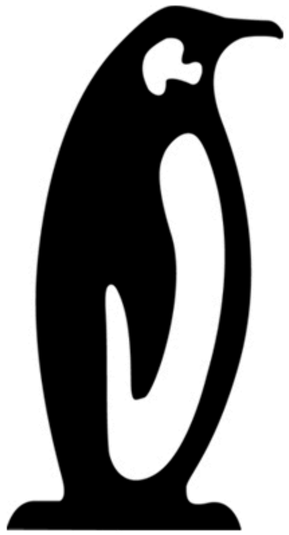


東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Demystifying Linux NPU Subsystem: From Vision to LLM at Edge

Jagan Teki <jagan@edgeble.ai>

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

NPUs are everywhere — but Linux still treats them like exceptions

Context & Scope of This Talk

→ Observations are based on real-world edge deployments using SoC and PCIe NPUs

→ Focus is on inference-time behavior, not training frameworks

→ Comparisons include upstream Linux and vendor kernels/drivers

→ Goal is not to propose a solution, but to highlight emerging gaps

...Examples shown are illustrative, not exhaustive.

Training

Dataset

Data loader & Augmentation

Training Engine

Forward Pass

Loss

Backward Pass

Optimizer Update

Model Export

Model

Inference

Input

Pre Process

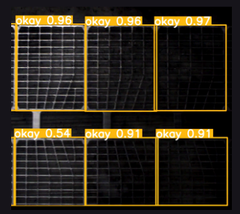
Model → Inference Engine

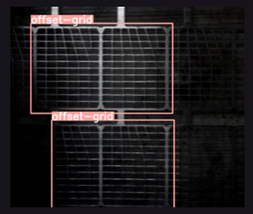
DMA Buffer

Post Process

No Detects

Detect





Training

Dataset

Data loader & Augmentation

Training Engine

Forward Pass

Loss

Backward Pass

Optimizer Update

Model Export

Model

Inference

Input

Pre Process

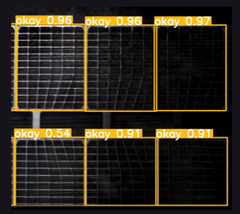
Model → Inference Engine

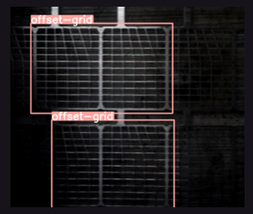
DMA Buffer

Post Process

No Detects

Detect





Self-learning

Input

Pre Process

Model → Inference Engine

DMA Buffer

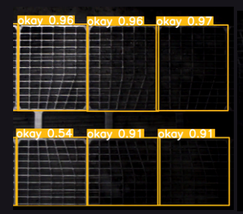
Event → Update Loop

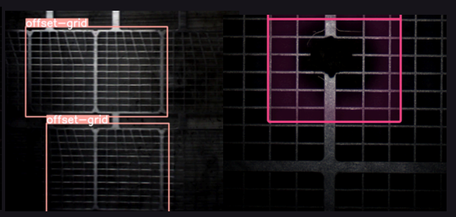
Post Process

No Detects

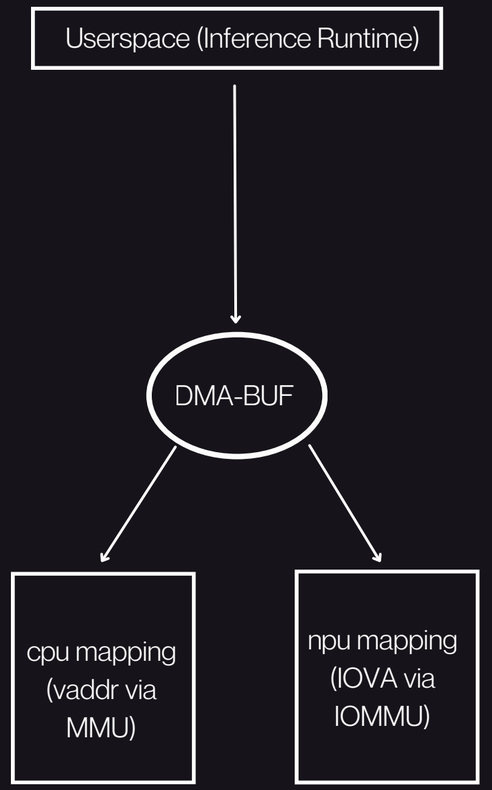
Detect

Unhandled pattern



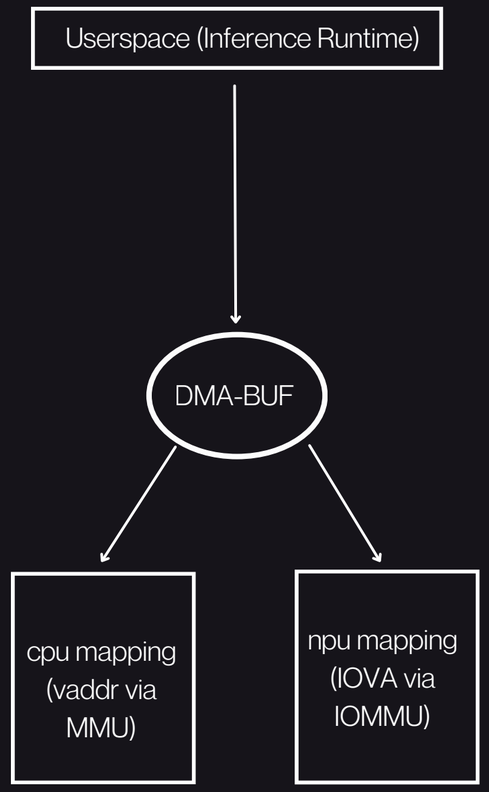


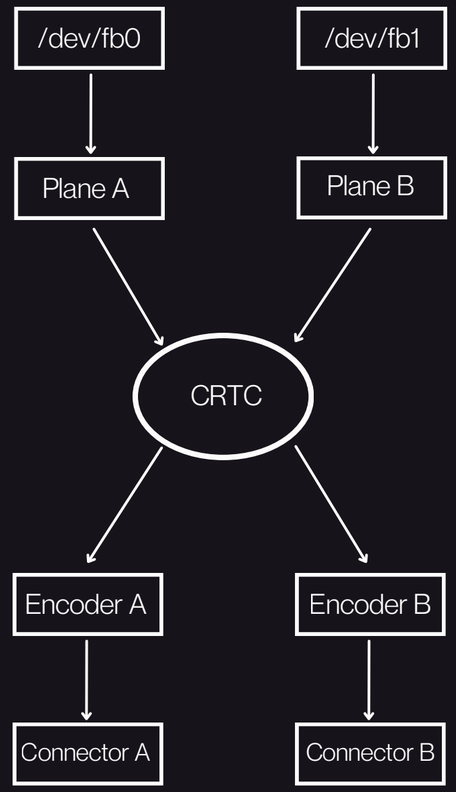
What Linux Does Well for NPUs?



DMA/IOMMU

What Linux Does Well for NPUs?

