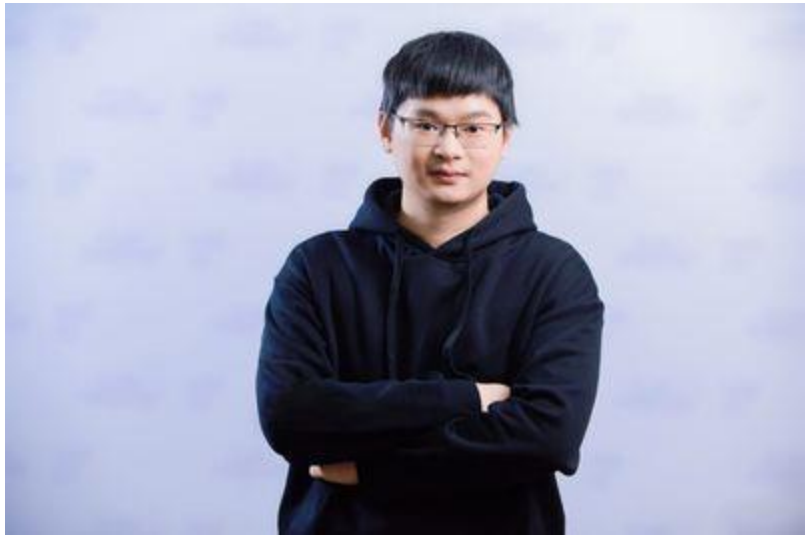
GOSIM

GOSIM AI Paris 2025

The Best Practice of Training and Inferencing on Ascend CANN



- Over 10 years opensource contribution.
- Ex OpenStack Keystone committer
- openEuler Cloud compute SIG maintainer
- vLLM Ascend maintainer

GOSIM

GOSIM AI Paris 2025

① Technical Overview and Ecology of CANN

②vLLM Best Practices of Ultimate Inference and Training and Promotion

GOSIM

GOSIM AI Paris 2025

Technical Overview and Ecology of CANN

Ascend CANN: Enable Efficient Development of Native Applications and Unleash Ascend Computing Power

GOSIM

1 Enable parallel computing acceleration for large models.

Optimize operators and algorithms by using mathematical power to release the computing power.

2 Efficient native development and ecological migration

Operator development period in typical scenarios < 1.5 man-weeks

3 Deep openness and ecosystem compatibility

Provides high-level component best practices to enable native innovation

AI framework

Supports AI frameworks in the industry and adapts to the PyTorch community version.

PyTorch | MindSpore 昇思 | TensorFlow | 飞桨 | Caffe | Jittor 计图 | ……

CANN

GE Graph Engine

Graph compilation, graph execution, and processor parallel acceleration, improving computing performance

Ascend C operator programming language

Simplified development of convergent operators is supported.
C/C++ Specification | High-Layer API | Hybrid Programming

Operator acceleration library

High Performance Fusion Operator for Large Model Structure Generalization over Transformer Networks

HCCL aggregate communication library

* Efficiently realizes single-machine multi-card and multi-machine multi-card distributed training.
* high performance set communication algorithm
* Unified hard scheduling for computing communication
* High-performance concurrent computing communication

BiSheng Compiler Bisheng Compiler

Heterogeneous Compilation | Global Optimization | Instruction Affinity

Runtime

Control flow|Memory management|Task scheduling|Device management

Driver driver

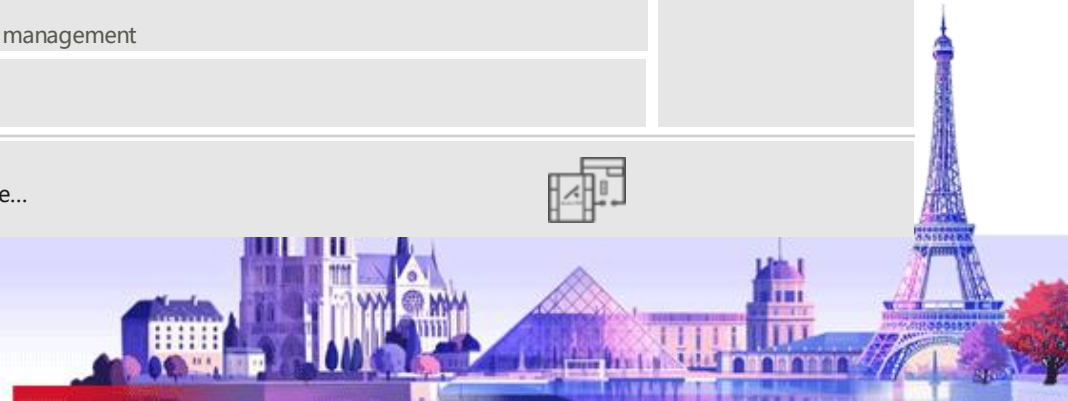
MindStudio E2E tool chain

* Operator development tool
* Training and inference development tool
* Visualization tool

Ascend hardware

Ascend

Ascend Series Hardware...



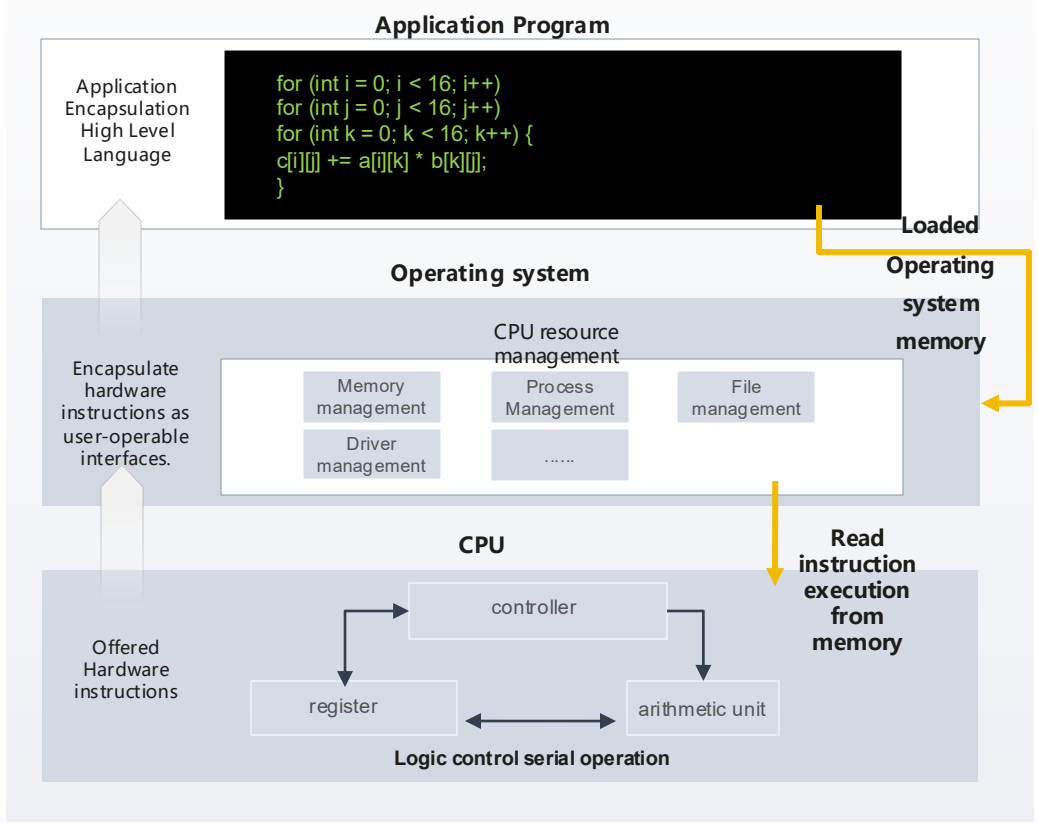
General Computing

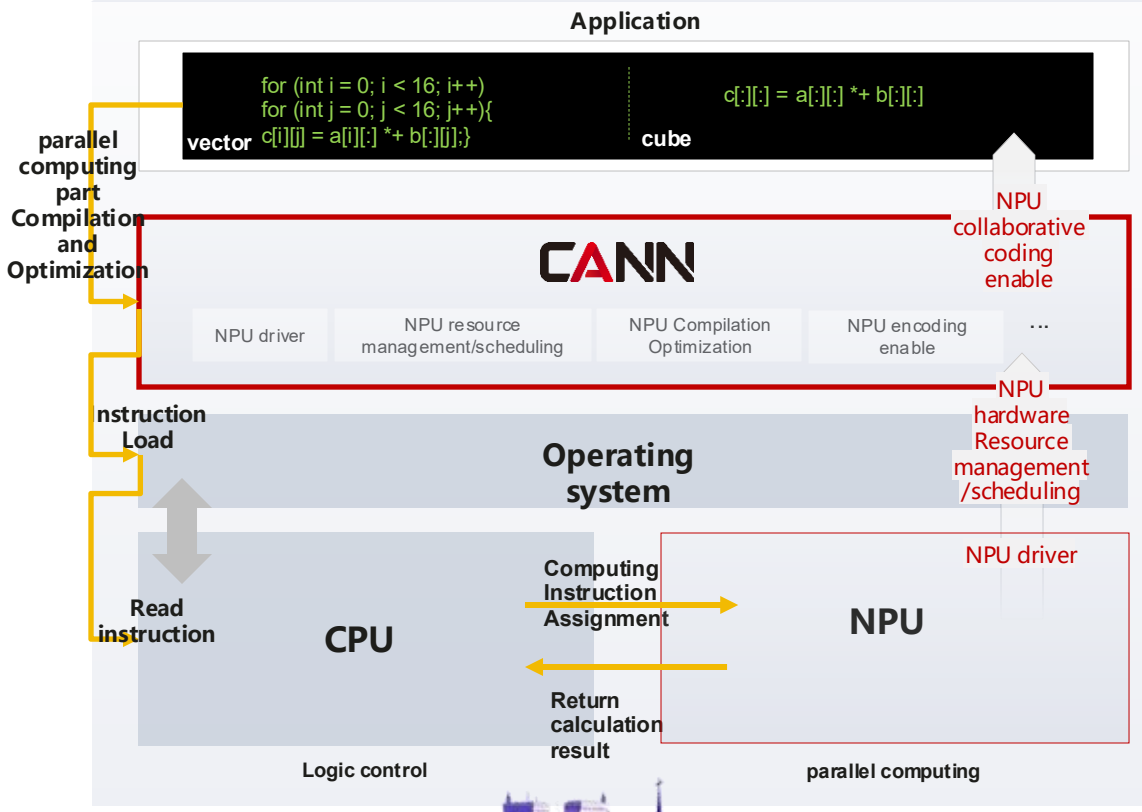
OS Enables CPU Hardware Instruction Abstract Encoding and Execution

