

Portable PyTorch Across AI Accelerators

A Triton Operator Stack from Eager Mode to vLLM

A Day-0 engineering story: new models, many chips, ready-to-run releases

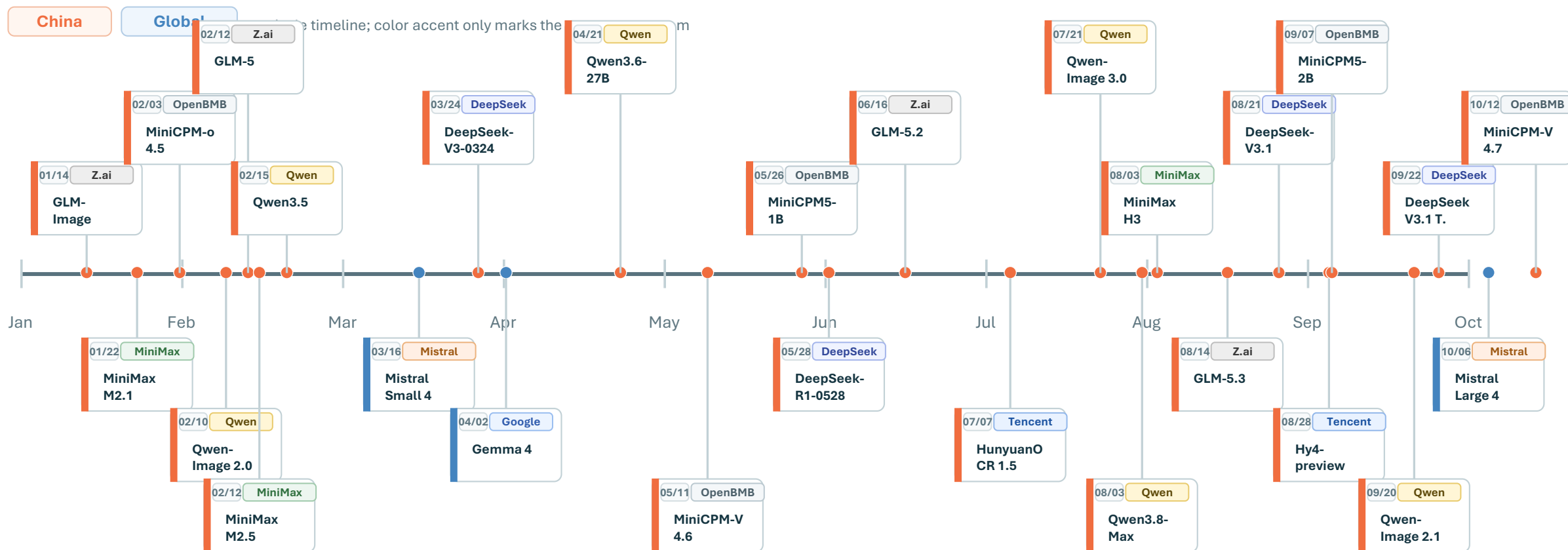
Yonghua Lin | Vice President & Chief Engineer
Beijing Academy of Artificial Intelligence

Chunlei Men | Compiler and Operator Lib Lead
Beijing Academy of Artificial Intelligence

Open models are iterating almost every week.

Selected 2026 open-model releases across China and the global ecosystem — the pace itself argues for portability.

27 selected releases in 10 months.
Aug–Sep alone included fresh
Qwen, GLM, DeepSeek, Hunyuan,
MiniMax, and MiniCPM updates.



Why this matters

When models change at this cadence, portability cannot be an afterthought — it must be built into the operator and runtime stack.

Hardware architecture are expanding at the same time.

A new model must fit the installed accelerator—not just a single reference GPU.

30+

chip vendors

50+

AI chips

GPU

NVIDIA / AMD

Other GPUs

Moore Threads / MetaX

NPU / DSA

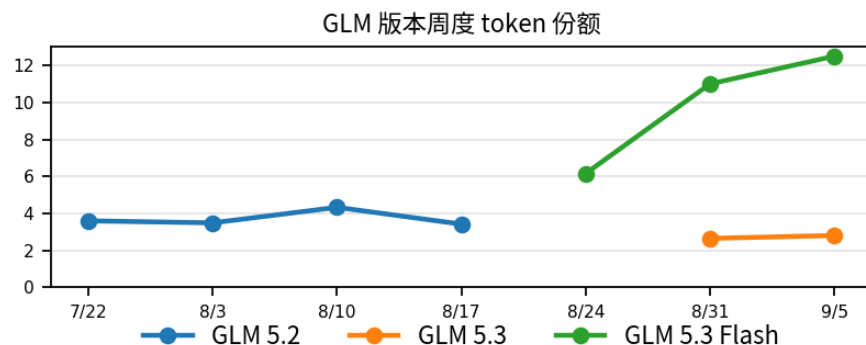
Ascend / T-Head

Edge CPUs

ARM / Apple M-series

Day-0 support decides who captures new model demand.

Model traffic shifts fast after release. Late support misses the window.



01

Demand shifts fast

New releases quickly reset usage.

02

Traffic follows the latest model

Users and developers move first.

Day-0 turns release momentum into usable demand.

It protects relevance for chips, frameworks, and deployment stacks.

Why it matters

- ✓ Capture demand
- ✓ Keep developers
- ✓ Stay relevant

One Day-0 case: Qwen3.8-2.4T-A95B.

Ultra-large MoE. Diverse chip capabilities. Several precision formats.

