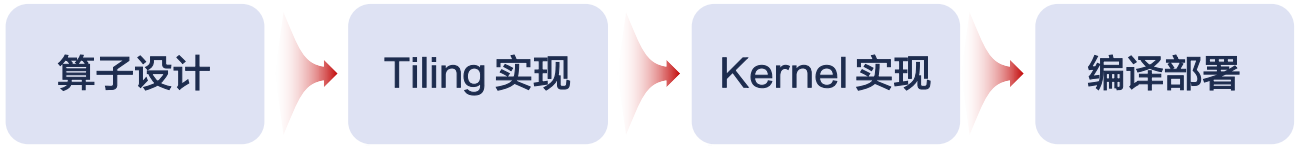
Ascend C算子模版库ATVC

吴林玉 CANN Ascend C技术专家

CANN

目录

Vector算子开发现状：开发复杂度高，重复工作量大



- 计算过程繁杂
- 需要感知多核切分
- 同类算子切分策略大同小异

- 同类算子数据搬入、计算、搬出、同步等核心逻辑基本相同
- 同类算子的基础操作代码重复

同类算子代码重复率高
需要额外的tiling实现

- **Host侧Tiling实现：**
- NPU片上高速缓存无法完全容纳输入输出的所有数据
- 数据分批次（切分）进行处理
- 切分算法称为Tiling算法

- **Device侧Kernel实现：**
- 算子核函数实现
- 包括 计算、数据搬运、内存管理、任务同步等API
- 多数为计算密集型任务，需要在NPU上执行

CANN

传统Vector算子开发

```cpp
uint32_t IsInfTiling::GetNeedCoreNum(uint32_t
maxCoreNum) const
{
    uint32_t coreNum = (uint32_t)CeilAlign(totalData,
DATA_BLOCK);
    if (coreNum < maxCoreNum) {
        return coreNum;
    } else {
        return maxCoreNum;
    }
}

void IsInfTiling::AssignDataToEachCore()
{
    perCoreData = totalData / needCoreNum;
    perCoreData = perCoreData / DATA_BLOCK *
DATA_BLOCK;
    uint32_t tailData = totalData - perCoreData *
needCoreNum;
    tailDataCoreNum = tailData / DATA_BLOCK;
    lastCoreData = perCoreData + tailData %
DATA_BLOCK;
}

uint32_t IsInfTiling::GetUsableUbMemory(uint64_t
ubSize)
{
    // The remaining UB size is split in two, double
buffer, input and output
    uint32_t usedUbSize = uint32_t(ubSize -
resevedUbSize - tilingData.GetDataSize()) / UB_PART;
    usedUbSize = usedUbSize / dataBlockSize *
dataBlockSize;
    return usedUbSize;
}
```

tiling切分逻辑

```cpp
// kernel function
extern "C" __global__ __aicore__ void is_inf(GM_ADDR
inputs, GM_ADDR outputs, GM_ADDR workspace, GM_ADDR
tiling)
{
    if (workspace == nullptr) {
        return;
    }
    GET_TILING_DATA(tilingData, tiling);

    GM_ADDR userWS = nullptr;

    const int16_t F16_INF_NUM = 0x7c00;
    const int16_t BF16_INF_NUM = 0x7f80;
    const int32_t FLOAT_INF_NUM = 0x7f800000;
    const int16_t SIGN_MASK = 0x7fff;

    if (TILING_KEY_IS(1)) {
        IsInf<half, SIGN_MASK, F16_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    } else if (TILING_KEY_IS(2)) {
        IsInf<float, SIGN_MASK, FLOAT_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    } else if (TILING_KEY_IS(3)) {
        IsInf<half, SIGN_MASK, BF16_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    }
}
```

kernel核函数

```cpp
template <typename T, auto MASK, auto INF_MASK>
__aicore__ inline void IsInf<T, MASK,
INF_MASK>::CompareInf(const int32_t dataLength)
{
    LocalTensor<int16_t> ubX =
inputQueue.DeQue<int16_t>();
    LocalTensor<uint8_t> result =
outputQueue.AllocTensor<uint8_t>();
    cacheTensor = cacheTensorBuff.Get<int16_t>();

    // 和sign_mask做按位与操作
    Duplicate(cacheTensor, (int16_t)MASK,
dataLength);
    And(ubX, ubX, cacheTensor, dataLength);

    uint32_t actualCalCount = dataLength /
selectInterval;
    Adds(ubX, ubX, (int16_t)-INF_MASK, dataLength);
    And(ubX, ubX, cacheTensor, dataLength);
    Mins(ubX, ubX, (int16_t)1, dataLength);
    Muls(ubX, ubX, (int16_t)-1, dataLength);
    Adds(ubX, ubX, (int16_t)1, dataLength);
    Cast(result, ubX.ReinterpretCast<half>(),
RoundMode::CAST_CEIL, actualCalCount);

    inputQueue.FreeTensor(ubX);
    outputQueue.EnQue(result);
}
```

