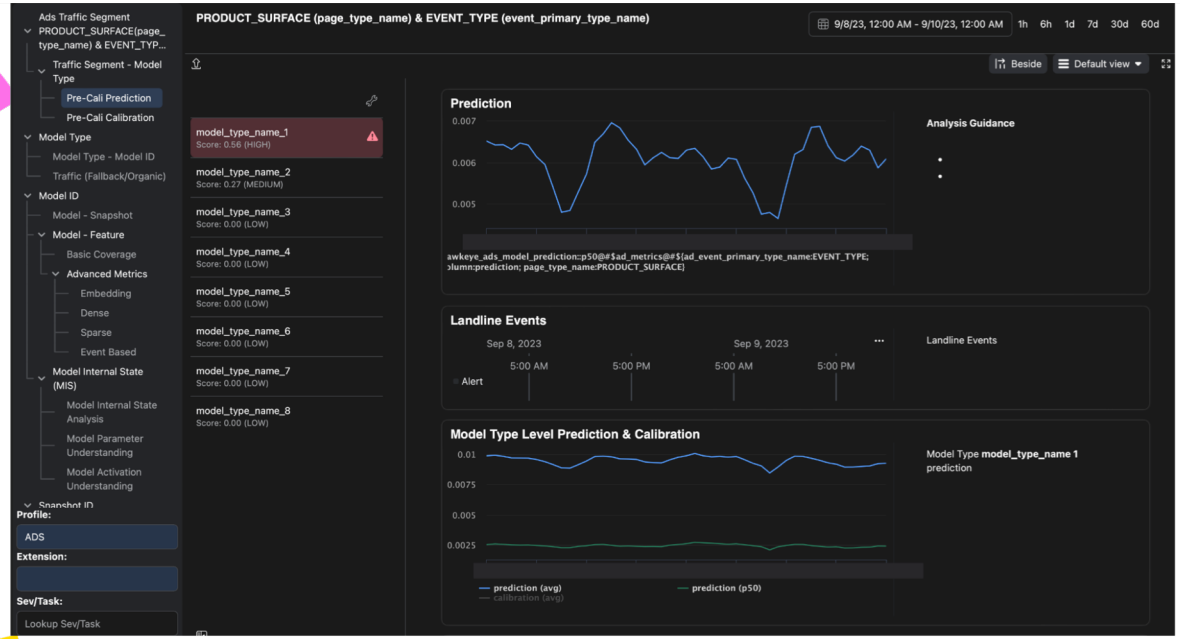
大模型可观测性

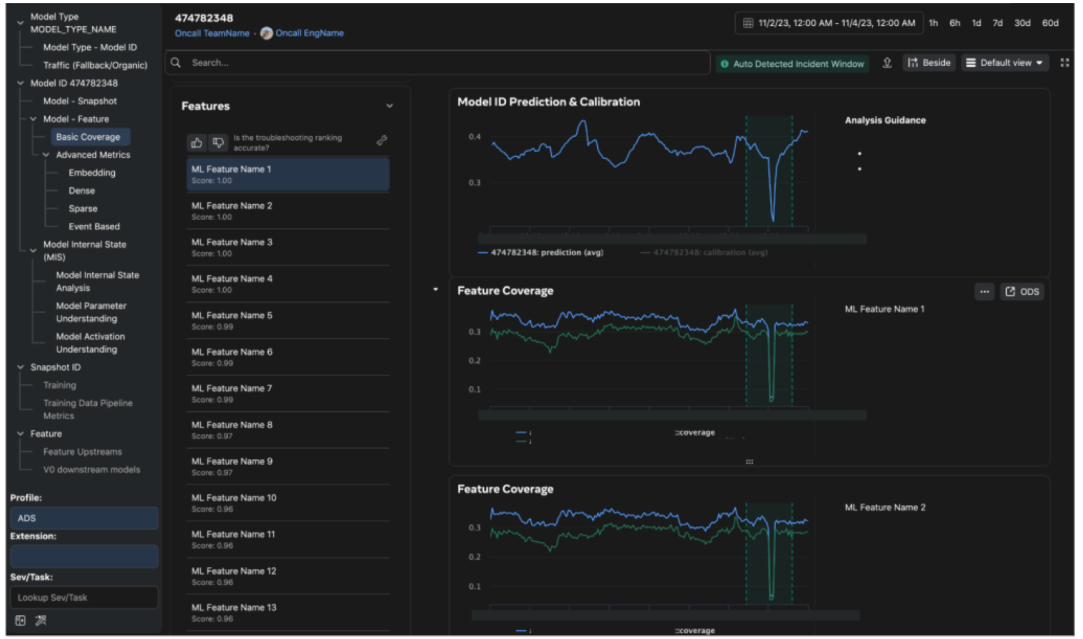

Search or jump to...
ctrl+k
Home › Dashboards › default › View panel
Public
Share
2024-07-18 16:30:34 to 2024-07-18 16:30:34
Home
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
CudaEvent
event1
event2
event3
event4
event5
event6
event7
event8
16:30:34.050
16:30:34.100
16:30:34.150
16:30:34.200
16:30:34.250
16:30:34.300
16:30:34.350
16:30:34.400
pytorch_call_impl
pytorch_transformer_forward
pytorch_transformer_encoder_forward
pytorch_transformer_decoder_forward
pytorch_transformer_encoder_layer_forward
pytorch_transformer_decoder_layer_forward
pytorch_tensor_add
pytorch_relu
pytorch_dropout_forward
pytorch_linear_forward

