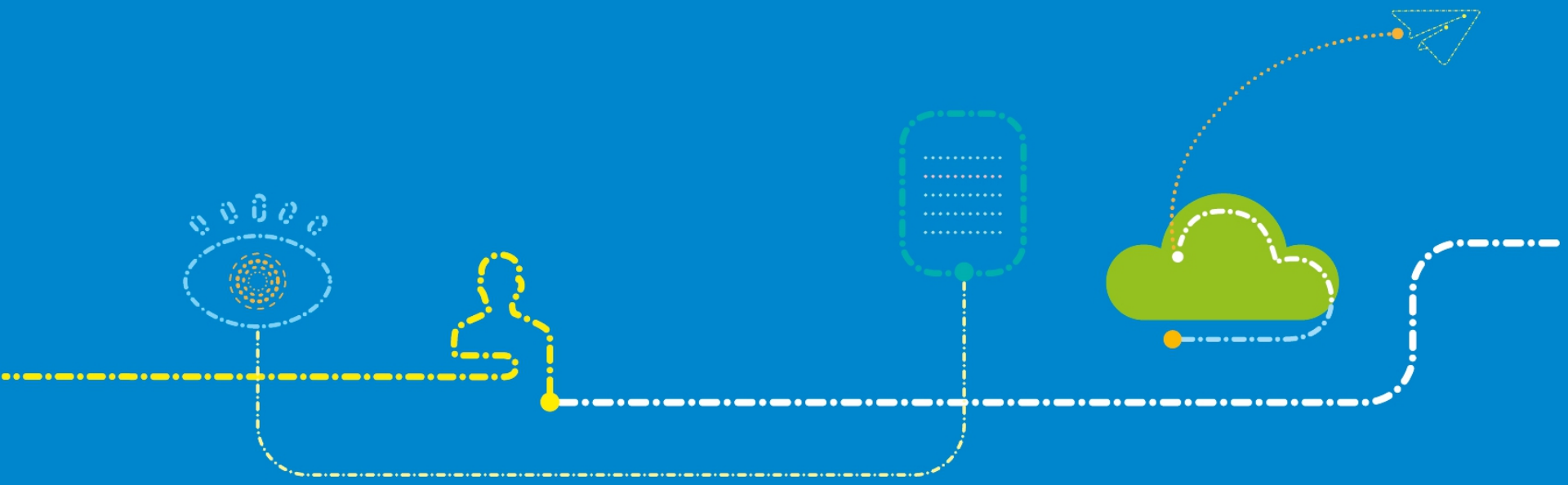
ZTE中兴

# Adlik

## 加速深度学习推理工具



# 目录

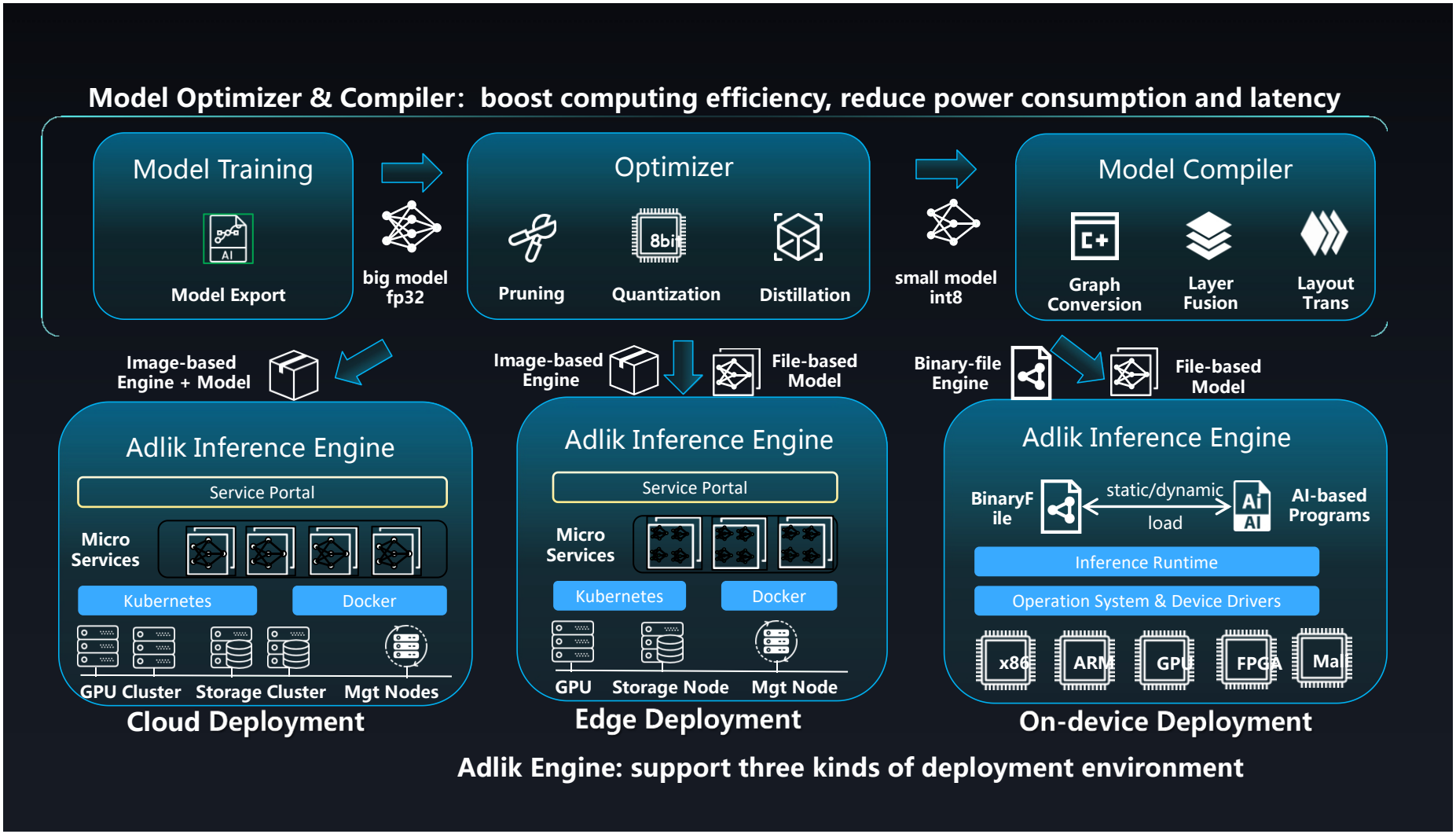


