Tengine
User Manual

文档版本 1.1

发布日期 2021-04-07

变更记录

日期	版本	说明	作者
2020-12-04	1.0	初版	Shunqi Fu
2021-04-07	1.1	新增 NPU 相关内容说明	Tang Qi

目录

1 简介

Tengine 是 **OPEN AI LAB** 推出的面向 AIoT 场景的 AI 应用开发平台，致力于解决 AIoT 产业链碎片化问题，加速 AI 产业化落地。Tengine 专为 AIoT 场景设计，同时具有跨平台、异构调度、芯片底层加速、超轻量无依赖、完整开发移植部署工具链几大特点。

Tengine 解决了深度神经网络模型在端侧设备上进行推理的问题，涵盖了深度神经网络模型的优化、推理和异构调度计算。Tengine 具有通用，开放，高性能等特点。

通用性体现在 Tengine 兼容多种操作系统，如 Linux、Android 等主流边缘侧操作系统，不依赖于除 C/C++以外的任何第三方库，支持 TensorFlow、TensorFlow-Lite、MXNet,、Caffe、ONNX 等主流框架以及 CNN、RNN/LSTM、GAN 等常用深度学习算法框架，并且可以支持 NCHW/NHWC（NHWC 有限支持）两种数据排布。

高性能是指 Tengine 对于端侧计算平台，做了大量定制和优化，从而实现在设备上高效运行神经网络。CPU 上核心运算的代码都是针对微架构手工优化的汇编代码，把 ARM CPU 的算力发挥到极致。针对不同参数和形状的卷积，高效实现了多种优化算法，保证了任意形状卷积推理的高性能。Tengine 运行模型时，自动对内存进行优化，如内存池、动态释放等，自动调度硬件资源。

Tengine 支持专用 NPU（神经网络处理器），如 Amlogic A311D 中的 AML NPU，并支持 CPU 和 NPU 的异构计算。

1.1 Tengine 框架

Tengine 的推理过程涵盖了模型的加载解析，格式转换，计算图的调度和优化，在多种架构 CPU 上高效运行。

在模型加载阶段，来自不同训练框架的模型，通过模型转换工具统一解析和转换成 Tengine 的唯一识别的模型格式 tmfile。模型转换工具通过对原有框架的计算图进行裁减，合并和优化，来实现统一的计算图表示。

此外，对于高级用户，还可以将网络模型的参数提取出来，通过 Tengine C API 接口直接构造计算图，从而省去修改或者实现一个序列化模块的工作。来帮助用户快速实现这个功能。

在图执行时，Tengine 会调度所有输入数据都准备好的计算子图，到绑定的设备上运行。当一个子图执行完成时，会更新所有依赖于该子图的其他计算子图的状态。如果之前处于等待状态的计算子图的所有依赖子图都已经计算完成，该子图会被调度执行。

Tengine 实现了一系列高效高性能的计算算子。例如，对于矩阵乘，这一基本运

算，Tengine 就针对不同平台，做了微架构级别的汇编优化。

当所有的计算子图都执行完毕，就可以通过计算图的输出 Tensor 取到计算结果。Tengine 同时提供了同步和异步两种方式来获取计算的结果。图 1 是 Tengine 的架构图。

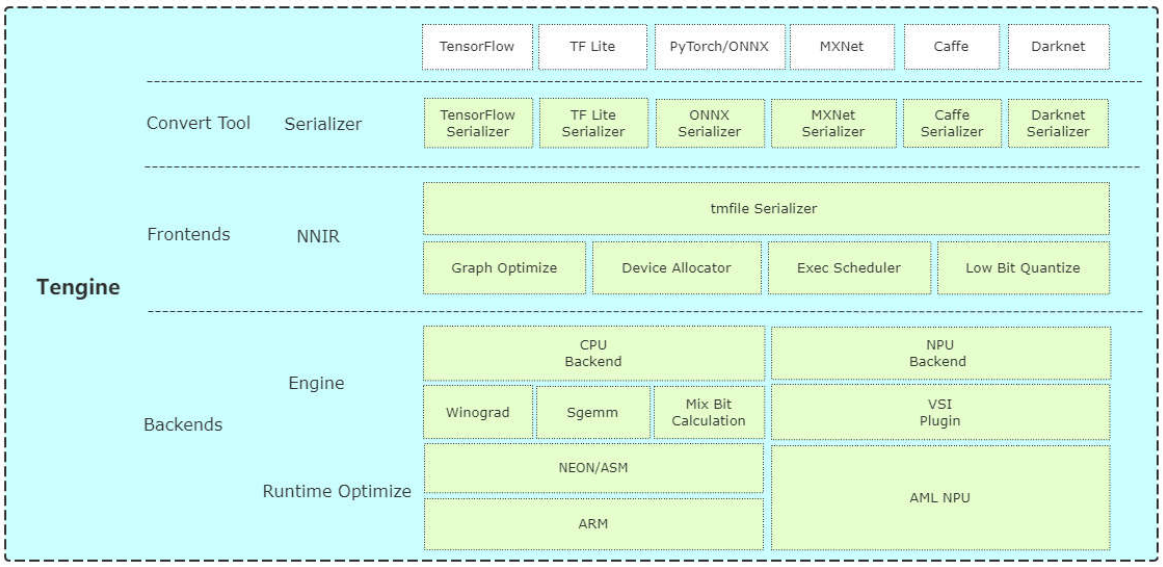

图 1 Tengine Architecture

Convert Tool：将不同训练框架的网络模型进行序列化处理，转换生成 Tengine 推理框架专用网络模型文件 tmfile；

Frontend：实现加载 Tengine 推理框架专用网络模型文件 tmfile，同时完成执行设备分配及调度功能；

Backend：在指定的执行设备上，通过计算方法和指令集优化实现网络模型推理加速功能。

1.2 Tengine 软件发布包

```
├── docs
│   ├── P3002-SW-TEG-Khadas-v1.1-20210622_TestReport.pdf
│   ├── Tengine_Convert_Tool_Technical_Spec.pdf
│   ├── Tengine_Convert_Tool_User_Manual.pdf
│   ├── Tengine_Quant_Tool_Technical_Spec.pdf
│   ├── Tengine_Quant_Tool_User_Manual.pdf
│   ├── Tengine_Release_Note.pdf
│   ├── Tengine_Technical_Spec.pdf
│   └── Tengine_User_Manual.pdf
```

```
├── sample
│   ├── 3rdparty
│   │   └── tim-vx
│   ├── README.md
│   └── console
│       ├── CMakeLists.txt
│       ├── apps
│       ├── build_android.sh
│       ├── cmake
│       ├── data
│       └── tengine
└── tools
    ├── convert_tool
    │   └── README.md
    └── quant_tool
        └── README.md
```

