GOTC 2023

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE，INTO THE FUTURE

「AI is everything」专场

OPPO移动端图形技术领域探索实践 - O3DE Mobile WG 及 ShaderNN

OPPO开源办公室　彭周虎　2023年05月28日

ABOUT OPPO

Established in 2004, sells to over 60 countries & regions

#4 Global Smartphone Market Share 2022 & 2021

40,000+ Employees on R&D

500 Million Active Users

Architected by Zaha-Hadid

Dr. Hongyu Sun

Sr. Director, OPPO

Hongyu.sun@oppo.com

Dr. Sun is the head of computing & graphics research institute in OPPO, responsible for converting state of the art graphics technologies into products. He is particularly focused on efficient and realistic rendering, both traditional and AI-powered. Before OPPO, he was chief software architect for Huawei, terminal OS dept, responsible for graphics and computer vision features. He holds a Ph.D from Iowa State University, where he worked under Dr. Robyn Lutz.

OPPO Computing & Graphics Research Institute

Technology investment area

Graphics & Imaging Algorithm
Game Engine
System Graphics
GPGPU& AI Computing
Rendering Pipeline

Mission & Duties

- Working on state-of-the-art mobile graphics and computing technologies.
- Building a technology advancement branding image in mobile graphics industry.
- Driving product ready technology’s implementation. Supporting gaming and operating system oriented mobile graphics applications.

Seattle, Shenzhen, Shanghai, Nanjing

POTENTIAL CONTRIBUTION FROM OPPO TO O3DE

Graphics & Imaging Algorithm

- Develop hybrid rendering pipeline (Rasterization + Ray Tracing)
- Mobile GPU features: Hardware accelerated Ray Tracing, Variable Rate Shading (VRS)
- Mobile Ray Tracing benchmark

Game Engine

- Mobile friendly game development ecosystem (IDE, build, test, deployment etc.)

System Graphic Technology

- Popular Mobile Game Engine Features: Adaptive Performance (ADP)
- Mobile perf tuning and profiling toolset

