



# AX ENGINE API 使用说明

文档版本: V1.4

发布日期: 2024/01/18

# 目 录

|                                    |    |
|------------------------------------|----|
| 前 言 .....                          | 5  |
| 修订历史 .....                         | 6  |
| 1 概述 .....                         | 7  |
| 2 功能 .....                         | 8  |
| 2.1 初始化 .....                      | 8  |
| 2.2 反初始化 .....                     | 8  |
| 2.3 创建模型句柄 .....                   | 8  |
| 2.4 销毁模型句柄 .....                   | 8  |
| 2.5 创建模型上下文 .....                  | 9  |
| 2.6 执行推理 .....                     | 9  |
| 2.7 获取模型信息 .....                   | 9  |
| 2.8 获取虚拟 NPU 信息 .....              | 9  |
| 2.9 设置 NPU 亲和性 .....               | 9  |
| 2.10 获取模型 CMM 信息 .....             | 10 |
| 2.11 获取 Engine 版本号 .....           | 10 |
| 2.12 获取 Pulsar2 版本号 .....          | 10 |
| 3 调用流程 .....                       | 11 |
| 4 API 参考 .....                     | 12 |
| AX_ENGINE_Init .....               | 12 |
| AX_ENGINE_Deinit .....             | 13 |
| AX_ENGINE_GetModelType .....       | 14 |
| AX_ENGINE_GetHandleModelType ..... | 15 |

|                                     |           |
|-------------------------------------|-----------|
| AX_ENGINE_CreateHandle.....         | 16        |
| AX_ENGINE_CreateHandleV2 .....      | 18        |
| AX_ENGINE_DestroyHandle .....       | 20        |
| AX_ENGINE_CreateContext .....       | 21        |
| AX_ENGINE_CreateContextV2 .....     | 22        |
| AX_ENGINE_RunSync.....              | 23        |
| AX_ENGINE_RunSyncV2 .....           | 25        |
| AX_ENGINE_GetIOInfo .....           | 27        |
| AX_ENGINE_GetVNPUAttr.....          | 29        |
| AX_ENGINE_SetAffinity.....          | 30        |
| AX_ENGINE_GetAffinity.....          | 31        |
| AX_ENGINE_GetCMMUsage.....          | 32        |
| AX_ENGINE_GetVersion.....           | 33        |
| AX_ENGINE_GetModelToolsVersion..... | 34        |
| <b>5 数据结构 .....</b>                 | <b>35</b> |
| AX_ENGINE_NPU_ATTR_T.....           | 35        |
| AX_ENGINE_NPU_MODE_T.....           | 36        |
| AX_ENGINE_HANDLE_EXTRA_T.....       | 37        |
| AX_ENGINE_IO_T .....                | 38        |
| AX_ENGINE_IO_BUFFER_T .....         | 39        |
| AX_ENGINE_IO_INFO_T.....            | 40        |
| AX_ENGINE_IOMETA_T .....            | 41        |
| AX_ENGINE_MODEL_TYPE_T.....         | 43        |
| AX_ENGINE_CMM_INFO_T .....          | 44        |
| <b>6 错误码.....</b>                   | <b>45</b> |

|                |    |
|----------------|----|
| 7 模型评测工具 ..... | 46 |
| 7.1 参数说明 ..... | 46 |
| 7.2 使用示例 ..... | 47 |

AEXRA CONFIDENTIAL FOR SIPEED

## 权利声明

爱芯元智半导体股份有限公司或其许可人保留一切权利。

非经权利人书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 注意

您购买的产品、服务或特性等应受商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非商业合同另有约定，本公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

# 前言

## 适用产品

AX630C、AX620Q

## 适读人群

- 软件开发工程师
- 技术支持工程师

## 符号与格式定义

| 符号/格式    | 说明   |
|----------|--|
| xxx      | 表示您可以执行的命令行。   |
| 斜体       | 表示变量。如，“安装目录/AX620E_SDK_Vx.x.x/build 目录”中的“安装目录”是一个变量，由您的实际环境决定。 |
| ☞ 说明/备注： | 表示您在使用产品的过程中，我们向您说明的事项。  |
| ！ 注意：    | 表示您在使用产品的过程中，需要您特别注意的事项。   |

## 修订历史

| 文档版本 | 发布时间       | 修订说明                       | 页码   |
|------|------------|----------------------------|------|
| V1.0 | 2023/08/25 | 文档初版                       |      |
| V1.1 | 2023/10/12 | 增加裁剪库说明                    |      |
| V1.2 | 2023/12/22 | 增加获取模型类型的 API 说明           |      |
| V1.3 | 2024/1/11  | 增加/etc/npu.conf 文件使用说明     |      |
|      |            | 说明 stride 字段默认需要清零         |      |
|      |            | 增加错误码处理说明                  |      |
| V1.4 | 2024/1/18  | AX_ENGINE_Init 增加 NPU 模式参数 | 8/12 |

# 1 概述

AXEngine（下文简称 Engine）是在本芯片平台上使用的神经网络模型芯片侧推理计算库，能够完成模型加载到执行的全部推理任务。

随 SDK 发布的有 libax\_engine\_tiny.so、libax\_engine.so 以及对应的静态库(在场景需要 ISP 打开时不推荐使用静态库，会导致和 AI-ISP 冲突)，分别对应全功能的 Engine、裁减了 CPU 算子的 Engine；对于大部分用户，只要容量允许，可以只使用其中之一即可。

SDK 中 Engine 动态的 so 文件的 ELF 字段里 SONAME 字段都是 libax\_engine.so，所以用户链接时只需要链接 libax\_engine.so；SDK 打包 axp 时，eMMC 的 axp 会打包 libax\_engine.so，而 NOR FLASH 的版本会打包 libax\_engine\_tiny.so 且重命名为 libax\_engine.so，所以终端用户一般无需关心这些细节。

## 2 功能

### 2.1 初始化

使用 Engine 时需要首先进行 Engine 初始化，初始化过程会初始化 Engine 内部所依赖使用的各种资源以及存储空间。

在用户使用时需要注意，Engine 只需要全局初始化一次，反复初始化无效。在配置有 AI-ISP 的芯片环境中，一般首先有 AI-ISP 侧应用初始化 Engine 并通过 Engine 初始化 NPU，用户侧不应该修改 AI\_ISP 的 NPU 初始化状态，否则会中断 ISP 通路的工作，引发异常。没有配置 AI-ISP 的解决方案中，NPU 全部算力都划分给用户自主使用，此时用户应自行初始化 Engine 以及 NPU 到希望的模式。