Through the NPU Offloading CPU Computing Po

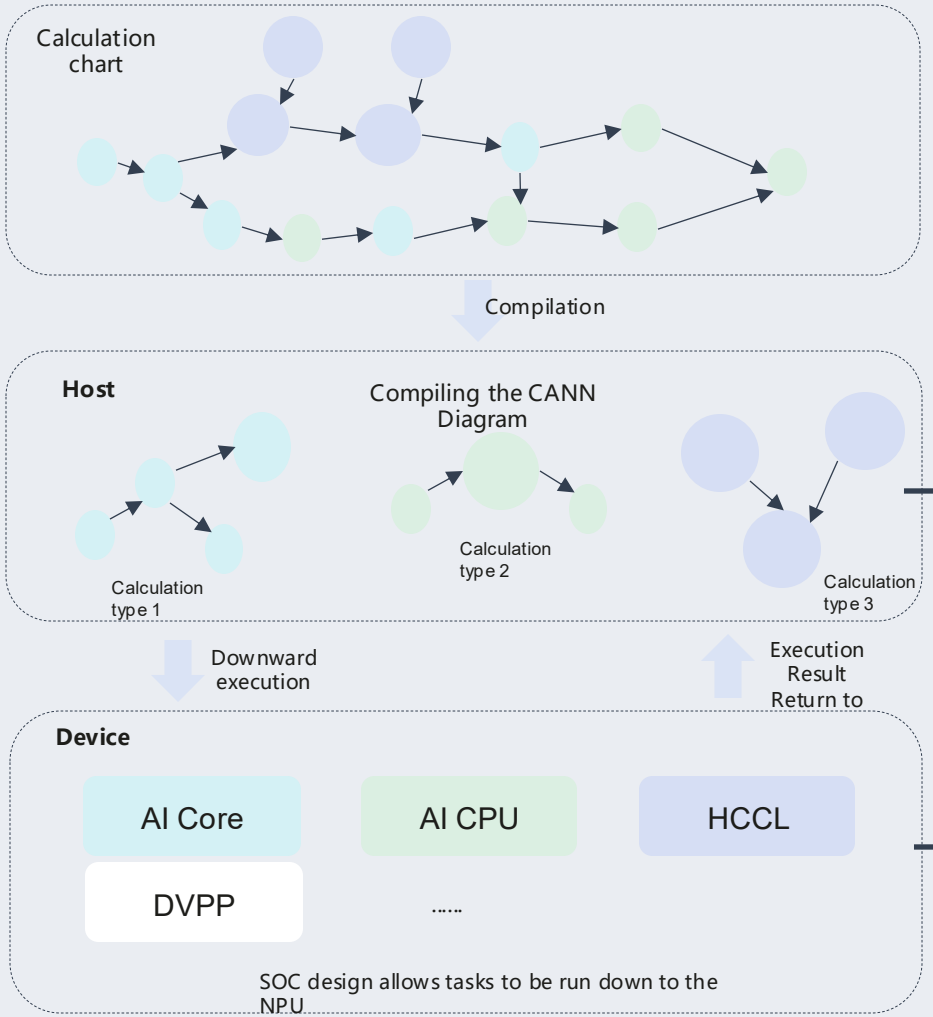
AI computing

CANN enables collaborative encoding between the CPU and the NPU to maximize the computing capability of the NPU.





The computing graph execution sinking technology implements closed-loop computing on the processor side, improving computing performance



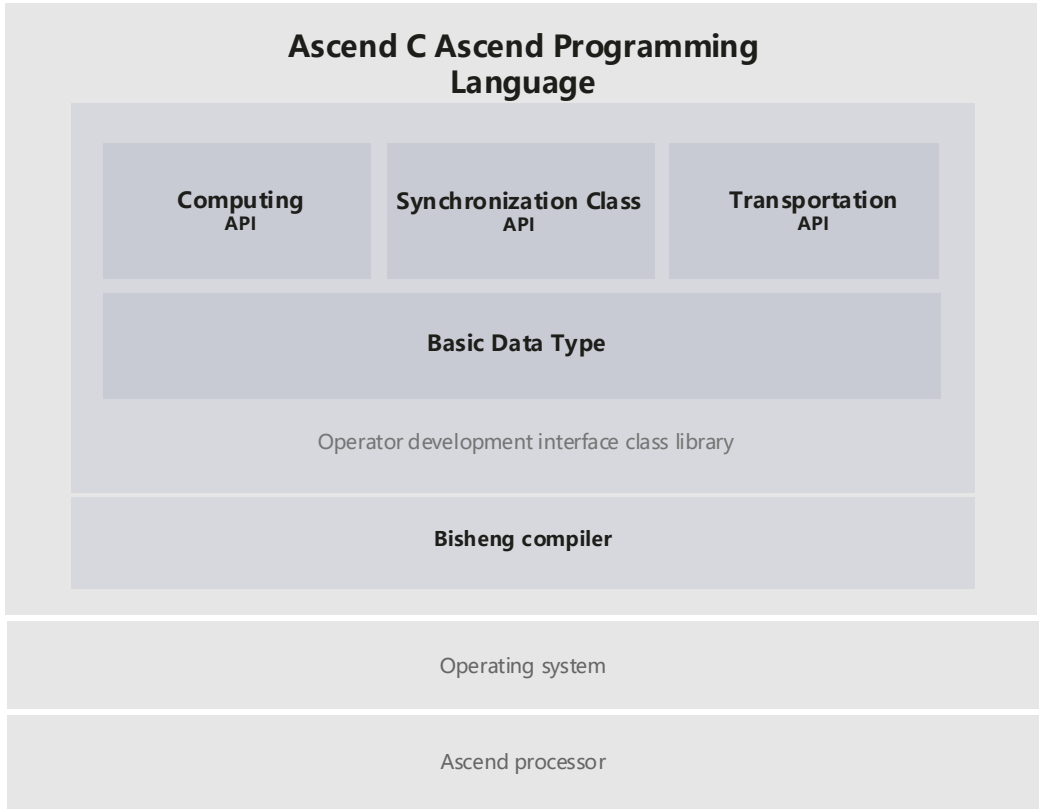
Whole picture sinking

Control flow, DVPP, aggregate communication, and computing are all moved downwards.
Reduce the time required for the interaction between the host and the device.

The calculation type is split during diagram compilation,
Perform computing, control flow, DVPP, and communication tasks in the downstream direction.
Multiple on-chip engines work together to balance computing power.

Ascend AI processor integrates multiple computing resources
Computing tasks can be scheduled at the hardware layer.
One-time delivery and unlimited execution
Iterative training tasks can be completely closed on the device side, eliminating scheduling overhead.

Ascend C operator programming language, enabling simplified operator development.




```
__aicore__ inline void ProcessLoop(GM_ADDR enc_chunk_data, GM_ADDR dec_chunk_data, GM_ADDR dec_v_weight,GM_ADDR buf_s_t, int32_t loop_count)
{
    enc_chunk_data_gm.SetGlobalBuffer((__gm__ float*) enc_chunk_data, seq_*dim_);
    dec_chunk_data_gm.SetGlobalBuffer((__gm__ float*) dec_chunk_data, beam_num_*dim_);
    dec_v_weight_gm.SetGlobalBuffer((__gm__ float*) dec_v_weight, dim_);
    buf_s_t_gm.SetGlobalBuffer((__gm__ float*) buf_s_t, beam_num_*seq_);
    for(int i=0;i<loop_count;i++)
    {
        CopyIn(prefix_sum_ + i);
        Compute();
        CopyOut(prefix_sum_ + i);
    }
}
```

structured kernel function programming

Operator Kernel Function Implementation

Intra-core automatic pipeline parallel scheduling

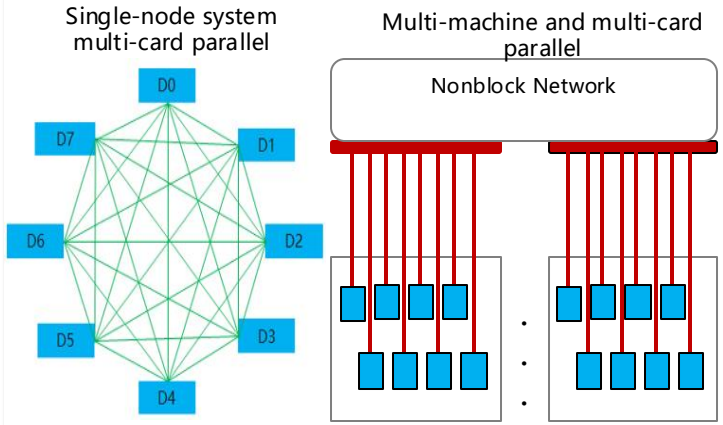
HCCL aggregate communication library, implementing high-performance communication

Huawei Collective Communication Library (HCCL) is a high-performance collective communication library based on the Ascend AI processor. Provides the single-node multi-card and multi-node multi-card aggregate communication primitives, and implements aggregate communication over the HCCS, RoCE, and PCIe high-speed links.

high performance set communication algorithm

Improving communication efficiency of massively parallel computing

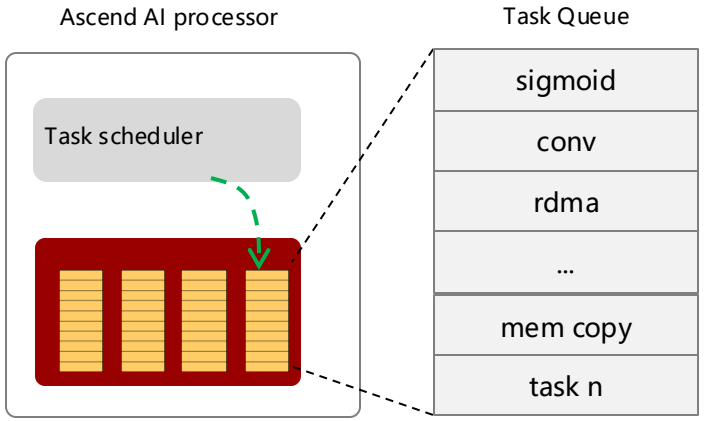
A basic topology inside the server is a full mesh interconnection topology, and a basic algorithm in each full mesh interconnection topology is a mesh algorithm. Supports communication algorithms such as HD, Ring, and NHR between servers.



