**PaddleSOT3.0.0**

**测试报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 日 期 | 版 本 | 说明 | 作 者 |
| 2025.4.10 | 1.0 | 最终版 | 肖宇檬 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目 录**

[1引言 3](#_Toc195452430)

[1.1编写目的 3](#_Toc195452431)

[1.2背景 3](#_Toc195452432)

[1.3定义 3](#_Toc195452433)

[1.4参考资料 3](#_Toc195452434)

[2测试概要 4](#_Toc195452435)

[2.1动态测试内容和方法 4](#_Toc195452436)

[2.1.1 测试内容 4](#_Toc195452437)

[2.1.2 测试方法 4](#_Toc195452438)

[2.2静态测试内容和方法 4](#_Toc195452439)

[2.2.1 测试内容和方法 4](#_Toc195452440)

[2.2.2 对规范的理解 4](#_Toc195452441)

[3 测试环境 5](#_Toc195452442)

[4测试总结 5](#_Toc195452443)

[4.1动态测试情况说明 5](#_Toc195452444)

[4.2缺陷情况说明 6](#_Toc195452445)

[4.3静态扫描情况说明 6](#_Toc195452446)

[4.4建议 6](#_Toc195452447)

[4.5评价 6](#_Toc195452448)

# 1引言

## 1.1编写目的

编写目的：对Paddle3.0的动转静SOT子项目功能进行测试，观察其在各种边界情况下能否正确执行动转静功能，将用户输入的Python代码转化为静态计算图。

在理想情况下，行为应该与TensorFlow和Pytorch对齐。

## 1.2背景

a. 被测软件：Paddle3.0子模块：PaddleSOT

b. 测试所用环境：CPU:12700H-laptop;GPU:3070TI-laptop;Python版本3.13;CUDA版本12.8;GCC版本11.4.0

用户可能在各种计算机设备上安装Paddle3.0，包括：个人PC，笔记本电脑，CPU超算，计算集群，大规模GPU集群等等。

使用的GPU来自的芯片厂商可能包括：英伟达、昆仑芯、海光、寒武纪、昇腾、燧原、太初；同时，也可以使用纯CPU版本的Paddle.

用户所使用的计算架构可能包括：CUDA,昆仑芯2代,昆仑芯P800,DTK24.04.01,NeuWare SDK v1.16.0,CANN 8.0.T13,TopsRider3.2.109,SDAA v2.1.0,x86(Intel, 海光等等),ArmV8(飞腾，鲲鹏等)等等。

## 1.3定义

Paddle: 百度开源的深度学习框架；

SOT: 全称是Symbolic Opcode Translator,是飞桨动转静体系下的子图提取子项目。

## 1.4参考资料

Paddle官方文档：

[飞桨PaddlePaddle-源于产业实践的开源深度学习平台](https://www.paddlepaddle.org.cn/)

# 2测试概要

## 2.1动态测试内容和方法

### 2.1.1 测试内容

测试Paddle3.0下，SOT的动转静功能是否能够正确地把用户输入的Python代码，也就是Python字节码，转换为静态图表示；在此基础上，追求面对动态Shape时的正确性，保证子图编译次数的准确性和在遇到0、1维度时的特化能力。

### 2.1.2 测试方法

因为PaddleSOT的项目体量过于庞大，因此我们采用的测试方法均为黑盒测试。针对Paddle的公开技术文档，编写针对性的测试用例。包括以下四个方面：

1. SOT的Tuple相关逻辑；
2. SOT的List相关逻辑；
3. SOT的动态Shape功能；
4. SOT的Dict相关逻辑。

## 2.2静态测试内容和方法

未采用静态测试方法

### 2.2.1 测试内容和方法

未采用静态测试方法

### 2.2.2 对规范的理解

Paddle项目由C++和Python组成，不需要遵守Java的开发规范；C++遵循RAII，ODR等原则，Python部分追求可读性大于高效性，优先保障Python代码的可读性和可理解性。

# 3 测试环境

硬件条件：

CPU:12700H-laptop

GPU:3070TI-laptop

Memory:32GB

VRAM:8GB

软件条件：

GCC version:11.4.0

Python version:3.13

CUDA version:12.8

Paddle version:3.0.0

环境变量：

SOT\_LOG\_LEVEL=3

SOT\_COLLECT\_INFO="subgraph\_relation"

# 4测试总结

## 4.1动态测试情况说明

测试对功能点的覆盖率达到了90%+，主要的风险点在于SOT面对动态Shape时行为的不确定性，可能会增加程序在面对该类型向量时的不确定性：

若干组len(size)相等的Tensor依次作为同一个位置参数传入同一个函数，程序会把它们识别为一个动态Shape的向量。此时，如果动态维度中出现了0或者1，那么当我们把编译好的子图传递给CINN和后端pass执行时，行为就会出现不确定性，为了满足broadcast等行为的合理性，会有大量多余的kernel被编译出来。