### 2.2 反初始化

Engine 使用完毕后(注意不是模型使用后)，如果准备释放资源不再使用 Engine，应进行反初始化，反初始化会释放 Engine 持有的所有资源。如果需要再次使用时，需要重新初始化。

需要注意的是，如果 ISP 通路使用了 AI-ISP 的相关功能，需要确保反初始化不会影响 ISP 通路的工作状态。

### 2.3 创建模型句柄

通过 axmodel 模型文件可以创建模型推理句柄，用于获取模型 IO 等信息，执行推理任务等接口。

### 2.4 销毁模型句柄

模型使用完毕后应销毁模型句柄，释放与此模型相关的所有资源。

## 2.5 创建模型上下文

可以认为 Engine 的工作是延迟生效的，在创建上下文时才真正初始化模型到 NPU，设置这样的接口可以使用户有机会避免任务流程在一些限制条件下卡加载模型，创建更多的任务优化机会。

需要注意的是，延迟机制并不需要在创建上下文前 axmodel 的内存都必须有效，创建句柄后，即可释放模型内存(对于内存非常紧张的系统，可以考虑使用 mmap 映射模型文件到内存)；调用创建上下文函数后，NPU 将配合 CMM 空间完成模型加载。

## 2.6 执行推理

通过模型句柄，输入数据和输出缓存进行推理，Engine 提供了提供同步推理接口，推理时接此接口会等待推理结束返回状态码，并写回推理结果到指定的 IO 地址。

需要注意的是，此接口允许用户更换用于输入模型数据，以及写回模型推理结果的 IO 地址，以便于用户使用不同帧或其他会产生不同缓冲区地址的情况。

在 1.1.0 以上版本的 Engine 中，支持推理动态 Batch；关于如何转换生成动态 Batch 的模型，请参考模型转换文档；Engine 在进行推理时，只需要填充一个 nBatchSize 字段即可。

还可以获取动态 Batch 的最大 Batch 情况，读取 nMaxBatchSize 即可。

## 2.7 获取模型信息

通过模型句柄获取模型各个输入和输出的信息。

## 2.8 获取虚拟 NPU 信息

通过模型句柄获取模型虚拟 NPU 的信息。在模型编译(即 pulsar2 build 命令行工具)时，可以由 --npu\_mode 参数根据需求指定模型的 VNPU 情况。可以设置为 Enable 或 Disable。

## 2.9 设置 NPU 亲和性

当 NPU 初始化为虚拟 NPU 模式后，允许用户根据任务需要设置模型的亲和性。在开启 AI-

ISP 的情况下，AI-ISP 默认亲和 VNPU0，用户的程序此时需要亲和 VNPU1，避免干扰 AI-ISP 导致丢帧或 pipeline 中断。

设置亲和性和 NPU 初始化是有关的，NPU 初始化时，可以初始化为 DISABLE、ENABLE 模式，其中适合 AI-ISP 的是 ENABLE 模式，此时 AI-ISP 亲和默认亲和 VNPU0，用户侧的 AI 算法需要亲和 VNPU1。当初始化为 DISABLE 模式时，没有 VNPU 概念，此时 NPU 视作是一体的，也就是这时只能进行纯 AI-ISP 任务，或纯用户侧任务；这个模式比较适合类似强 AI 应用类的高算力需求类任务或极端 AI-ISP 效果的需求。

设置亲和时，初始化成 DISABLE 的 NPU 只能亲和到掩码 0b01 核心；同理，ENABLE 模式下，可以设置掩码 0b01 或者 0b10。

需要注意的是，如果 AI-ISP 使能，用户不能亲和 AI-ISP 使用的虚拟 NPU。

## 2.10 获取模型 CMM 信息

用户可以通过 API 获取模型运行时占用的 CMM 信息，这样方便用户规划 CMM 的划分和使用；需要注意的是，CMM 在设计上是有分块大小的，此 CMM 接口获取的是原始 CMM 占用的大小，会比真实大小最多小三个分块。关于分块的情况，请阅读《AX SYS API 文档》获取进一步的信息。