算子计算逻辑

传统Vector算子开发

```cpp
uint32_t IsInfTiling::GetNeedCoreNum(uint32_t
maxCoreNum) const
{
    uint32_t coreNum = (uint32_t)CeilAlign(totalData,
DATA_BLOCK);
    if (coreNum < maxCoreNum) {
        return coreNum;
    } else {
        return maxCoreNum;
    }
}

void Is
{
    per
    per
DATA_BL
    uint
needCoreNum;
    tailDataCoreNum = tailData / DATA_BLOCK;
    lastCoreData = perCoreData + tailData %
DATA_BLOCK;
}

uint32_t IsInfTiling::GetUsableUbMemory(uint64_t
ubSize)
{
    // The remaining UB size is split in two, double
buffer, input and output
    uint32_t usedUbSize = uint32_t(ubSize -
resevedUbSize - tilingData.GetDataSize()) / UB_PART;
    usedUbSize = usedUbSize / dataBlockSize *
dataBlockSize;
    return usedUbSize;
}
```

- Tiling计算过程繁复
- 用户需要感知多核切分、UB内存空间分配等底层信息
- 同类算子的**切分策略大同小异**

tiling切分逻辑

```cpp
// kernel function
extern "C" __global__ __aicore__ void is_inf(GM_ADDR
inputs, GM_ADDR outputs, GM_ADDR workspace, GM_ADDR
tiling)
{
    if (workspace == nullptr) {
        return;
    }
    GET_TILING_DATA(tilingData, tiling);

    GM_ADDR

    const
    const
    const
    const

    if (TILI
        IsInf<half, SIGN_MASK, F16_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    } else if (TILING_KEY_IS(2)) {
        IsInf<float, SIGN_MASK, FLOAT_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    } else if (TILING_KEY_IS(3)) {
        IsInf<half, SIGN_MASK, BF16_INF_NUM> op;
        op.Init(inputs, outputs, userWS, &tilingData);
        op.Process();
    }
}
```

- **Kernel核函数需要完成数据搬运、流水同步等操作**
- 这些操作实现**逻辑较固定但易错**

kernel核函数

```cpp
template <typename T, auto MASK, auto INF_MASK>
__aicore__ inline void IsInf<T, MASK,
INF_MASK>::CompareInf(const int32_t dataLength)
{
    LocalTensor<int16_t> ubX =
inputQueue.DeQue<int16_t>();
    LocalTensor<uint8_t> result =
outputQueue.AllocTensor<uint8_t>();
    cacheTensor = cacheTensorBuff.Get<int16_t>();

    // 和sign mask做按位与操作

data

sele
    Adds(ubX, ubX, (int16_t)-1, dataLength);
    And(ubX, ubX, cacheTensor, dataLength);
    Mins(ubX, ubX, (int16_t)1, dataLength);
    Muls(ubX, ubX, (int16_t)-1, dataLength);
    Adds(ubX, ubX, (int16_t)1, dataLength);
    Cast(result, ubX.ReinterpretCast<half>(),
RoundMode::CAST_CEIL, actualCalCount);

    inputQueue.FreeTensor(ubX);
    outputQueue.EnQue(result);
}
```