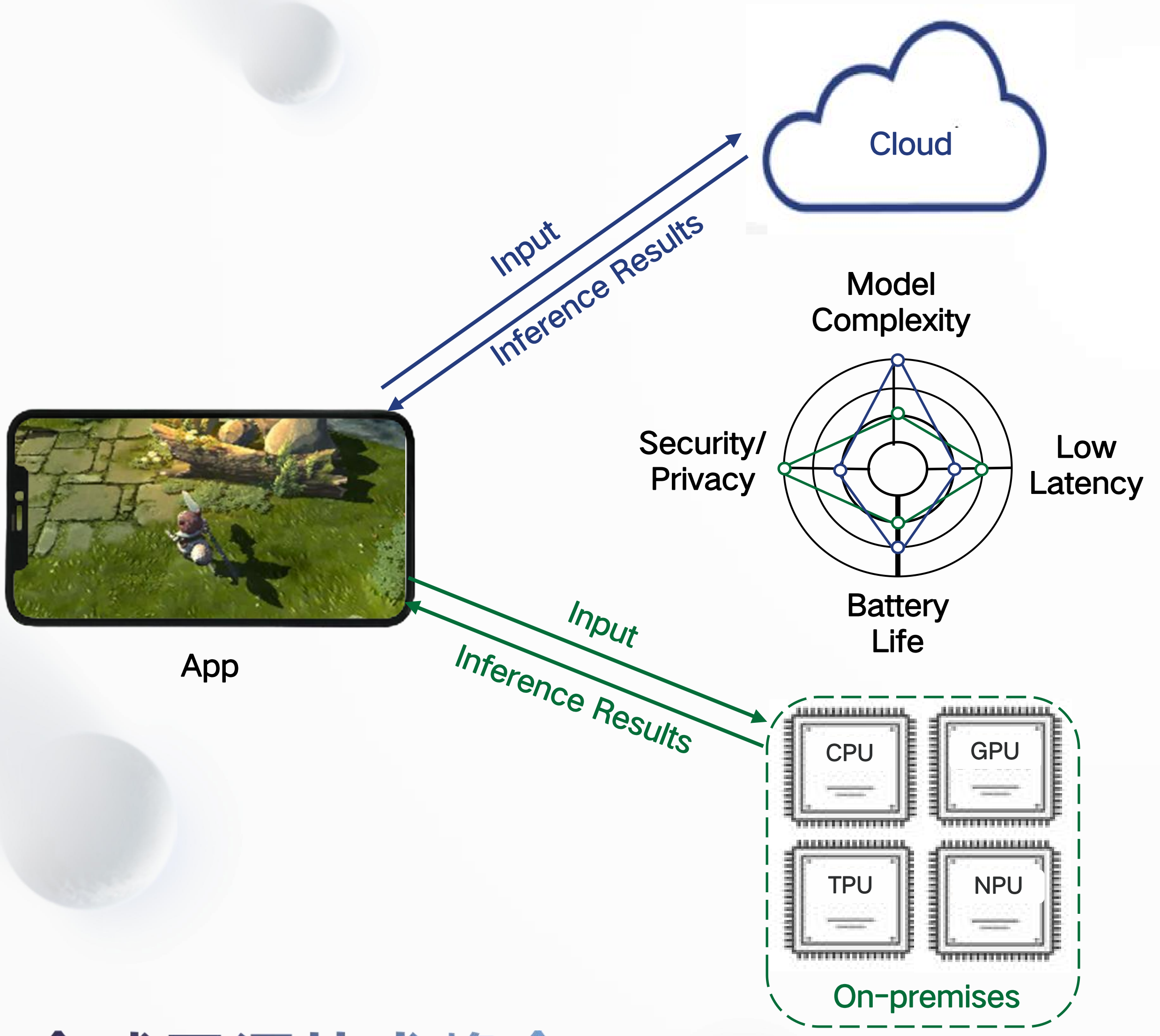
VALUE OPPO BRINGS TO O3DE

Drive technology's propagation and adoption:

- Bring our deep understanding of mobile game platforms and users to O3DE society.
- Push mobile technology evolvement in the industry, drive the formalization of best practice and common standards.
- Speed up O3DE's adoption to mobile game developers.
- **Support ShaderNN2.0 Plugin.**

Mobile Inference Engine Overview

GOTC



Cloud Inferencing

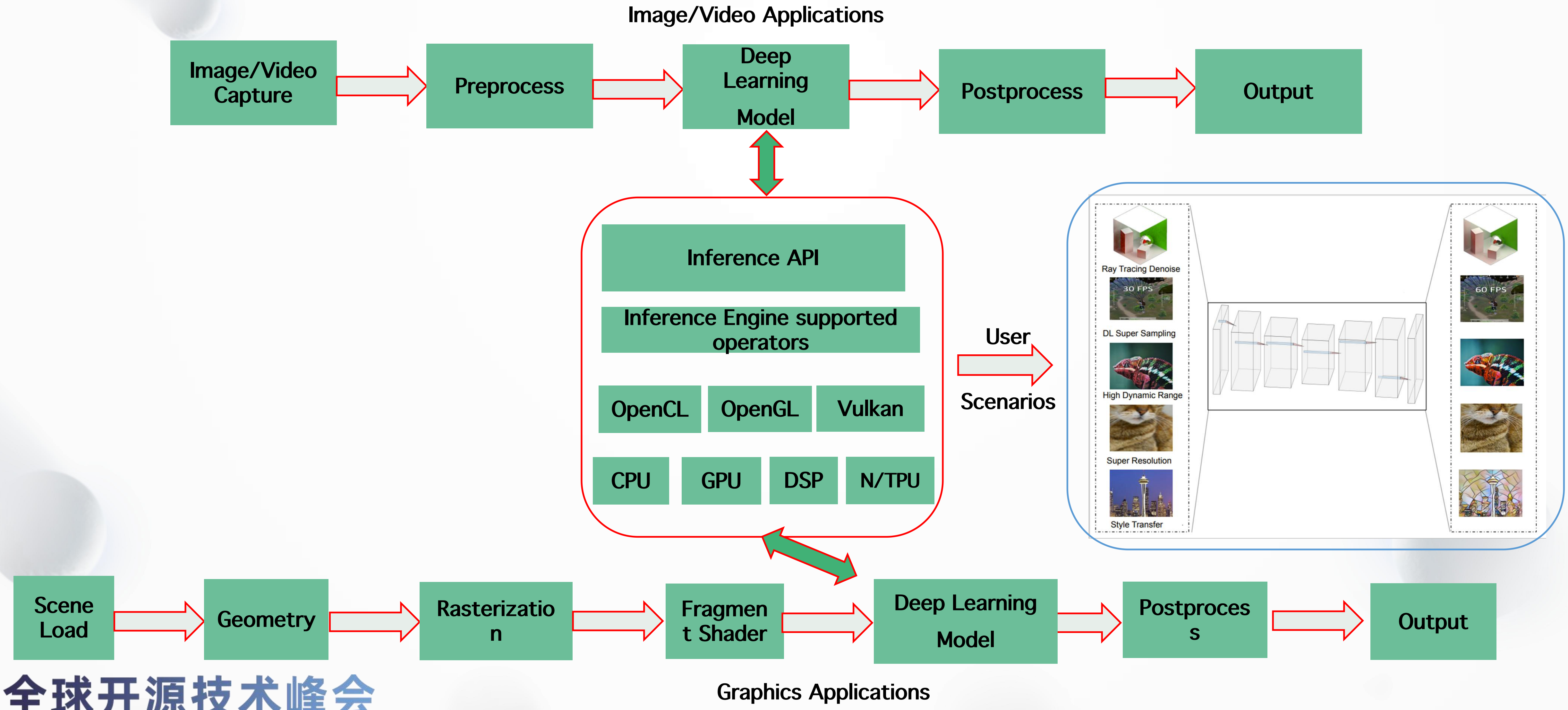
On-premises Inferencing

Major challenges for on-premises inference for mobile devices:

- Limited computational capacity.
- Low power budget.
- Model compatibility.
- Customizable and lightweight implementation.
- Deeply coupled with image/graphic applications.
- Varied memory access methods and I/O bus bandwidth.

	CPU	SIMD	OpenCL	OpenGL Compute Shader	OpenGL Fragment Shader	Vulkan	NPU/ DSP
TensorFlow Lite	V	V	V	V			V
MNN	V	V	V	V		V	V
NCNN	V	V				V	
TNN	V	V	V				V
BOLT	V	V	V				
MACE	V	V	V				V
ShaderNN	V			V	V	V	

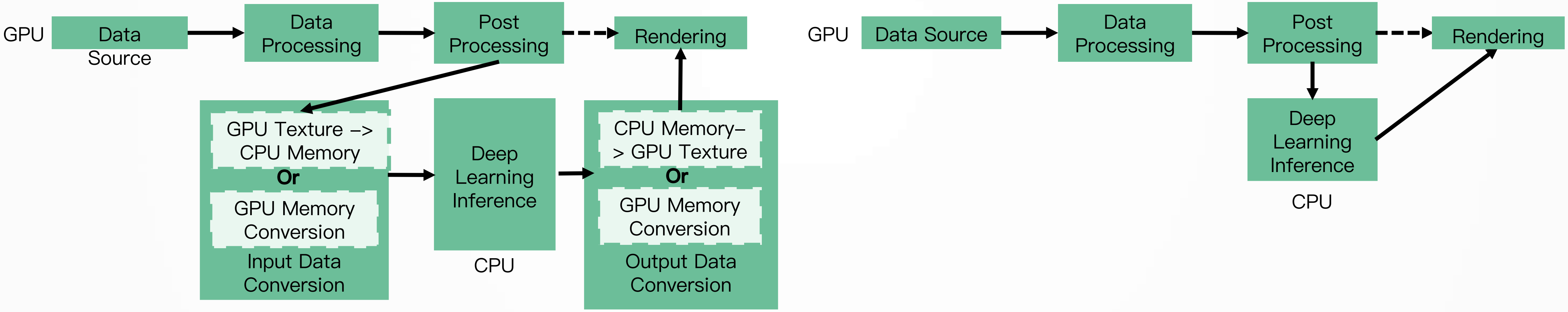
ShaderNN: A Shader Based Lightweight and Efficient Inference Engine for Mobile GPU



Innovations of ShaderNN

GOTC

- Use texture-based input/output, which provides an efficient, zero-copy integration with real-time graphics pipeline or image processing applications, thereby saving expensive data transfers & format conversion between CPU and GPU.