## 2.11 获取 Engine 版本号

此接口会返回 Engine 的版本号，用以快速识别 Engine 的版本。

## 2.12 获取 Pulsar2 版本号

此接口会返回编译工具 Pulsar2 的版本号，用以快速识别编译模型的 Pulsar2 的工具链版本。

# 3 调用流程

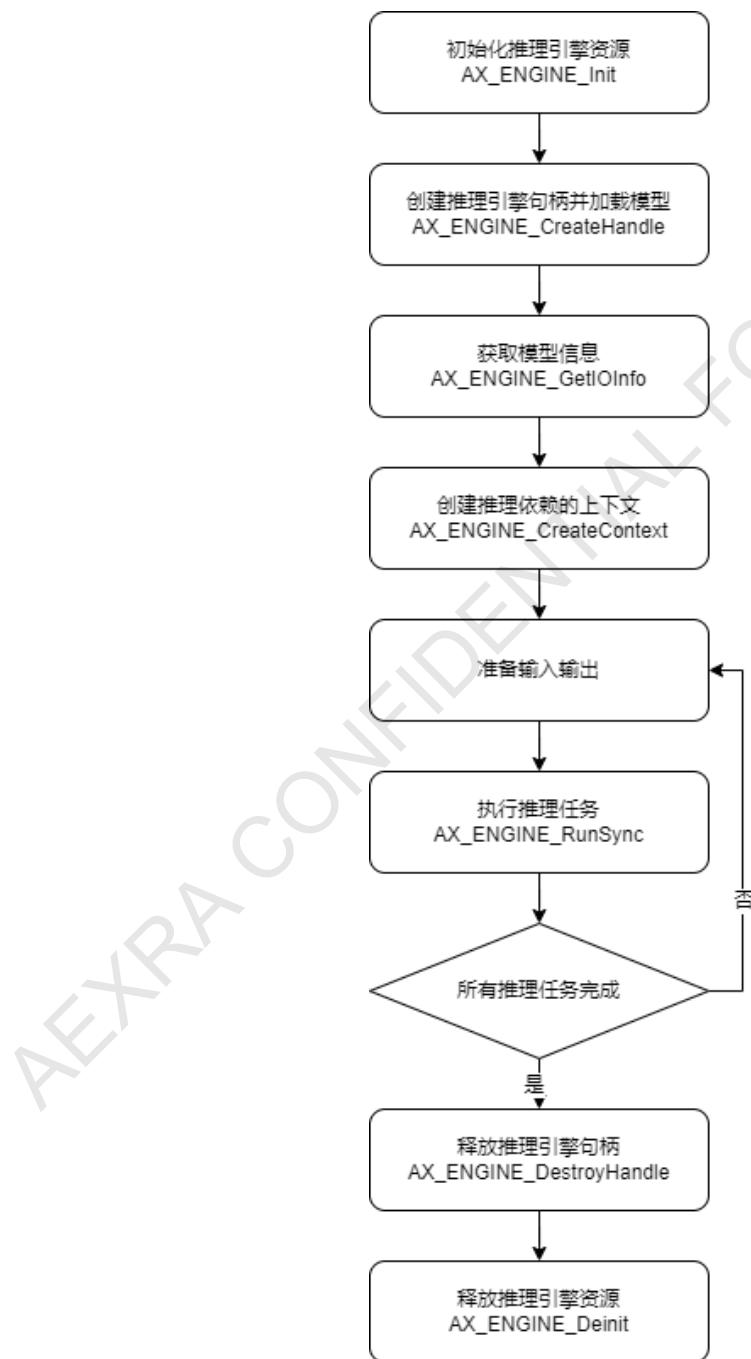


图3-1 调用流程

# 4 API 参考

## AX\_ENGINE\_Init

### 【描述】

初始化 Engine 运行所需要的资源，同时指定 NPU 工作模式。

### 【语法】

```
AX_S32 AX_ENGINE_Init(AX\_ENGINE\_NPU\_ATTR\_T *pNpuAttr);
```

### 【参数】

| 参数名称   | 描述       | 输入/输出 |
|--|----------|-------|
| <a href="#">AX_ENGINE_NPU_ATTR_T</a> *pNpuAttr | NPU 运行模式 | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件：ax\_engine\_api.h
- 库文件：libax\_engine.so

### 【注意】

同一线程中，只能调用一次，退出该线程之前需要与 AX\_ENGINE\_Deinit 成对使用。

### 【相关主题】

[AX\\_ENGINE\\_Deinit](#)

## AX\_ENGINE\_Deinit

### 【描述】

释放 IO 和内存和模型，退出线程。

### 【语法】

```
AX_S32 AX_ENGINE_Deinit(AX_VOID);
```

### 【参数】

无

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 无   | 无        |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

同一线程中，只能调用一次，且需要在 [AX\\_ENGINE\\_Init](#) 调用之后成对使用。

### 【相关主题】

[AX\\_ENGINE\\_Init](#)

## AX\_ENGINE\_GetModelType

### 【描述】

通过读取模型内存 buffer 获取模型类型。

### 【语法】

```
AX_S32 AX_ENGINE_GetModelType(const AX_VOID *pData, AX_U32 nDataSize,  
AX_ENGINE_MODEL_TYPE_T* pModelType);
```

### 【参数】

| 参数名称                               | 描述       | 输入/输出 |
|------------------------------------|----------|-------|
| const AX_VOID *pData               | 模型数据起始地址 | 输入    |
| AX_U32 nDataSize                   | 模型数据长度   | 输入    |
| AX_ENGINE_MODEL_TYPE_T* pModelType | 模型类型枚举   | 输出    |

### 【返回值】

| 返回值 | 描述        |
|-----|-----------|
| 0   | 成功        |
| 非 0 | 失败, 返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

无

### 【相关主题】

[AX\\_ENGINE\\_MODEL\\_TYPE\\_T](#)

[AX\\_ENGINE\\_GetHandleModelType](#)

## AX\_ENGINE\_GetHandleModelType

### 【描述】

通过读取模型内存 buffer 获取模型类型。

### 【语法】

```
AX_S32 AX_ENGINE_GetHandleModelType (AX_ENGINE_HANDLE nHandle,  
AX_ENGINE_MODEL_TYPE_T* pModelType);
```

### 【参数】

| 参数名称                               | 描述     | 输入/输出 |
|------------------------------------|--------|-------|
| AX_ENGINE_HANDLE nHandle           | 模型句柄   | 输入    |
| AX_ENGINE_MODEL_TYPE_T* pModelType | 模型类型枚举 | 输出    |

### 【返回值】

| 返回值 | 描述        |
|-----|-----------|
| 0   | 成功        |
| 非 0 | 失败, 返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

无

### 【相关主题】

[AX\\_ENGINE\\_MODEL\\_TYPE\\_T](#)

[AX\\_ENGINE\\_GetModelType](#)

## AX\_ENGINE\_CreateHandle

### 【描述】

创建推理引擎句柄，加载 axmodel 模型文件数据，axmodel 模型文件的大小就是模型加载后会占用的最大 CMM 空间大小。

### 【语法】

```
AX_S32 AX_ENGINE_CreateHandle(AX_ENGINE_HANDLE *pHandle, const AX_VOID *pData, AX_U32 nDataSize);
```

### 【参数】

| 参数名称                      | 描述       | 输入/输出 |
|---------------------------|----------|-------|
| AX_ENGINE_HANDLE *pHandle | 推理引擎句柄   | 输入    |
| const AX_VOID *pData      | 模型数据起始地址 | 输入    |
| AX_U32 nDataSize          | 模型数据长度   | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件：ax\_engine\_api.h
- 库文件：libax\_engine.so

### 【注意】

需要在 [AX\\_ENGINE\\_Init](#) 调用之后使用。

### 【相关主题】

[AX\\_ENGINE\\_CreateHandleV2](#)

## [AX\\_ENGINE\\_DestroyHandle](#)

AEXRA CONFIDENTIAL FOR SPEED

## AX\_ENGINE\_CreateHandleV2

### 【描述】

创建推理引擎句柄，加载 axmodel 模型文件数据，axmodel 模型文件的大小就是模型加载后会占用的最大 CMM 空间大小。

### 【语法】

```
AX_S32 AX_ENGINE_CreateHandleV2(AX_ENGINE_HANDLE *pHandle, const AX_VOID *pData, AX_U32 nDataSize, AX_ENGINE_HANDLE_EXTRA_T* pExtraParam);
```