各个目录简要说明：

Docs：版本相关文档，只包含中文版本；

Sample/apps：随版本发布的 example 源码；

Sample/console/tengine：随版本发布的预编译的二进制库及头文件，针对 Amlogic A311D 中的 AML NPU 平台目前提供预编译二进制库；

Tools/convert_tool：Linux x86-64 的模型转换工具使用说明；

Tools/quant_tool_uint8：Linux x86-64 的模型量化工具使用说明；

2 Tengine 模型转换工具

详情请参考《Tengine_Convert_Tool_User_Manual》。

3 Tengine 模型量化工具

详情请参考《Tengine_Quant_Tool_User_Manual》。

4 Tengine C API

4.1 数据类型和数据结构

Tengine C API 相关的数据类型和数据结构定义如下：

4.1.1 MAX_SHAPE_DIM_NUM

【说明】

定义 shape 最大维度。

【定义】

```
#define    MAX_SHAPE_DIM_NUM    4
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.2 Tensor 数据类型

【说明】

定义 Tensor 的数据类型。

【定义】

```
#define    TENGINE_DT_FP32    0
#define    TENGINE_DT_FP16    1
#define    TENGINE_DT_INT8    2
#define    TENGINE_DT_UINT8   3
#define    TENGINE_DT_INT32   4
#define    TENGINE_DT_INT16   5
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.3 数据布局格式

【说明】

定义数据的布局格式。（NHWC 有限支持）

【定义】

```
#define    TENGINE_LAYOUT_NCHW    0
#define    TENGINE_LAYOUT_NHWC    1
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.4 Tensor 类型

【说明】

定义 Tensor 类型。

【定义】

```
#define    TENSOR_TYPE_UNKNOWN    0
#define    TENSOR_TYPE_VAR        1
#define    TENSOR_TYPE_CONST      2
#define    TENSOR_TYPE_INPUT      3
#define    TENSOR_TYPE_DEP        4
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.5 Node 转储操作类型

【说明】

定义节点转储类型。

【定义】

```
#define    NODE_DUMP_ACTION_DISABLE    0
#define    NODE_DUMP_ACTION_ENABLE     1
#define    NODE_DUMP_ACTION_START      2
#define    NODE_DUMP_ACTION_STOP       3
```

```
#define    NODE_DUMP_ACTION_GET          4
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.6 Graph 性能统计动作类型

【说明】

定义 Graph 性能统计动作类型。

【定义】

```
#define    GRAPH_PERF_STAT_DISABLE       0
#define    GRAPH_PERF_STAT_ENABLE        1
#define    GRAPH_PERF_STAT_STOP          2
#define    GRAPH_PERF_STAT_START         3
#define    GRAPH_PERF_STAT_RESET         4
#define    GRAPH_PERF_STAT_GET           5
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.7 Log 级别

【说明】

定义 Log 级别。

【定义】

```
enum log_level
{
        LOG_EMERG,
        LOG_ALERT,
```

```
        LOG_CRIT,
        LOG_ERR,
        LOG_WARNING,
        LOG_NOTICE,
        LOG_INFO,
        LOG_DEBUG
    };
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

- set_log_level

4.1.8 Graph 执行事件类型

【说明】

定义 Graph 执行事件类型。

【定义】

```
    enum graph_exec_event
    {
        GRAPH_EXEC_START,
        GRAPH_EXEC_SUSPEND,
        GRAPH_EXEC_RESUME,
        GRAPH_EXEC_ABORT,
        GRAPH_EXEC_DONE
    };
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.9 Graph 执行状态类型

【说明】

定义 Graph 执行状态类型。

【定义】

```
enum graph_exec_stat
{
        GRAPH_STAT_CREATED,
        GRAPH_STAT_READY,
        GRAPH_STAT_RUNNING,
        GRAPH_STAT_DONE,
        GRAPH_STAT_ERROR
};
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.10 设备运行策略

【说明】

定义设备运行策略。

【定义】

```
enum device_policy
{
        DEFAULT_POLICY,
        LATENCY_POLICY,
        LOW_POWER_POLICY
};
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.11 上下文

【说明】

定义上下文。

【定义】

```
typedef void * context_t;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.12 Graph 句柄

【说明】

定义 Graph 句柄。

【定义】

```
typedef void * graph_t;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.13 Tensor 句柄

【说明】

定义 Tensor 句柄。

【定义】

```
typedef void * tensor_t;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.14 Node 句柄

【说明】

定义 Node 句柄。

【定义】

```
typedef void * node_t;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.15 事件回调函数

【说明】

定义事件回调函数。

【定义】

```
typedef int (*event_handler_t)(graph_t, int, void* arg);
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.16 Log 输出函数

【说明】

定义 Log 输出函数。

【定义】

```
typedef void (*log_print_t)(const char*);
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

4.1.17 运行选项 options 的数据类型

【说明】

定义 Tengine 推理模型时采用的运行选项。

【定义】

```
/* graph exec options */
struct options
{
    int num_thread;
    int cluster;
    int precision;
    uint64_t affinity;
};
```

【成员】

opt.num_thread:设置执行图采用的线程数

opt.cluster：设置运行 CPU 的核心类型，可设置为 TENGINE_CLUSTER_ALL/TENGINE_CLUSTER_BIG/ TENGINE_CLUSTER_ MEDIUM/TENGINE_CLUSTER_ LITTLE

opt.presion:设置运行精度模式，如 TENGINE_MODE_FP32

opt. affinity 设置 CPU 亲和性使用位掩码(bitmask)表示，每一位都表示一个 CPU，置 1 表示"绑定"，如绑定 CPU0、CPU1，可设置为 0x03;

【注意事项】

无。

【相关数据类型及接口】

详见 C API 介绍 prerun_graph_multithread。

4.2 C API 介绍

Tengine 支持的主要 C APIs 是：

- init_tengine：初始化 Tengine.
- release_tengine：释放 Tengine 资源
- get_tengine_version：获取 Tengine 版本号
- request_tengine_version：检查 Tengine 版本号是否支持
- create_graph：创建 graph
- set_graph_layout：设置 graph 的布局类型(layout)
- set_graph_input_node：设置 graph 的输入节点(node)
- set_graph_output_node：设置 graph 的输出节点(node)
- destroy_graph：销毁 graph
- get_graph_input_node_number：获取 graph 的输入节点(node)的个数
- get_graph_input_node：通过 ID 获取 gragh 的输入节点(node)
- get_graph_output_node_number：获取 graph 的输出节点(node)的个数
- get_graph_output_node：通过 ID 获取 gragh 的输出节点(node)
- get_graph_output_tensor：通过 ID 获取 graph 的输出节点的张量(tensor)
- get_graph_input_tensor：通过 ID 获取 graph 的输入节点的张量(tensor)
- create_graph_node：创建 graph 的节点(node)
- get_graph_node：获取节点(node)句柄
- get_node_name：获取节点(node)的名称
- get_node_op：获取节点(node)的操作(operation)
- release_graph_node：释放 graph 的节点(node)
- get_node_input_tensor：通过 ID 获取节点(node)的输入张量(tensor)
- get_node_output_tensor：通过 ID 获取节点(node)的输出张量(tensor)
- set_node_input_tensor：设置节点(node)的输入张量(tensor)
- set_node_output_tensor：设置节点(node)的输出张量(tensor)
- get_node_output_number：获取节点(node)的输出张量(tensor)个数
- get_node_input_number：获取节点(node)的输入张量(tensor)个数
- add_node_attr：添加节点(node)参数(attribute)
- get_node_attr_int：获取节点(node)的整型参数
- get_node_attr_float：获取节点(node)的浮点型参数
- get_node_attr_pointer：获取节点(node)的指针型参数
- get_node_attr_generic：获取节点(node)的通用型参数