大模型故障定因

✓ 知识图谱驱动的渐进式根因定位方法



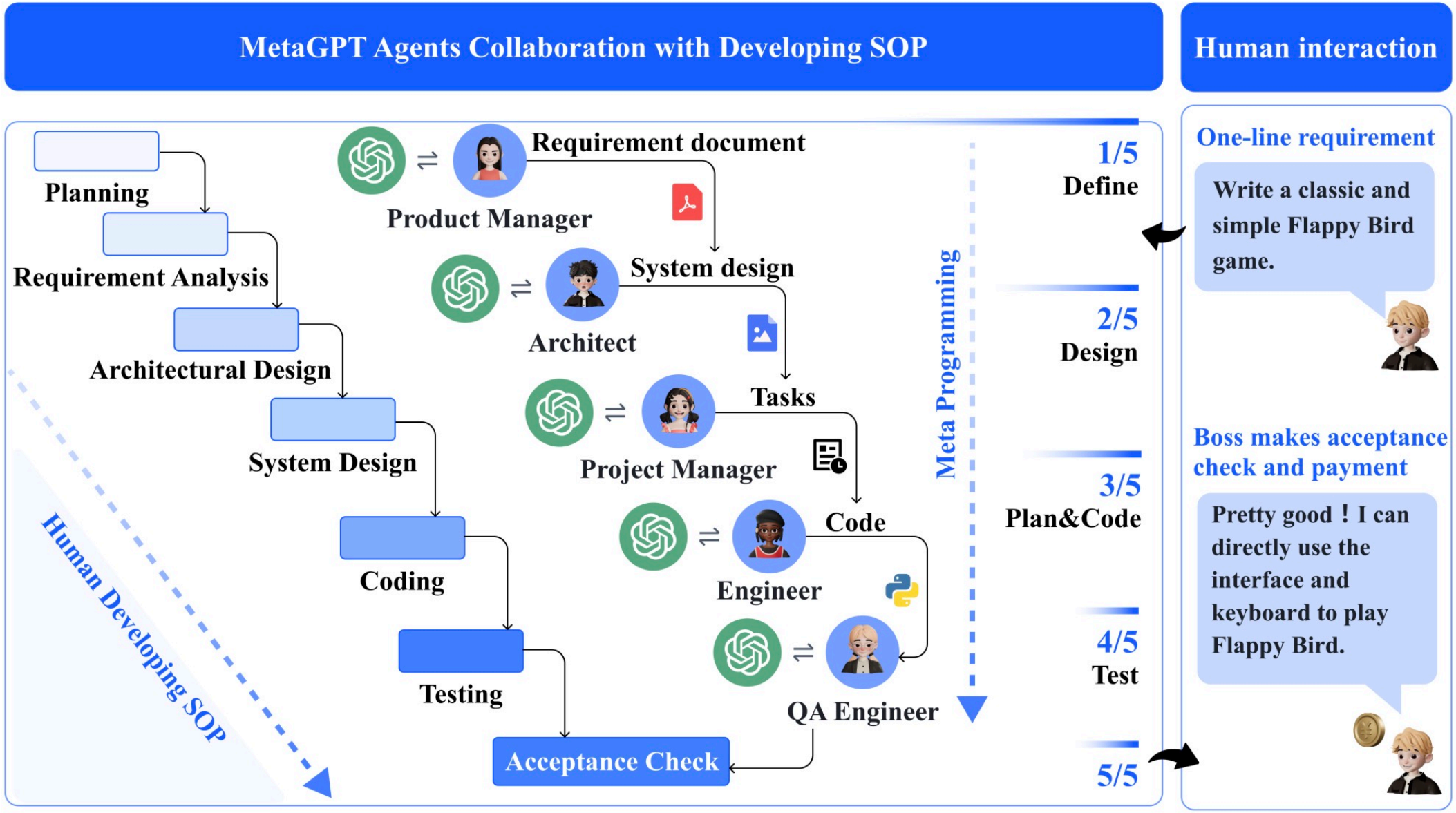




https://engineering.fb.com/2023/12/19/data-infrastructure/hawkeye-ai-debugging-meta/