2.4T

total parameters

95B

active parameters

9

chip platforms on Aug 13 (release timeline)

10

platforms listed in current model card

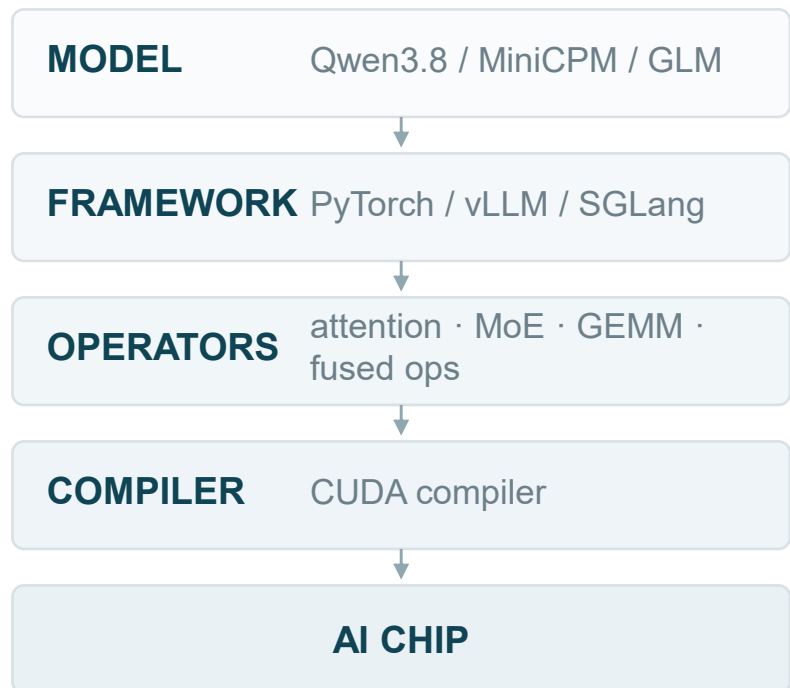
BF16 / FP8 / INT8

different deployment precisions

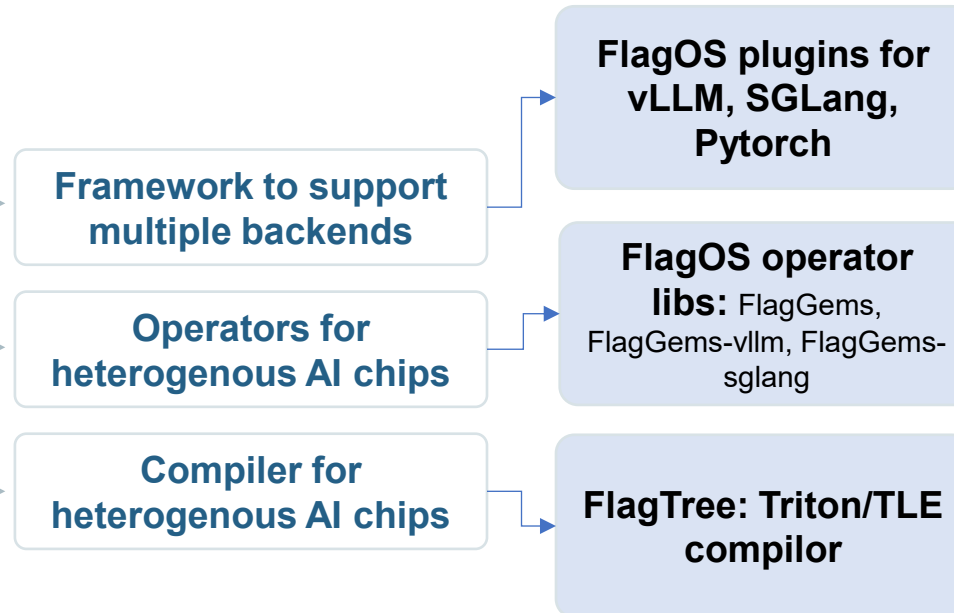
What must change for one new model?

Day-0 adaptation spans the full stack — from hardware enablement to model runtime.

STACK FOR ONE MODEL



DAY-0 ADAPTATION TO MULTIPLE BACKENDS



FOR MISSING/NEW OPERATORS

KernelGen operator generation tool

FOR PRECISION ADAPTATION

FlagOS-Compressor

Day-0 is not one fix. It is coordinated adaptation across the full stack.

The FlagOS path: reuse what exists; specialize only the gaps.

A repeatable workflow instead of one private software fork per chip.



EAGER MODE

Keep torch calls; replace supported ATen ops

SERVING

Keep vLLM API; use backend plugins

CHIP BACKENDS

Shared operator and compiler ecosystem

Start from Ops Libraries: 1400+ Triton Operators, 20+ AI chips

Supporting 20+ AI chip platforms, with 8 operator libraries spanning LLMs, multiple domains, and high-performance computing — over 800 operators in total

Weekly downloads: >50k

Operator Coverage for ATen, vLLM, and SGLang

FlagGems → **PyTorch ATen** | **1200** operators

Pytorch Ecosystem project



FlagGems-vLLM → **vLLM** | **70** key operators

Multi-chip inference · Fast adaptation to new models

FlagGems-sglang → **SGLang** | **40** key operators

Common / FlashInfer operators · Flexible multi-backend support · Mainstream model inference

12 **FlagDNN**
↔ cuDNN

20 **FlagBLAS**
↔ cuBLAS

10 **FlagFFT**
↔ cuFFT

12 **FlagSparse**
↔ cuSPARSE

8 **FlagTensor**
↔ torch.Tensor

8 **FlagAudio**
↔ torchaudio

6 Domain Libraries • 70 Operators

General AI Operator Library

<https://github.com/flagos-ai/FlagGems>

High-Performance Inference Operator Library

<https://github.com/flagos-ai/FlagGems-vllm>

Deep Learning Operator Library

<https://github.com/flagos-ai/FlagDNN>

Linear Algebra Operator Library

<https://github.com/flagos-ai/FlagBLAS>

Signal Processing Operator Library

<https://github.com/flagos-ai/FlagFFT>

Sparse Computing Operator Library

<https://github.com/flagos-ai/FlagSparse>

Tensor Operator Library

<https://github.com/flagos-ai/FlagTensor>

Audio Processing Operator Library

<https://github.com/flagos-ai/FlagAudio>

FlagOS Triton Ops Libraries have **6%** speedup vs. NV CUDA libraries, and **27%** speedup vs. AMD libraries.

01 Easy Integration

Keep familiar APIs.
Choose which operators to replace.

➤ PyTorch • Enable in one line

```
gems.enable(unused=["sum"])
```

Replace supported ATen operators.
Exclude specific operators as needed.

➤ vLLM • Native plugin integration

```
VLLM_PLUGINS=fl \
vllm serve <model>
```

Use inference operators via vllm-plugin-FL while keeping existing serving APIs.