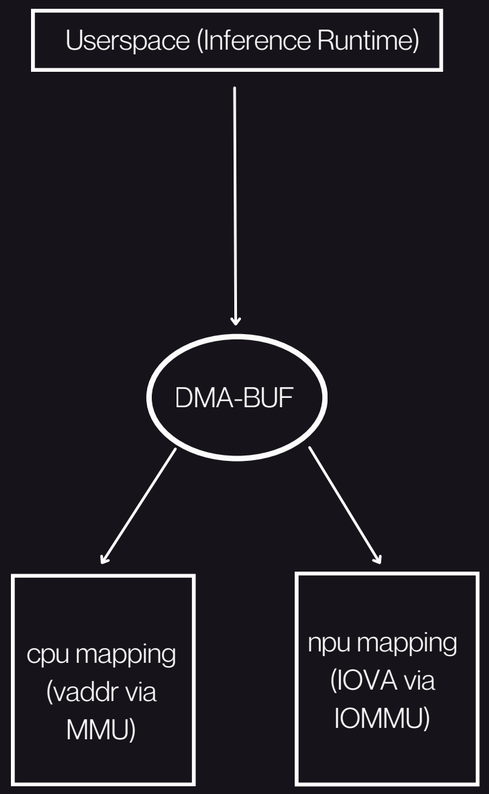




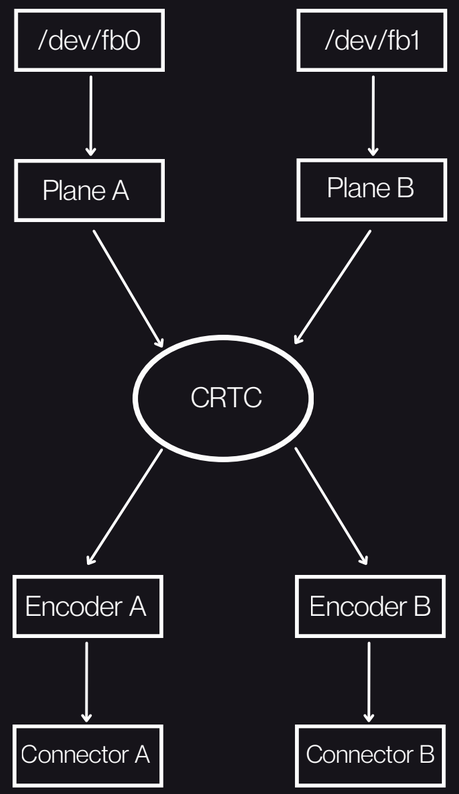
DMA/IOMMU

DRM/KMS

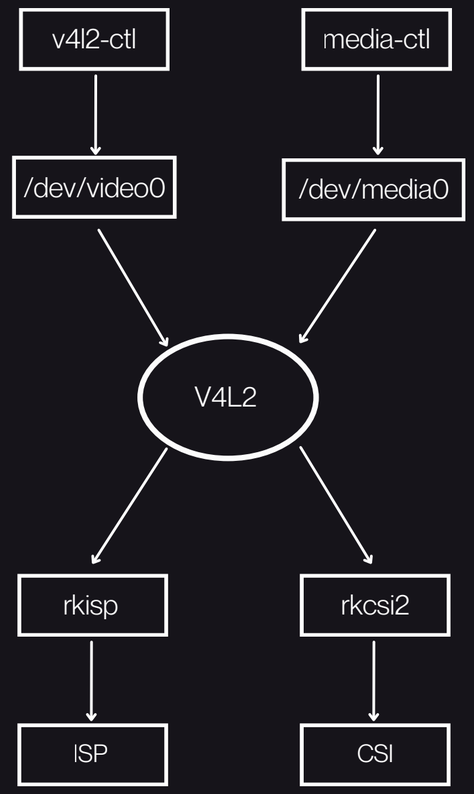
What Linux Does Well for NPUs?



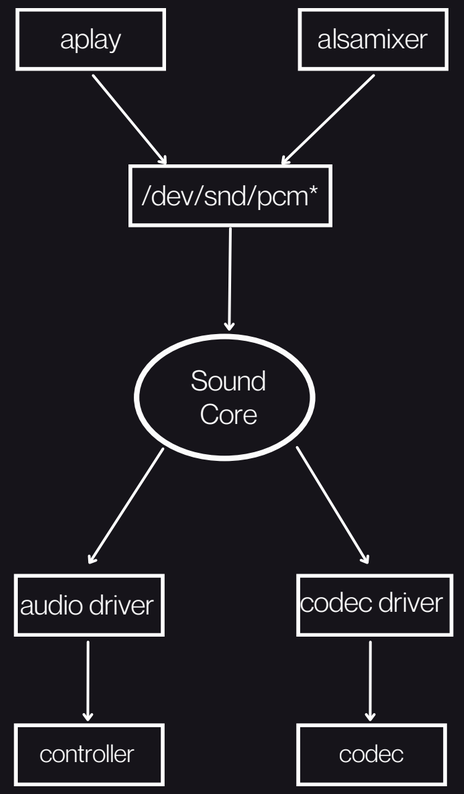
DMA/IOMMU



DRM/KMS



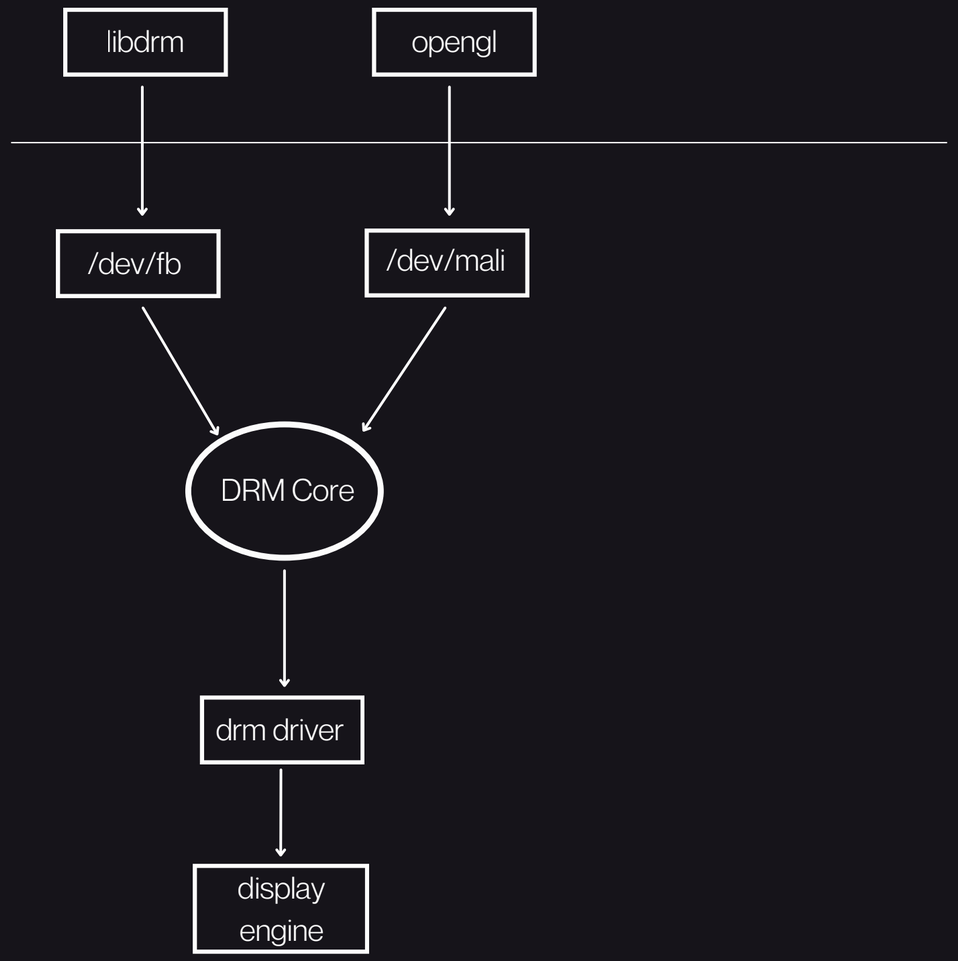
V4L2 (Vison)