# 概述

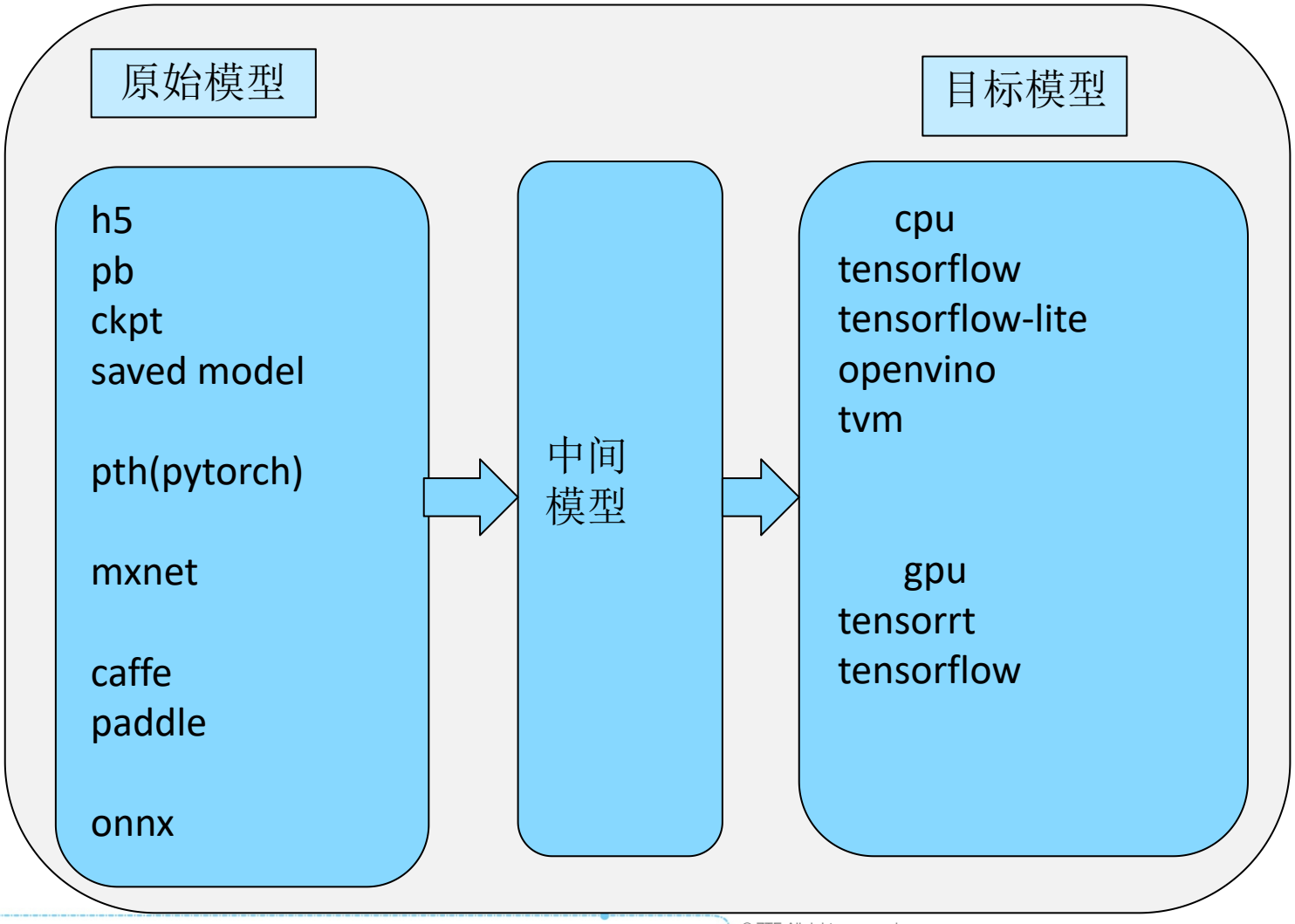
- Adlik主要致力于加速深度学习和机器学习推理的执行，提供便于集成的开发框架，来帮助用户快速实现特定场景需求。