Unified hard scheduling for computing communication

Reduce scheduling overhead and optimize hardware resource utilization.

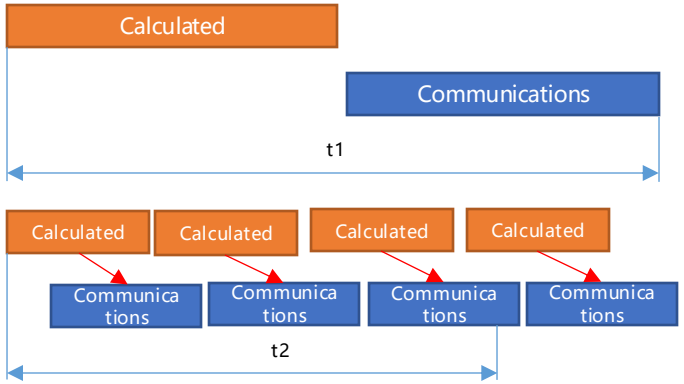
Dedicated hardware scheduling engine and hardware communication primitives, implementing full-hardening scheduling of computing and communication tasks, reducing scheduling overhead and accurately controlling jitter.



High-performance concurrent computing communication

The system performance is further improved.

Convergence communication operations (AllReduce, ReduceScatter, and Reduce) are implemented in channel-associated mode. Computing communication tasks cover each other and are executed concurrently without occupying computing resources.



Six convergent operators, 27 Ascend C high-level APIs, and 7 HCCL communication algorithm examples have been launched in the community.

fusion operator	Ascend C high-level API				HCCL aggregate communication library
FA	AdjustSoftMaxRes	GeGlu	ReGlu	SoftmaxFlashV2	HCCL communication API
FAG	AscendAntiQuant	Gelu	RmsNorm	SoftmaxGrad	HCCL communication domain management
FFN	AscendDequant	LayerNorm	Sigmoid	SoftmaxGradFront	HCCL Communication Algorithm-HD
FIA	AscendQuant	LayerNormGrad	Silu	Swiglu	HCCL Communication Algorithm-Mesh
IFA	BatchNorm	LayerNormGradBeta	SimpleSoftmax	Swish	HCCL Communication Algorithm-Pairwise
PFA	DeepNorm	LogSoftMax	SoftMax	TopK	HCCL Communication Algorithm-Ring
	FasterGelu	Matmul	SoftmaxFlash		HCCL Communication Algorithm-Star

https://gitee.com/ascend/cann-ops-adv

model

Application

Hugging Face Tool chain

deepspeed
Acceleration Library/Engine
vLLM

PyTorch framed

CANN

Computing power

Currently, Ascend supports the following open source software:

Fine-tuning, tool chain:

•Huggingface transformers(since v4.32, 2023): huggingface/transformers/pull/24879

•Huggingface peft (since 0.5.0, 2023): huggingface/peft/pull/772

•Huggingface accelerate(since 0.22.0, 2023): huggingface/accelerate/pull/1676

•LLaMA-Factory (since v0.7.1, 2024): hiyouga/LLaMA-Factory/pull/975

•FastChat (since v0.2.29, 2023): lm-sys/FastChat/pull/2422

•stable-diffusion-webui (since v1.8.0, 2024): stable-diffusion-webui/pull/14801

•text-generation-webui (since v1.8, 2024): text-generation-webui/pull/5541

•OpenCompass (since v0.3.4, 2024): opencompass/pull/1250 & 1618

•lm-evaluation-harness (since v0.4.4, 2024): lm-evaluation-harness/pull/1886

•ComfyUI (since Dec.2024): ComfyUI/pull/5436

•DeepSpeed (since 2024.01)

Inference engine:

vLLM vllm-project/vllm-ascend

ONNX Runtime (since v1.13.1) microsoft/onnxruntime/pull/12416

llama.cpp (since July.2024) llama.cpp/pull/6035

Whisper.cpp (since Aug. 2024) whisper.cpp/pull/2336

AI framework:

PyTorch (since 2.1, 2023) pytorch/releases/tag/v2.1.0

MindSpore(since 1.0, 2020)

GOSIM

GOSIM AI Paris 2025

vLLM Best Practices of Ultimate Inference and Training and Promotion

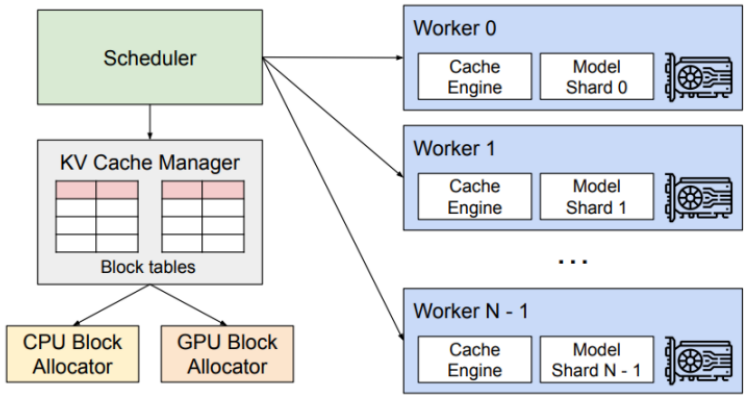
Easy-to-use and high-performance open-source full-stack inference service framework

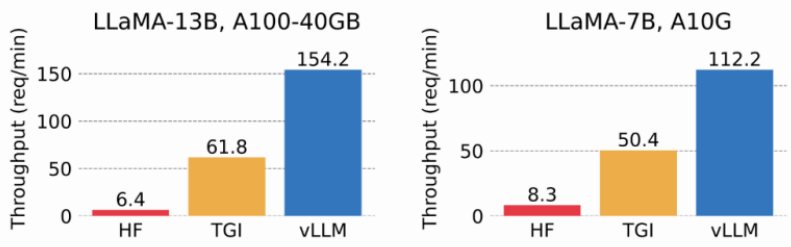
GOSIM

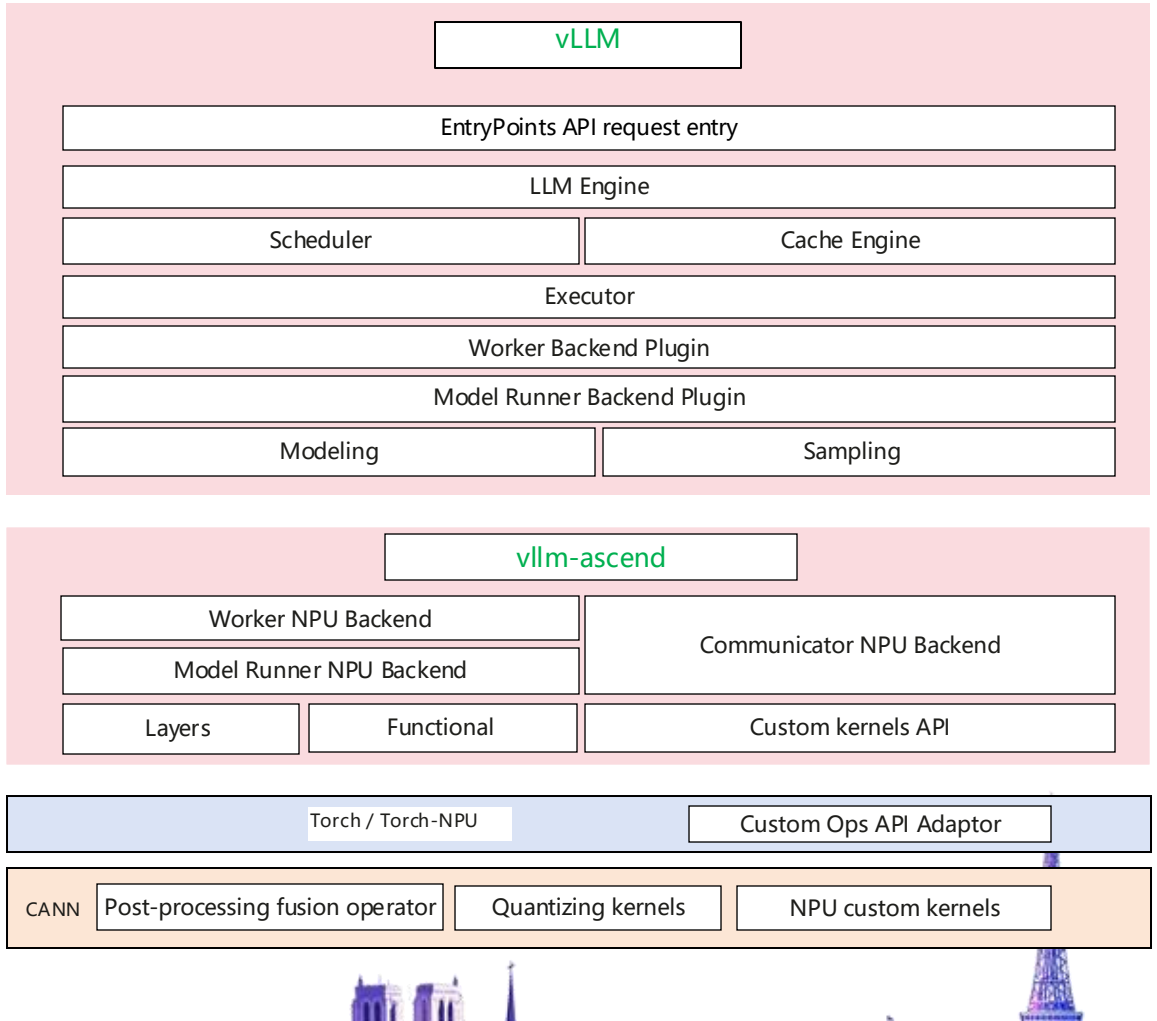
- **Open source date: June 2023**
- **Open source organization:** Berkeley Sky Computing Lab (formerly RiseLab and AMPLab), which has hatched famous open source projects such as Apache Spark, Ray, and Alluxio.
- **Project introduction:** vLLM uses the PagedAttention algorithm as the core algorithm to improve memory utilization, supports more than 100 generative large language models, and achieves 24 times the throughput of Hugging Face Transformers.
- **Competitive features:** Start from PagedAttetion, gradually add mainstream inference features, learn from Pytorch, and focus on usability and diversity.