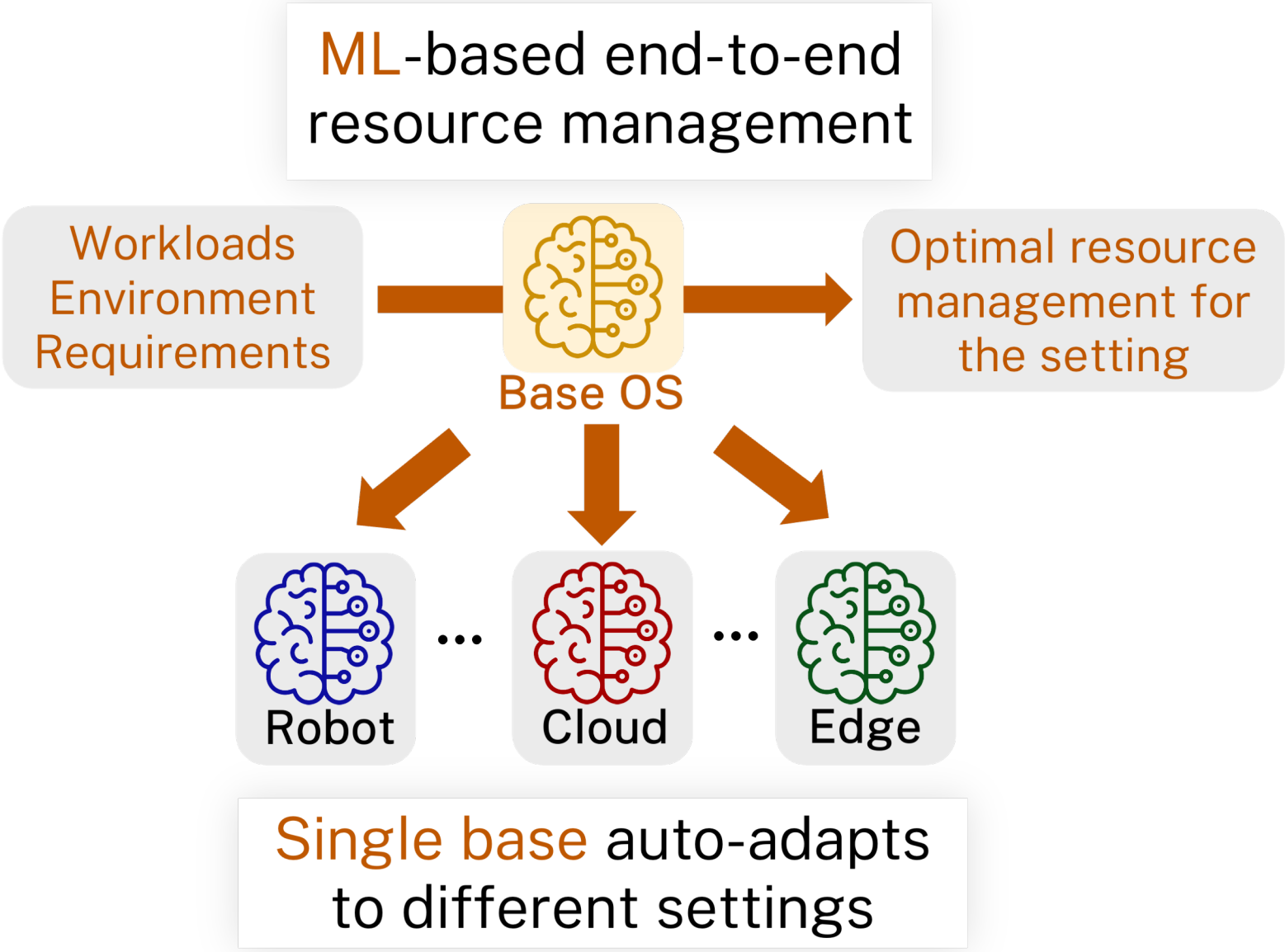
目录

多智能体协作的智能运维



METAGPT: META PROGRAMMING FOR A MULTI-AGENT COLLABORATIVE FRAMEWORK

大模型驱动的操作系统


ML-based end-to-end resource management
Workloads Environment Requirements
Base OS
Optimal resource management for the setting
Robot
Cloud
Edge
Single base auto-adapts to different settings

确定性运维

从“基本运维”能力
迈向“确定性运维”能力

无序向有序进阶

第一级 基本运维

组织

无严格定义的流程角色和职责，依赖个人经验。

流程

无严格定义的流程和步骤，靠事件驱动，以及个人经验，缺乏计划性。

确定性运维

- 无 ITSM 工具；
- 无运维作业工具。

流程管控向软件工程转型

第二级 标准化运维

组织

基于ITIL或其他运维标准定义的流程角色和职责。

流程

运维流程覆盖关键的告警、事件、变更活动。

确定性运维

- 有 ITSM 工具，能满足基本运维管理需要；
- 有烟囱式运维作业工具，尚未形成智能运维体系。