- set_node_attr_int：设置节点(node)的整型参数
- set_node_attr_float：设置节点(node)的浮点型参数
- set_node_attr_pointer：设置节点(node)的指针型参数
- set_node_attr_generic：设置节点(node)的通用型参数
- set_custom_kernel：设置用户内核(kernel)
- remove_custom_kernel：移除用户内核(kernel)
- create_graph_tensor：创建 graph 的张量(tensor)
- get_graph_tensor：通过名称获取 graph 的张量(tensor)
- get_tensor_name：获取张量(tensor)的名称
- release_graph_tensor：释放 graph 的张量(tensor)
- get_tensor_shape：获取张量(tensor)的形状(shape)
- set_tensor_shape：设置张量(tensor)的形状(shape)
- get_tensor_buffer_size：获取张量(tensor)的缓存大小
- get_tensor_buffer：获取张量(tensor)的缓存(buffer)
- set_tensor_buffer：设置张量(tensor)的缓存(buffer)
- get_tensor_data：获取张量(tensor)的数据(data)
- set_tensor_data：设置张量(tensor)的数据(data)
- get_tensor_data_type：获取张量(tensor)的数据类型
- set_tensor_data_type：设置张量(tensor)的数据类型
- set_tensor_quant_param：设置 tensor 的量化参数
- get_tensor_quant_param：获取 tensor 的量化参数
- set_graph_attr：设置 graph 的参数
- get_graph_attr：获取 graph 的参数
- prerun_graph：graph 的预运行
- prerun_graph_multithread：graph 的预运行，配置运行选项
- run_graph：运行 graph
- wait_graph：等待 graph
- postrun_graph：停止运行 graph 并释放 graph 占据的资源
- set_default_device：设置默认设备
- set_graph_device： 设置 graph 的执行设备
- get_node_device： 获取 node 的执行设备
- do_graph_perf_stat： 启用或禁用性能统计
- get_graph_perf_stat：获取性能统计信息
- get_device_number：获取设备编号
- get_device_name： 按 index 获取设备名称

- get_default_device：获取默认设备
- create_context：创建 context
- destroy_context：销毁 context
- get_context_device_number：获取 context 设备数量
- add_context_device：将设备添加到 context 中
- remove_context_device：将设备移除出 context
- set_context_attr：设置 context 属性
- get_context_attr：获取 context 属性
- get_tengine_errno：获取错误数量
- set_log_level：设置 log 级别
- set_log_output：设置日志打印功能
- dump_graph：dump 图的结构
- load_tengine_plugin：加载 Tengine 插件
- unload_tengine_plugin：移除 Tengine 插件
- get_tengine_plugin_number：获取 Tengine 插件数量
- get_tengine_plugin_name：通过索引获取插件的名称
- register_custom_op：注册用户自定义算子
- unregister_custom_op：注销用户自定义算子

4.2.1 Tengine 创建与释放相关 API

4.2.1.1 init_tengine

名称	init_tengine	
介绍	使用Tengine库前，首先需要执行init_tengine()，对Tengine库进行初始化	
参数	无	
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int init_tengine(void);	

4.2.1.2 release_tengine

名称	release_tengine	
介绍	执行 release_tengine()，对 Tengine 库进行资源释放操作	
参数	无	
返回值	无	
原型	void release_tengine(void);	

4.2.1.3 get_tengine_version

名称	get_tengine_version	
介绍	获取Tengine库的版本信息	
参数	无	
返回值	const char *	版本的字符串指针
原型	const char * get_tengine_version(void);	

4.2.1.4 request_tengine_version

名称	request_tengine_version	
介绍	初始化 Tengine 后，通过该接口获取对版本的兼容情况	
参数	const char * version	(I) 版本号字符串的指针
返回值	int	1 ： 表示支持 0 ： 表示不支持
原型	int request_tengine_version(const char * version);	

4.2.2 graph 相关操作

4.2.2.1 create_graph

名称	create_graph	
介绍	创建 graph	
参数	context_t context	(I) context的句柄
	const char * model_format	(I) 采用模型的类型，目前仅支持“tengine”
	const char * file_name	(I)文件名称
	...	(I) 其他参数
返回值	graph_t	Graph 句柄， 如果为NULL 表示失败
原型	graph_t create_graph(context_t context, const char * model_format, const char * file_name,...);	

4.2.2.2 set_graph_layout

名称	set_graph_layout	
介绍	设置graph的layout	
参数	graph_t graph	(I) graph的句柄

	int layout_type	Graph 的layout 类型，比如： TENGINE_LAYOUT_NCHW TENGINE_LAYOUT_NHWC
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_graph_layout(graph_t graph, int layout_type);	

4.2.2.3 set_graph_input_node

名称	set_graph_input_node	
介绍	设置graph的input node	
参数	graph_t graph	(I) graph的句柄
	const char * input_nodes[]	输入节点数组的指针，char类型
	int input_number	(I) 输入节点的数量
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_graph_input_node(graph_t graph, const char * input_nodes[], int input_number);	

4.2.2.4 set_graph_output_node

名称	set_graph_output_node	
介绍	设置graph的output node	
参数	graph_t graph	(I) graph的句柄
	const char * output_nodes[]	(I) 输出节点数组的指针，char类型
	int output_number	(I) 输出节点的数量

返回值	int	0 ： 表示成功 -1 ： 表示失败
原型	int set_graph_output_node(graph_t graph, const char * output_nodes[], int output_number);	

4.2.2.5 merge_graph

名称	merge_graph	
介绍	将多个graph合并为一个graph	
参数	int graph_num	graph句柄的个数
	graph_t graph0	graph 句柄0
	graph_t graph1	(I) graph 句柄1
	...	其他参数
返回值	graph_t	graph 句柄，如果为NULL 表示失败
原型	graph_t merge_graph(int graph_num , graph_t graph0, graph_t graph1, ...);	