vLLM

Easy, fast, and cheap LLM serving for everyone





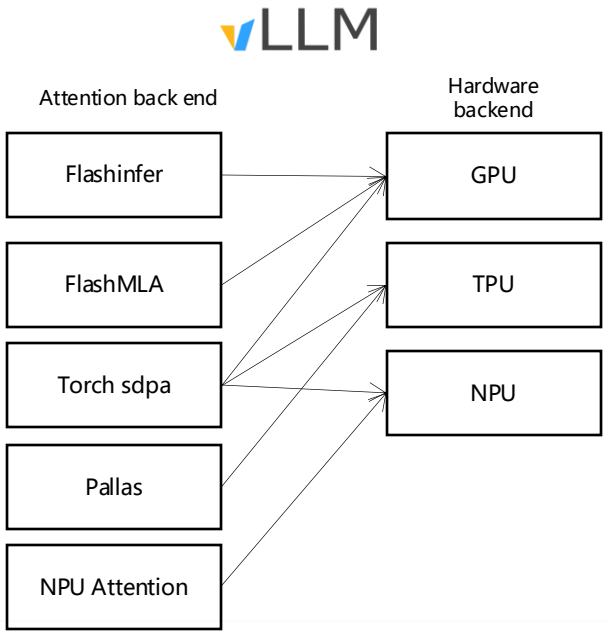


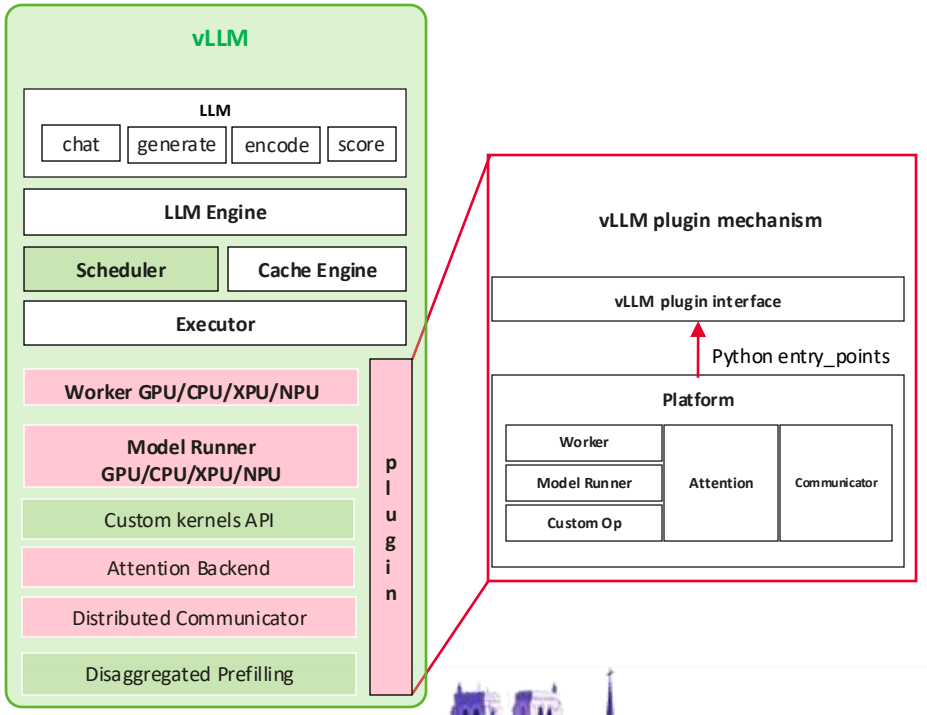
GOSIM AI Paris 2025

Core issue: Different hardware platforms (such as GPU and NPU) have different architecture features. Operator implementation needs to be optimized for each hardware. Different hardware platforms also support attention backends differently. How to ensure efficient running of multiple backends on various hardware is a difficult problem.

Solution:

1. Supports multiple types of attention backends and supports many-to-many use of attention backends and hardware devices.
2. Plug-in design, hardware device scalability, and customized module implementation
3. Optimize with specific hardware acceleration libraries (e.g., CUDA, ROCm, CANN) for different hardware platforms

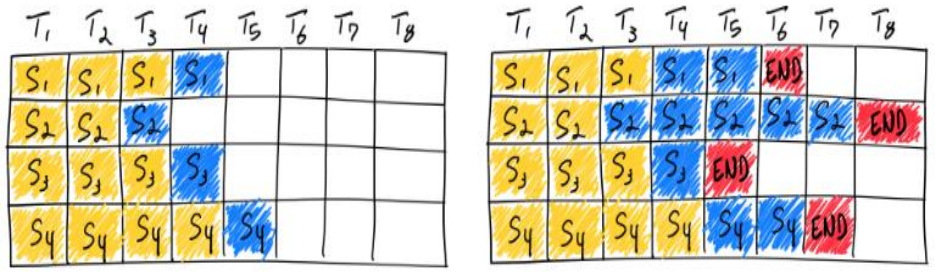




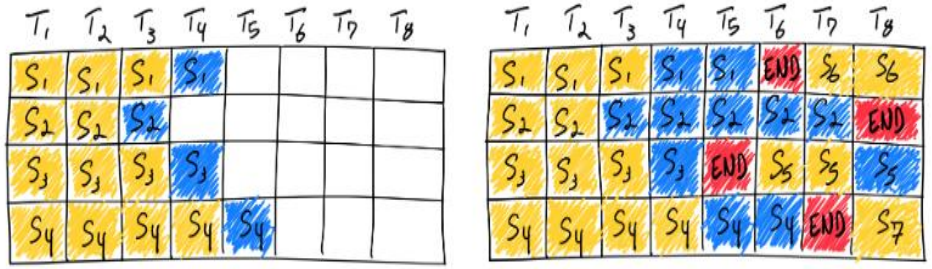
Core issue: In scenarios such as input and output length differences and dynamic addition of new requests, NPU resources are idle in the case of batch inference requests, wasting computing power.

Solution:

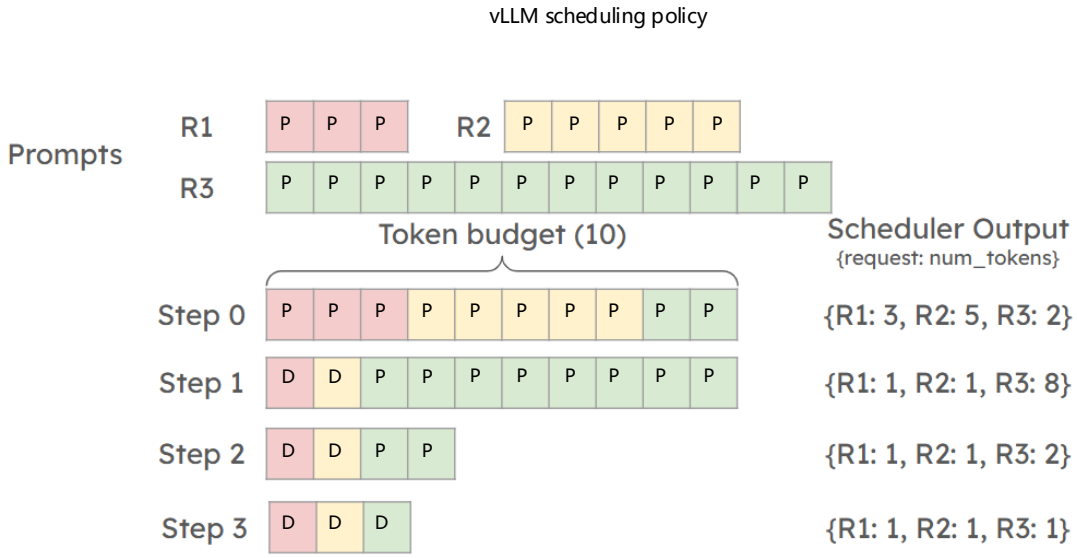
- In the multi-batch (multiple users in parallel) inference scenario, the time sequence of the batch is optimized to eliminate bubbles and improve the NPU usage and throughput.



Before the optimization: Computing resources are empty.



After optimization: Make full use of resources

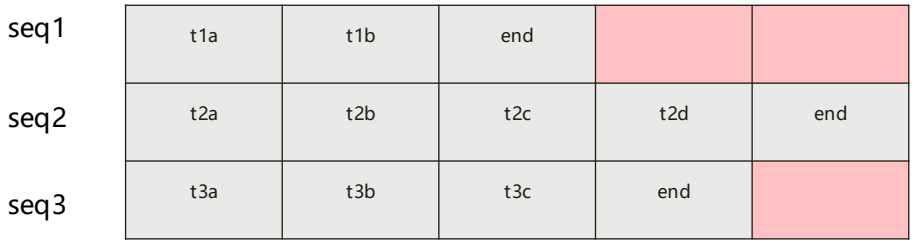


Core issues:

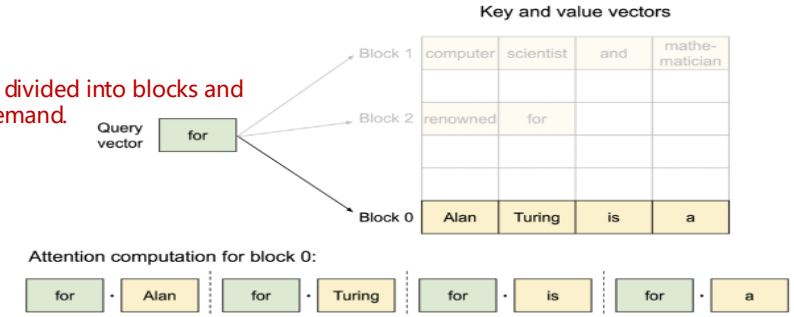
1. In the traditional attention mechanism, key-value pairs (Key-Value Pairs) need to allocate a continuous large block of video memory for each sequence, resulting in low memory utilization, difficult to expand, and high video memory usage
2. During long sequence inference, attention computing requires frequent access and storage of a large amount of KV cache data, which may cause memory bandwidth bottlenecks and affect the inference speed.
3. The sequence lengths of different requests may vary greatly. Traditional methods cannot flexibly process sequences of dynamic lengths, which may cause video memory waste or low scheduling efficiency.