A. Integrate with other inference engines

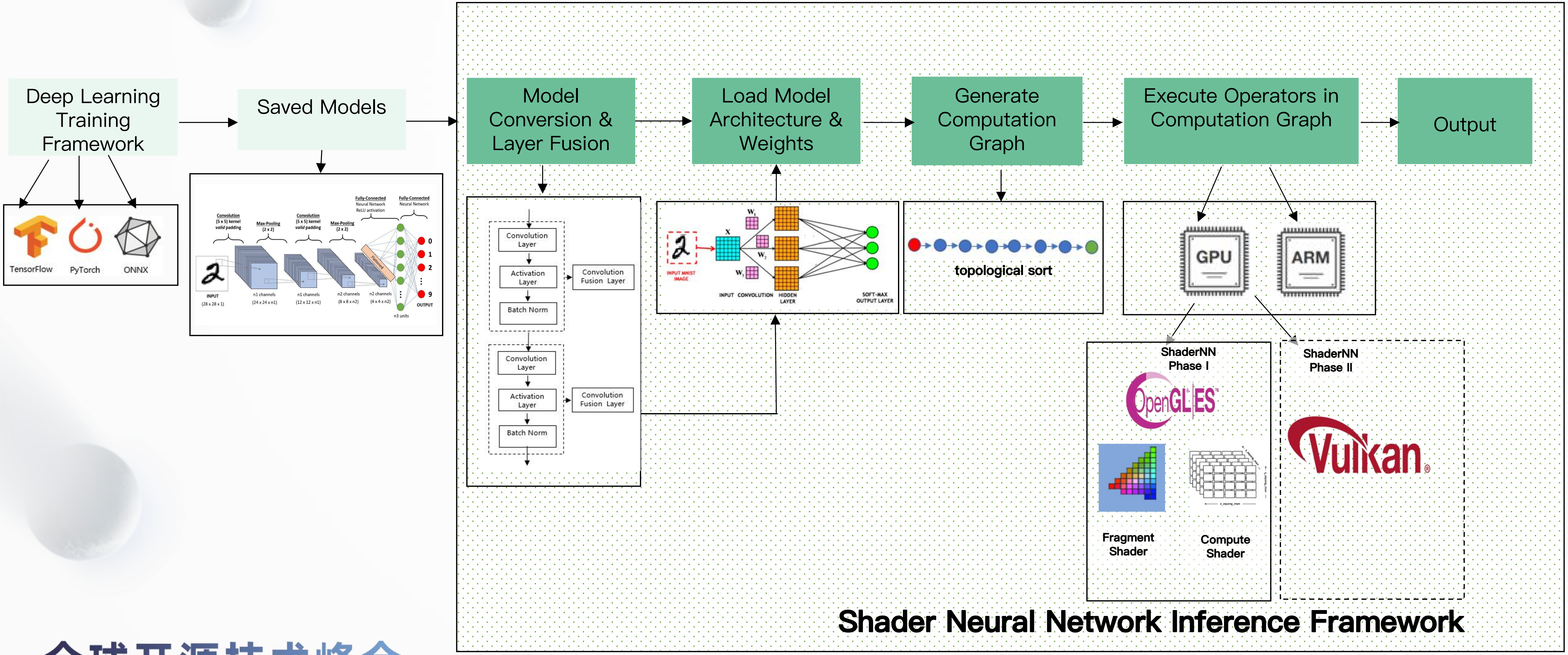
B. Integrate with ShaderNN

- Built on **native OpenGL ES and Vulkan**, which can be easily integrated with the graphics rendering pipeline to maximize the use of computing resources, suits for rendering, image/video and game AI applications.

- Leverage the **fragment shader** based on OpenGL backend in the neural network inference operators, which is advantageous when deploying parametrically small neural network modes.

- Enable a **hybrid implementation of compute and fragment shaders**, with the ability to select layer-level shaders for performance optimization.

ShaderNN Workflow



ShaderNN Framework Architecture

GOTC

Model Preparation	Framework	TensorFlow	PyTorch	ONNX	
	Conversion Tool	TensorFlow Converter	PyTorch Converter	ONNX Converter	
	Model Optimizations	Model Compressions	Layer Fusion	Grouping Optimization	Operator Optimization
Inference Engine	Inference Graph	Computation Graph Generation	Topological Sort Schedule		
	Compile Optimization	Shader Optimization	Equivalent Layers Fusion		
	Runtime Optimization	Convolutional Optimization	Texture Reuse	Multi Thread	CPU、GPU Memory Reuse
		C4 Data Layout Cache Vectorization			

Supported Operators (Inference Engine)

	OpenGL Fragment Shader	OpenGL Compute Shader	CPU	Vulkan Compute Shader
Conv2D	X	X		X
Conv2DTranspose	X			
DepthwiseConv2D	X	X		X
Concatenate	X	X		X
Add	X	X		X
Average Pooling	X	X		X
Max Pooling	X	X		X
Flatten		X	X	X
Dense		X	X	X
Upsampling	X	X		X
Yolo Layer			X	
Padding	Vaild/None, Same, Replicate(mirrored padding), Checkerboard(Repeat padding)			
Activation Functions	Linear, Relu, LeakyRelu, Tanh, Sigmoid, SiLU			
BatchNorm				
Layers/Operators Fusion	Padding, Activation Function and BatchNorm are combined with Conv2D, Conv2Dtranspose and DepthwiseConv2D			

Inference Engine	Hardware Backends	OpenGL GPU backend	Vulkan GPU backend	CPU backend

CNN Applications	Common CNN Scenarios	Classification	Object Detection	Image Segmentation	Image Enhancement
	Model Zoo	ResNet18, MobileNetV2	YoloV3-Tiny	Unet	ESPCN, Spatial Denoise