4.2.2.6 destroy_graph

名称	destroy_graph	
介绍	销毁 graph	
参数	graph_t graph	graph句柄
返回值	int	0 ： 表示成功 -1 ： 表示失败
原型	int destroy_graph(graph_t graph);	

4.2.2.7 get_graph_input_node_number

名称	get_graph_input_node_number

介绍	获取graph的输入节点(node)数目	
参数	graph_t graph	graph句柄
返回值	int	>0 ： node 个数 -1 ： failure
原型	int get_graph_input_node_number(graph_t graph);	

4.2.2.8 get_graph_input_node

名称	get_graph_input_node	
介绍	通过ID获取gragh的输入节点(node)	
参数	graph_t graph	graph句柄
	int idx	Node 下标(>=0)
返回值	node_t	node 句柄，如果为NULL 表示失败
原型	node_t get_graph_input_node(graph_t graph, int idx);	

4.2.2.9 get_graph_output_node_number

名称	get_graph_output_node_number	
介绍	获取graph的输出节点(node)个数	
参数	graph_t graph	graph句柄
返回值	int	>0 ： node 编号 -1 ： 失败
原型	int get_graph_output_node_number(graph_t graph);	

4.2.2.10 get_graph_output_node

名称	get_graph_output_node
介绍	通过ID获取gragh的输出节点(node)

参数	graph_t graph	graph句柄
	int idx	Node 下标(>=0)
返回值	node_t	node 句柄，如果为NULL 表示失败
原型	node_t get_graph_output_node(graph_t graph, int idx);	

4.2.2.11 get_graph_output_tensor

名称	get_graph_output_tensor	
介绍	按ID获取graph的输出张量(tensor)	
参数	graph_t graph	graph句柄
	int output_node_idx	Node 下标(>=0)
	int tensor_idx	tensor 下标（>=0）
返回值	tensor_t	tensor句柄，如果为NULL 表示失败
原型	tensor_t get_graph_output_tensor(graph_t graph, int output_node_idx, int tensor_idx);	

4.2.2.12 get_graph_input_tensor

名称	get_graph_input_tensor	
介绍	按ID获取graph的输入张量(tensor)	
参数	graph_t graph	graph句柄
	int input_node_idx	Node 下标(>=0)
	int tensor_idx	tensor 下标（>=0）
返回值	tensor_t	tensor句柄，如果为NULL 表示失败
原型	tensor_t get_graph_input_tensor(graph_t graph, int input_node_idx, int tensor_idx);	

4.2.2.13 get_graph_node_by_idx

名称	get_graph_node_by_idx	
介绍	通过下标获取graph的node	
参数	graph_t graph	graph句柄
	int node_idx	node的下标
返回值	node_t	node 句柄，如果为NULL 表示失败
原型	node_t get_graph_node_by_idx(graph_t graph, int node_idx);	

4.2.3 node 相关操作

4.2.3.1 create_graph_node

名称	create_graph_node	
介绍	创建graph的node	
参数	graph_t graph	graph句柄
	const char * node_name	node的名称
	const char * op_name	操作名称
返回值	node_t	node 句柄，如果为NULL 表示失败
原型	node_t create_graph_node(graph_t graph, const char * node_name, const char * op_name);	

4.2.3.2 get_graph_node

名称	get_graph_node	
介绍	获取graph的node 的句柄	
参数	graph_t graph	graph句柄
	const char * node_name	node的名称

返回值	node_t	node 句柄，如果为NULL 表示失败
原型	node_t get_graph_node(graph_t graph, const char * node_name);	

4.2.3.3 get_node_name

名称	get_node_name	
介绍	获取node 的名称	
参数	node_t node	node句柄
返回值	const char *	node 句柄名称指针，如果为NULL 表示失败
原型	const char * get_node_name(node_t node);	

4.2.3.4 get_node_op

名称	get_node_op	
介绍	获取node使用的操作(op)	
参数	node_t node	node句柄
返回值	const char *	node 的op名称指针，如果为NULL 表示失败
原型	const char * get_node_op(node_t node);	

4.2.3.5 release_graph_node

名称	release_graph_node	
介绍	释放graph的node	
参数	node_t node	node句柄
返回值	无	
原型	void release_graph_node(node_t node);	

4.2.3.6 get_node_input_tensor

名称	get_node_input_tensor	
介绍	获取 node 的输入张量(tensor)	
参数	node_t node	node句柄
	int input_idx	Input 的下标（>=0）
返回值	tensor_t	tensor 的句柄，如果为NULL 表示失败
原型	tensor_t get_node_input_tensor(node_t node, int input_idx);	

4.2.3.7 get_node_output_tensor

名称	get_node_output_tensor	
介绍	获取node的输出张量(tensor)	
参数	node_t node	node句柄
	int output_idx	Output 的下标（>=0）
返回值	tensor_t	tensor 的句柄，如果为NULL 表示失败
原型	tensor_t get_node_output_tensor(node_t node, int output_idx);	

4.2.3.8 set_node_input_tensor

名称	set_node_input_tensor	
介绍	设置node的输入张量(tensor)	
参数	node_t node	node句柄
	int input_idx	Input 的下标（>=0）
	tensor_t tensor	tensor 的句柄
返回值	int	0 ：表示设置成功 -1 ：表示设置失败

原型	int set_node_input_tensor(node_t node, int input_idx, tensor_t tensor);

4.2.3.9 set_node_output_tensor

名称	set_node_output_tensor	
介绍	设置node的输出张量(tensor)	
参数	node_t node	node句柄
	int output_idx	Input 的下标（>=0）
	tensor_t tensor	tensor 的句柄
	int tensor_type	tensor 类型
返回值	int	0 ： 表示成功 -1 ： 表示失败
原型	int set_node_output_tensor(node_t node, int output_idx, tensor_t tensor, int tensor_type);	

4.2.3.10 get_node_input_number

名称	get_node_input_number	
介绍	获取node 中输入张量(tensor)的个数	
参数	node_t node	node句柄
返回值	int	>0 ： 表示输入张量(tensor)的个数 -1 ： 表示失败
原型	int get_node_input_number(node_t node);	

4.2.3.11 add_node_attr

名称	add_node_attr
介绍	添加node属性

参数	node_t node	node句柄
	const char* attr_name	属性名称
	const char* type_name	属性类型名称
	int size	属性大小
返回值	int	0 ： success -1 ： failure
原型	int add_node_attr(node_t node, const char *attr_name, const char* type_name, int size);	

4.2.3.12 get_node_attr_int

名称	get_node_attr_int	
介绍	获取node的int类型参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	int * attr_val	int参数返回值指针
返回值	int	0 ： 表示获取成功 -1 ： 表示失败
原型	int get_node_attr_int(node_t node, const char * attr_name, int * attr_val);	