软件工程向可用性架构转型

第三级 SRE转型

组织

开展SRE变革，用软件工程的方法解决问题。

流程

- 优化运维流程
- 初步建立 SRE 流程，强调软件工程方法的落地。

确定性运维

- 开始基于故障模式设计产品高可用架构；
- 开始主动设计 SLI/SLO；
- 初步构建智能运维平台。

向确定性质量结果进阶

第四级 初步确定性

组织

- 职责分工：SRE、开发、产品管理团队协同承担 SLO 目标；
- 合作模式：SRE 作为可用性专家深度参与到产品设计和上线活动

流程

SRE深入参与到前端的产品设计，落实可用性架构

确定性运维

- 部分高可用架构；
- 完整的智能运维平台(包括监测、自动修复、自动巡检、自动变更等)；
- 资源健康自动化检查能力。

第五级 高度确定性

组织

- 成员具备四大关键能力：软件编码、工程方法、自动化、SLO。

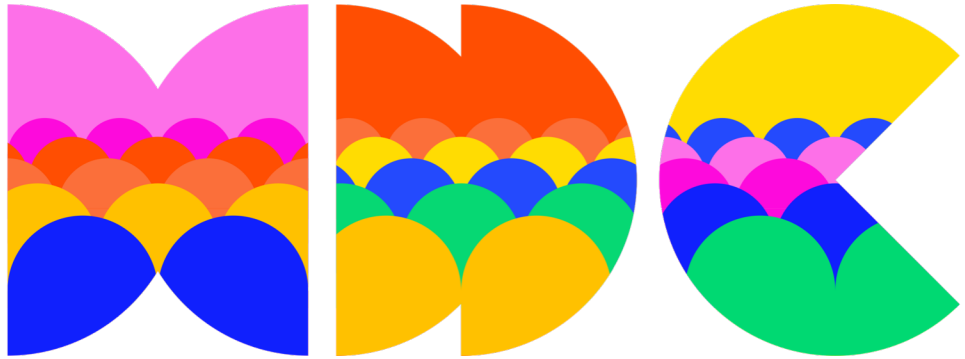
流程

质量文化融入
所有技术人员的血液

确定性运维

- 全面高可用架构；
- 全面 SLI/SLO 设计；
- 高度确定性恢复能力；
- AIOPS；
- 全面数据运营；
- 全面动态风控；全面混沌工程；
- 全面自动化变更。

Thank you.



把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.