Native integration • Configurable scope

02 Flexible Use

Enable in existing code,
or call operators directly.

➤ Local scope • Keep torch calls

```
with gems.use_gems():
    c = torch.mm(a, b)
```

Use FlagGems for matrix multiplication within the selected code block.

➤ Direct calls • Explicit operator selection

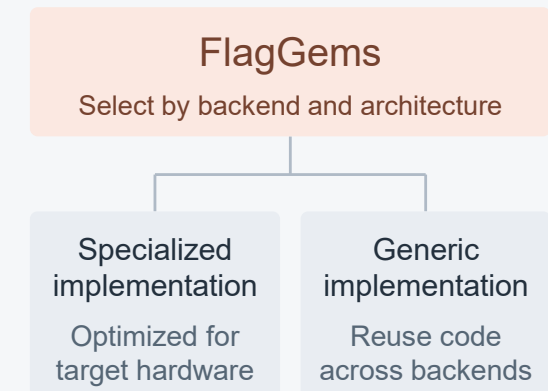
```
c = gems.ops.mm(a, b)
```

Call matrix multiplication directly for custom code and benchmarking.

Local scope • Direct calls

03 Hardware Portability

Unified operator interfaces.
Automatic dispatch to target hardware.



Use the generic implementation when no matching specialized implementation exists.

Unified APIs • Automatic dispatch

Example: `import torch; import flag_gems as gems`. Run snippets independently; a and b are compatible device matrices.

FlagOS Ops Libraries: High Performance



Operator libraries: FlagGems / FlagGems-vLLM / FlagGems-SGLang

Compiler: FlagTree + Triton-TLE

Operator	Models	Speedup by Hardware Platform	Language
FlashMLA Sparse	DeepSeek V3/V4, GLM-5.2/GLM-5.3-flash	NVIDIA: 2.3× vs Triton; Ascend: 83× vs Triton;	Triton-TLE
TopK Selector	DeepSeek V4 (DSA sparse attention)	NVIDIA: 1.5× vs FlashInfer; 2.5× vs TRT-LLM; Hygon: 2.95× vs native	Triton-TLE
CausalConv1D Decode	DeepSeek V4 (GDN hybrid attention)	Ascend: up to 64×, average 11.87× vs Triton; NVIDIA: 1.3× vs Triton	Triton-TLE
MHC Pre (preprocessing)	DeepSeek V4 (MLA)	NVIDIA: 2.73× vs TileLang; 8.25× vs Torch	Triton
MHC Post (postprocessing)	DeepSeek V4 (MLA)	NVIDIA: 25.88× vs Torch	Triton
Grouped TopK	DeepSeek V4 / MoE routing	Hygon: 27.48× vs Torch; Ascend: 4.65× vs Torch; Moore Threads: 7.25× vs Torch	Triton
Fused MoE INT4 (Marlin)	DeepSeek V4 / INT4-quantized MoE	NVIDIA: up to 2.92× vs Triton; 1.27× vs native; MetaX: 1.29× vs native	Triton
FP8 MatMul (w8a8_block_fp8)	DeepSeek V4 / FP8 inference	Moore Threads: 4.48~13.9× vs Triton; NVIDIA: 2.3× vs Triton	Triton
Sparse Attention	DeepSeek V4 (sparse attention)	Moore Threads: average 6× vs Triton; NVIDIA: 1.89× vs TileLang	Triton
GDN2 (Chunked Gated Delta Rule)	Qwen3.8-Flash-Next (GDN compressed attention)	NVIDIA: 1.5~2× vs Flash Triton; up to 3.37×; Hygon: 2.4~6.3× vs native	Triton-TLE
GLA (Gated Linear Attention)	Qwen / MiniCPM (GLA models)	NVIDIA: 1.18~1.27× vs FLA Triton; MetaX: 15× vs native; Moore Threads: 10× vs native;	Triton
KDA (Kimi Delta Attention)	Kimi series	NVIDIA: ~1.15× vs current SOTA	Triton-TLE
FlashQLA	Qwen series (QLA quantized attention)	Hygon: 4.8~6.5× vs native; MetaX: 3~4× vs native; Ascend: 1.4× vs native; NVIDIA: 1.15× vs TileLang	Triton-TLE
Act Quant (Activation Quantization)	DeepSeek V4 (FP8 quantization)	NVIDIA: 1.42× vs TileLang	Triton
Hadamard Transform	DeepSeek V4 (Hadamard Transform)	NVIDIA: 1.13× vs TileLang (automatically generated by KernelGen)	Triton
MoeAlignBlockSize	MoE models (expert scheduling)	NVIDIA: 3× vs Triton; up to 4× vs SGLang; 1.27× vs CUDA	Triton-TLE
per_token_group_fp8_quant	FP8-quantized LLM inference	NVIDIA: 2.38× vs Triton; 1.29× vs CUDA; Ascend: 2.58× vs Triton; Moore Threads: 2.27× vs Triton	Triton

FlagTree: one Triton ecosystem, many backends, Triton-Language-Extension (TLE) for high performance