### 【参数】

| 参数名称                                  | 描述       | 输入/输出 |
|---------------------------------------|----------|-------|
| AX_ENGINE_HANDLE *pHandle             | 推理引擎句柄   | 输入    |
| const AX_VOID *pData                  | 模型数据起始地址 | 输入    |
| AX_U32 nDataSize                      | 模型数据长度   | 输入    |
| AX_ENGINE_HANDLE_EXTRA_T* pExtraParam | 附加参数     | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件：ax\_engine\_api.h
- 库文件：libax\_engine.so

### 【注意】

需要在 [AX\\_ENGINE\\_Init](#) 调用之后使用。

### 【相关主题】

[AX\\_ENGINE\\_HANDLE\\_EXTRA\\_T](#)

[AX\\_ENGINE\\_CreateHandle](#)

[AX\\_ENGINE\\_DestroyHandle](#)

AEXRA CONFIDENTIAL FOR SPEED

## AX\_ENGINE\_DestroyHandle

### 【描述】

释放推理引擎句柄，释放与此模型相关的所有资源，包含推理引擎相关资源。

### 【语法】

```
AX_S32 AX_ENGINE_DestroyHandle(AX_ENGINE_HANDLE handle);
```

### 【参数】

| 参数名称                    | 描述   | 输入/输出 |
|-------------------------|------|-------|
| AX_ENGINE_HANDLE handle | 模型句柄 | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需在 [AX\\_ENGINE\\_CreateHandle](#) 调用之后使用，且必须与 [AX\\_ENGINE\\_CreateHandle](#) 成对使用。

如果是多 Context 的 handle，那么 destroy 后多 Context 也一并被释放资源，不可继续使用。

### 【相关主题】

[AX\\_ENGINE\\_CreateHandle](#)

## AX\_ENGINE\_CreateContext

### 【描述】

创建计算图执行所需要的上下文，初始化计算图资源。

### 【语法】

```
AX_S32 AX_ENGINE_CreateContext(AX_ENGINE_HANDLE handle);
```

### 【参数】

| 参数名称                    | 描述   | 输入/输出 |
|-------------------------|------|-------|
| AX_ENGINE_HANDLE handle | 模型句柄 | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需在 [AX\\_ENGINE\\_CreateHandle](#) 调用之后运行。

### 【相关主题】

无

## AX\_ENGINE\_CreateContextV2

### 【描述】

创建计算图执行所需要的上下文，初始化计算图资源。

### 【语法】

```
AX_S32 AX_ENGINE_CreateContext(AX_ENGINE_HANDLE handle, AX_ENGINE_CONTEXT_T*  
pContext);
```

### 【参数】

| 参数名称                          | 描述            | 输入/输出 |
|-------------------------------|---------------|-------|
| AX_ENGINE_HANDLE handle       | 模型句柄          | 输入    |
| AX_ENGINE_CONTEXT_T* pContext | 模型 Context 句柄 | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需在 [AX\\_ENGINE\\_CreateHandle](#) 调用之后运行。同一个模型 handle 可以创建多个 Context，这样就可以不必创建多 handle 完成多 handle 任务了。

还需要注意，AX\_ENGINE\_CreateContextV2 / AX\_ENGINE\_RunSyncV2 需要配套使用。

### 【相关主题】

[AX\\_ENGINE\\_RunSyncV2](#)

## AX\_ENGINE\_RunSync

### 【描述】

推理引擎执行计算图推理计算，同步模式。

### 【语法】

```
AX_S32 AX_ENGINE_RunSync(AX_ENGINE_HANDLE handle, AX_ENGINE_IO_T *pIO);
```

### 【参数】

| 参数名称                    | 描述          | 输入/输出 |
|-------------------------|-------------|-------|
| AX_ENGINE_HANDLE handle | 模型句柄        | 输入    |
| AX_ENGINE_IO_T *pIO     | 模型 IO 结构体地址 | 输入    |

### 【返回值】

| 返回值 | 描述       |
|-----|----------|
| 0   | 成功       |
| 非 0 | 失败，返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需在 [AX\\_ENGINE\\_CreateHandle](#) 调用之后运行。

在执行推理前，通过 [AX\\_ENGINE\\_GetIOInfo](#) 可以获得每一个输入输出的名字和需要的空间大小，此 CMM 空间需要提前申请，并填充 [AX\\_ENGINE\\_IO\\_T](#) 里面的对应结构体 [AX\\_ENGINE\\_IO\\_BUFFER\\_T](#) 中的字段。

当需要进行动态 Batch 推理时，填写 nBatchSize 的大小，需要注意的是，此时申请的 CMM 空间必须大于等于模型每一个 IO 的 nBatchSize 倍大小。

也就是说，当实际推理任务需要进行 1~10Batch 大小的动态 Batch 推理时，首先申请 10 倍的每一个 IO 空间，然后将此 IO 的地址信息填入 `AX_ENGINE_IO_BUFFER_T` 中的字段，并正确填写当前需要的 `nBatchSize`(比如 4)，最后运行本接口即可。运行后，此 4Batch 的推理结果就已经写入此 10 倍 IO 的 CMM 空间中的前 4Batch 中了。

### 【相关主题】

[AX\\_ENGINE\\_GetIOInfo](#)

## AX\_ENGINE\_RunSyncV2

### 【描述】

推理引擎执行计算图推理计算，同步模式。该 API 允许传入 context 用于不同的上下文。

### 【语法】

```
AX_S32 AX_ENGINE_RunSyncV2(AX_ENGINE_HANDLE handle, AX_ENGINE_CONTEXT_T context,  
AX_ENGINE_IO_T* pIO);
```

### 【参数】

| 参数名称                        | 描述          | 输入/输出 |
|-----------------------------|-------------|-------|
| AX_ENGINE_HANDLE handle     | 模型句柄        | 输入    |
| AX_ENGINE_CONTEXT_T context | Context 句柄  | 输入    |
| AX_ENGINE_IO_T *pIO         | 模型 IO 结构体地址 | 输入    |

### 【返回值】

| 返回值 | 描述          |
|-----|-------------|
| 0   | 获取 CMM 信息成功 |
| 非 0 | 获取失败        |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需在 [AX\\_ENGINE\\_CreateHandle](#) 调用之后运行。