- 主要由两部分组成：编译器、推理引擎

# Adlik 架构

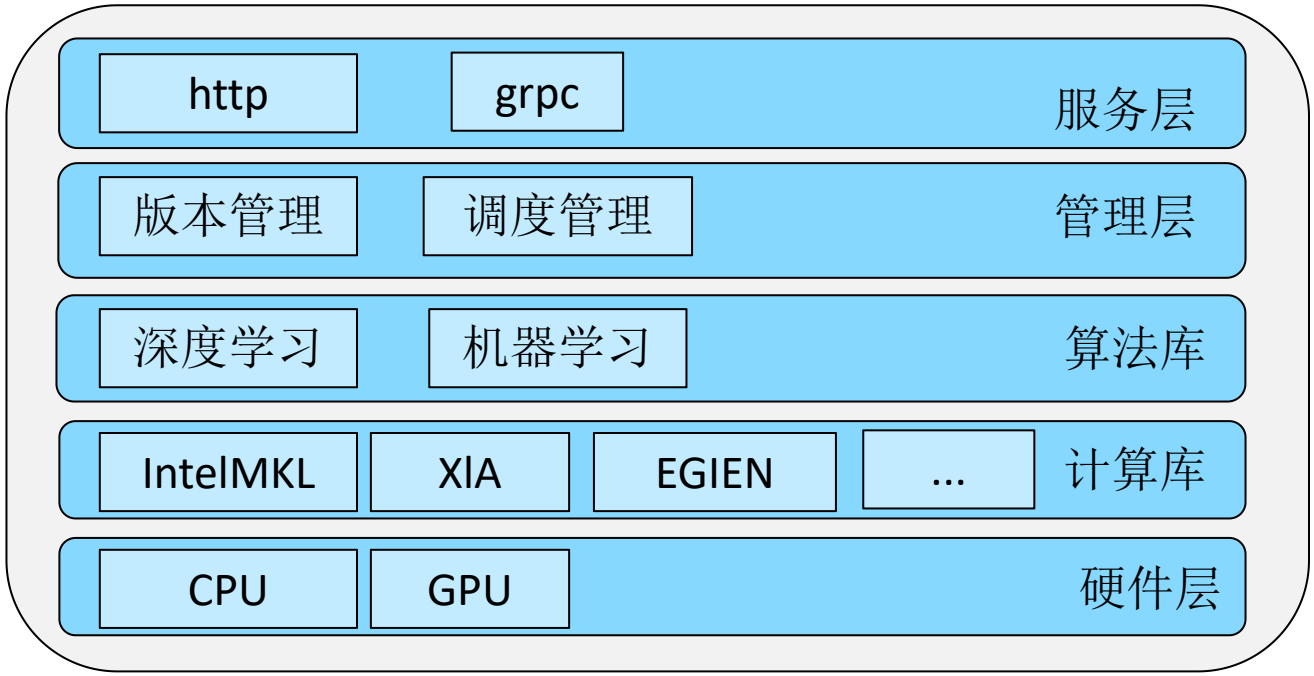


# Adlik编译器路线图



# Adlik运行时总体架构

- 服务层：提供统一的对外grpc，同时提供扩展api，方便用户自定义服务接口
- 管理层：提供对模型和任务的各种管理功能，模型的添加|删除|查询|激活，任务的调度协作管理
- 算法层：深度学习、机器学习算法的支持，比如lstm|cnn|regression|xgboost等等，其中机器学习可以根据需要进行定制化
- 基于各类硬件环境的线性代数加速运算库：用户应该可以根据自身需求和硬件环境选取合适底层加速库
- 硬件层：可以部署在多种硬件环境中，比如云环境，基站等，还可以支持多种CPU。



# 支持特性

- 功能特性：
  - 支持多种深度学习框架
  - 支持自定义机器学习算法
  - 支持grpc接口模型管理
  - 支持grpc接口推理
  - 支持sdk接口推理
  - 支持多模型多实例调度器

- 部署特性：
  - 支持docker部署
  - 支持exe可执行文件启动服务
  - 支持含服务层运行时so方式加载
  - 支持无服务层sdk so方式加载

# 支持特性

## 支持grpc接口推理(由portobuf定义)

深度学习推理:

- rpc predict(PredictRequest) returns (PredictResponse);
- rpc getModelMeta(GetModelMetaRequest) returns (GetModelMetaResponse);

机器学习推理:

- rpc create(CreateTaskRequest) returns (CreateTaskResponse);
- rpc countIdleInstance(ModelStatusRequest) returns (ModelStatusResponse);

# 支持特性

## 支持grpc接口模型管理(由portobuf定义)

- rpc addModel(ModelOperateRequest) returns (ModelOperateResponse);
- rpc addModelVersion(ModelOperateRequest) returns (ModelOperateResponse);
- rpc deleteModel(ModelOperateRequest) returns (ModelOperateResponse);
- rpc deleteModelVersion(ModelOperateRequest) returns (ModelOperateResponse);