4.2.3.13 get_node_attr_float

名称	get_node_attr_float	
介绍	获取node的float类型参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称

	float * attr_val	(O) float参数返回值的指针
返回值	int	0 ：表示获取成功 -1 ：表示失败
原型	int get_node_attr_float(node_t node, const char * attr_name, float * attr_val);	

4.2.3.14 get_node_attr_pointer

名称	get_node_attr_pointer	
介绍	获取node的指针类型参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	void * attr_val	(O) pointer 参数的指针
返回值	int	0 ：表示获取成功 -1 ：表示获取失败
原型	int get_node_attr_pointer(node_t node, const char * attr_name, void * attr_val);	

4.2.3.15 get_node_attr_generic

名称	get_node_attr_generic	
介绍	获取node的通用参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	const char * type_name	参数类型名称
	void * buf	(O) 数据的buffer
	int size	数据的buffer 的尺寸

返回值	int	0 ： 表示获取成功 -1 ： 表示获取失败
原型	int get_node_attr_generic(node_t node, const char * attr_name, const char* type_name, void * buf, int size);	

4.2.3.16 set_node_attr_int

名称	set_node_attr_int	
介绍	设置node的int类型参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	const int * attr_val	int参数返回值的指针
返回值	int	0 ： 表示获取成功 -1 ： 表示获取失败
原型	int set_node_attr_int(node_t node, const char * attr_name, const int *attr_val);	

4.2.3.17 set_node_attr_float

名称	set_node_attr_float	
介绍	设置node的float 类型参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	const float * attr_val	float型返回值的指针
返回值	int	0 ： 表示获取成功 -1 ： 表示获取失败
原型	int set_node_attr_float(node_t node, const char * attr_name, const float *attr_val);	

4.2.3.18 set_node_attr_pointer

名称	set_node_attr_pointer	
介绍	设置node的指针类型参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	const void * attr_val	指针类型参数的指针
返回值	int	0 ：表示获取成功 -1 ：表示获取失败
原型	int set_node_attr_pointer(node_t node, const char * attr_name, const void *attr_val);	

4.2.3.19 set_node_attr_generic

名称	set_node_attr_generic	
介绍	设置node的通用参数	
参数	node_t node	node句柄
	const char * attr_name	参数名称
	const char* type_name	类型名称
	const void * buf	数据的buffer
	int size	数据的buffer 的尺寸
返回值	int	0 ：表示获取成功 -1 ：表示获取失败
原型	int set_node_attr_generic(node_t node, const char * attr_name, const char * type_name, const void * buf, int size);	

4.2.4 Tensor 相关操作

4.2.4.1 create_graph_tensor

名称	create_graph_tensor	
介绍	创建graph的tensor	
参数	graph_t graph	graph的句柄
	const char * tensor_name	tensor的名称
	int data_type	数据类型
返回值	tensor_t	tensor 的句柄，如果为NULL 表示失败
原型	tensor_t create_graph_tensor(graph_t graph, const char * tensor_name, int data_type);	

4.2.4.2 get_graph_tensor

名称	get_graph_tensor	
介绍	获取graph的tensor	
参数	graph_t graph	graph 的句柄
	const char * tensor_name	tensor的名称
返回值	tensor_t	tensor 的句柄，如果为NULL 表示失败
原型	tensor_t get_graph_tensor(graph_t graph, const char * tensor_name);	

4.2.4.3 get_tensor_name

名称	get_tensor_name	
介绍	获取tensor的名称	
参数	tensor_t tensor	tensor的句柄
返回值	const char *	tensor 名称，如果为NULL 表示失败

原型	const char * get_tensor_name(tensor_t tensor);

4.2.4.4 release_graph_tensor

名称	release_graph_tensor	
介绍	释放graph的tensor	
参数	tensor_t tensor	tensor的句柄
返回值	无	
原型	void release_graph_tensor(tensor_t tensor);	

4.2.4.5 get_tensor_shape

名称	get_tensor_shape	
介绍	获取tensor的shape	
参数	tensor_t tensor	tensor的句柄
	int dims[]	(O) shape的数组
	int dim_number	数组的尺寸
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int get_tensor_shape(tensor_t tensor, int dims[], int dim_number);	

4.2.4.6 set_tensor_shape

名称	set_tensor_shape	
介绍	设置tensor的shape	
参数	tensor_t tensor	tensor的句柄
	const int dims[]	shape的数组

	int dim_number	数组的尺寸
返回值	int	0 ： 表示设置成功 -1 ： 表示设置失败
原型	int set_tensor_shape(tensor_t tensor, const int dims[], int dim_number);	

4.2.4.7 get_tensor_buffer_size

名称	get_tensor_buffer_size	
介绍	获取tensor的buffer的大小	
参数	tensor_t tensor	tensor的句柄
返回值	Int	>0 ： buffer size -1 ： failure
原型	int get_tensor_buffer_size(tensor_t tensor);	

4.2.4.8 get_tensor_buffer

名称	get_tensor_buffer	
介绍	获取tensor的buffer	
参数	tensor_t tensor	tensor的句柄
返回值	void *	Buffer 的指针，如果为NULL 表示失败
原型	void * get_tensor_buffer(tensor_t tensor);	

4.2.4.9 set_tensor_buffer

名称	set_tensor_buffer	
介绍	设置tensor的buffer	
参数	tensor_t tensor	tensor的句柄

	void * buffer	Buffer 的指针
	int buffer_size	数据buffer 的尺寸
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_tensor_buffer(tensor_t tensor, void * buffer, int buffer_size);	

4.2.4.10 get_tensor_data

名称	get_tensor_data	
介绍	获取tensor的data	
参数	tensor_t tensor	tensor的句柄
	void * output_data	(O) 输出buffer的指针
	int data_size	(I) 输出数据buffer的大小
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int get_tensor_data(tensor_t tensor, void * output_data, int data_size);	

4.2.4.11 set_tensor_data

名称	set_tensor_data	
介绍	设置tensor的数据	
参数	tensor_t tensor	tensor的句柄
	const void * input_data	input 数据buffer 的指针
	int data_size	Input 数据的尺寸
返回值	int	0 ：表示成功 -1 ：表示失败

原型	int set_tensor_data(tensor_t tensor, const void * input_data, int data_size);

4.2.4.12 get_tensor_data_type

名称	get_tensor_data_type	
介绍	获取tensor的数据类型	
参数	tensor_t tensor	tensor的句柄
返回值	int	Tensor 类型。比如： TENGINE_DT_FP32、 TENGINE_DT_FP16、 TENGINE_DT_IN8，等等 -1 ： 失败
原型	int get_tensor_data_type(tensor_t tensor);	