20

vendor backends in the FlagOS 2.2 compiler repository

TLE-Lite

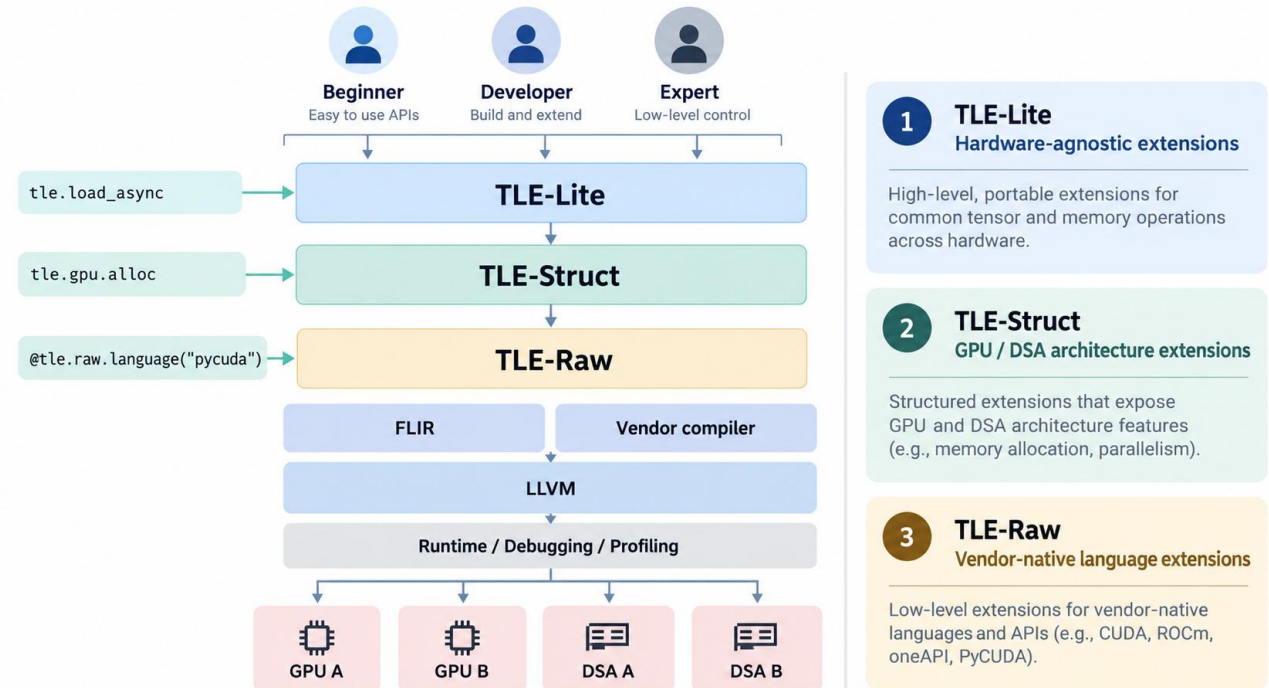
Portable extensions

TLE-Struct

GPU / DSA architecture control

TLE-Raw

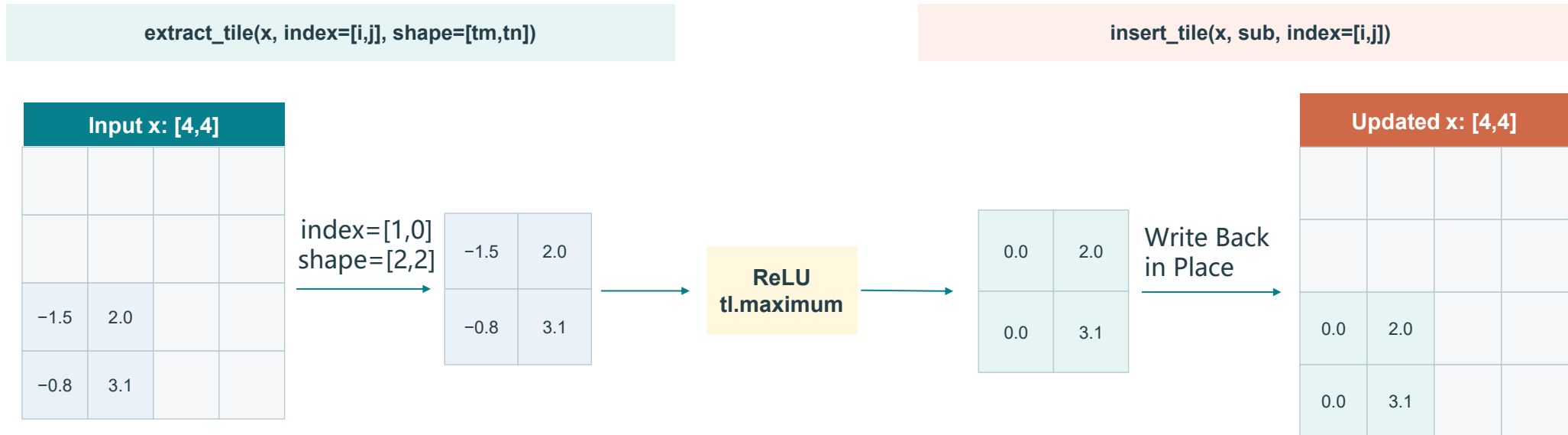
Vendor-native primitives



Portable interface.
Architecture-aware code generation.

TLE-Lite example : Tile Slicing for Local Computation

extract_tile / insert_tile: extract by sub-tile grid coordinates, compute, then write back.
Save I/O time.



example:

```
sub = tl.extract_tile(x, index=[1, 0], shape=[2, 2])
sub = tl.maximum(sub, 0.0)
x = tl.insert_tile(x, sub, index=[1, 0])
```

benchmark:

Operator	Speedup(TLEvs CUDA)
GLU	1.25x
rotary_pos_emb	5.87x

TLE-Lite example : Pipelining and Synchronization with Pipe

FlagOS

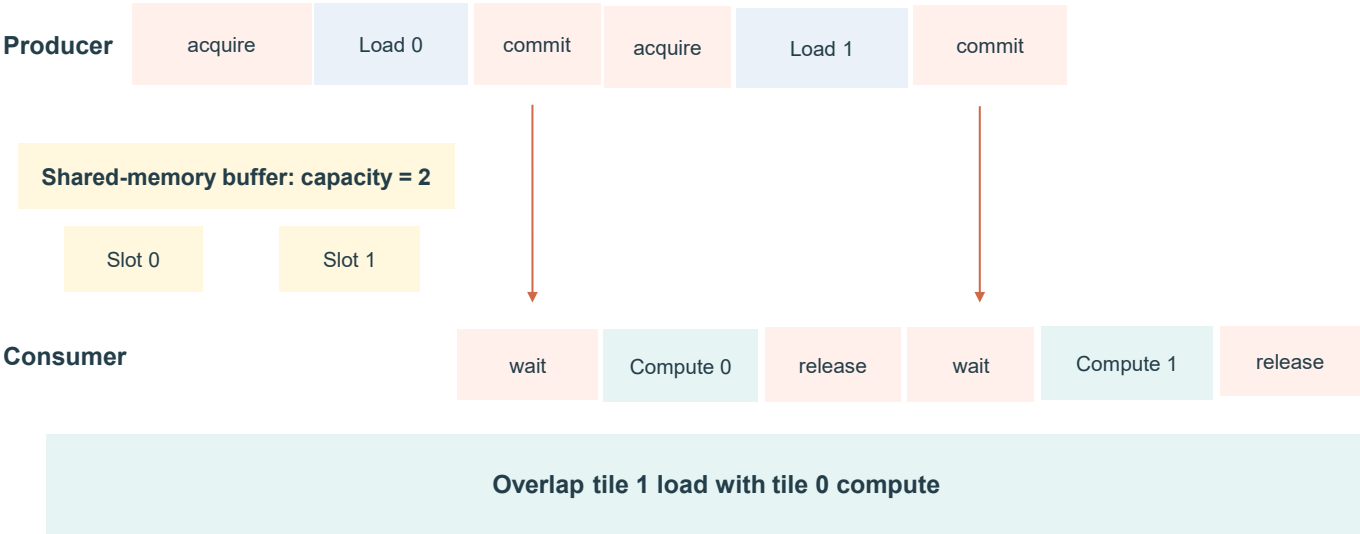
Producers write and commit; consumers wait, read and release. Slots are reused by lifecycle.