在执行推理前，通过 [AX\\_ENGINE\\_GetIOInfo](#) 可以获得每一个输入输出的名字和需要的空间大小，此 CMM 空间需要提前申请，并填充 [AX\\_ENGINE\\_IO\\_T](#) 里面的对应结构体 [AX\\_ENGINE\\_IO\\_BUFFER\\_T](#) 中的字段。

当需要进行动态 Batch 推理时，填写 nBatchSize 的大小，需要注意的是，此时申请的 CMM

空间必须大于等于模型每一个 IO 的 nBatchSize 倍大小。

也就是说，当实际推理任务需要进行 1~10Batch 大小的动态 Batch 推理时，首先申请 10 倍的每一个 IO 空间，然后将此 IO 的地址信息填入 `AX_ENGINE_IO_BUFFER_T` 中的字段，并正确填写当前需要的 nBatchSize(比如 4)，最后运行本接口即可。运行后，此 4Batch 的推理结果就已经写入此 10 倍 IO 的 CMM 空间中的前 4Batch 中了。

还需要注意，`AX_ENGINE_CreateContextV2` / `AX_ENGINE_RunSyncV2` 需要配套使用。

### 【相关主题】

[AX\\_ENGINE\\_CreateContextV2](#)

## AX\_ENGINE\_GetIOInfo

### 【描述】

获取推理引擎已创建的句柄中加载的模型的输入输出信息。

### 【语法】

```
AX_S32 AX_ENGINE_GetIOInfo(AX_ENGINE_HANDLE handle, AX_ENGINE_IO_INFO_T **pIO);
```

### 【参数】

| 参数名称                      | 描述     | 输入/输出 |
|---------------------------|--------|-------|
| AX_ENGINE_HANDLE handle   | 模型句柄   | 输入    |
| AX_ENGINE_IO_INFO_T **pIO | 输出输出数据 | 输入/输出 |

### 【返回值】

|     |           |
|-----|-----------|
| 0   | 成功        |
| 非 0 | 失败, 返回错误码 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需在 [AX\\_ENGINE\\_CreateHandle](#) 调用之后运行。

对于动态 Batch 的模型, nMaxBatchSize 字段描述了其动态 Batch 执行时, 能够组装的最大 Batch 能力。如果是非动态 Batch 模型, 此值为固定为 1。

当推理超过 nMaxBatchSize 大小的输入时, Engine 按动态 Batch 模型的 Batch 组成进行处理 (比如, 用户希望处理 9Batch 的输入, 模型此时 nMaxBatchSize 是 8, 那么就会组成 8+1 的执行序列完成任务), 并不会因为超过此值大小而返回错误。

模型的 nMaxBatchSize 值的大小和模型编译过程的设置有关, 详见模型编译使用说明。

## 【相关主题】

无

AEXRA CONFIDENTIAL FOR SPEED

## AX\_ENGINE\_GetVNPUAttr

### 【描述】

获取 NPU 运行模式。

### 【语法】

```
AX_S32 AX_ENGINE_GetVNPUAttr(AX_ENGINE_NPU_ATTR_T *pNpuAttr)
```

### 【参数】

| 参数名称                           | 描述       | 输入/输出 |
|--------------------------------|----------|-------|
| AX_ENGINE_NPU_ATTR_T *pNpuAttr | 运行模式值的指针 | 输入/输出 |

### 【返回值】

| 返回值 | 描述     |
|-----|--------|
| 0   | 获取属性成功 |
| 非 0 | 获取属性失败 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

无

### 【相关主题】

无

## AX\_ENGINE\_SetAffinity

### 【描述】

设置虚拟 NPU 亲和性。

### 【语法】

```
AX_S32 AX_ENGINE_SetAffinity(AX_ENGINE_HANDLE handle, AX_ENGINE_NPU_SET_T nNpuSet);
```

### 【参数】

| 参数名称                        | 描述        | 输入/输出 |
|-----------------------------|-----------|-------|
| AX_ENGINE_HANDLE handle     | 模型句柄      | 输入    |
| AX_ENGINE_NPU_SET_T nNpuSet | 希望亲和的核心掩码 | 输入    |

### 【返回值】

| 返回值 | 描述      |
|-----|---------|
| 0   | 设置亲和性成功 |
| 非 0 | 设置失败    |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

亲和性是对应的比特位掩码；设置亲和性和 NPU 初始化状态以及模型参数有关。

### 【相关主题】

[AX\\_ENGINE\\_GetAffinity](#)

## AX\_ENGINE\_GetAffinity

### 【描述】

获取虚拟 NPU 亲和性。

### 【语法】

```
AX_S32 AX_ENGINE_GetAffinity(AX_ENGINE_HANDLE handle, AX_ENGINE_NPU_SET_T *nNpuSet);
```

### 【参数】

| 参数名称                         | 描述      | 输入/输出 |
|------------------------------|---------|-------|
| AX_ENGINE_HANDLE handle      | 模型句柄    | 输入    |
| AX_ENGINE_NPU_SET_T* nNpuSet | 亲和性值的指针 | 输入/输出 |

### 【返回值】

| 返回值 | 描述      |
|-----|---------|
| 0   | 获取亲和性成功 |
| 非 0 | 获取失败    |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

亲和性是对应的比特位掩码；设置亲和性和 NPU 初始化状态以及模型参数有关。

### 【相关主题】

[AX\\_ENGINE\\_SetAffinity](#)

## AX\_ENGINE\_GetCMMUsage

### 【描述】

获取模型的 CMM 占用情况。

### 【语法】

```
AX_S32 AX_ENGINE_GetCMMUsage(AX_ENGINE_HANDLE handle, AX_ENGINE_CMM_INFO_T*  
cmm_info);
```

### 【参数】

| 参数名称                           | 描述         | 输入/输出 |
|--------------------------------|------------|-------|
| AX_ENGINE_HANDLE handle        | 模型句柄       | 输入    |
| AX_ENGINE_CMM_INFO_T* cmm_info | CMM 结构体的指针 | 输出    |

### 【返回值】

| 返回值 | 描述          |
|-----|-------------|
| 0   | 获取 CMM 信息成功 |
| 非 0 | 获取失败        |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

需要注意的是, 此接口获取的是非对齐到 CMM 分块的数值。

### 【相关主题】

无

## AX\_ENGINE\_GetVersion

### 【描述】

获取 Engine 的版本号。

### 【语法】

```
const AX_CHAR* AX_ENGINE_GetVersion(AX_VOID);
```

### 【参数】

无

### 【返回值】

| 返回值   | 描述            |
|-------|---------------|
| 版本字符串 | Engine 版本的字符串 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