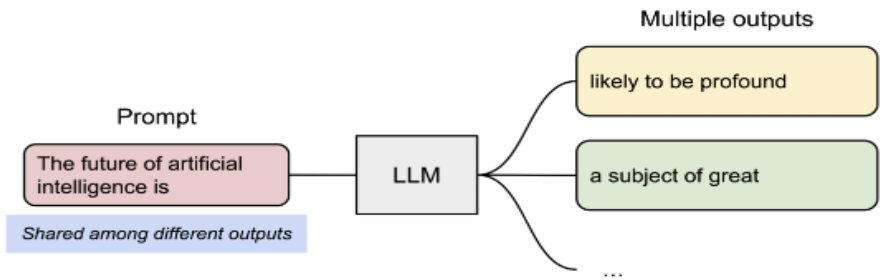
Solution:

1.The core idea of PagedAttention is to introduce the paging mechanism to divide the KV cache of each sequence into blocks of a fixed size. Each block contains a fixed number of tokens. In this way, large blocks of video memory are avoided for the entire sequence at one time.
2.Dynamic video memory management. By allocating and releasing video memory blocks on demand, PagedAttention can dynamically adjust the video memory usage according to actual requirements, avoiding the problem of video memory waste in traditional methods.

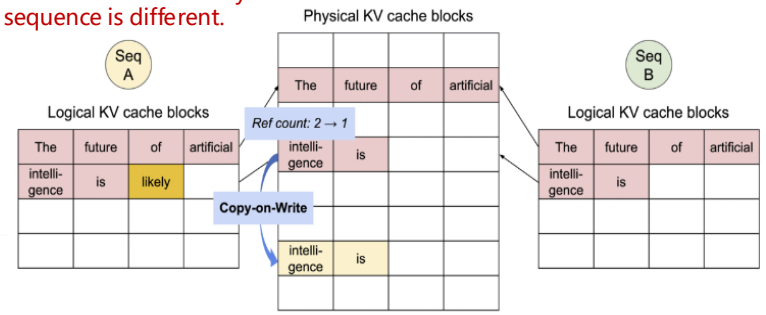


The memory is divided into blocks and allocated on demand.





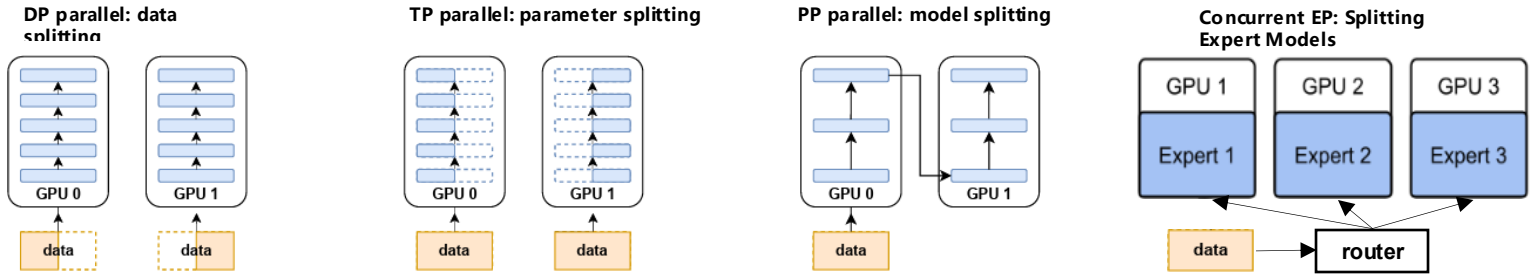
An independent block is allocated only when the output of each sequence is different.



Core issue: Single-card inference cannot meet the requirements for installation (insufficient video memory) and fast push (low latency).

- TP parallel based on single-node system
- TP/PP/EP parallelism based on multiple computers

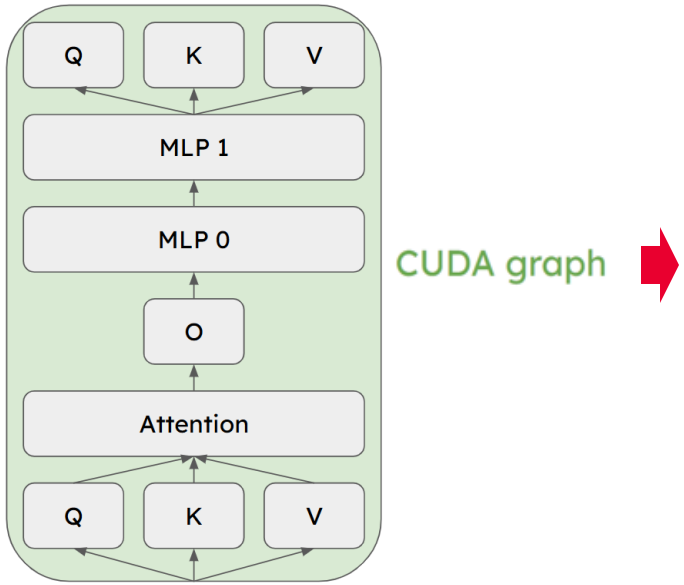
Solution: When the number of model parameters is large, the TP, PP, and EP segments are used to reduce the overhead of the video memory occupied by a single card by weight. More video memory is reserved for the KVCache to support a larger number of requests, meeting the cost-effective requirements.



Core issue: In the single-operator scenario, each operation in the computing graph needs to be scheduled separately, which causes frequent waiting time and context switchover, increasing the delay. As the inference process progresses, the allocation and release of the video memory may cause fragmentation and reduce the video memory usage.

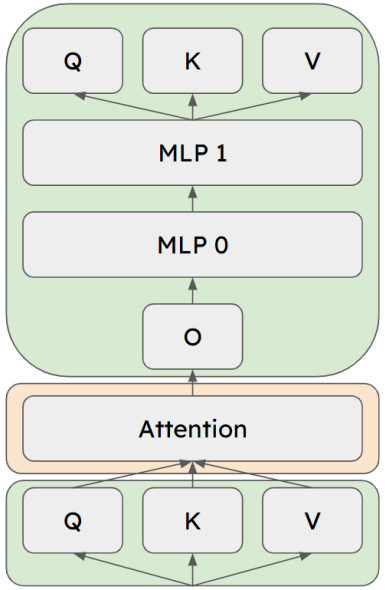
Solution:

1. Divide the network-wide calculation graph into multiple subgraphs to reduce scheduling overhead because multiple operations can be processed in batches at a time.
2. Precompile the calculation graph to further improve the utilization of hardware resources.



CUDA graph

V0: Entire image compilation. Dynamic shape is not supported.



CUDA graph N

PyTorch Eager

CUDA graph N-1

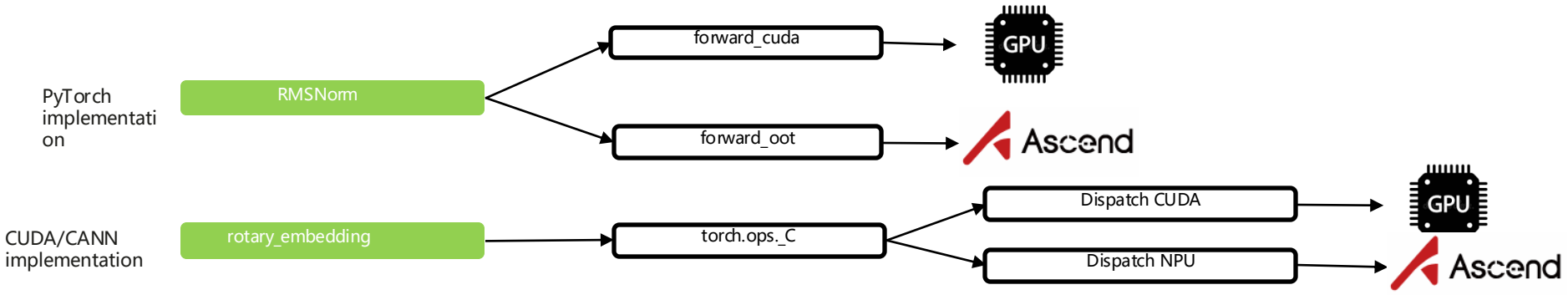
V1: Segmented compilation, dynamic shape, CPU overhead slightly improved, and overall performance

vllm-ascend's ACLGraph is being supported.

Core problem: The native operator may not fully utilize hardware resources. For example, in a multi-card parallel environment, the operator implementation needs to consider the overhead of distributed communication.

Solution:

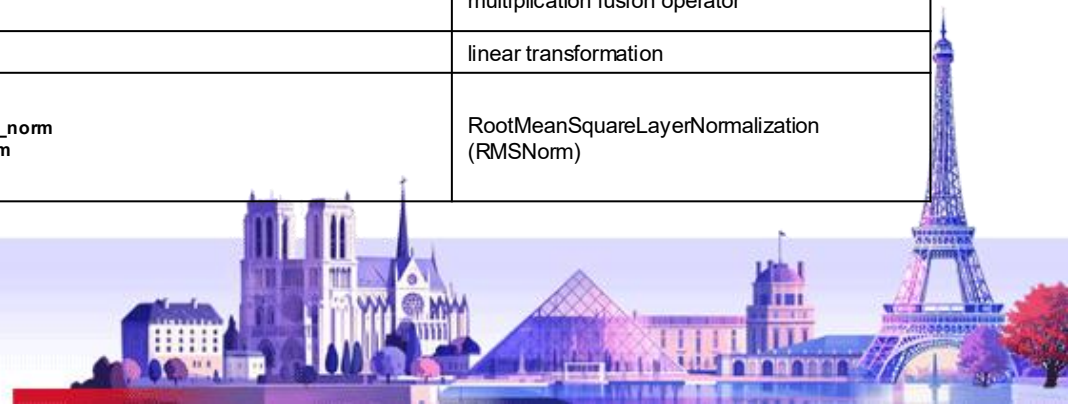
1. Using NPU CANN Programming Technology to Design High Efficiency custom operator
2. Multiple operators involved in self-attention calculation (e.g. QKV calculation, Softmax, matrix multiplication, etc.) Integrate into an efficient operator, such as FlashAttention, which is well known in the industry.



The vLLM supports the access and implementation of customized and convergent operators based on PyTorch and CUDA/CANN.

Take Qwen3 as an example. What support does vLLM+Ascend+CANN provide?