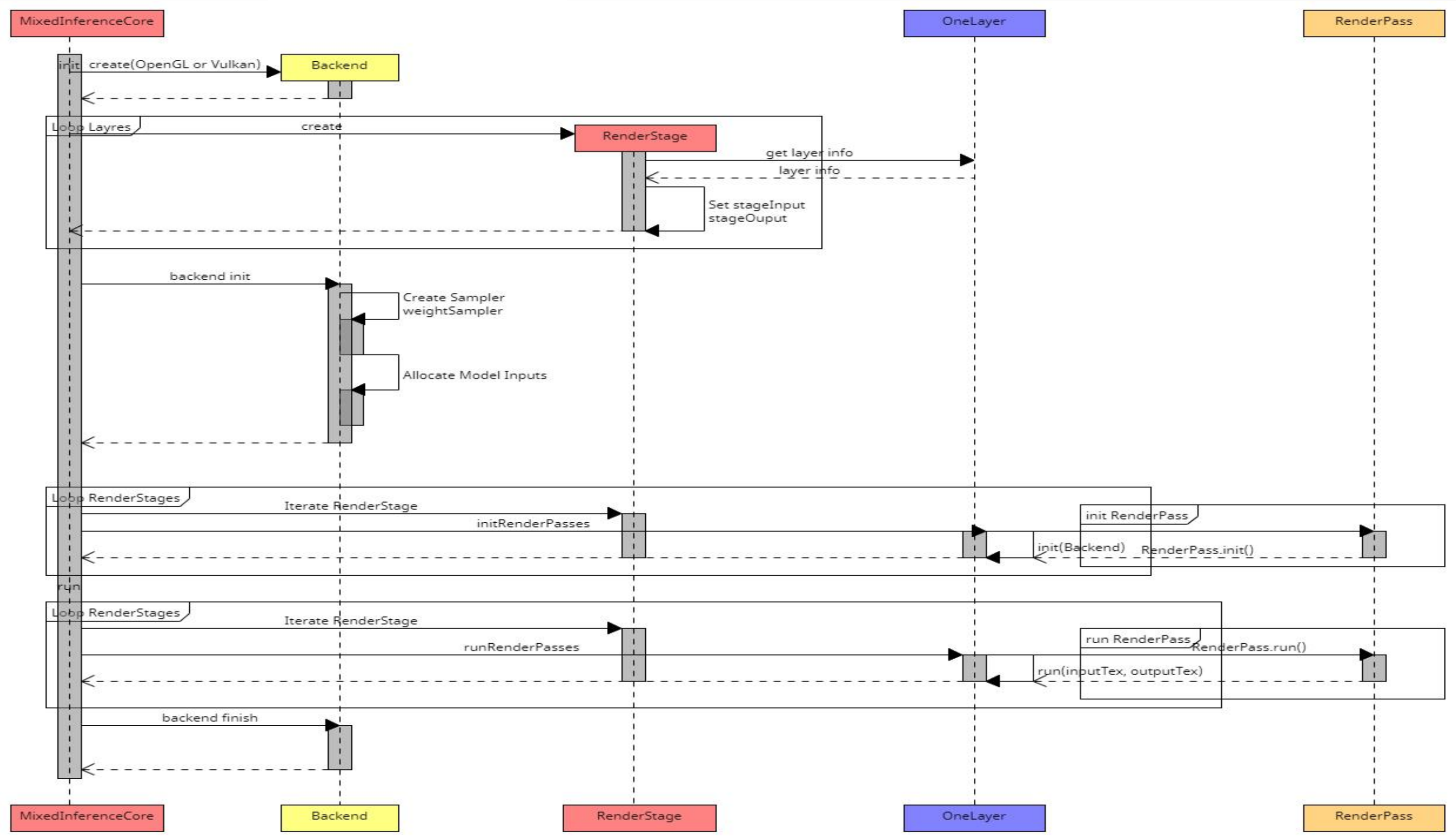
ShaderNN Inference Core Algorithms

Input: InferenceGraph
Output: RenderStage
Function `init()`:
  $layers \leftarrow InferenceGraph \rightarrow layers$
  $M \leftarrow layers.size()$
  for $i \leftarrow 0$ *to* M **do**
    $stage[i] \leftarrow new\ RenderStage()$
    $stage[i] \rightarrow layer \leftarrow layers[i]$
    $N \leftarrow layers[i].inputs.size()$
    for $j \leftarrow 0$ *to* N **do**
      $input \leftarrow layers[i].inputs[j]$
      if $input.isStageOutput$ *is true* **then**
        $texture \leftarrow$
        $input.stageOutputs[0].texture$
      else
        $texture \leftarrow modelInputs[j].texture$
      end
      $stage[i].stageInputs[j].texture \rightarrow$
      $attach(texture)$
    end
    $stage[i].stageOutputs[0].texture \rightarrow allocate()$
    $P \leftarrow layers[i].passes.size()$
    for $k \leftarrow 0$ *to* P **do**
      $stage[i].renderPasses[k].init()$
    end
  end