4.2.4.13 set_tensor_data_type

名称	set_tensor_data_type	
介绍	设置tensor的数据类型	
参数	tensor_t tensor	tensor的句柄
	int data_type	数据类型，比如 ： TENGINE_DT_FP32、 TENGINE_DT_FP16、 TENGINE_DT_IN8，等等
返回值	int	0：success -1 ： failure
原型	int set_tensor_data_type(tensor_t tensor, int data_type);	

4.2.4.14 set_tensor_quant_param

名称	set_tensor_quant_param	
介绍	设置tensor的量化参数	
参数	tensor_t tensor	tensor的句柄
	const float *scale	scale 指针
	const int *zero_point	起点指针
	int number	数据大小
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_tensor_quant_param(tensor_t tensor, const float *scale, const int *zero_point, int number);	

4.2.4.15 get_tensor_quant_param

名称	get_tensor_quant_param	
介绍	获取tensor的量化参数	
参数	tensor_t tensor	张量句柄
	float *scale	scale 指针
	int *zero_point	起点指针
	int number	数据大小
返回值	int	0 ：表示成功 -1 ：表示失败

4.2.5 与 graph 运行相关 API

4.2.5.1 set_graph_attr

名称	set_graph_attr

介绍	设置graph的参数	
参数	graph_t graph	graph 的句柄
	const char * attr_name	要设置的参数名称
	const void * buf	设置的参数的指针
	int size	数据的尺寸
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_graph_attr(graph_t graph, const char * attr_name, const void * buf, int size);	

4.2.5.2 get_graph_attr

名称	get_graph_attr	
介绍	获取graph的参数	
参数	graph_t graph	graph 的句柄
	const char * attr_name	要获取的参数名称
	void * buf	要获取参数的指针
	int size	数据的大小
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int get_graph_attr(graph_t graph, const char * attr_name, void * buf, int size);	

4.2.5.3 prerun_graph

名称	prerun_graph	
介绍	准备运行graph，并准备资源	
参数	graph_t graph	graph 的句柄

返回值	int	0 ：表示成功 -1 ：表示失败
原型	int prerun_graph(graph_t graph);	

4.2.5.4 prerun_graph_multithread

名称	prerun_graph_multithread	

介绍	准备运行graph，并准备资源，并通过options结构设置不同的CPU Cluster、不同的CPU号来运行网络模型，同时支持多线程提升计算效率。
	以设置CPU Cluster中的大核心，线程数为2，绑定到CPU0、CPU1举例：
	struct options opt;
	opt.num_thread = 2;
	opt.cluster = TENGINE_CLUSTER_BIG;
	opt. affinity= 0x03;
	prerun_graph_multithread(graph, opt);	

参数	graph_t graph	graph 的句柄
	struct options opt	Options 的结构体内容
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int prerun_graph_multithread(graph_t graph, struct options opt);	

4.2.5.5 run_graph

名称	run_graph	
介绍	运行graph，执行推理，可以反复多次调用	
参数	graph_t graph	graph 的句柄
	int block	1：阻塞. 需要配置 GRAPH_DONE 函数 0：非阻塞：需要调用wait_graph()以获取结果

返回值	int	0 ：表示成功 -1 ：表示失败
原型	int run_graph(graph_t graph, int block);	

4.2.5.6 wait_graph

名称	wait_graph	
介绍	等待graph运行的结果	
参数	graph_t graph	graph 的句柄
	int try_wait	1：检查状态，然后返回 0：立即返回
返回值	int	-1 ：graph 运行未完成 0 ：运行完成
原型	int wait_graph(graph_t graph, int try_wait);	

4.2.5.7 postrun_graph

名称	postrun_graph	
介绍	停止运行graph 并释放graph 占据的资源	
参数	graph_t graph	graph 的句柄
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int postrun_graph(graph_t graph);	

4.2.6 与设备相关操作

4.2.6.1 set_default_device

名称	set_default_device	
介绍	设置默认运行设备	
参数	const char * device	运行的设备名称
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_default_device(const char * device);	

4.2.6.2 set_graph_device

名称	set_graph_device	
介绍	设置graph运行设备	
参数	graph_t graph	graph 的句柄
	const char * dev_name	设备的名称
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int set_graph_device(graph_t graph, const char * dev_name);	

4.2.6.3 get_node_device

名称	get_node_device	
介绍	获取node 运行设备	
参数	node_t node	node的句柄
返回值	const char *	设备的名称： NULL意味着没有可利用的设备

原型	const char * get_node_device(node_t node);

4.2.6.4 get_default_device

名称	get_default_device	
介绍	获取默认使用设备名称	
参数	无	
返回值	const char *	设备的名称： NULL意味着失败
原型	const char * get_default_device(void);	

4.2.7 与 context 相关部分

4.2.7.1 create_context

名称	create_context	
介绍	创建context	
参数	const char * context_name	context的名称
	int empty_context	(I) 1：没有为该设备分配可用的设备 0：所有声明的设备都将添加到context中
返回值	context_t	Context 的句柄, 返回NULL表示创建失败
原型	context_t create_context(const char * context_name, int empty_context);	

4.2.7.2 destroy_context

名称	destroy_context	
介绍	销毁context	
参数	context_t context	context 的句柄

返回值	无	
原型	void destroy_context(context_t context);	

4.2.7.3 get_context_device_number

名称	get_context_device_number	
介绍	获取context设备编号	
参数	context_t context	context 的句柄
返回值	int	context使用的设备编号
原型	int get_context_device_number(context_t context);	

4.2.7.4 add_context_device

名称	add_context_device	
介绍	将设备添加到context中	
参数	context_t context	context 的句柄
	const char * dev_name	设备名称
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int add_context_device(context_t context, const char * dev_name);	

4.2.7.5 remove_context_device

名称	remove_context_device	
介绍	将设备移除出context	
参数	context_t context	context 的句柄
	const char * dev_name	device名称

返回值	int	0 ： 表示成功 -1 ： 表示失败
原型	int remove_context_device(context_t context, const char * dev_name);	

4.2.7.6 set_context_attr

名称	set_context_attr	
介绍	设置context参数	
参数	context_t context	context 的句柄
	const char * attr_name	参数名称
	const void * val	设置参数值的指针
	int val_size	参数的值所占的大小
返回值	int	0 ： 表示成功 -1 ： 表示失败
原型	int set_context_attr(context_t context, const char * attr_name, const void * val, int val_size);	

4.2.7.7 get_context_attr

名称	get_context_attr	
介绍	获取context属性	
参数	context_t context	context 的句柄
	const char * attr_name	参数名称
	void * val	(O) 参数返回值的指针
	int val_size	参数的值所占的大小
返回值	int	0 ： 表示成功 -1 ： 表示失败

原型	int get_context_attr(context_t context, const char * attr_name, void * val, int val_size);

4.2.8 其他的 API

4.2.8.1 get_tengine_errno

名称	get_tengine_errno	
介绍	获取错误码	
参数	无	
返回值	int	Tengine错误码。定义遵循glic
原型	int get_tengine_errno(void);	