Synchronization: acquire → write → commit → wait → read → release

Without Pipe: Serial Load and Compute



With Pipe: Overlap Production and Consumption



Primitives in Code

```
# producer
slot = writer.acquire(i)
# write payload into slot
writer.commit(i)

# consumer
ready = reader.wait(i)
# read ready and compute
reader.release(i)
```

benchmark:

Operator	Speedup(TLEvs CUDA)
flash_mla	1.0x
chunk_kda	3.98x

TLE-RAW example : Native Capabilities inside Triton

All-Gather GEMM: native communication and synchronization with Triton matrix computation.

① CUDA Device Function

```
extern "C" __device__ void
ag_wait_ready(uint64_t *ready, int idx) {
    nvshmem_signal_wait_until(...);
}
```

② Bind Native Functions in Python

Bind vendor language and source
Declare external device functions
Expose callable entry points to Triton JIT

③ Triton Kernel: Wait for the Current Tile, Then Compute

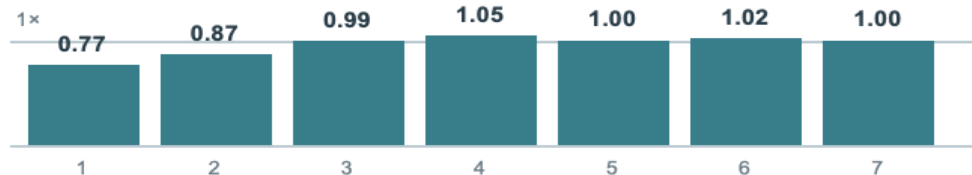
```
tle.raw.call(wait_ready, [ready, src_rank])
acc = tl.dot(a, b.T, acc)
```

Communication: P2P copy → signal

Compute: wait until ready → GEMM

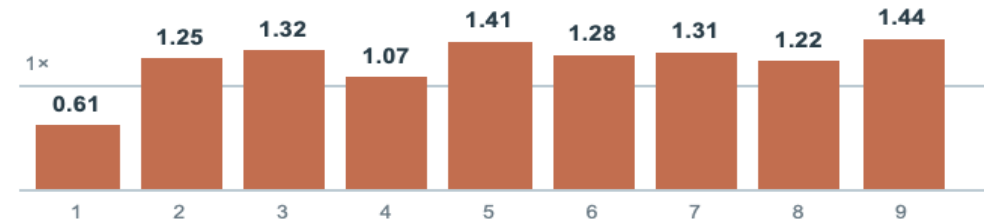
Keep compute in Triton; use native paths for specific communication and synchronization. Dependencies vary by hardware.