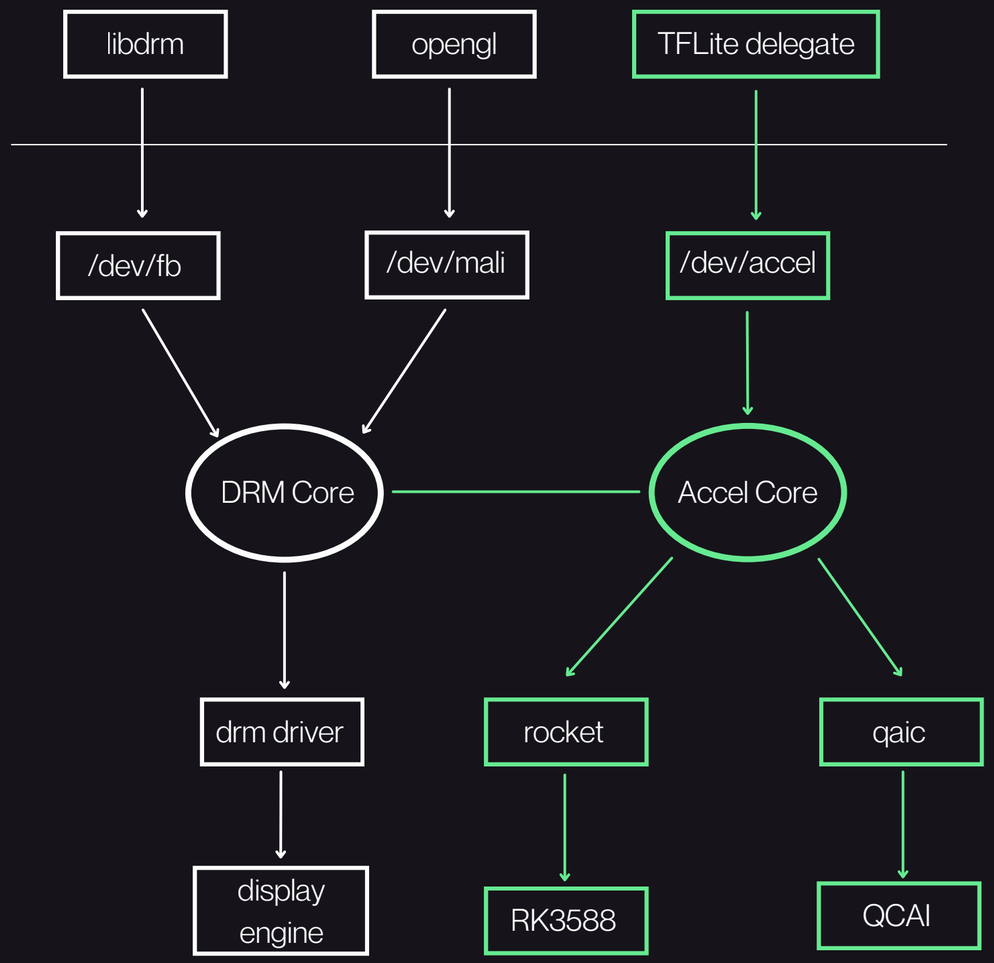
ALSA (Acoustic)

From kernel's POV, LLM I/O is just userspace buffers

Where Linux Falls Short:

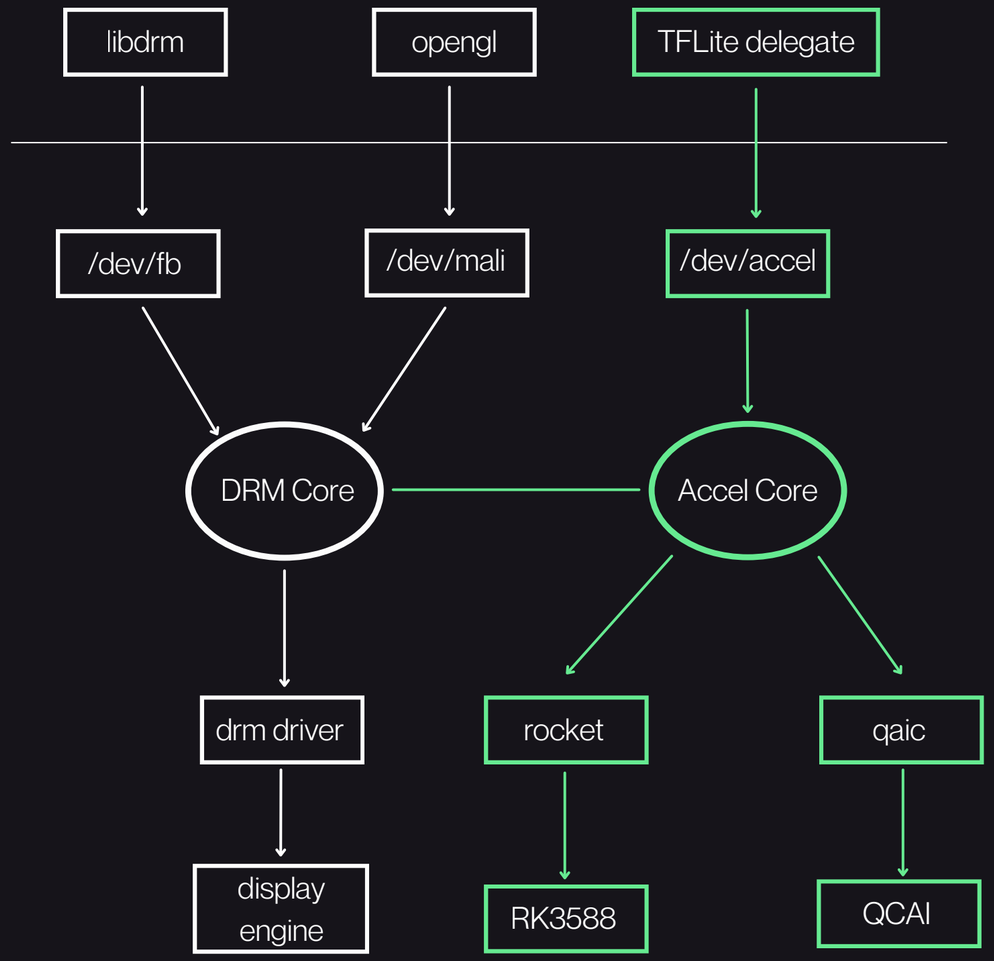


Where Linux Falls Short: NPU as a Compute Accelerator



Linux Compute Accelerator System → NPU

Where Linux Falls Short: NPU as a Compute Accelerator



→ device class (/dev/accelX)