Model			Qwen method	vLLM operator	Corresponding torch port	Description
Qwen3Model			embed_tokens	VocabParallelEmbedding	torch.nn.funcational.embedding	Input token vectorization
	Qwen3Decoder Layer		input_layernorm	RMSNorm	torch_npu.npu_add_rms_norm torch_npu.npu_rms_norm	RootMeanSquareLayerNormalization (RMSNorm)
		Qwen3Attention	qkv_proj	QKVParallelLinear	torch.nn.linear	linear transformation
			q_norm,k_norm	RMSNorm	torch.rsqrt	QK root mean square normalization
			rotary_emb	RotaryEmbedding	torch_npu._npu_rotary_embedding	Rotary Position Coding (RoPE:RotaryPostionEmbedding)
			attention	Attention	torch_npu._npu_reshape_and_cache torch_npu._npu_flash_attention torch_npu._npu_paged_attention	transformerself-attention
			o_proj	RowParallelLinear	torch.nn.linear	linear transformation
			post_layernorm	RMSNorm	torch_npu.npu_add_rms_norm torch_npu.npu_rms_norm	RootMeanSquareLayerNormalization (RMSNorm)
		Qwen3MLP	gate_up_proj	MergedColumnParallelLinear	torch.nn.linear	linear transformation
			act_fn	SiluAndMul	torch_npu.npu_swiglu	SiluAndMul:SwiGLU activation function + multiplication fusion operator
			down_proj	RowParallelLinear	torch.nn.linear	linear transformation
			norm	RMSNorm	torch_npu.npu_add_rms_norm torch_npu.npu_rms_norm	RootMeanSquareLayerNormalization (RMSNorm)



Take DeepSeek as an example. What are the optimizations made by vLLM+Ascend+CANN?

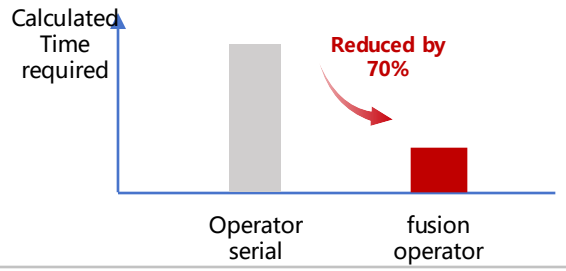
MLA pre-processing fusion operator, implementing CV parallelism
The entire network performance is improved by more than 10%.

Large operators are integrated to eliminate the problems of many MLA pre-processing operators and small shapes in the inference scenario.

AS-IS: Operators are serialized and frequently occupy memory and communication resources.

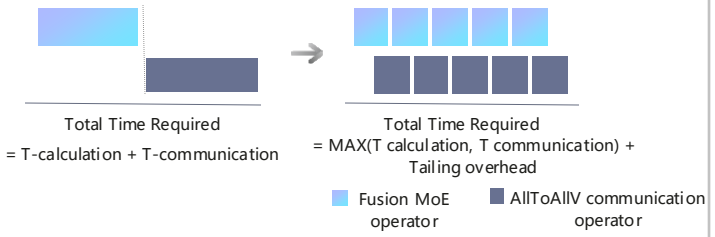
Rope	3	Full cover-up
Rmsnorm	3	Partial cover-up
Concat	2	Eliminates
Split	2	Eliminates
ReshapeandCache	1	Full cover-up
Matmul	3	Head-down overhead

TO-BE: multi-operator convergence, one-time delivery
Implement CV parallel computing.



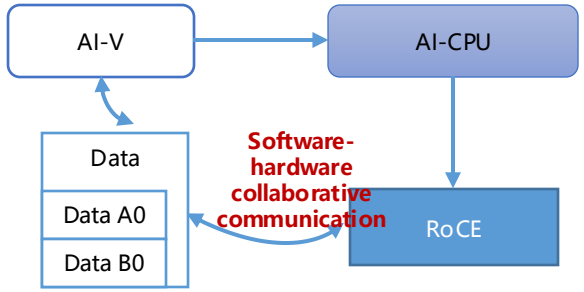
General Computing Fusion and Soft-Hardware Cooperative Communication Algorithm
The performance of the entire network is improved by 100%+.

Convergence of communication and computing to mask general computing overheads



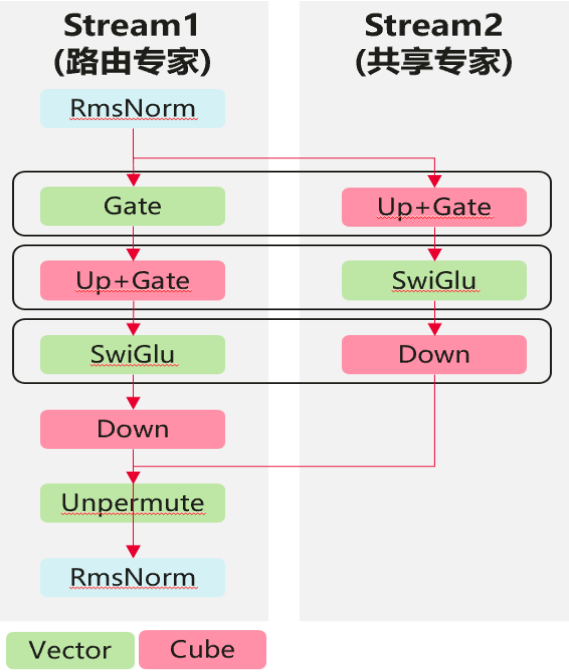
The communication efficiency is improved by 15% and the entire network performance is improved by 10%.

Unique Multi-computing Soft-Hardware Cooperative Communication Algorithm
Significantly reduces the latency of a single communication.



MoE expert dual-stream parallel technology
The performance of the entire network is improved by 5% to 10%.

Two streams can be used to perform parallel computing on the Cube and Vector of shared experts and routing experts, shortening the computing latency.

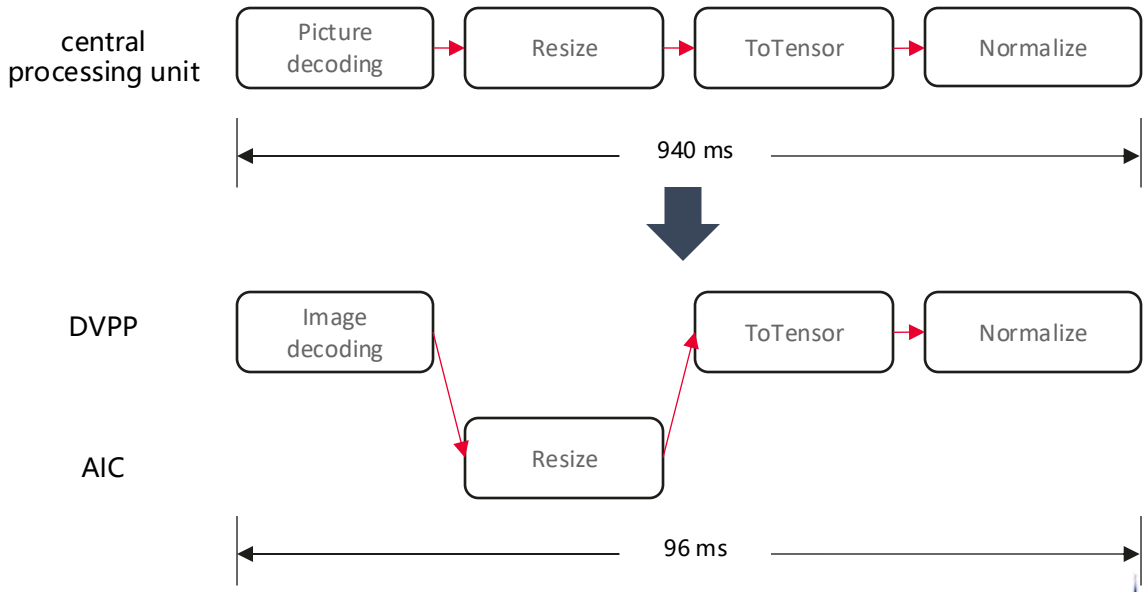


Key process nodes of the MoE model are optimized, and the performance of the fusion operator is improved by 20% to 70%.

operator	Converged solution	Benefits
MoeGatingTopkSoftmax	Softmax and topk are integrated to reduce memory access.	Increment scenario: 70% Full scenario: 100%+
MoeInitRouting	Re-arranges the tokens by experts, supports the dropless, drop, and pad capabilities, and calculates the number of tokens for each expert. All these capabilities are integrated into one operator, avoiding repeated calculation logic and inefficient histogram logic in small operator concatenation.	Increment scenario: same Full scenario: 20%
GroupedMatMul	Supports grouping MatMul. Concurrent calculation can be performed on the device side based on the number of experts, avoiding host device synchronization problems caused by grouping on the host side. Great performance improvement	Incremental memory access utilization: 90% Full long sequence cube usage: 90%

Multi-modal understanding image loading and preprocessing sinking, decoding and preprocessing speed up 88%

The Ascend device supports hardware decoding and preprocessing, greatly reducing the preprocessing time.



Models need to be split to AI clusters in parallel with finer granularity.

Save on-chip memory and put down a larger model.

The optimizer shortens the TTA for large model training.

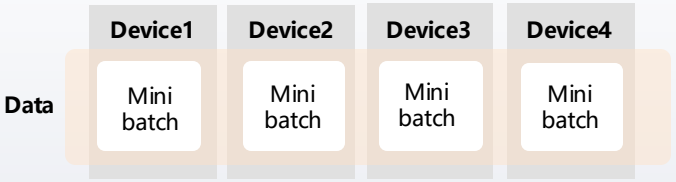
Attention module accelerated training

Cut well.

Saved

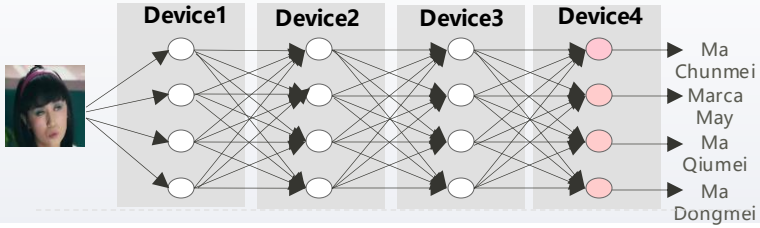
Data is split to multiple devices.

Data parallelism



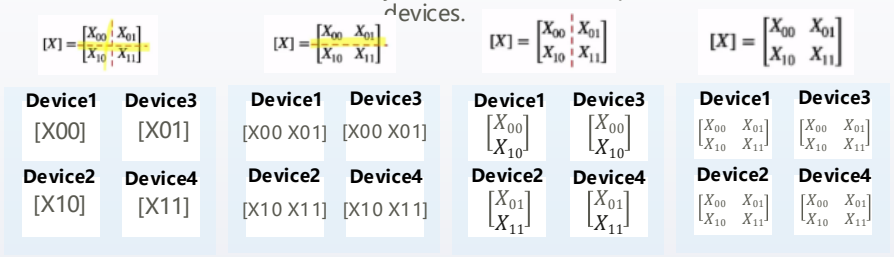
Segmenting multiple devices by model layer

Pipeline parallelism



Divide data at a layer of a model to multiple devices.