- 只有用户计算逻辑实现代码有差异
- 但实际的算子开发，**上手太难了！**

算子计算逻辑

ATVC模板库设计目标：简单易上手，兼顾开发效率与性能

ATVC :Ascend C Templates for Vector Compute

轻松上手

屏蔽底层操作

新手小白也能快速上手

开发效率显著提升

使用内置tiling

代码量小

模板可扩展

开箱性能有保障

内置Tiling贴合硬件特性

最大化发挥硬件性能

良好的调试调优能力

编译期静态检查

支持通用的调试能力

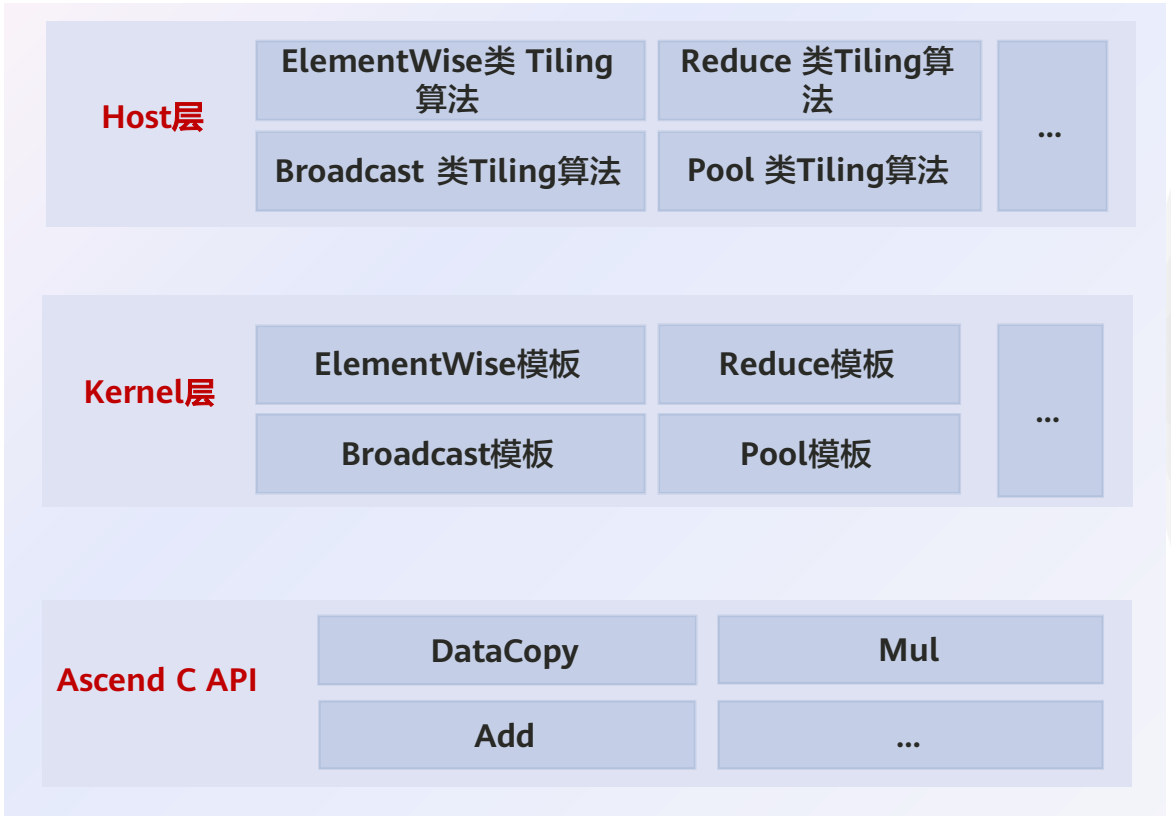
超参调优简单易用

CANN

ATVC（Ascend C Templates for Vector Compute）

•Kernel 层：提供典型Vector算子模板类，解耦算子固定模块和自定义模块。

•Host 层：提供多核、单核切分策略



- **极简编程**

  繁杂易错的底层实现由模板库实现，用户只需关注核心计算逻辑实现;