All-Gather: Single Node,



Bandwidth ratio, TLE / Torch: 0.77–1.05×

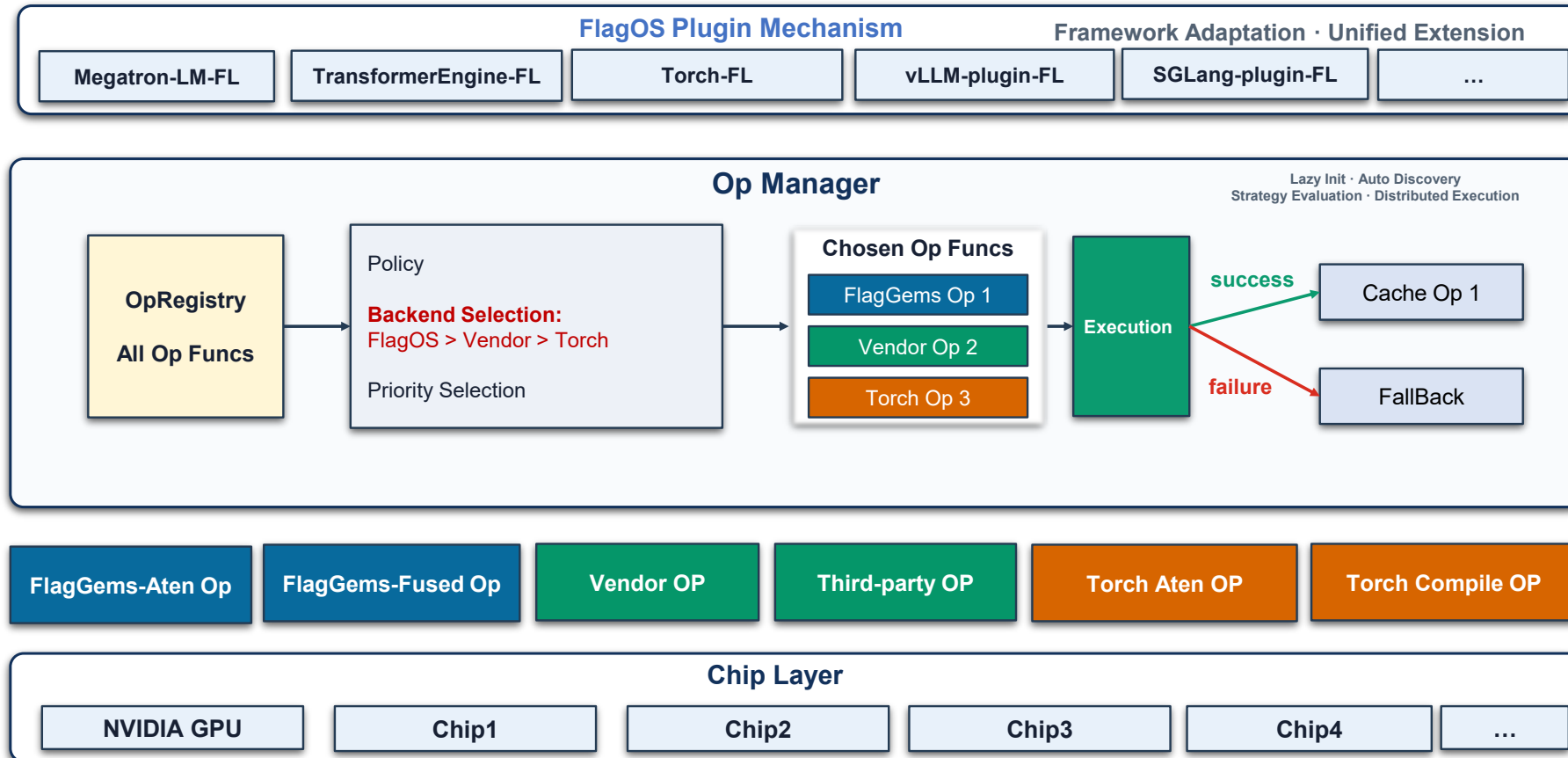
All-Gather GEMM



Speedup normalized to Torch: 0.61–1.44×

Framework plugins: PyTorch, vLLM, SGLang unchanged

The new model reaches runtime-specific paths **without a long-lived private fork**.



```
VLLM_PLUGINS=fl vllm serve "$MODEL"
```

Seamless Framework Integration Through FlagOS Plugins

vLLM-Plugin-FL

<https://github.com/flagos-ai/vllm-plugin-FL>

For high-efficient inference and service deployment

Megatron-LM-FL

<https://github.com/flagos-ai/Megatron-LM-FL>

For large-scale model training

TransformerEngine-FL

<https://github.com/flagos-ai/TransformerEngine-FL>

For large-scale and high-efficient model training

SGLang-Plugin-FL

<https://github.com/flagos-ai/sglang-plugin-FL>

For high-efficient inference and service deployment

PyTorch-Plugin-FL

<https://github.com/flagos-ai/PyTorch-Plugin-FL>

For general deep learning (training and inference)

VeRL-FL → VeRL Plugin

<https://github.com/verl-project/verl-hardware-plugin>

For reinforcement learning

LightX2V-Plugin-FL

<https://light-ai.top/LightX2V-BLOG/posts/FlagOS-Plugin/>

A Plugin Design for Cross-Chip Video Generation Inference

	Ascend	Hygon	Mthreads	Metax	PPU	Kunlun	Iluvatar	Cambricon	TsingMicro	Enflame	Nvidia
vLLM-Plugin-FL	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
Megatron-LM-FL	✓	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓
TransformerEngine-FL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SGLang-Plugin-FL	✓	✓	✓		✓	✓	✓	✗	✓	✓	✓
PyTorch-Plugin-FL	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
VeRL-FL	✓	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓

FlagOSCompressor: make precision a deployment choice.

The same model family may need different checkpoint precision and runtime kernels.

REAL CLI · ONE QUANTIZATION WORKFLOW

```
flagos-compressor inspect --input /path/to/model
```

```
flagos-compressor quantize \  
--input /path/to/model \  
--output /path/to/output \  
--select linear --bits 8 --strategy group \  
--group-size 128 --backend cuda
```

```
flagos-compressor validate --input /path/to/output
```

<https://github.com/flagos-ai/FlagOS-Compressor>

QWEN3.8-2.4T RELEASE

BF16

FP8

INT8

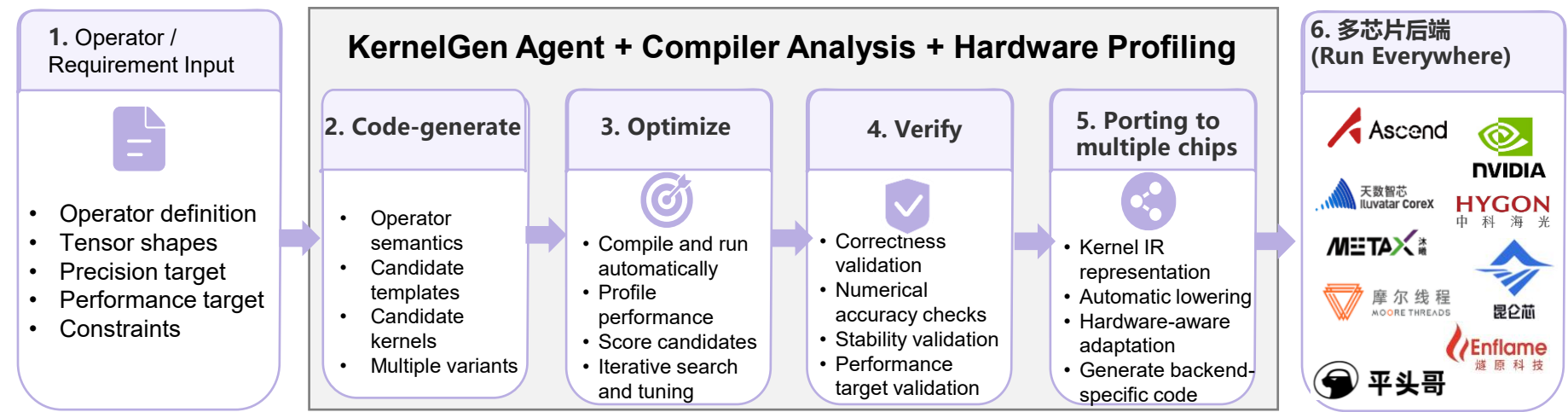
Each variant needs both weight-format checks and numerical evaluation.