→ fops glue

→ drm_ioctl, drm_file, drm_gem

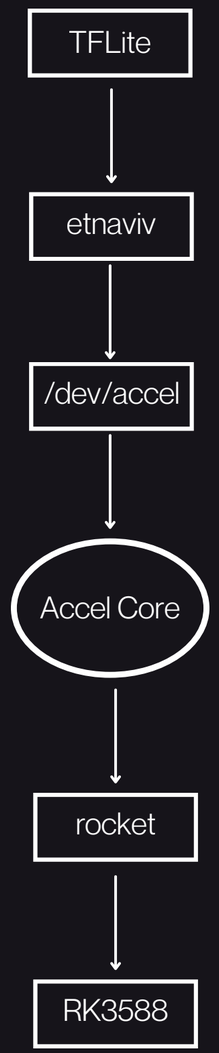
→ open the device – not what the device does?

```
#define DRM_ACCEL_FOPS \
        .open = accel_open, \
        .unlocked_ioctl = drm_ioctl, \
        .mmap = drm_gem_mmap
```

→ TensorFlow Lite delegate works? then

Linux Compute Accelerator System → NPU

drm/accel → TFLite Works.



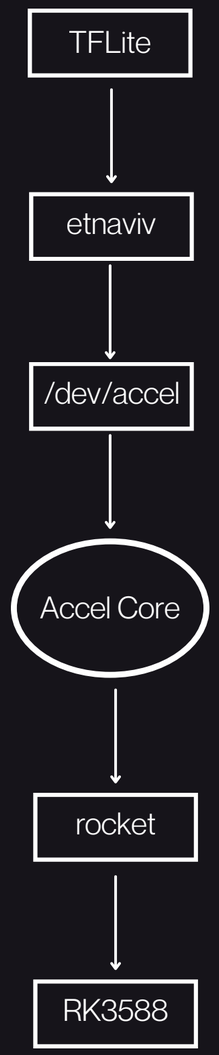
→ fixed model graph

→ known tensors

→ static memory

```
$ TEFLON_DEBUG=verbose ETNA_MESA_DEBUG=ml_dbgs python3.10
src/gallium/frontends/teflon/tests/classification.py \
       -i ~/tensorflow/assets/grace_hopper.bmp \
       -m src/gallium/targets/teflon/tests/models/mobilenetv1/mobilenet_v1_1_224_quant.tflite \
       -l src/gallium/frontends/teflon/tests/labels_mobilenet_quant_v1_224.txt \
       -e build/src/gallium/targets/teflon/libteflon.so

Loading external delegate from build/src/gallium/targets/teflon/libteflon.so with args: {}
Teflon delegate: loaded etnaviv driver

teflon: compiling graph: 89 tensors 28 operations
idx scale     zp has_data size
=======================================
0 0.023528   0 no       1x1x1x1024
1 0.166099  42 no       1x1x1x1001
2 0.000117   0 yes      1001x0x0x0
3 0.004987  4a yes      1001x1x1x1024
```

drm/accel → TFLite Works. Vision Tolerates.



→ fixed model graph

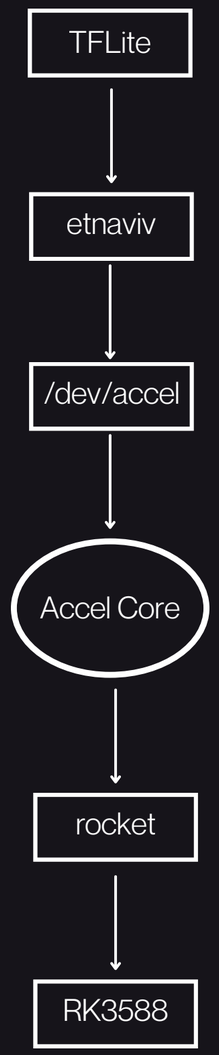
→ known tensors

→ static memory

```
$ TEFLON_DEBUG=verbose ETNA_MESA_DEBUG=ml_dbgs python3.10
src/gallium/frontends/teflon/tests/classification.py \
       -i ~/tensorflow/assets/grace_hopper.bmp \
       -m src/gallium/targets/teflon/tests/models/mobilenetv1/mobilenet_v1_1_224_quant.tflite \
       -l src/gallium/frontends/teflon/tests/labels_mobilenet_quant_v1_224.txt \
       -e build/src/gallium/targets/teflon/libteflon.so

Loading external delegate from build/src/gallium/targets/teflon/libteflon.so with args: {}
Teflon delegate: loaded etnaviv driver

teflon: compiling graph: 89 tensors 28 operations
idx scale     zp has_data size
=======================================
0 0.023528   0 no       1x1x1x1024
1 0.166099  42 no       1x1x1x1001
2 0.000117   0 yes      1001x0x0x0
3 0.004987  4a yes      1001x1x1x1024
```

→ vision models works, need strong userspace

drm/accel → TFLite Works. Vision Tolerates. LLM Exposes Gaps.