- **极速开发**

  ✓ 内置通用Tiling计算，开箱性能有保障；

  ✓ 提供简便的调试调优功能

- **支持扩展**

① EleWise

② Reduce

③ Broadcast

④ Reduce+ EleWise融合

⑤ Broadcast+ EleWise融合

⑥ Pool

https://gitcode.com/cann

CANN

ATVC极速编程

```
using AddOpTraits = ATVC::OpTraits<ATVC::OpInputs<float, float>, ATVC::OpOutputs<float>>;

template <typename Traits>
struct TanhGradComputeFunc {

  template <typename D_DY, typename D_X, typename D_Z>
  __aicore__ inline void operator()(LocalTensor<D_DY> dy, LocalTensor<D_X> y,  LocalTensor<D_Z> z) {
    auto length = y.GetSize();
    AscendC::Mul(y, y, y, length);
    AscendC::Mul(y, dy, y, length);
    AscendC::Sub(z, dy, y, length);
  }

};

__global__ __aicore__ void TanhGrad(GM_ADDR dy, GM_ADDR y, GM_ADDR z, ATVC::EleWiseParam param)
{
  KERNEL_TASK_TYPE_DEFAULT(KERNEL_TYPE_AIV_ONLY);
  auto op = ATVC::Kernel::EleWiseOpTemplate<TanhGradComputeFunc<Traits>>();
  op.Run(dy, y, z, &param);
}

ATVC::Host::CalcEleWiseTiling<AddOpTraits>(eleNum, param);
TanhGrad<AddOpTraits><<<blockNum, nullptr, stream>>>(dyDevice, yDevice, zDevice, param);
```

ATVC极速编程

```
using AddOpTraits = ATVC::OpTraits<ATVC::OpInputs<float, float>, ATVC::OpOutputs<float>>;

template <typename Traits>
struct TanhGradComputeFunc {
  template <typename D_DY, typename D_X, typename D_Z>
  __aicore__ inline void operator()(LocalTensor<D_DY> dy, LocalTensor<D_X> y, LocalTensor<D_Z> z) {
    auto length = y.GetSize();
    AscendC::Mul(y, y, y, length);
    AscendC::Mul(y, dy, y, length);
    AscendC::Sub(z, dy, y, length);
  }
};

__global__ __aicore__ void TanhGrad(GM_ADDR dy, GM_ADDR y, GM_ADDR z, ATVC::EleWiseParam param)
{
  KERNEL_TASK_TYPE_DEFAULT(KERNEL_TYPE_AIV_ONLY);
  auto op = ATVC::Kernel::EleWiseOpTemplate<TanhGradComputeFunc<Traits>>();
  op.Run(dy, y, z, &param);
}

ATVC::Host::CalcEleWiseTiling<AddOpTraits>(eleNum, param);
TanhGrad<AddOpTraits><<<blockNum, nullptr, stream>>>(dyDevice, yDevice, zDevice, param);
```

1. 描述算子输入输出信息

2. 用户自定义算子计算逻辑

屏蔽底层信息，聚焦计算实现

3. 使用Kernel模板

完成核函数功能

4. Host侧切分计算

kernel调用

ATVC算子开发效率与性能表现

新手算子开发者

极速开发

普通算子开发者

快速开发、快速性能调优

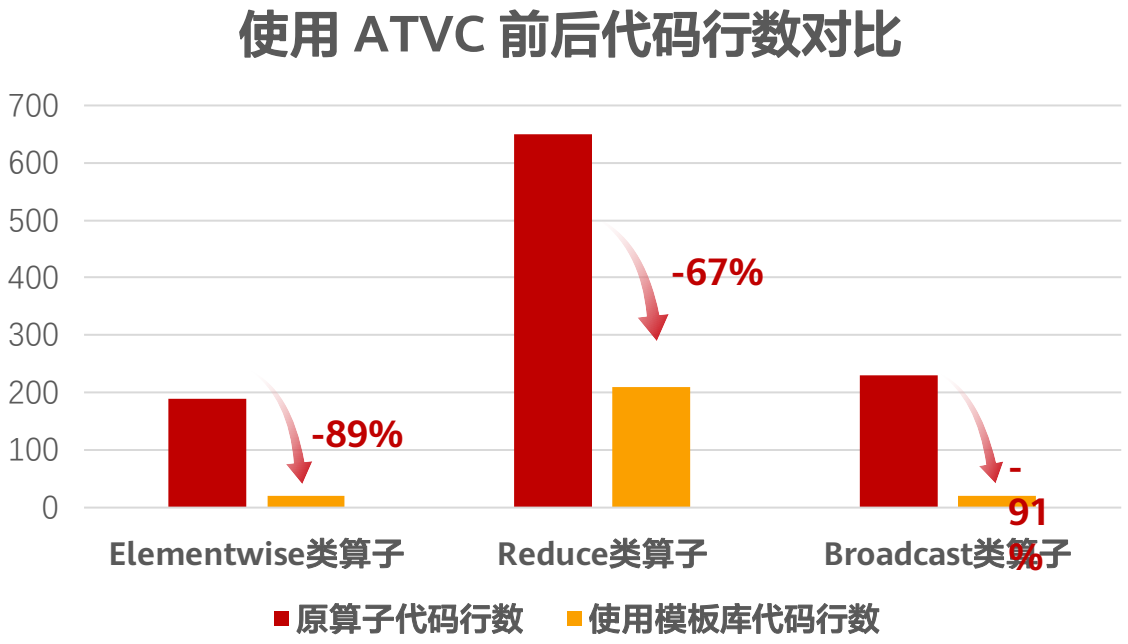
针对不同用户能力模型，提供多种Vector算子开发路径，用户开发Vector类算子效率提升65%以上（1人月 -> 1人周）