4.2.8.2 set_log_level

名称	set_log_level	
介绍	设置log级别	
参数	log_level level	上下文的句柄
返回值	无	
原型	void set_log_level(enum log_level level);	

4.2.8.3 set_log_output

名称	set_log_output	
介绍	设置log输出函数	
参数	log_print_t func	输出函数的指针
返回值	无	
原型	void set_log_output(log_print_t func);	

4.2.8.4 dump_graph

名称	dump_graph	
介绍	dump图的结构	
参数	graph_t graph	graph 的句柄
返回值	无	0 ：表示成功 -1 ：表示失败
原型	void dump_graph(graph_t graph);	

4.2.8.5 load_tengine_plugin

名称	load_tengine_plugin	
介绍	加载Tengine插件	
参数	const char * plugin_name	插件的名称
	const char * fname	文件的名称
	const char * init_func_name	初始化函数的名称
返回值	int	0 ：表示成功 -1 ：表示失败
原型	int load_tengine_plugin(const char * plugin_name, const char * fname, const char * init_func_name);	

4.2.8.6 unload_tengine_plugin

名称	unload_tengine_plugin	
介绍	移除Tengine插件	
参数	const char * plugin_name	插件的名称
	const char * rel_func_name	release函数的指针

返回值	int	0 ： 表示成功 -1 ： 表示失败
原型	int unload_tengine_plugin(const char * plugin_name, const char * rel_func_name);	

4.2.8.7 get_tengine_plugin_number

名称	get_tengine_plugin_number	
介绍	获取Tengine插件的编号	
参数	无	
返回值	int	插件数量
原型	int get_tengine_plugin_number(void);	

4.2.8.8 get_tengine_plugin_name

名称	get_Tengine_plugin_name	
介绍	获取Tengine插件的名称	
参数	int idx	插件的编号(>=0)
返回值	const char *	插件名称
原型	const char * get_Tengine_plugin_name(int idx);	

4.2.8.9 register_custom_op

名称	register_custom_op	
介绍	Tengine 初始化后，使用该接口注册用户自定义的操作。	
参数	struct custom_op *op	用户自定义操作结构体指针。
返回值	int	0 ： 表示成功。 -1 ： 表示失败。

原型	int register_custom_op(struct custom_op *op);

4.2.8.10 unregister_custom_op

名称	unregister_custom_op	
介绍	注销用户自定义操作。	
参数	struct custom_op *op	用户自定义操作结构体指针
返回值	int	0 ： 表示成功。 -1 ： 表示失败。
原型	int unregister_custom_op(struct custom_op *op);	

4.3 错误码

Tengine C API 错误码如下表所示。

错误代码	描述
1	Operation not permitted
2	No such file or directory
3	No such process
4	Interrupted system call
5	I/O error
6	No such device or address
7	Arg list too long
8	Exec format error
9	Bad file number
10	No child processes

更多错误码信息请参考 Linux Error Code 标准定义。

5 调试技巧

5.1 模型的性能评估

在模型的性能评估时会用到该环境变量。该功能开启后，会在模型退出运行时打印出模型的整个运行状况，包括以下内容：

- 模型中所有 OP 运行时间。从高到底排序；

- 模型中每一层的运行时间，输入数据结构，输出数据结构，kernel 结构等。

打印输出示例如下：

```
---------------------------------------
 0 [ 8.09% :  1.2 ms]   Convolution idx:  3 shape: {1   3 227 227} -> {1  64 113 113}     fp32 K: 3x3 | S: 2x2 | P: 0 0 0 0        MFLOPS: 44.13 Rate:3736
 1 [11.65% :  1.7 ms]      Pooling idx:  6 shape: {1  64 113 113} -> {1  64  56  56}     fp32 K: 3x3 | S: 2x2 | P: 0 0 0 0        Max
 2 [ 1.76% :  0.3 ms]   Convolution idx: 11 shape: {1  64  56  56} -> {1  16  56  56}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:2498
 3 [ 5.70% :  0.8 ms]   Convolution idx: 12 shape: {1  16  56  56} -> {1  64  56  56}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 57.80 Rate:6939
 4 [ 1.34% :  0.2 ms]   Convolution idx: 13 shape: {1  16  56  56} -> {1  64  56  56}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:3276
 5 [ 0.40% :  0.1 ms]       Concat idx: 16 shape: {1  64  56  56} -> {1 128  56  56}     fp32
 6 [ 3.79% :  0.6 ms]   Convolution idx: 21 shape: {1 128  56  56} -> {1  16  56  56}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS: 12.85 Rate:2323
 7 [ 5.72% :  0.8 ms]   Convolution idx: 22 shape: {1  16  56  56} -> {1  64  56  56}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 57.80 Rate:6915
 8 [ 1.32% :  0.2 ms]   Convolution idx: 23 shape: {1  16  56  56} -> {1  64  56  56}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:3326
 9 [ 0.47% :  0.1 ms]       Concat idx: 24 shape: {1  64  56  56} -> {1 128  56  56}     fp32
10 [ 5.91% :  0.9 ms]      Pooling idx: 27 shape: {1 128  56  56} -> {1 128  28  28}     fp32 K: 3x3 | S: 2x2 | P: 0 1 0 1        Max
11 [ 1.12% :  0.2 ms]   Convolution idx: 32 shape: {1 128  28  28} -> {1  32  28  28}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:3938
12 [ 3.62% :  0.5 ms]   Convolution idx: 33 shape: {1  32  28  28} -> {1 128  28  28}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 57.80 Rate:10946
13 [ 0.83% :  0.1 ms]   Convolution idx: 34 shape: {1  32  28  28} -> {1 128  28  28}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:5314
14 [ 0.16% :  0.0 ms]       Concat idx: 37 shape: {1 128  28  28} -> {1 256  28  28}     fp32
15 [ 2.08% :  0.3 ms]   Convolution idx: 42 shape: {1 256  28  28} -> {1  32  28  28}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS: 12.85 Rate:4226
16 [ 3.53% :  0.5 ms]   Convolution idx: 43 shape: {1  32  28  28} -> {1 128  28  28}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 57.80 Rate:11226
17 [ 0.80% :  0.1 ms]   Convolution idx: 44 shape: {1  32  28  28} -> {1 128  28  28}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:5492
18 [ 0.18% :  0.0 ms]       Concat idx: 45 shape: {1 128  28  28} -> {1 256  28  28}     fp32
19 [ 3.04% :  0.4 ms]      Pooling idx: 40 shape: {1 256  20  20}  > {1 256  14  14}     fp32 K: 3x3 | S: 2x2 | P: 0 1 0 1        Max
20 [ 0.74% :  0.1 ms]   Convolution idx: 53 shape: {1 256  14  14} -> {1  48  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  4.82 Rate:4464
21 [ 2.50% :  0.4 ms]   Convolution idx: 54 shape: {1  48  14  14} -> {1 192  14  14}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 32.51 Rate:8908
22 [ 0.44% :  0.1 ms]   Convolution idx: 55 shape: {1  48  14  14} -> {1 192  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  3.61 Rate:5563
23 [ 0.05% :  0.0 ms]       Concat idx: 58 shape: {1 192  14  14} -> {1 384  14  14}     fp32
24 [ 1.10% :  0.2 ms]   Convolution idx: 63 shape: {1 384  14  14} -> {1  48  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  7.23 Rate:4511
25 [ 2.51% :  0.4 ms]   Convolution idx: 64 shape: {1  48  14  14} -> {1 192  14  14}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 32.51 Rate:8884
26 [ 0.42% :  0.1 ms]   Convolution idx: 65 shape: {1  48  14  14} -> {1 192  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  3.61 Rate:5943
27 [ 0.05% :  0.0 ms]       Concat idx: 68 shape: {1 192  14  14} -> {1 384  14  14}     fp32
28 [ 1.27% :  0.2 ms]   Convolution idx: 73 shape: {1 384  14  14} -> {1  64  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  9.63 Rate:5206
29 [ 4.18% :  0.6 ms]   Convolution idx: 74 shape: {1  64  14  14} -> {1 256  14  14}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 57.80 Rate:9463
30 [ 0.69% :  0.1 ms]   Convolution idx: 75 shape: {1  64  14  14} -> {1 256  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:6354
31 [ 0.08% :  0.0 ms]       Concat idx: 78 shape: {1 256  14  14} -> {1 512  14  14}     fp32
32 [ 1.75% :  0.3 ms]   Convolution idx: 83 shape: {1 512  14  14} -> {1  64  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS: 12.85 Rate:5040
33 [ 4.22% :  0.6 ms]   Convolution idx: 84 shape: {1  64  14  14} -> {1 256  14  14}     fp32 K: 3x3 | S: 1x1 | P: 1 1 1 1        MFLOPS: 57.80 Rate:9369
34 [ 0.64% :  0.1 ms]   Convolution idx: 85 shape: {1  64  14  14} -> {1 256  14  14}     fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:  6.42 Rate:6833
35 [ 0.08% :  0.0 ms]       Concat idx: 86 shape: {1 256  14  14} -> {1 512  14  14}     fp32
36 [ 0.00% :  0.0 ms]      Dropout idx: 89 shape: {1 512  14  14} -> {1 512  14  14}     fp32
37 [16.53% :  2.4 ms]   Convolution idx: 90 shape: {1 512  14  14} -> {1 1000  14  14}    fp32 K: 1x1 | S: 1x1 | P: 0 0 0 0        MFLOPS:200.70 Rate:8311
38 [ 1.15% :  0.2 ms]      Pooling idx: 91 shape: {1 1000  14  14} -> {1 1000   1   1}    fp32 K: 14x14 | S: 1x1 | P: 0 0 0 0       Avg
39 [ 0.12% :  0.0 ms]      Softmax idx: 92 shape: {1 1000   1   1} -> {1 1000   1   1}    fp32
```