无

### 【相关主题】

无

## AX\_ENGINE\_GetModelToolsVersion

### 【描述】

获取编译该模型的 Pulsar2 工具链的版本号。

### 【语法】

```
const AX_CHAR* AX_ENGINE_GetModelToolsVersion(AX_ENGINE_HANDLE handle);
```

### 【参数】

| 参数名称                    | 描述   | 输入/输出 |
|-------------------------|------|-------|
| AX_ENGINE_HANDLE handle | 模型句柄 | 输入    |

### 【返回值】

| 返回值   | 描述                |
|-------|-------------------|
| 版本字符串 | Pulsar2 工具链版本的字符串 |

### 【需求】

- 头文件: ax\_engine\_api.h
- 库文件: libax\_engine.so

### 【注意】

无

### 【相关主题】

无

# 5 数据结构

## AX\_ENGINE\_NPU\_ATTR\_T

### 【说明】

输入或输出缓存，包括用户缓存和 SDK 内部使用的缓存。

### 【定义】

```
1. typedef struct {  
2.     AX_ENGINE_NPU_MODE_T eHardMode;  
3.     AX_U32 reserve[8];  
4. } AX_ENGINE_NPU_ATTR_T;
```

### 【成员】

| 成员名称                           | 描述       |
|--------------------------------|----------|
| AX_ENGINE_NPU_MODE_T eHardMode | NPU 运行模式 |

### 【注意】

无

### 【相关数据类型及接口】

#### [AX\\_ENGINE\\_NPU\\_MODE\\_T](#)

## AX\_ENGINE\_NPU\_MODE\_T

### 【说明】

虚拟 NPU 模式的枚举。

### 【定义】

```
1. typedef enum {
2.     AX_ENGINE_VIRTUAL_NPU_DISABLE = 0,
3.     AX_ENGINE_VIRTUAL_NPU_ENABLE = 1,
4.     AX_ENGINE_VIRTUAL_NPU_BUTT   = 2
5. } AX_ENGINE_NPU_MODE_T;
```

### 【成员】

| 枚举名称                          | 描述         |
|-------------------------------|------------|
| AX_ENGINE_VIRTUAL_NPU_DISABLE | 非虚拟 NPU 模式 |
| AX_ENGINE_VIRTUAL_NPU_ENABLE  | 虚拟 NPU 模式  |
| AX_ENGINE_VIRTUAL_NPU_BUTT    | 占位符        |

### 【注意】

非虚拟 NPU 模式时，NPU 作为整体；虚拟模式时，有 VNPU0 和 VNPU1 两个虚拟 NPU。

### 【相关数据类型及接口】

#### AX\_ENGINE\_NPU\_ATTR\_T

## AX\_ENGINE\_HANDLE\_EXTRA\_T

### 【说明】

模型输入输出的基本信息。

### 【定义】

```
1. typedef struct {  
2.     AX_ENGINE_NPU_SET_T nNpuSet;  
3.     AX_S8 *pName;  
4.     AX_U32 reserve[8];  
5. } AX_ENGINE_HANDLE_EXTRA_T;
```

### 【成员】

| 成员名称                        | 描述       |
|-----------------------------|----------|
| AX_ENGINE_NPU_SET_T nNpuSet | 输入预设的亲和性 |
| AX_S8 *pName                | 模型名字     |

### 【注意】

无

### 【相关数据类型及接口】

[AX\\_ENGINE\\_CreateHandleV2](#)

## AX\_ENGINE\_IO\_T

### 【说明】

模型输入输出的各项参数。

### 【定义】

```
1. typedef struct _AX_ENGINE_IO_T
2. {
3.     AX_ENGINE_IO_BUFFER_T *pInputs;
4.     AX_U32 nInputSize;
5.     AX_ENGINE_IO_BUFFER_T *pOutputs;
6.     AX_U32 nOutputSize;
7.     AX_U32 nBatchSize;           // 0 for auto detection
8.     AX_ENGINE_IO_SETTING_T *pIoSetting;
9. } AX_ENGINE_IO_T;
```

### 【成员】

| 成员名称                               | 描述               |
|------------------------------------|------------------|
| AX_ENGINE_IO_BUFFER_T *pInputs     | 输入 Buffer 有关的结构体 |
| AX_U32 nInputSize                  | 输入数量             |
| AX_ENGINE_IO_BUFFER_T *pOutputs    | 输出 Buffer 有关的结构体 |
| AX_U32 nOutputSize                 | 输出数量             |
| AX_U32 nBatchSize                  | 作业 Batch, 默认是 0  |
| AX_ENGINE_IO_SETTING_T *pIoSetting | 保留 IO 设置, 默认不设置  |

### 【注意】

nBatchSize 用以设置本次运行的 Batch 数量, 设置后模型即按要求 Batch 数优化运行;

### 【相关数据类型及接口】

#### [AX\\_ENGINE\\_IO\\_BUFFER\\_T](#)

## AX\_ENGINE\_IO\_BUFFER\_T

### 【说明】

模型 IO 的内存缓冲区信息。

### 【定义】

```
1. typedef struct _AX_ENGINE_IO_BUFFER_T
2. {
3.     AX_ADDR phyAddr;
4.     AX_VOID *pVirAddr;
5.     AX_U32 nSize; // total size of memory
6.     AX_S32 *pStride;
7.     AX_U8 nStrideSize;
8. } AX_ENGINE_IO_BUFFER_T;
```

### 【成员】

| 成员名称              | 描述          |
|-------------------|-------------|
| AX_ADDR phyAddr   | CMM 内存的物理地址 |
| AX_VOID *pVirAddr | CMM 内存的虚拟地址 |
| AX_U32 nSize      | Buffer 大小   |
| AX_S32 *pStride   | 对齐数组，需清零    |
| AX_U8 nStrideSize | 对齐数组大小，需清零  |

### 【注意】

无

### 【相关数据类型及接口】

[AX\\_ENGINE\\_IO\\_T](#)

## AX\_ENGINE\_IO\_INFO\_T

### 【说明】

模型输入输出的基本信息。

### 【定义】

```
1. typedef struct _AX_ENGINE_IO_INFO_T
2. {
3.     AX_ENGINE_IOMETA_T *pInputs;
4.     AX_U32 nInputSize;
5.     AX_ENGINE_IOMETA_T *pOutputs;
6.     AX_U32 nOutputSize;
7.     AX_U32 nMaxBatchSize;
8.     AX_BOOL bDynamicBatchSize;
9. } AX_ENGINE_IO_INFO_T;
```