除了基本的Python类型处理之外，我们还考虑了动态Shape的存在，因此，测试设计既全面，又合理。

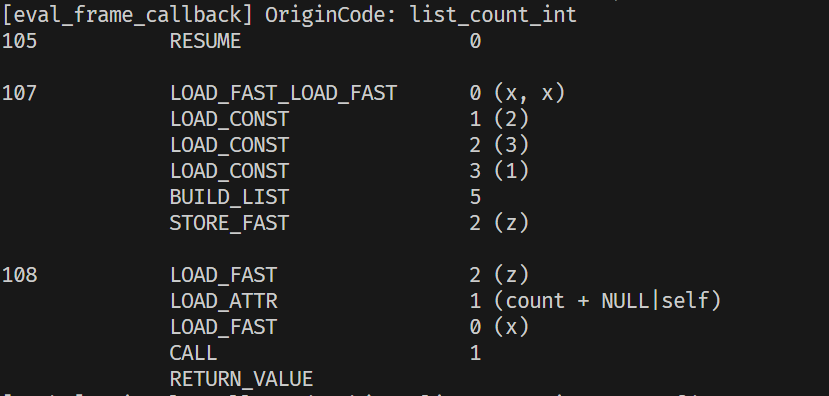
经过测试，我们可以认定：PaddleSOT能够正确处理绝大多数的Python字节码，并且给出高效的动转静方案和尽可能高效的子图。

测试环境与实际运行环境可能存在的差异：

我们只能构造出一个相当有限的Python字节码子集，但是在真实的运行环境中，用户可能写出千奇百怪的Python代码，PaddleSOT需要在各种极端和复杂的情况下都能正确地对输入字节码进行转换。

例如，在我们测试List相关逻辑的过程中：

这是用户输入的原始字节码：



输入到PaddleSOT之后，我们会把它置于我们创建的VM当中，进行模拟执行，这是模拟执行的步骤之一：

图片包含 图形用户界面

AI 生成的内容可能不正确。

可以看到，SOT的虚拟环境对传入的字节码进行了逐条模拟执行。

最后，SOT会输出自适应的新字节码，这个case由于输入的字节码较为简单，没有必要进行动转静的封装，所以我们直接返回原始字节码：



## 4.2缺陷情况说明

PaddleSOT能够在逻辑上正确地处理用户输入的Python字节码，并且成功地把动态图逻辑转换为静态图输出。

发现的缺陷：

程序在处理包含0,1Shape的动态Shape时，子图的编译次数不符合预期，观察其运行日志，发现有部分不应该命中的cache命中了，说明大小为0或者1的Shape没有被正确特化，而Pytorch是能够正确处理这种边界case的，说明现阶段PaddleSOT的动态图处理逻辑暂未和Pytorch完全对齐。

## 4.3静态扫描情况说明

未使用静态扫描手段

## 4.4建议

缺陷：

PaddleSOT目前能正确处理绝大多数的动态Shape输入，但是在遇到维度为0或者1的输入时，不能正确处理这种特化逻辑，使得原本应该被特化的逻辑被作为普通输入而泛化了，最后，在编译器端，为了满足broadcast的合理性，与0、1相关的动态Shape会被重复处理，从而编译出多余的kernel，拖累了编译时间，提高了时间复杂度。

改进建议：

我们可以调研Pytorch目前的实现：

@property

def specialize\_zero\_one(self) -> bool:

return self.settings.specialize\_zero\_one

可以发现，Pytorch设置了一个默认flag，会在遇到所有大小为0或者1的动态Shape时，为他们进行特化，走单独的处理路径，与>=2的Shape区分开来。

紧急程度：

中

因为该缺陷并不影响PaddleSOT模块整体的正确性，带来的负面影响是稍微下降的性能和更长的编译时间，所以属于紧急程度中等的修复项，

## 4.5评价

PaddleSOT经过长期的演进，已经能够将绝大多数的Python动态图逻辑转换为最终的静态图输出，并且在Python3.13发版后，快速进行了适配，面对新增和改动的字节码也能达到1 00%的转换率。目前，基本的使用场景下能够同时保证高正确率和良好的性能；但是在面对可能性达到指数级别的动态Shape场景时，由于该场景天然的复杂度，会导致最终在编译器端产生多余的kernel，冗余的数量也达到了和输入对应的指数级别。该缺陷虽然不影响正确性，但是会延长后端编译时间，影响用户的使用体验，有待修复。

结论：

PaddleSOT能够出色地完成生产环境中的Python动转静转化，在部分复杂的corner case上，性能有待进一步优化，修复项紧急程度为中。