- rpc activateModel(ModelOperateRequest) returns (ModelOperateResponse);
- rpc queryModel(ModelOperateRequest) returns (ModelOperateResponse);

# 支持特性

## grpc接口客户端使用demo c++ client

// create grpc stub

```
const std::string grpcUrl = "localhost:8500";
std::shared_ptr<Channel> channel = grpc::CreateChannel(grpcUrl, grpc::InsecureChannelCredentials());
std::unique_ptr<PredictService::Stub> predictServiceStub(PredictService::NewStub(channel));
```

// get predict result

```
ClientContext predictContext;
PredictRequest predictRequest = make_predict_request(getModelMetaResponse);
PredictResponse predictResponse;
Status predictStatus = predictServiceStub->predict(&predictContext, predictRequest,  &predictResponse);
```

# 部署特性

## docker部署

基于k8s或者手动执行 docker run -d registry.cn-beijing.aliyuncs.com/adlik/serving-tensorflow-cpu:v0.2.0

# 部署特性

## 可执行文件部署：服务进程

```
adlik-serving  --model_base_path=”/model_repos/” --grpc_port=8500
```

# 部署特性

## 含服务层运行时so方式加载

```
int main(int argc, char** argv) {
  auto adlik_handle = loadLibrary("libadlik_serving.so");
  auto ran = loadLibrary("libran_demo.so");

  auto start = (void (*)(int argc, const char** argv))dlsym(adlik_handle,
"StartServer");
  start(argc, (const char**)argv);

  return 0;
}
```