### 【成员】

| 成员名称                         | 描述               |
|------------------------------|------------------|
| AX_ENGINE_IOMETA_T *pInputs  | 输入 IO 信息结构体      |
| AX_U32 nInputSize            | 输入 IO 个数         |
| AX_ENGINE_IOMETA_T *pOutputs | 输出 IO 信息结构体      |
| AX_U32 nOutputSize           | 输出 IO 个数         |
| AX_U32 nMaxBatchSize         | 最大作业 Batch       |
| AX_BOOL bDynamicBatchSize    | 使能动态 Batch, 暂时无效 |

### 【注意】

无

### 【相关数据类型及接口】

[AX\\_ENGINE\\_IOMETA\\_T](#)

## AX\_ENGINE\_IOMETA\_T

### 【说明】

模型一个 IO 的元信息。

### 【定义】

```
1. typedef struct _AX_ENGINE_IOMETA_T
2. {
3.     AX_CHAR *pName;
4.     AX_S32 *pShape;
5.     AX_U8 nShapeSize;
6.     AX_ENGINE_TENSOR_LAYOUT_T eLayout;
7.     AX_ENGINE_MEMORY_TYPE_T eMemoryType;
8.     AX_ENGINE_DATA_TYPE_T eDataType;
9.     AX_ENGINE_IOMETA_EX_T* pExtraMeta;
10.    AX_U32 nSize;
11.    AX_U32 nQuantizationValue;
12.    AX_S32 *pStride;
13. } AX_ENGINE_IOMETA_T;
```

### 【成员】

| 成员名称                                | 描述                    |
|-------------------------------------|-----------------------|
| AX_CHAR *pName                      | IO 的名字，也就是模型 tensor 名 |
| AX_S32 *pShape                      | IO 的 shape            |
| AX_U8 nShapeSize                    | IO 的 shape 的维度        |
| AX_ENGINE_TENSOR_LAYOUT_T eLayout   | IO 的 layout           |
| AX_ENGINE_MEMORY_TYPE_T eMemoryType | IO 的存储类型              |
| AX_ENGINE_DATA_TYPE_T eDataType     | IO 的数值类型              |
| AX_ENGINE_IOMETA_EX_T* pExtraMeta   | IO 的额外信息              |
| AX_U32 nSize                        | IO 的缓冲区大小             |
| AX_U32 nQuantizationValue           | IO 的量化值，暂时无效          |

| 成员名称            | 描述        |
|-----------------|-----------|
| AX_S32 *pStride | 对齐信息，暂时无效 |

**【注意】**

无

**【相关数据类型及接口】**

[AX\\_ENGINE\\_IO\\_INFO\\_T](#)

## AX\_ENGINE\_MODEL\_TYPE\_T

### 【说明】

模型算力需求类型。

### 【定义】

```
1. typedef enum {  
2.     AX_ENGINE_MODEL_TYPE0      = 0,  
3.     AX_ENGINE_MODEL_TYPE1      = 1,  
4.     AX_ENGINE_MODEL_TYPE_BUTT = 2,  
5. } AX_ENGINE_MODEL_TYPE_T;
```

### 【成员】

| 枚举名称                      | 描述          |
|---------------------------|-------------|
| AX_ENGINE_MODEL_TYPE0     | 虚拟 NPU 的模型  |
| AX_ENGINE_MODEL_TYPE1     | 非虚拟 NPU 的模型 |
| AX_ENGINE_MODEL_TYPE_BUTT | 占位符         |

### 【注意】

当 NPU 初始化成 AX\_ENGINE\_VIRTUAL\_NPU\_DISABLE 模式时，全部模型类型均可以运行；

当 NPU 初始化成 AX\_ENGINE\_VIRTUAL\_NPU\_ENABLE 模式时，NPU 物理上进行了分割，此时各自虚拟 NPU 只能运行 AX\_ENGINE\_MODEL\_TYPE0 的模型。

### 【相关数据类型及接口】

#### AX\_ENGINE\_NPU\_MODE\_T

## AX\_ENGINE\_CMM\_INFO\_T

### 【说明】

CMM 占用情况结构体。

### 【定义】

```
1. typedef struct _AX_ENGINE_CMM_INFO_T  
2. {  
3.     AX_U32 nCMMSize;  
4. } AX_ENGINE_CMM_INFO_T;
```

### 【成员】

| 成员名称            | 描述      |
|-----------------|---------|
| AX_U32 nCMMSize | CMM 的大小 |

### 【注意】

CMM 是有对齐的，获取 CMM 信息的接口返回时不计算对齐。

### 【相关数据类型及接口】

无

## 6 错误码

错误码详见《55 - AX 软件错误码文档》文档。

# 7 模型评测工具

为了方便 Engine 用户测评模型，在开发板上预制了 `ax_run_model` 工具，此工具有若干参数，可以很方便地测试模型速度和精度。

```
root@AXERA:~# ax_run_model

usage: ax_run_model --model=string [options] ...
options:
  -m, --model           path to a model file (string)
  -r, --repeat          repeat times running a model (int [=1])
  -w, --warmup          repeat times before running a model to warming up (int [=1])
  -v, --vnpu             type of Visual-NPU initied {0=Disable, 1=Enable} (int [=0])
  -a, --affinity         npu affinity when running a model (int [=1])
  -b, --batch            the batch will running (int [=0])
  -i, --input-folder    the folder of each inputs (folders) located (string [=])
  -o, --output-folder   the folder of each outputs (folders) will saved in (string [=])
  -l, --list              the list of inputs which will test (string [=])
  --inputs-is-folder    each time model running needs inputs stored in each standalone
input folders
  --outputs-is-folder   each time model running saved outputs stored in each standalone
output folders
  --use-tensor-name     using tensor names instead of using tensor indexes when loading
& saving io files
  --verify               verify outputs after running model
  --save-benchmark      save benchmark result(min, max, avg) as a json file
  -?, --help              print this message
```