KernelGen: shorten the time to a missing operator for multiple chips

An AI-native kernel engineering system for Triton / Triton-TLE across 9 AI chip platforms

More than half of FlagOS operator libraries are now generated with KernelGen, significantly improving development efficiency.

URL:
<https://kernelgen.flagos.io>



Rapid Support for New Model Kernels

DeepSeek V4: 3 new kernels across 10 chip platforms

Kernel	Speedup vs. DS-V4 Original
FP8 MatMul	2.47×
Sparse Attention	1.89×
Hadamard Transform	1.13×

Automatically Generated High-Performance TLE Kernels

Kernel	Gain
hc_head_fused_kernel	TLE vs Triton 1.30×
Top-K Radix Select	1.16–2.03× TLE vs. Triton at large batch sizes

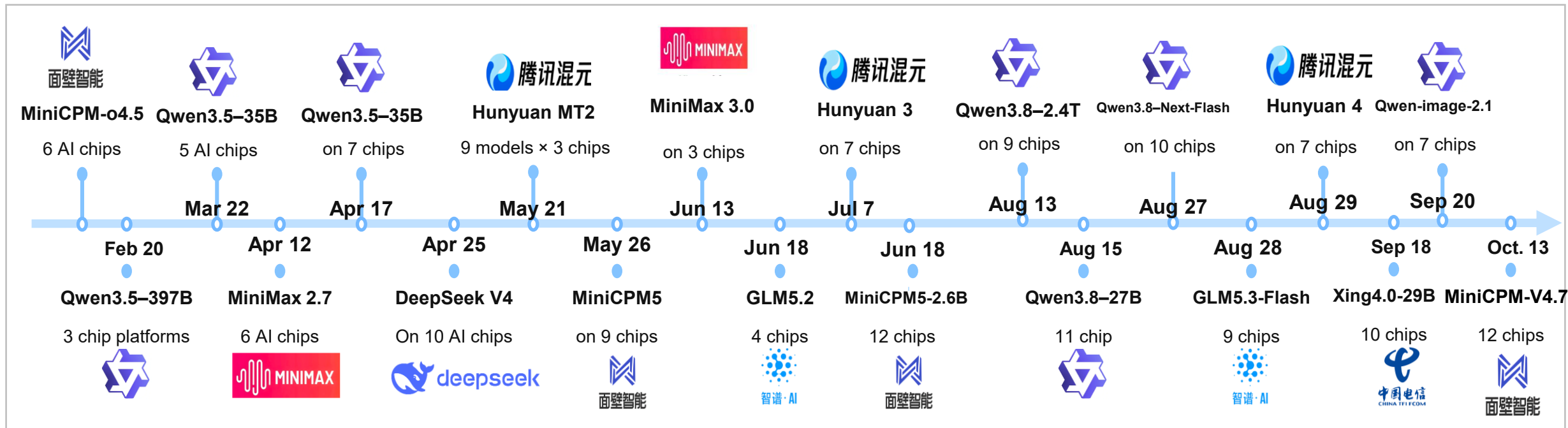
Competitive Performance on KernelBench

Compared with other automated kernel-generation approaches on KernelBench

		Easy	Medium	Hard
Triton	DR.Kernel	39.3%	80.9%	29.8%
	DeepSeek R1 (G+E+P)	43%	72%	18%
	Kernel smith	70%	93%	46%
	KernelGen (ours)	74.1%	92.9%	87.8%

One workflow, repeated on new model releases — FlagRelease

Selected Day-0 milestones from the FlagOS 2.2 timeline.



URL for multi-chip open-model version: <https://huggingface.co/FlagRelease>

AUG 13

Qwen3.8-2.4T

9 chips

AUG 27

Next-Flash

10 chips

SEP 20

Qwen-Image-2.1

7 chips

OCT 13

MiniCPM-V4.7

12 chips

For developer: from a published model card to a running vLLM API

Example: the published MiniCPM-V4.7 35B-A3B FlagOS release for MetaX.

PUBLISHED MODEL

MiniCPM-V4.7-35B-A3B

FlagRelease / BF16 / MetaX

MODEL WEIGHTS

Chip-specific checkpoint on huggingface

RUNTIME IMAGE

Versioned FlagOS Docker image

VALIDATION

Reference accuracy results on model card

USER GUIDE

Download → container → vLLM → API

ONLY 3 STEPS:

1 · DOWNLOAD THE VERIFIED MODEL

```
huggingface-cli download \  
--model FlagRelease/MiniCPM-V-4.7-35B-A3B-BF16-metax-FlagOS \  
--local_dir /data/MiniCPM-V-4.7-35B-A3B
```

2 · USE THE PUBLISHED IMAGE + SERVE

```
# After starting the chip-specific Docker image with the card's device flags:  
device flags:  
export VLLM_PLUGINS=fl  
vllm serve /data/MiniCPM-V-4.7-35B-A3B \  
--served-model-name minicpm-v-4.7-35b-a3b --port 8032
```

3 · CALL THE OPENAI-COMPATIBLE ENDPOINT

```
curl http://127.0.0.1:8032/v1/chat/completions \  
-H "Content-Type: application/json" \  
-d '{"messages":[{"role":"user","content":"Explain GPQA"}],"max_tokens":128}'
```

FlagOS: Open-sourced Stack for Open Computing — Run AI Everywhere

The AI System Software Stack Supporting the Largest Number of Chip Types Worldwide

FlagOS is a unified, open-source AI system software stack that enables AI models and applications to run seamlessly across diverse hardware platforms.

20+
Chip Vendors

30+
AI Chips

There are >100 organizations contributing to this open-sourced project, including BAAI, Peking Univ., Tsinghua Univ., CSA, Alibaba, Huawei, AMD, Nvidia, etc. And FlagOS also created global collaboration with

Open-weight AI Models

Language models

Multimodal models

World models

Embodied AI

Unified Device Plugin for Training/Inference framework

- Megatron-FL
- Transforme-rEngine-FL
- vLLM-plugin-FL
- VeRL-plugin-FL
- SGLang-plugin-FL
- Pytorch-plugin-FL

Domain solution

- FlagOS-Robo
- FlagQuantu-ms

FlagOS Skills

- Deploy, code-gen, migration, benchmark

FlagOS core libraries

- FlagGEMs operator library
- FlagGEMs - vLLM fused-kernels
- FlagGEMs, FlagBLAS, FlagSparse, FlagFFT, FlagTensor, FlagAudio

- FlagTree : Unified AI compiler
- FlagTree: AI chips
- FlagTree-CPU: ARM

- FlagScale parallel framework

- FlagCX Unified comm. lib

FlagOS tools

- KernelGen: operator auto-gen tool
- FlagRelease: Model auto-migration tool
- FlagPerf: AI chip evaluation

FlagOS

Diverse Compute

AI Compute Hardware (Nvidia, AMD, Huawei Ascend, ARM, etc.)