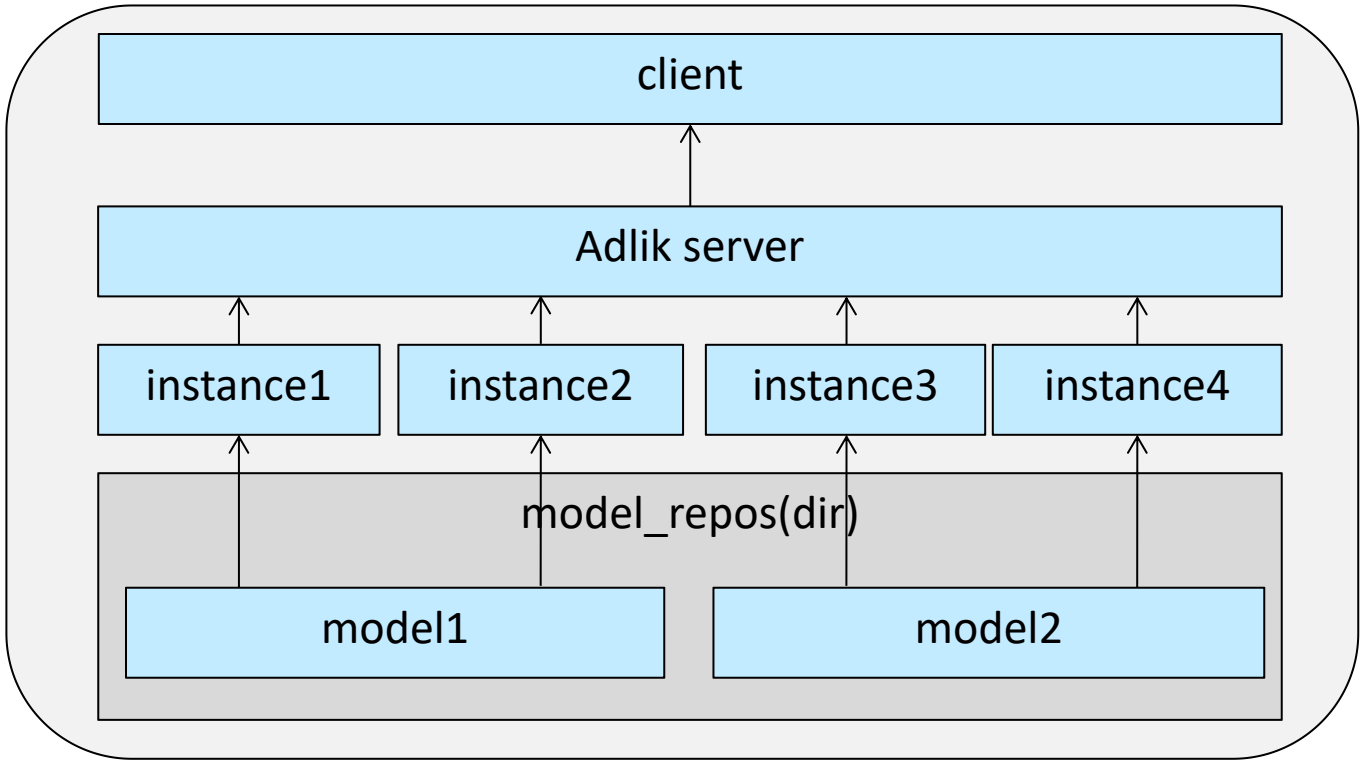
# 部署特性

## c++ sdk 推理 接口

```
struct InferModel {
  virtual ~InferModel() {
  }

  virtual cub::StatusWrapper predict(const PredictRequestProvider& req, PredictResponseProvider& rsp) = 0;

  static cub::StatusWrapper create(const std::string& modelPath,
                                   const std::map<std::string, std::string>& condigMap,
                                   std::unique_ptr<InferModel>* model);
};

std::map<pthread_t, std::string>& GetThreadRegistry();
```