Algorithm 1: Initialization of Inference Core

Input: RenderStages, InputTextures
Output: OutputTexture
Function `run()`:
  $L \leftarrow length(InputTextures)$
  for $i \leftarrow 0$ *to* L **do**
    $modelInputs[i].texture(0) \rightarrow$
    $attach(InputTextures[i])$
  end
  $M \leftarrow RenderStages.size()$
  for $j \leftarrow 0$ *to* M **do**
    $renderPasses \leftarrow RenderStages[j].renderPasses$
    $N \leftarrow renderPasses.size()$
    for $k \leftarrow 0$ *to* N **do**
      $renderPasses[k].run()$
    end
  end

Algorithm 2: Run of Inference Core



Key Features of ShaderNN

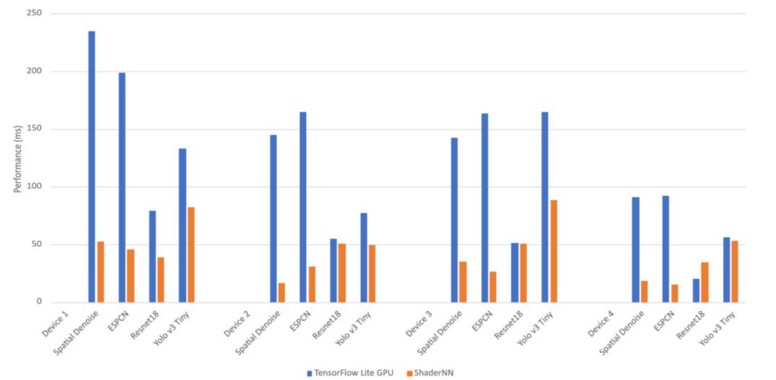
- ## High Performance
  - **Utilize GPU Shader:** Implement core operators using GPU Shader to leverage parallel computing capabilities for optimal performance.
  - **Pre-built Static Computation Graph:** Optimize with constant folding and operator fusion to accelerate forward operation speed.

- ## Lightweight & Portability & Extensibility
  - **No Third-Party Library Dependencies:** Ensure independence from external libraries, reducing overhead and simplifying integration.
  - **Mobile Platform Optimization:** Optimize specifically for mobile platforms, enabling effortless portability, deployment, and upgrades.
  - **Simple Input/Output Interface:** Provide a user-friendly interface compatible with GPU processing for streamlined interactions.

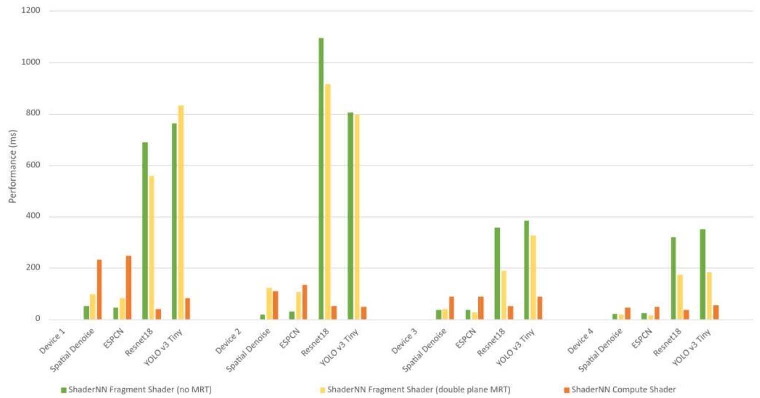
- ## Versatility
  - **Framework & CNN network Compatibility:** Support popular framework formats like TensorFlow, PyTorch, and ONNX. Support common classification, detection, segmentation, and enhancement networks.
  - **User-Defined Operators:** Enable easy implementation of new models by supporting user-defined operators.
  - **Flexible backend configure:** Select the running backend statically or dynamically according to the platform resources during model execution, dynamically adjusting kernel running parameters for minimal energy consumption at runtime.

ShaderNN Performance and Power Consumption Comparison – OpenGL backend with TensorFlow Lite