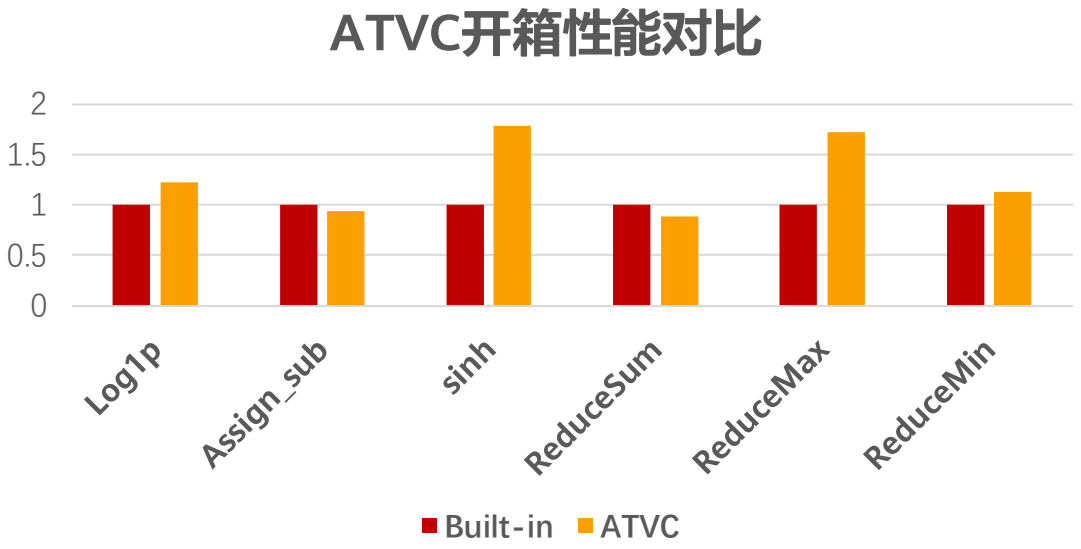
- **新手开发者：平均性能达成0.8x built-in算子**

  用户根据算子模板样例，快速完成算子计算实现，开箱性能达成0.8x built-in 算子。

- **普通开发者：**

  提供超参调优，精细化调优，可达极致性能。





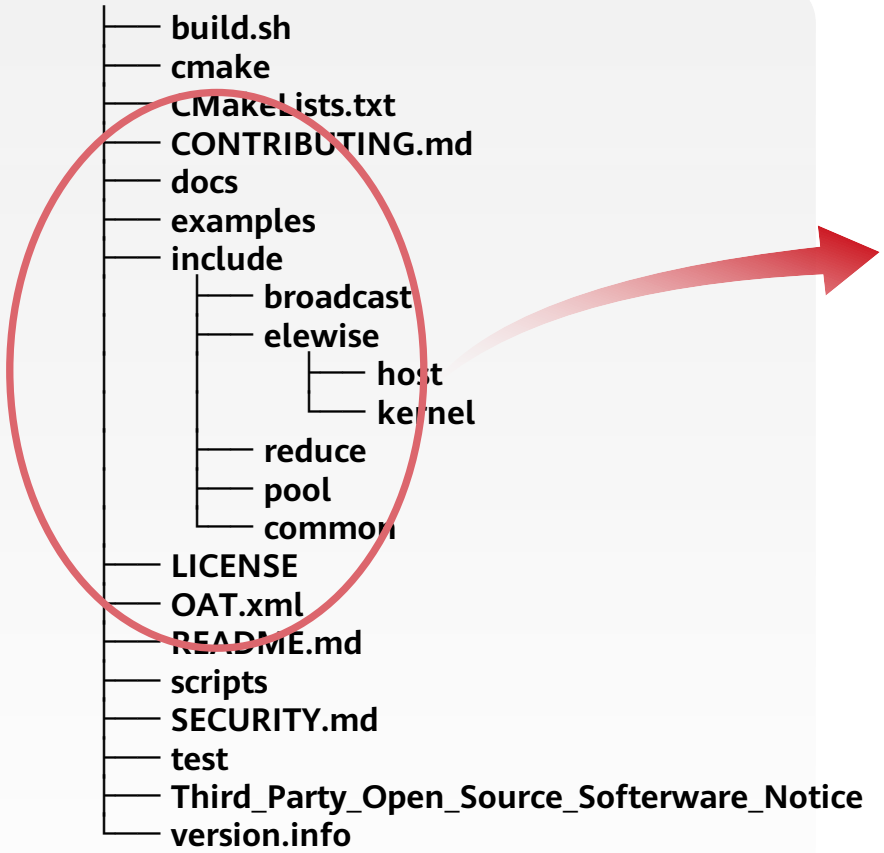
*基于2000+ case的平均性能结果

https://gitcode.com/cann

CANN

目录

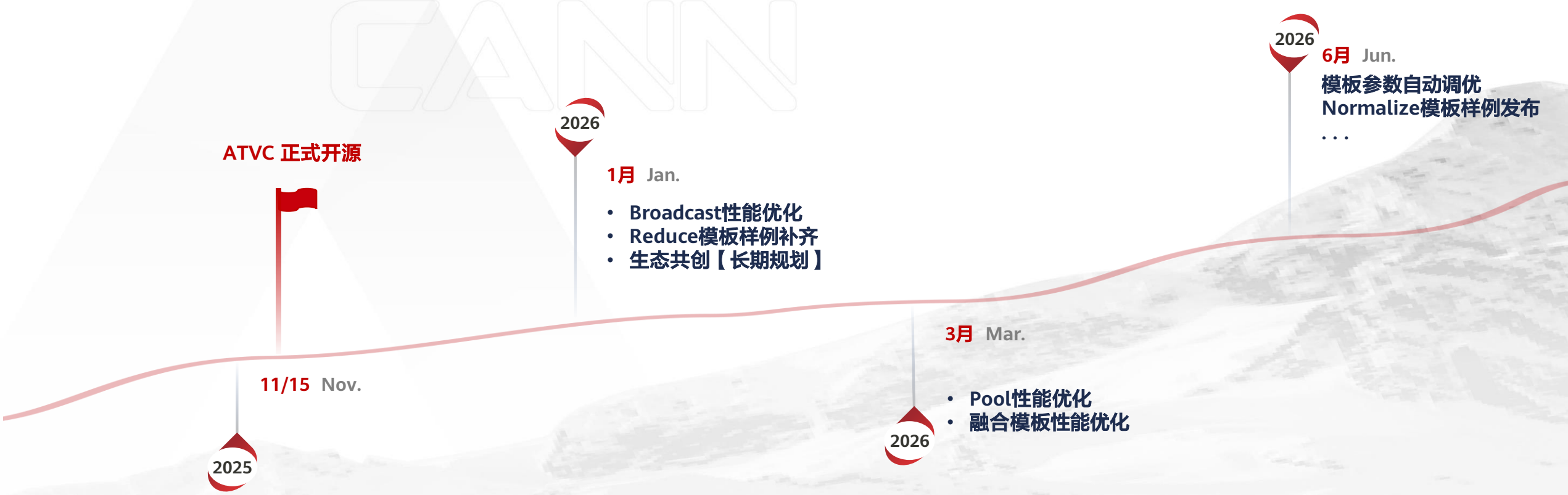
ATVC仓介绍



- **Docs：**包含模板使用开发指南等文档。
- **Examples：**算子开发样例。
- **include：**模板实现头文件库。
  - ELementwise
  - Reduce
  - Broadcast
  - Pool
- **各类模板：**
  - ✓ host：模板动态参数计算实现；
  - ✓ kernel：模板实现，完成kernel侧数据搬运、计算调度。

代码仓链接：https://gitcode.com/cann/atvc

ATVC未来规划：更丰富、更优、更易用



https://gitcode.com/cann

CANN

欢迎开发者交流 & 贡献

扫码关注ATVC 代码仓

https://gitcode.com/cann/atvc

欢迎交流！欢迎贡献！欢迎共创合作！