Servers of different AI chipsets



Edge devices



AI book



ARM devices



RISC-V AI chips



Quantum Compute

Quantum



Open Computing Is a Team Sport — FlagOS broadly collaborating with open communities

From v1.6 to v2.2: release evolution, upstream framework integrations & ready-to-run images — enabling every community with zero user code changes

FlagOS	PyTorch	vLLM	SGLang	Megatron-LM	TransformerEngine	VeRL	Triton · via FlagTree
V1.6 Jan 5, 2026	FlagGems Aten	V0.13.0 vLLM-plugin-fl v0.1.0		v0.15.0rc7 Megatron-LM-FL v0.1.0	v2.9.0 TE-FL v0.1.0		v3.2 – 3.5 FlagTree v0.4 · TLE + KernelGen v1.0
V2.0 Mar 27, 2026	FlagGems Aten	V0.13.0 vLLM-plugin-fl v0.1.0		v0.15.0rc7 Megatron-LM-FL v0.1.0	v2.9.0 TE-FL v0.1.0 · 6 vendors	v0.7.0 verl-FL v0.1.0	v3.3 / 3.5 FlagTree v0.5
V2.1 LATEST STABLE Jun 24, 2026	V2.11.0	V0.20.2 vLLM-plugin-fl v0.2.0 · 10 platforms	v0.5.11 sglang-plugin-FL v0.1.0	v0.17.0 Megatron-LM-FL v0.2.0	v2.14.0 TE-FL v0.2.0	v0.7.0 verl-FL v0.2.0	v3.2 / 3.3 / 3.5 / 3.6 FlagTree v0.6
V2.2 RC · COMING Late Sep. 2026	V2.8 – 2.10 Torch-FL (planned)	V0.24.0/0.28.0 vllm-plugin-FL v0.3.0-rc0 · Empty-mode		v0.18.2 target	v2.17 target	HW plugins +4 chips	v3.6 FlagTree v0.7-rc0

COMMUNITY SCALE

18 vendors

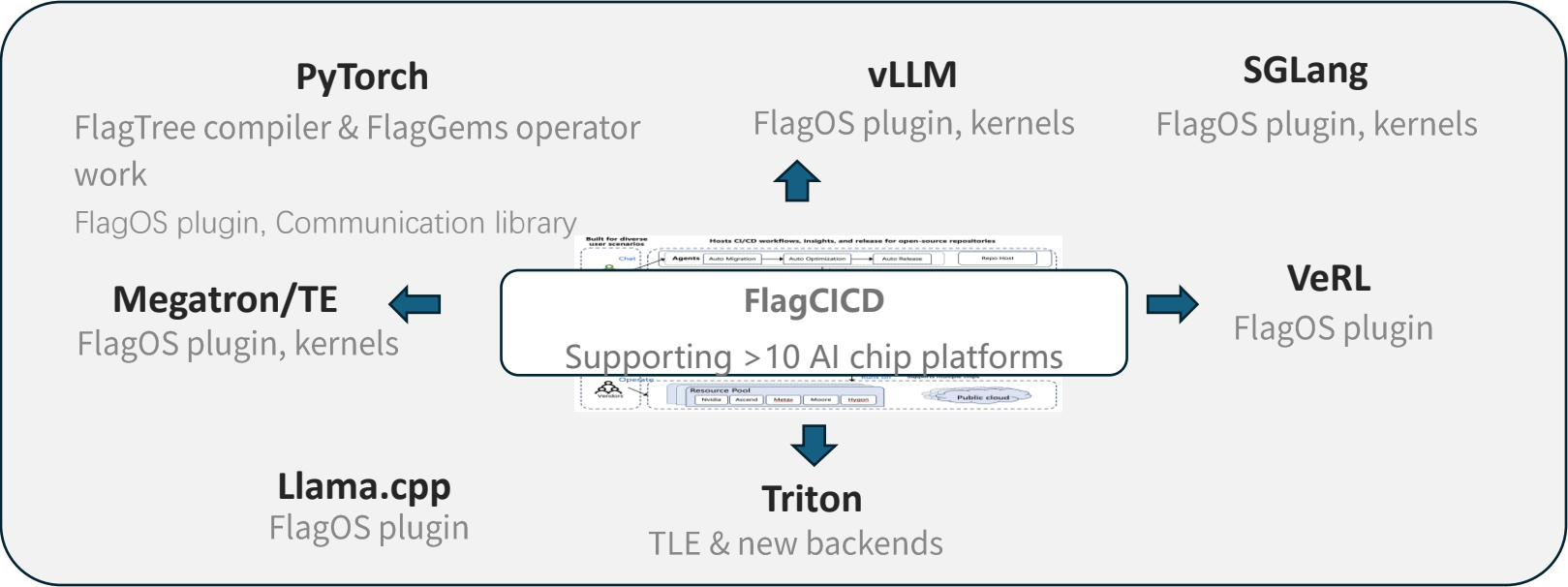
32 AI chips supported across NVIDIA, Ascend, Hygon, Metax, Moore Threads, Cambricon, T-Head, Kunlunxin, Iluvatar, TsingMicro & more

730+

active contributors · 80+ member organizations

55,800+

developers reached across 30+ countries & regions



One Open Ecosystem. Many Accelerators.

Portable performance is built together — from PyTorch operators and Triton backends to serving plugins and verified Day-0 releases.

01 / CONTRIBUTE

Reusable operators
and correctness tests

02 / EXTEND

Triton backends and
framework plugins

03 / SHIP

Reproducible
Day-0 model releases

Let's make portability a shared upstream capability.