→ fixed model graph

→ known tensors

→ static memory

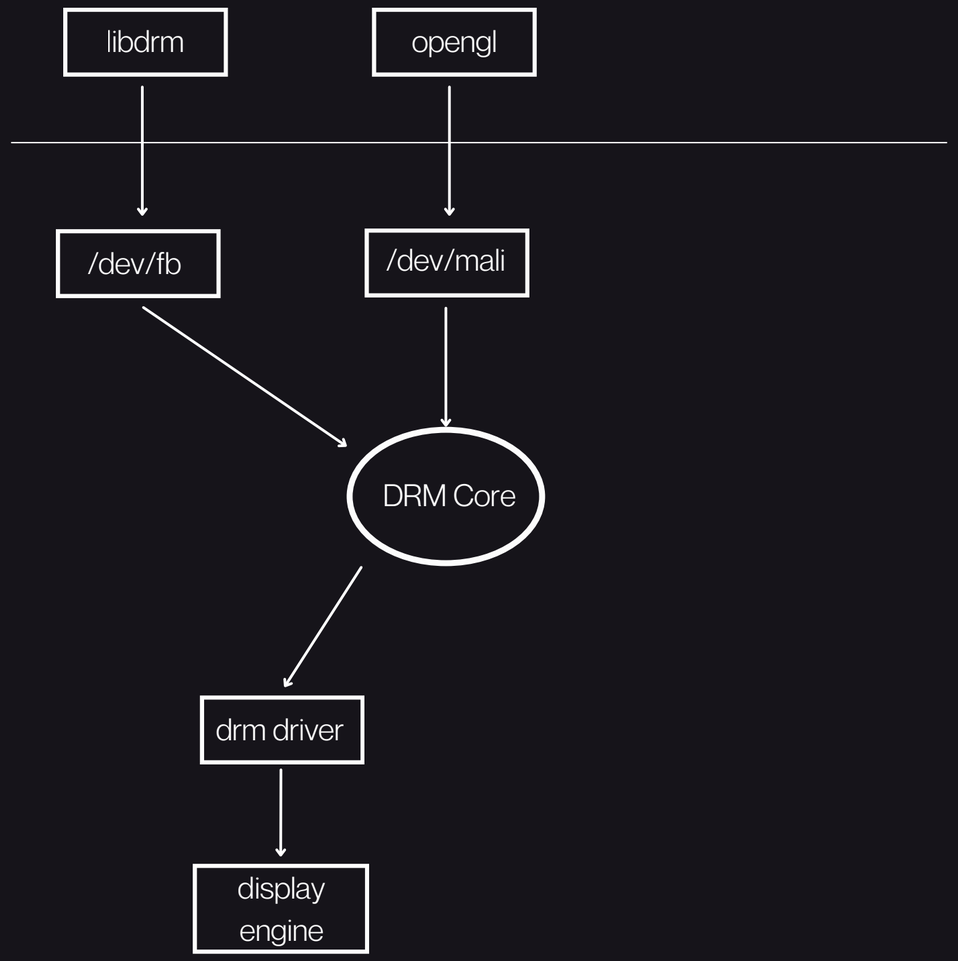
```
$ TEFLON_DEBUG=verbose ETNA_MESA_DEBUG=ml_dbgs python3.10
src/gallium/frontends/teflon/tests/classification.py \
       -i ~/tensorflow/assets/grace_hopper.bmp \
       -m src/gallium/targets/teflon/tests/models/mobilenetv1/mobilenet_v1_1_224_quant.tflite \
       -l src/gallium/frontends/teflon/tests/labels_mobilenet_quant_v1_224.txt \
       -e build/src/gallium/targets/teflon/libteflon.so

Loading external delegate from build/src/gallium/targets/teflon/libteflon.so with args: {}
Teflon delegate: loaded etnaviv driver

teflon: compiling graph: 89 tensors 28 operations
idx scale     zp has_data size
=======================================
0 0.023528   0 no       1x1x1x1024
1 0.166099  42 no       1x1x1x1001
2 0.000117   0 yes      1001x0x0x0
3 0.004987  4a yes      1001x1x1x1024
```

→ vision models works, need strong userspace

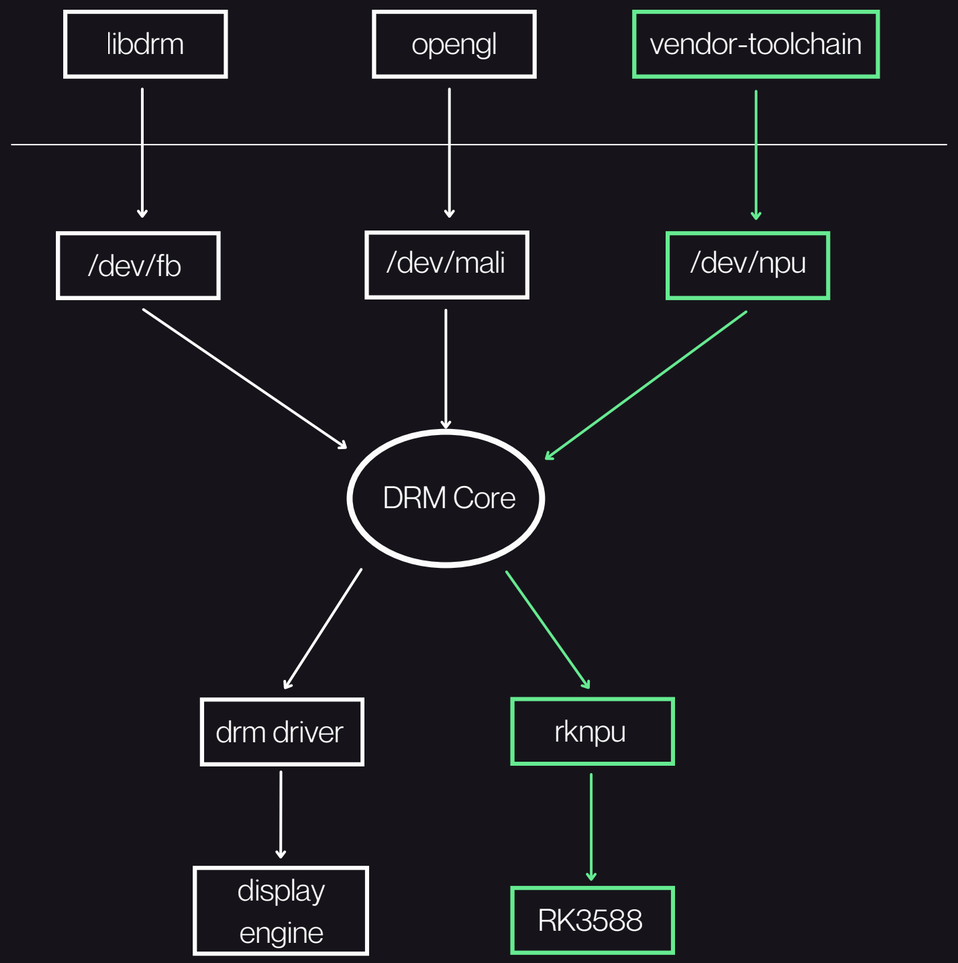
→ LLM workloads expose need for

- job queues
- execution contexts
- scheduling hints
- persistent allocations

Why Vendor NPUs Work Today:

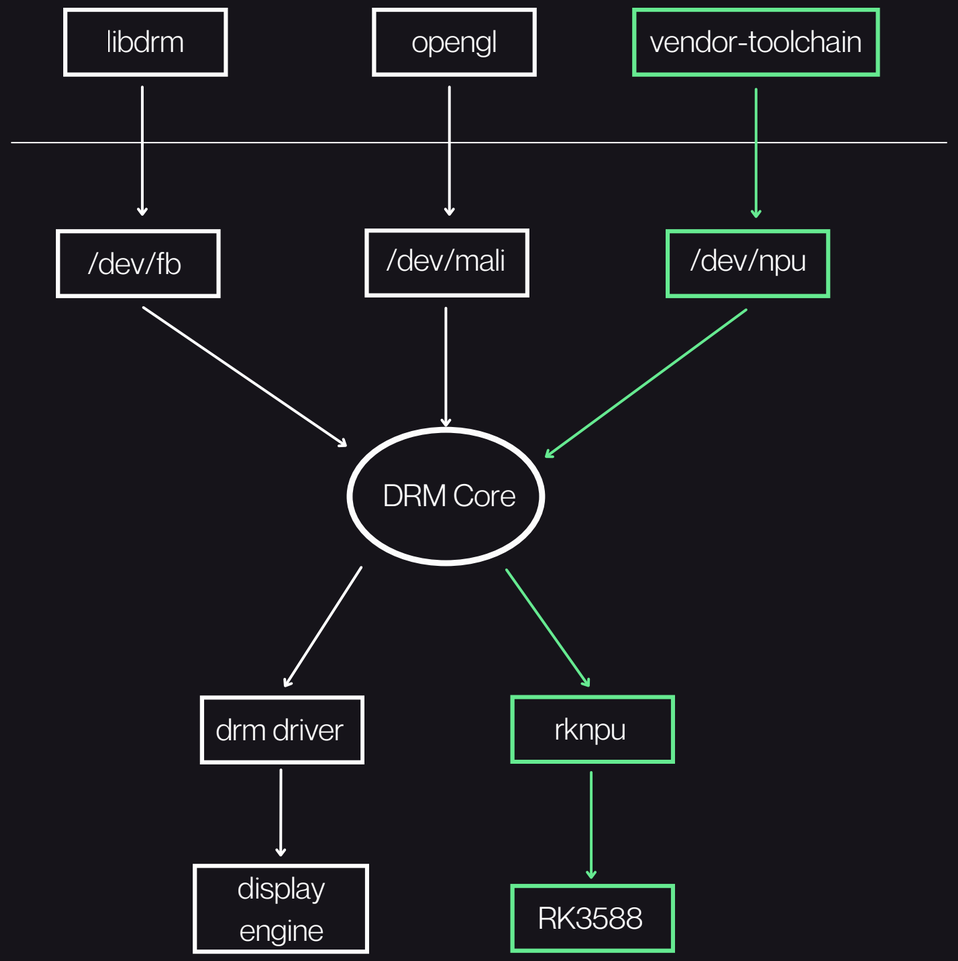


Why Vendor NPUs Work Today: Complete Stacks Inside DRM



Vendor NPU stacks resemble early DRM-era GPUs

Why Vendor NPUs Work Today: Complete Stacks Inside DRM



→ strong vendor-toolchain

```
struct rknpu_subcore_data {
        struct list_head todo_list;
        struct rknpu_job *job;
        wait_queue_head_t job_done_wq;
};
```

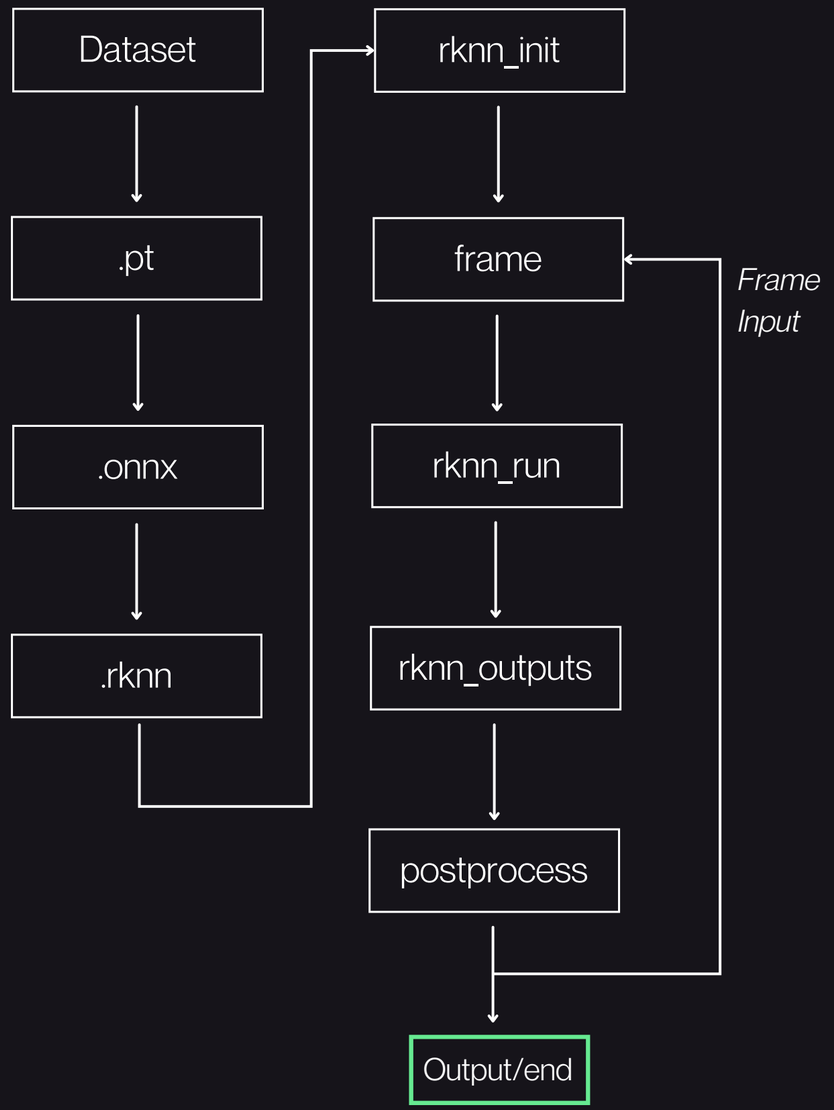
```
phys_addr_t sram_start;
phys_addr_t sram_end;
struct rknpu_mm *sram_mm;
phys_addr_t nbuf_start;
phys_addr_t nbuf_end;
```

```
atomic_t sequence;
struct rknpu_fence_context *fence_ctx;
```

```
struct devfreq *devfreq;
struct rockchip_opp_info opp_info;
struct thermal_cooling_device *devfreq_cooling;
```

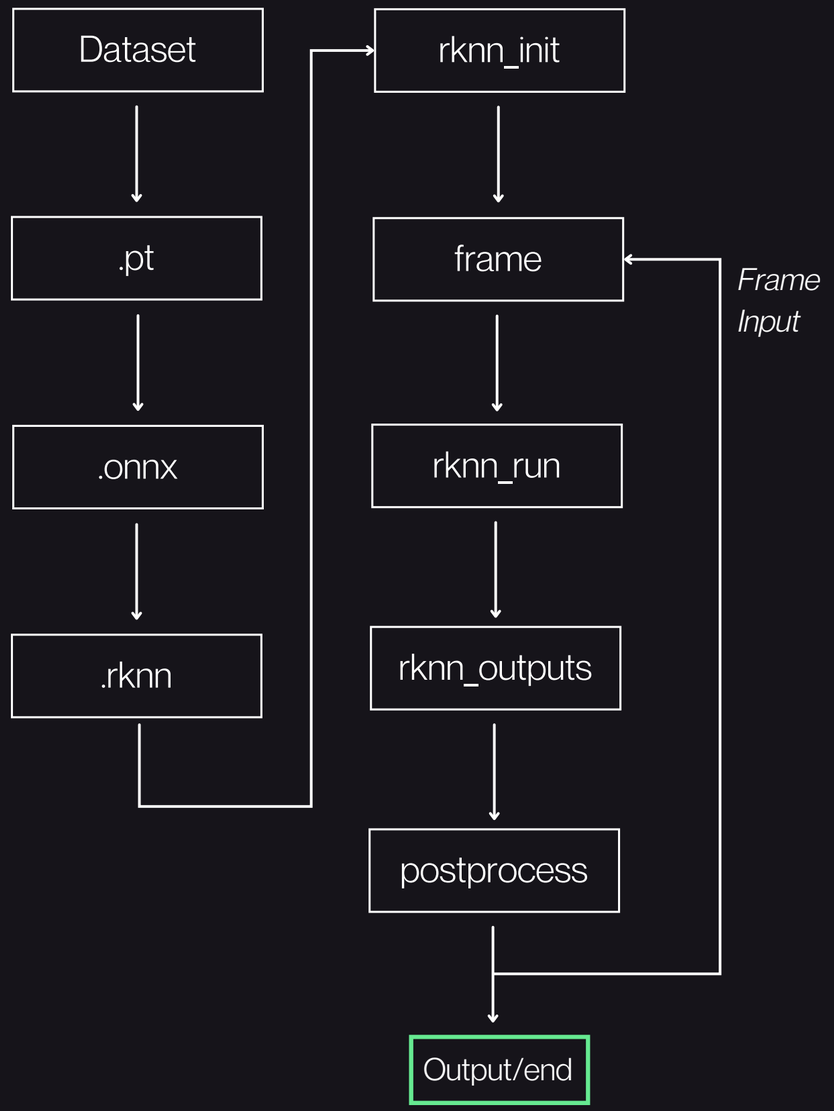
Vendor NPU stacks resemble early DRM-era GPUs