Model parallelism



Optimizer parallelism, Multi-copy parallelism, Fine-grained pipeline parallelism...

recalculate

Time-to-space optimization policy: Clear some forward calculation results. When the calculation results need to be used again, the cached CheckPoint is used for recalculation.

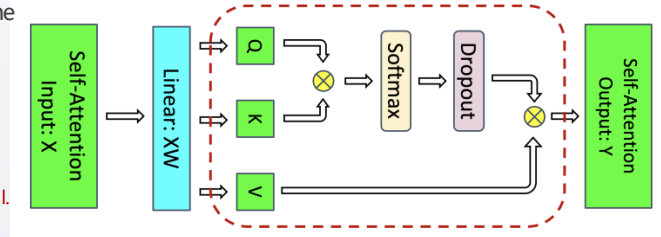
Each value in the node is saved to calculate the gradient in a backpropagation.

Only some nodes are saved in memory, while others are recalculated as needed.

Calculation results of the softmax and dropout operators in Self-Attention

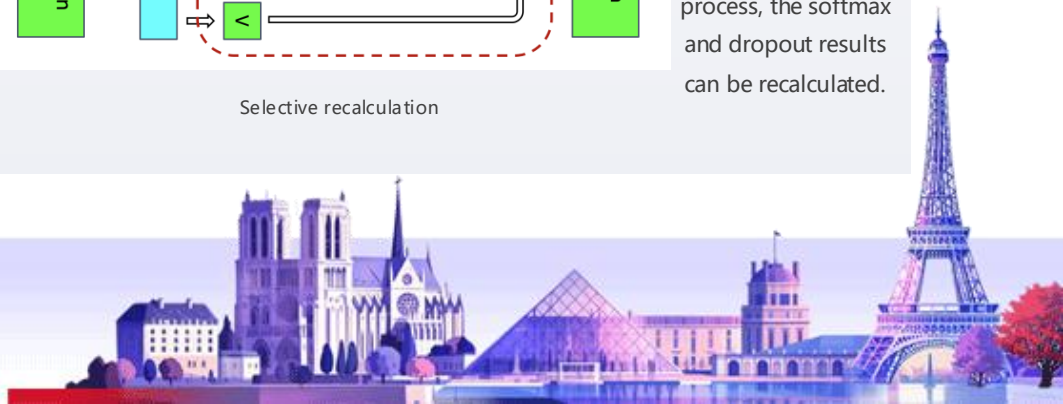
Large memory usage

But the calculation is small.



Selective recalculation

The forward calculation result is not retained. In the reverse calculation process, the softmax and dropout results can be recalculated.



Communication characteristics of large model training

periodicity

Calculation and communication are periodically iterated, and the communication mode is consistent in each iteration.

Model training traffic

Small amount of flow is large

In each round of iteration, the number of flows is small and the traffic volume is huge. The traffic synchronization between different nodes and the data volume transmitted between different nodes is greater than GB.

Intra-node

Hundreds of gigabytes of traffic, Mainly allreduce.

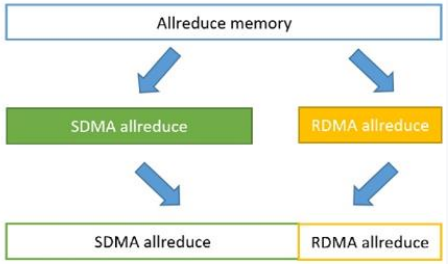
Inter-node

The traffic is in the gigabyte level, include allreduce and p2p, Most of it can be calculated to be masked.

Reduce communication overhead

Parallel mode	Features	The need for communication	
Tensor parallelism (TP)	Large traffic volume (hundred GB) and communication time cannot be obscured.	**Intra-node allreduce**	Ultra-high bandwidth
Pipeline parallel (PP)	Large traffic volume (model-related, 100 M-GB) Communication time cannot be masked/flow can be masked	Inter-node P2P	Medium bandwidth
Data parallelism (DP)	Large traffic (GB level) Communication time calculations can be mostly masked	**Inter-node allreduce**	High bandwidth

Intra-Node All-Reduce Scenario - Reuse RDMA Bandwidth for TP Communication



Intra-node communication requires ultra-high bandwidth. During intra-node communication, the RDMA communication link between nodes is usually idle,

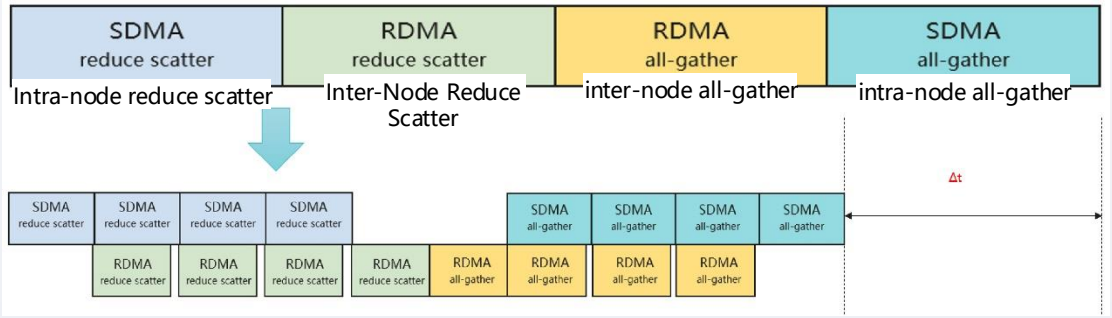
Data is split in a certain proportion, SDMA communication within a node and RDMA communication between nodes are simultaneously transmitted, thereby making full use of the concurrent communication link between nodes, Improve communication performance.

Inter-Node All-Reduce Scenario - SDMA and RDMA Communication Pipelining

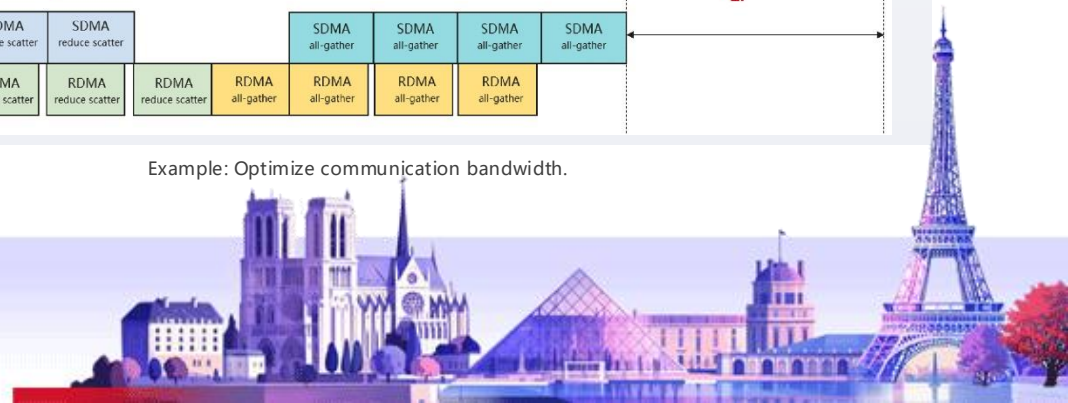
At the same time, only one inter-node link and one intra-node link are working, which wastes bandwidth.

The communication data slices are arranged into pipelines to make concurrent use of intra-node and inter-node links.

Quickly run in small steps to save communication time.