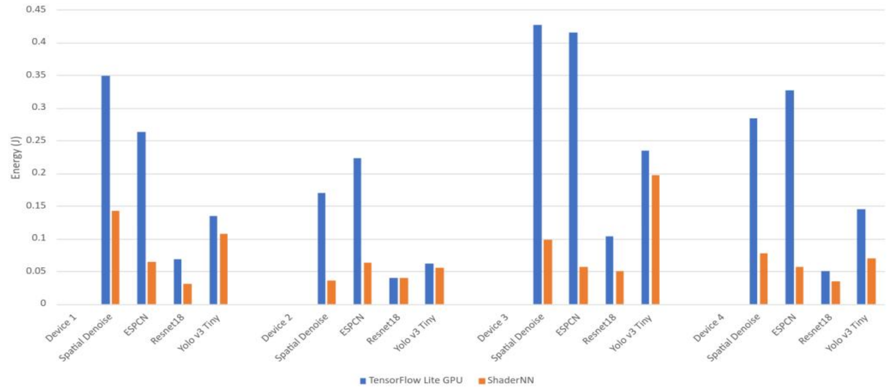
GOTC



Performance comparison



Performance comparison over MRT and Fragment/Compute Shader



Power consumption comparison

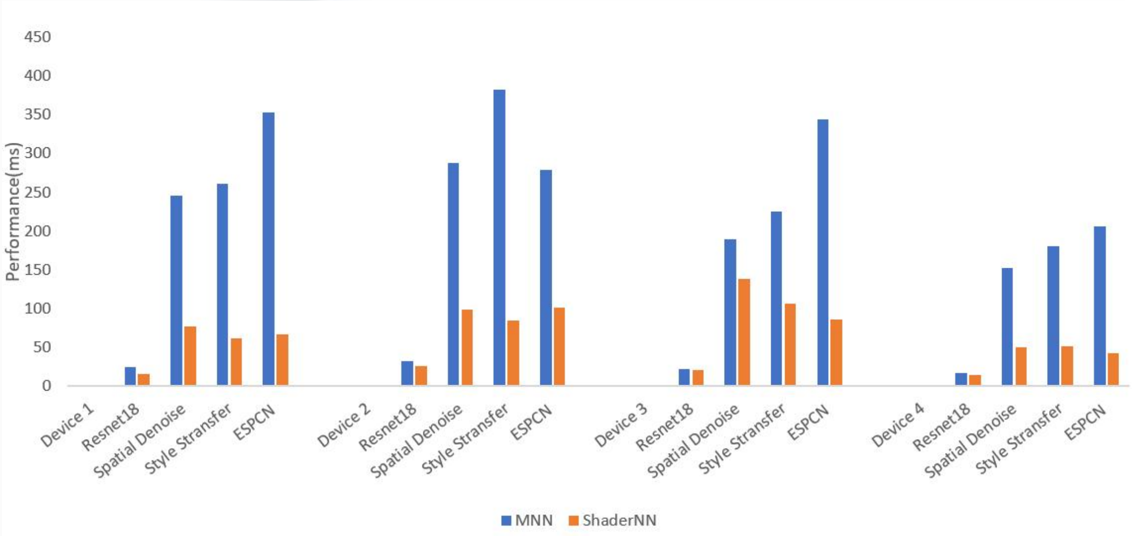
- On selected target processor chipsets, ShaderNN outperforms TensorFlow Lite on certain tasks, with 75%-90% better performance on spatial denoise and ESPCN, and up to 50% better performance on Resnet18 and YOLO v3 tiny.

Device	Chipset	GPU
1	Dimensity 1300 (MT6893)	Mali G77
2	Dimensity 9000 (MT6983)	Mali G710
3	Snapdragon 888 (SM8350)	Adreno 660
4	Snapdragon 8 Gen 1 (SM8450)	Adreno 730

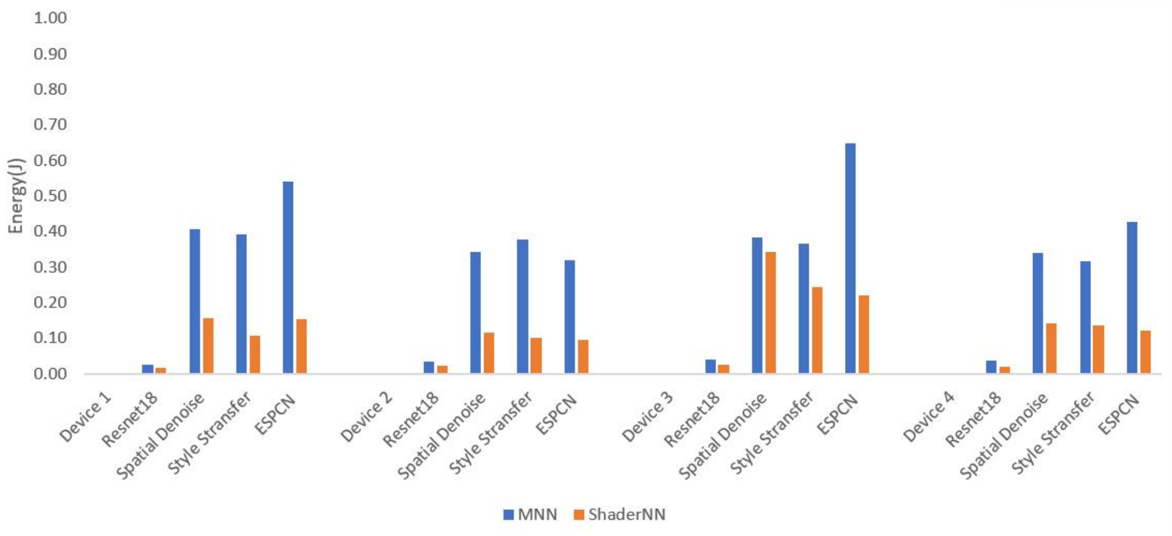
- The fragment shader pipeline offers the option to execute as either no MRT (single render target) or double plane MRT.
- On certain Qualcomm chipsets like Snapdragon SM8350 and SM8450, MRT optimization can provide additional speed up.