Deployed Vision workload

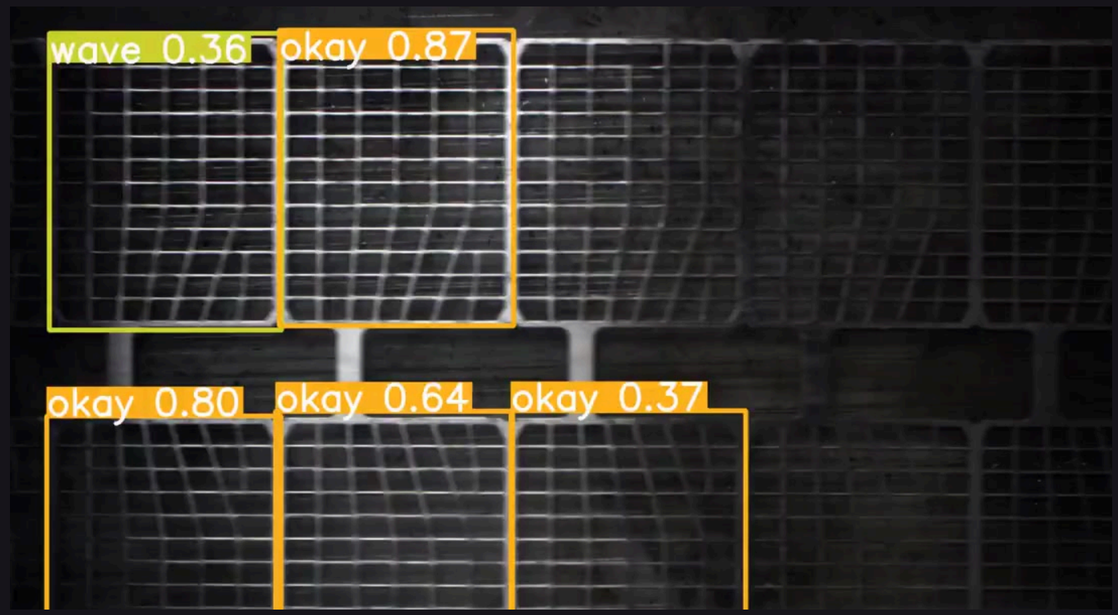


single job, short lifetime

Deployed Vision workload (YOLO / defects)

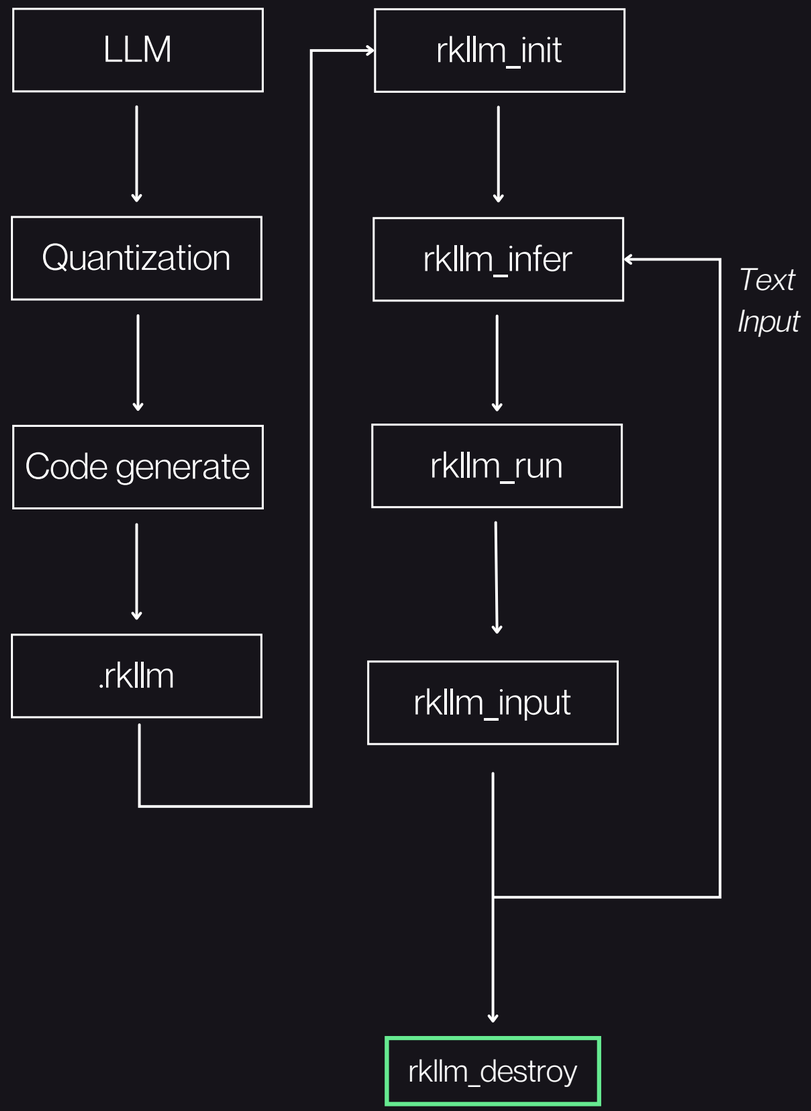



```
./edgegpt-0 model/metal-defects.rknn MIPI 11
Loading model...
sdk version: 1.5.2 (c6b7b351a@2023-08-23T15:28:22) driver version: 0.9.6
model input num: 1, output num: 3
model is NHWC input fmt
model input height=640, width=640, channel=3
QSettings::value: Empty key passed
QSettings::value: Empty key passed
img width = 0, img height = 0
rga_api version 1.9.1_[4]
loadLabelName ./model/coco_80_labels_list.txt
okay @ (84 784 276 939) 0.375294
okay @ (84 783 279 939) 0.317701
okay @ (81 783 279 939) 0.317701
okay @ (81 783 279 939) 0.317701
okay @ (81 783 279 939) 0.375294
okay @ (81 784 276 939) 0.218586
offset-grid @ (81 806 273 939) 0.218586
```