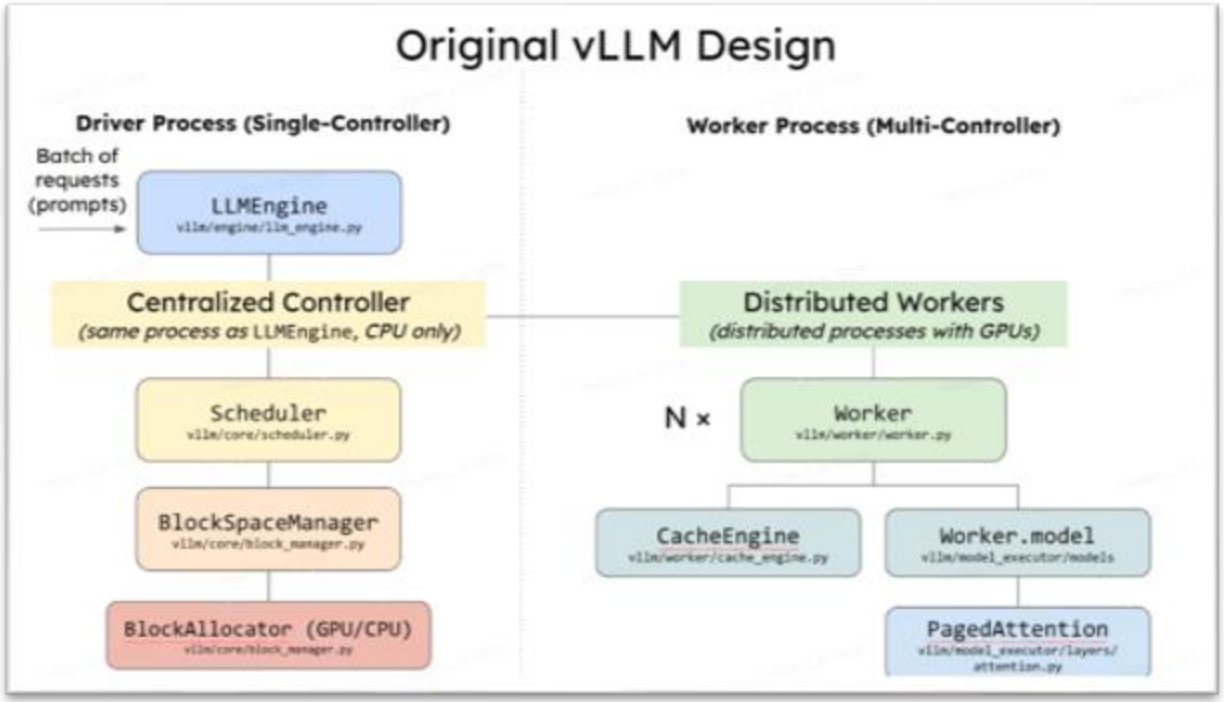


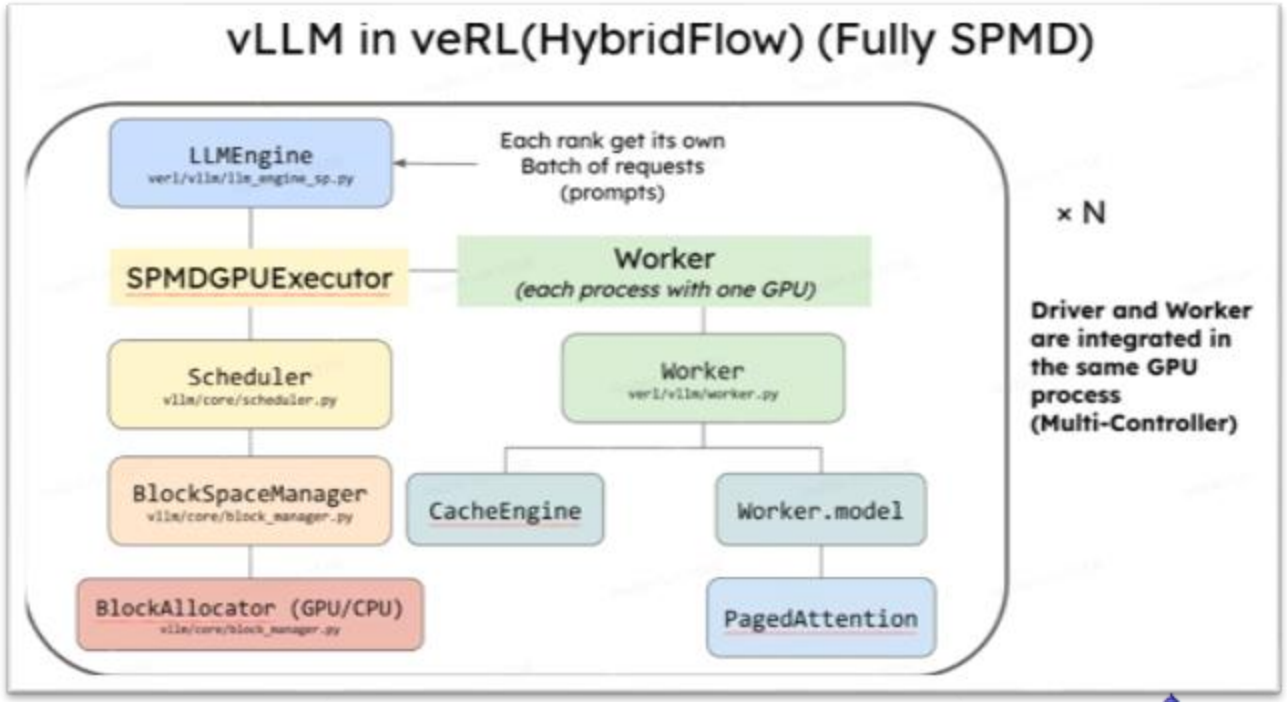
Example: Optimize communication bandwidth.



Take RLHF as an example. What optimizations are made in vLLM+verl+Ascend+CANN?

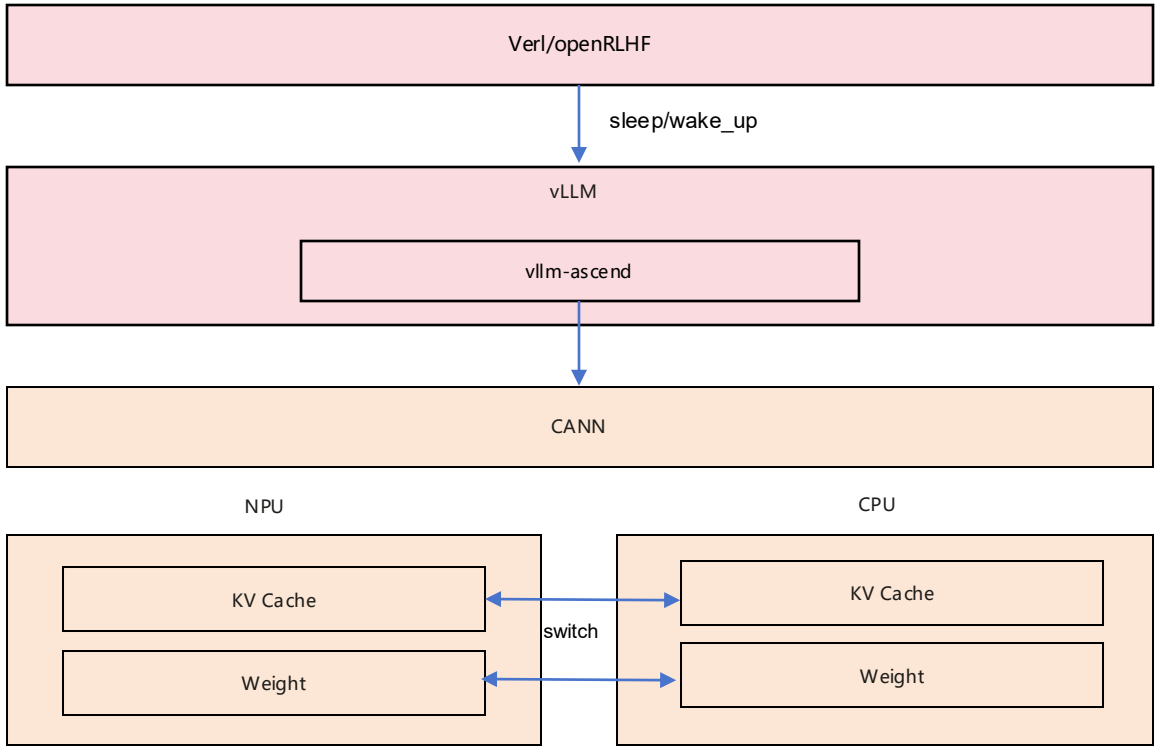
vLLM supports SPMD deployment, improving throughput in RLHF scenarios.





https://github.com/vllm-project/vllm/issues/11400

The vllm ascend supports the sleep mode feature, meeting the requirement of sharing a card in the RLHF scenario.



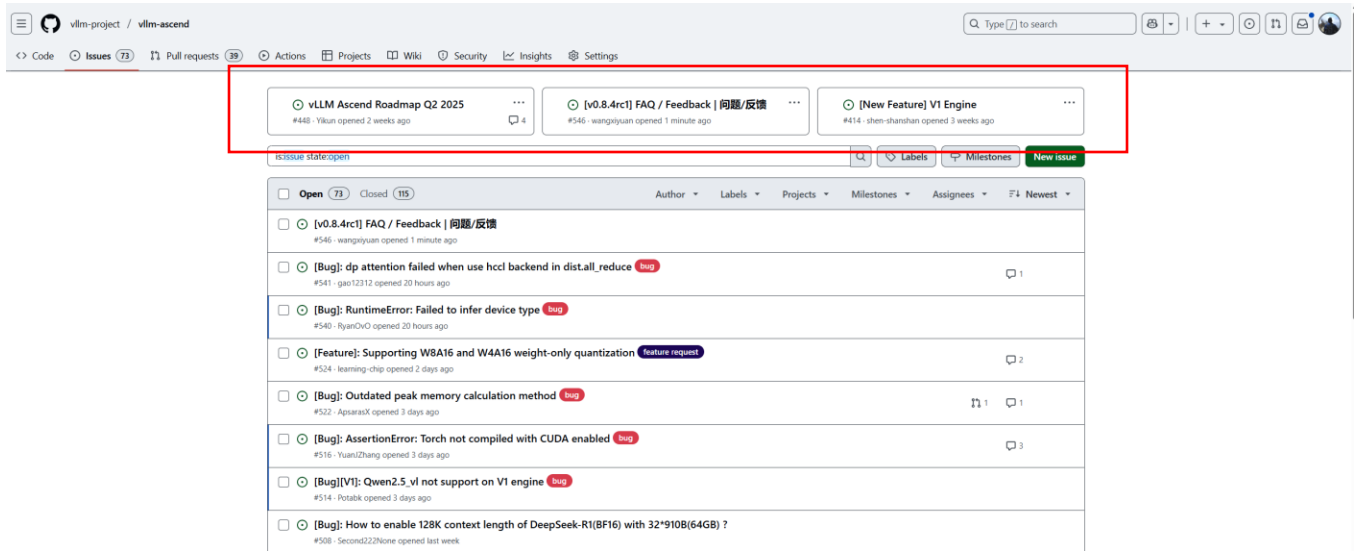
- aclrtCreateContext
- aclrtSetCurrentContext
- aclrtMallocPhysical
- aclrtFreePhysical
- aclrtMapMem
- aclrtUnmapMem
- aclrtDrvMemHandle

vLLM Ascend v0.8.4rc1 release

vLLM × Ascend

vLLM Ascend First RC Release for vLLM v0.8.4

$ docker pull quay.io/ascend/vllm-ascend:v0.8.4rc1
$ pip install vllm vllm-ascend
Doc: https://vllm-ascend.readthedocs.io
Feedback: github.com/vllm-project/vllm-ascend/issues



Feature	vLLM V0 Engine	vLLM V1 Engine	Next Step
Chunked Prefill	WIP	WIP	Functional, waiting for CANN 8.1 nnal package release
Automatic Prefix Caching	WIP	WIP	Functional, waiting for CANN 8.1 nnal package release
LoRA	Functional	WIP	vllm-ascend#396, CI needed, working on V1 support
Prompt adapter	No plan	Planned	Plan in 2025.06.30
Speculative decoding	Functional	WIP	CI needed; working on V1 support
Pooling	Functional	Functional	CI needed and adapting more models; V1 support rely on vLLM support.
Enc-dec	NO plan	Planned	Plan in 2025.06.30
Multi Modality	Functional	Functional	Tutorial, optimizing and adapting more models
LogProbs	Functional	Functional	CI needed
Prompt logProbs	Functional	Functional	CI needed
Async output	Functional	Functional	CI needed
Multi step scheduler	Functional	Deprecated	vllm#8779, replaced by vLLM V1 Scheduler)
Best of	Functional	Deprecated	vllm#13361, CI needed
Beam search	Functional	Functional	CI needed
Guided Decoding	Functional	Functional	vllm-ascend#177
Tensor Parallel	Functional	Functional	CI needed
Pipeline Parallel	Functional	Functional	CI needed
Expert Parallel	NO plan	Functional	CI needed; No plan on V0 support
Data Parallel	NO plan	Functional	CI needed; No plan on V0 support
Prefill Decode Disaggregation	Functional	Functional	1P1D available, working on xPyD and V1 support.
Quantization	Functional	Functional	W8A8 available, CI needed; working on more quantization method support
Graph Mode	NO plan	Functional	Functional, waiting for CANN 8.1 nnal package release
Sleep Mode	Functional	Functional	level=1 available, CI needed, working on V1 support

GOSIM

GOSIM AI Paris 2025

THANK YOU