## 7.1 参数说明

测评工具参数主要有两部分；第一部分是与测速有关的参数：

`--model` 指定测试模型的路径； `string` 类型；

`--repeat` 指定要测试的循环次数，然后显示 `min/max/avg` 的速度； `int` 类型；

--warmup 循环测试前，预热的次数； int 类型；

--vnpu 希望测试的 VNPU 模式， 默认为 Disable；

--affinity 亲和性的 mask 值； int 类型， 大于 1(0b001)， 小于 7(0b111)；

--batch 指定测试的 batch； int 类型；

这部分参数中，只有 --model 是必须参数。第二部分的参数主要用来测试精度：

--input-folder 指定用于精度测试的输入文件夹； string 类型；

--output-folder 指定用于精度测试的输出文件夹； string 类型；

--list 指定测试列表； string 类型；

--inputs-is-folder 指定输入路径参数--input-folder 是由文件夹组成的； string 类型； 此参数是 AX650 芯片的兼容参数，**不加也生效**；

--outputs-is-folder 指定输出路径参数--out-folder 是由文件夹组成的； string 类型； 此参数是 AX650 芯片的兼容参数，**不加也生效**；

--use-tensor-name 指定按模型输入输出名字查找激励文件，不设置是按索引查找； string 类型； 此参数是 AX650 芯片的兼容参数，**不加也生效**；

--verify 指定不保存模型输出且指定的目录输出文件已存在，进行逐 byte 比较； string 类型；

## 7.2 使用示例

以测速需求为例，假设已经转换完成了一个单核心的 yolo 模型，现在想要知道上板子运行的速度，那么可以参考运行如下命令：

```
1. root@AXERA:~# ax_run_model -m /opt/data/npu/models/yolov5s.axmodel -  
w 10 -r 100  
2. [Axera version]: libax_sys.so V1.13.0 Apr 26 2023 16:24:35  
3. Run AxModel:  
4.     model: /opt/data/npu/models/yolov5s.axmodel  
5.     type: HalfOCM  
6.     vnpu: Enable  
7.     affinity: 0b11
```

```
8.      repeat: 100
9.      warmup: 10
10.     batch: 1
11. pulsar2 ver: 1.2-patch2 7e6b2b5f
12. engine ver: [Axera version]: libax_engine.so V1.13.0 Apr 26 2023 16:48
     :53 1.1.0
13.     tool ver: 1.0.0
14.     cmm size: 12730188 Bytes
15.     -----
16.     min = 7.658 ms   max = 7.672 ms   avg = 7.662 ms
17.     -----
```

用户可以通过 -a 参数进行亲和性设置：

```
1. root@AXERA:~# ax_run_model -m /opt/data/npu/models/yolov5s.axmodel -
   w 10 -r 100 -v 1 -a 2
2. [Axera version]: libax_sys.so V1.13.0 Apr 26 2023 16:24:35
3. Run AxModel:
4.     model: /opt/data/npu/models/yolov5s.axmodel
5.     type: HalfOCM
6.     vnpu: Enable
7.     affinity: 0b10
8.     repeat: 100
9.     warmup: 10
10.    batch: 1
11. pulsar2 ver: 1.2-patch2 7e6b2b5f
12. engine ver: [Axera version]: libax_engine.so V1.13.0 Apr 26 2023 16:48
     :53 1.1.0
13.     tool ver: 1.0.0
14.     cmm size: 12730188 Bytes
15.     -----
16.     min = 7.656 ms   max = 7.672 ms   avg = 7.660 ms
17.     -----
```

从打印的 log 可以看出，VNPU 被初始化成 ENABLE 模式，此时 NPU 被分作两份；并且这次测速时亲和性设置为亲和序号最大的那个核心。如果希望改变 VNPU 模式，可以修改-v 参数；而通过设置亲和性，可以在 VNPU Enable 时很方便地在不编写代码的情况下，同时跑多个模型进行测速。

比如，在一个 SSH 终端窗口里，运行模型 a 数万次，然后在另一个 SSH 终端里，设置不同的亲和性，观察模型 b 速度相较于没有运行模型 a 时的速度下降，就可以得知极高负载情况下，模型 b 受模型 a 运行的影响(这可能比真实情况更严苛)。

另一个很常见的需求是转完了模型，想要知道板子上的精度如何，这可以通过精度的参数进行测试。以分类模型为例，说明目录结构和参数的使用：

```
1. root@AXERA:~# tree /opt/data/npu/temp/
2. /opt/data/npu/temp/
3. |-- input
4. |   |-- θ
5. |   |   |-- data.bin
6. |   |-- list.txt
7. |   |-- mobilenet_v1.axmodel
8. `-- output
9.   |-- θ
10.    |-- prob.bin
11.
12. 4 directories, 4 files
```

测试精度时必须的参数是 `-m -i -o -L`，分别指定模型、输入文件夹、输出文件夹、和待测试的输入列表。

如果输出文件夹都非空，在运行命令时输出文件夹的已有文件会被覆盖；但如果是已经从 Pulsar2 仿真拿到的输出 golden 文件，则可以通过附加`--verify`参数不覆写输出文件，而是读取输出文件夹的已有文件，和当前模型的输出在内存中进行逐位比较，这个模式在怀疑仿真和上板精度不对齐时特别有用：

```
1. root@AXERA:~# cat /opt/data/npu/temp/list.txt
2. θ
3. root@AXERA:~#
```

激励文件(夹) 指定参数在数据集很大时非常有用，比如输入文件夹是完整的 ImageNet 数据集，文件非常多；但这次测试时只希望测 10 个文件验证一下，如果没有异常再跑全量的测试，那么这样的需求可以通过创建两个 list.txt 完成，一个 list 里保存的只有 10 行激励，一个 list 文件里是全部的激励。

以进行模型 verify 的需求进行举例，`ax_run_model` 参数运行示例如下：

```
1. root@AXERA:~# ax_run_model -m /opt/data/npu/temp/mobilenet_v1.axmodel -  
i /opt/data/npu/temp/input/ -o /opt/data/npu/temp/output/ -  
l /opt/data/npu/temp/list.txt --verify  
2. [Axera version]: libax_sys.so V1.13.0 Apr 26 2023 16:24:35  
3. total found {1} input drive folders.  
4. infer model, total 1/1. Done.  
5. -----  
6. min = 3.347 ms max = 3.347 ms avg = 3.347 ms  
7. -----  
8.  
9. root@AXERA:~#
```

可以看出，这个模型在这组输入输出 binary 文件下，输出是逐位对齐的。如果没有对齐，打印会报告没有对齐的 byte 偏移量。