打开方式

设置环境变量：

```
export TG_DEBUG_TIME=1
```

关闭方式

设置环境变量：

```
unset TG_DEBUG_TIME
```

5.2 模型的精度评估

在模型的精度评估时会用到该环境变量。该功能开启后，会在模型退出运行时将每一层的输入输出数据保存到执行程序所在的路径下创建一个 output 文件夹：

output 文件夹包含的输出示例如下（SqueezeNet v1.1）：

```
bug1989@DESKTOP-SGN0H2A:/mnt/d/Release/Tengine-Lite/[illegible]/sample/build-linux$ ls output/
conv10_in_blob_data.txt              fire3-expand1x1_in_blob_data.txt    fire5-expand1x1_in_blob_data.txt
conv10_out_blob_data.txt             fire3-expand1x1_out_blob_data.txt   fire5-expand1x1_out_blob_data.txt
conv1_in_blob_data.txt               fire3-expand3x3_in_blob_data.txt    fire5-expand3x3_in_blob_data.txt
conv1_out_blob_data.txt              fire3-expand3x3_out_blob_data.txt   fire5-expand3x3_out_blob_data.txt
drop9_in_blob_data.txt               fire3-squeeze1x1_in_blob_data.txt   fire5-squeeze1x1_in_blob_data.txt
drop9_out_blob_data.txt              fire3-squeeze1x1_out_blob_data.txt  fire5-squeeze1x1_out_blob_data.txt
fire2-concat_in_blob_data.txt        fire4-concat_in_blob_data.txt       fire6-concat_in_blob_data.txt
fire2-concat_out_blob_data.txt       fire4-concat_out_blob_data.txt      fire6-concat_out_blob_data.txt
fire2-expand1x1_in_blob_data.txt     fire4-expand1x1_in_blob_data.txt    fire6-expand1x1_in_blob_data.txt
fire2-expand1x1_out_blob_data.txt    fire4-expand1x1_out_blob_data.txt   fire6-expand1x1_out_blob_data.txt
fire2-expand3x3_in_blob_data.txt     fire4-expand3x3_in_blob_data.txt    fire6-expand3x3_in_blob_data.txt
fire2-expand3x3_out_blob_data.txt    fire4-expand3x3_out_blob_data.txt   fire6-expand3x3_out_blob_data.txt
fire2-squeeze1x1_in_blob_data.txt    fire4-squeeze1x1_in_blob_data.txt   fire6-squeeze1x1_in_blob_data.txt
fire2-squeeze1x1_out_blob_data.txt   fire4-squeeze1x1_out_blob_data.txt  fire6-squeeze1x1_out_blob_data.txt
fire3-concat_in_blob_data.txt        fire5-concat_in_blob_data.txt       fire7-concat_in_blob_data.txt
fire3-concat_out_blob_data.txt       fire5-concat_out_blob_data.txt      fire7-concat_out_blob_data.txt
```

打开方式

设置环境变量：

```
export TG_DEBUG_DATA=1
```

关闭方式

设置环境变量：

```
unset TG_DEBUG_DATA
```

6 演示例程

6.1 编译

```
$ cd sample/console
$ mkdir build && cd build
$ cmake ..
$ make -j4

$ export LD_LIBRARY_PATH=<project root>/tengine/linux-arm64-v8a/lib/
```

6.2 运行

```
$./apps/test_classification -m <your/sample/path>/tengine/data/mobilenet_uint8.tmfile -i <your/sample/path>/tengine/data/cat.jpg

$./apps/test_yolov3 -m <your/sample/path>/tengine/data/yolov3_uint8.tmfile -i <your/sample/path>/tengine/data/ssd_dog.jpg
```

更多信息参考'sample/readme.md'。