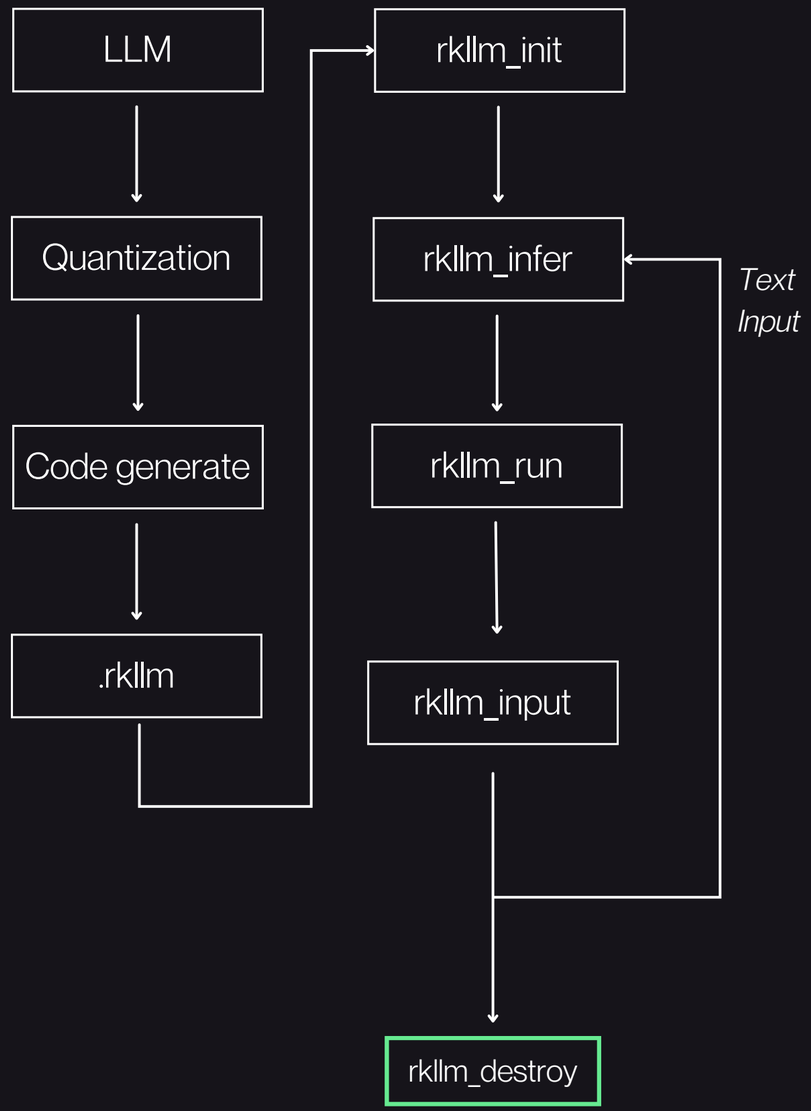
single job, short lifetime

Deployed LLM at Edge workload



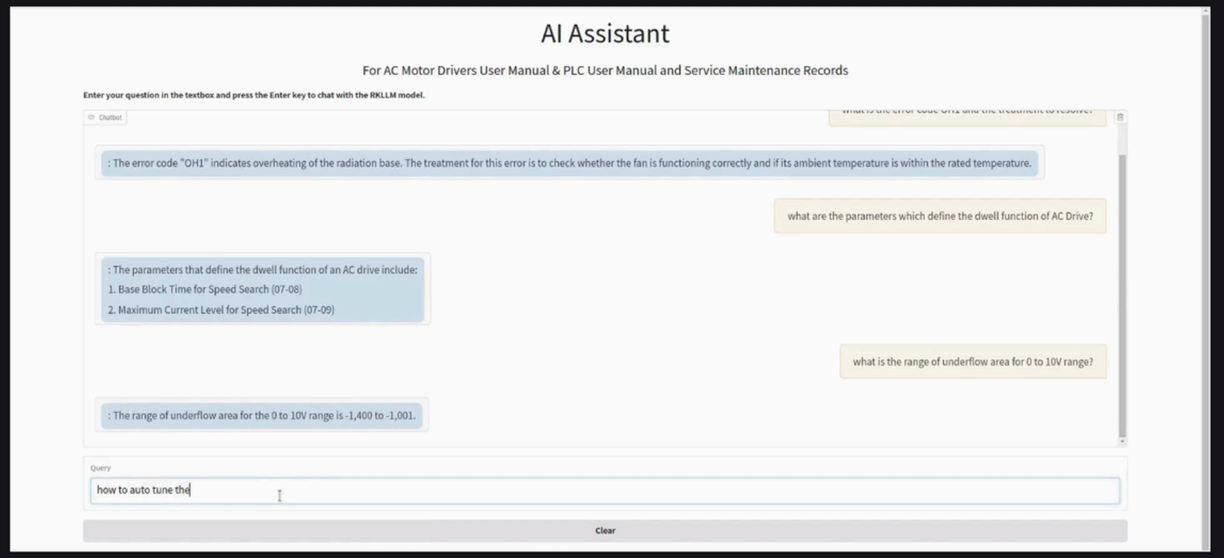
Repeated execution with
persistent buffers

Deployed LLM at Edge workload (DeepSeek-R1)



Repeated execution with persistent buffers

```
./llm_demo DeepSeek-R1-Distill-Qwen-1.5B_W8A8_RK3588.rkllm 2048 4096

rkllm init start
rkllm init success

user: There is a cage with chickens and rabbits in it. There are 14 heads and 38 legs. How many
chickens and rabbits are there?
robot: <think>
First, I'll define the variables for the number of chickens and rabbits.

Let \( c \) represent the number of chickens and \( r \) represent the number of rabbits.

Since each chicken has 1 head and each rabbit also has 1 head, the total number of heads is equal to
the sum of chickens and rabbits. Therefore:

<skip>

**Step 4:** Find the value of \( c \) using the value of \( r \).
\[
c = 14 - 5 = 9
\]

**Final Answer:**
- **Number of Chickens:** \( \boxed{9} \)
- **Number of Rabbits:** \( \boxed{5} \)
```



Verdicts: drivers/accel vs vendor npu

Aspect	drivers/accel	vendor npu
Job model	IOCTL only	Explicit jobs
SRAM	invisible	explicit
Persistence	none	managed
Scheduling	userspace	driver
Vision/Sound	tolerate	works
LLM viability	constrained	works

What Would a First-Class Linux NPU Subsystem Need?

→ Explicit job abstractions

→ Execution contexts

→ On-chip memory visibility

→ Persistence & residency

→ Scheduling hints (latency vs throughput)

...This mirrors the early evolution of DRM for GPUs.

Thank You

Question?