# 支持特性

## 支持多模型多实例调度



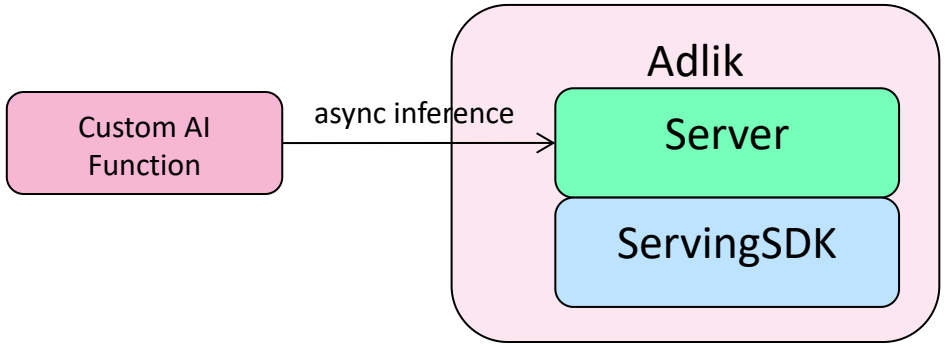
# 编译方案

## 编译容器环境配置：

- 操作系统： ubuntu 16.04.7
- 构建工具：bazel 4.0.0
- 工具链 ：gcc6.5.0
- cpp版本 ：c++14

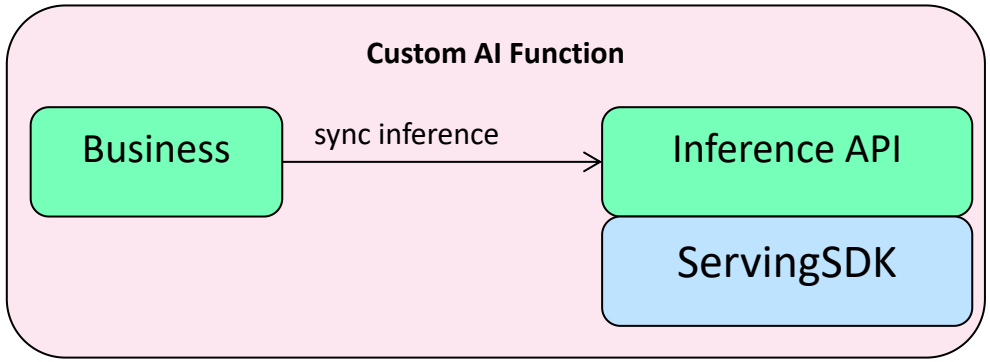
# 应用案例 嵌入式网元解决方案

- 根据推理引擎对实时性要求不同，Adlik存在两种部署方式：
  - 异步方式。推理引擎和AIF（使用AI推理的网络App）是分离的。推理请求通过异步通讯的方式进行交互（进程间通信或网络接口）




Adlik可以作为微服务应用单独部署，也可以使用独立进程方式部署

  - 同步方式。这种方式对实时性要求更高。推理引擎将以SDK的方式呈现。由有推理需求的网络App调用引擎SDK接口函数，对推理引擎进行加载，并通过调用推理函数接口进行推理，应用和推理引擎的交互都在进程内通过函数调用的方式呈现




网络App可以使用SDK API，在一个进程内以函数调用的方式完成推理

# 资源消耗

| 镜像部署 | 文件大小(MB) | 内存大小(空载) |
| --- | --- | --- |
| adlik_serving(Tensorflow-cpu) | 233.9 | 31.1 |
| adlik_serving(Tensorflow-lite) | 78.8 | 9.3 |
| adlik_serving(Openvino) | 136.1 | 14.0 |

| 可执行文件部署 | 文件大小(MB) | 内存大小（空载） |
| --- | --- | --- |
| adlik_serving(Tensorflow-cpu) | 162.0 | 31.1 |
| adlik_serving(Tensorflow-lite) | 14.0 | 9.3 |
| adlik_serving(Openvino) | 12.0(不含依赖so) | 14.0 |

# 问题

1、服务端python训练的模型怎么通过编译变为C++可调用访问的方式

2、编译python训练模型的环境怎么部署；运行的服务环境如何部署

3、客户端如何通过API接口调用服务端服务（包括远程调用和so方式调用）

4、demo示例演示

# 谢谢！

5G 先锋