- When inferring Spatial Denoise, ESPCN, Resnet18, and YOLO v3 tiny, ShaderNN can save up to 80%, 70%, 55%, and 51% of energy, respectively.

ShaderNN Performance and Power Consumption Comparison – Vulkan backend with MNN



Performance comparison



Power consumption comparison

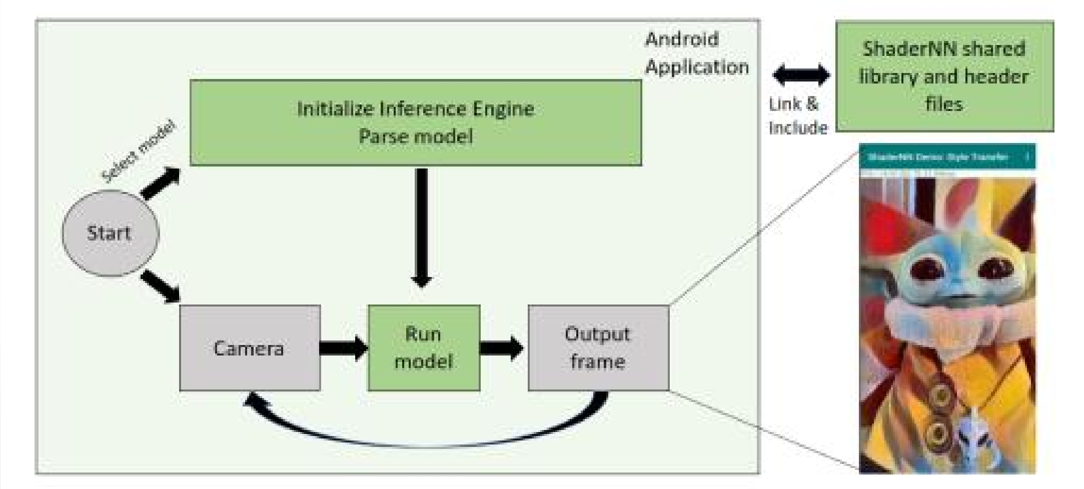
- ShaderNN outperforms MNN on selected target processor chipsets, with 50%-80% better performance on tasks such as spatial denoise and ESPCN, and 6%-60% better performance on tasks such as Resnet18 and Style Transfer.

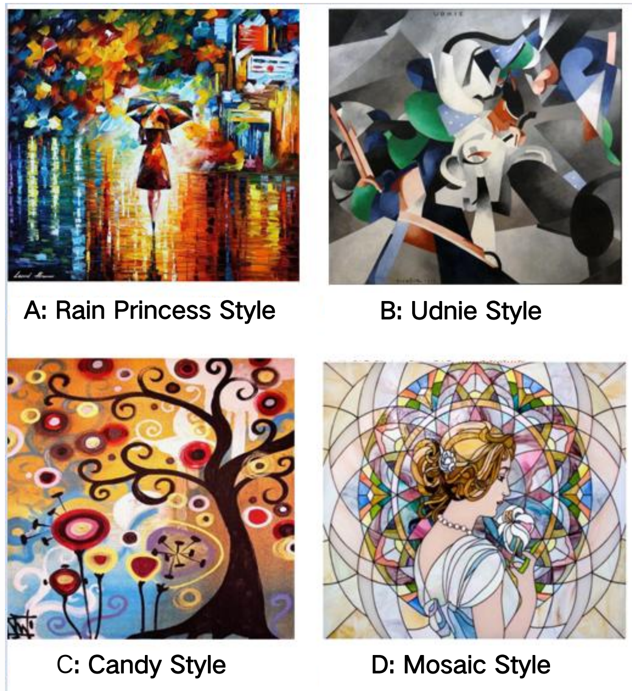
Device	Chipset	GPU
1	Snapdragon 8 Gen 1(SM8450)	Adreno 730
2	Snapdragon 8 Gen 2(SM8550)	Adreno 740
3	Dimensity 9000 (MT6983)	Mali G710
4	Dimensity 9200 (MT6985)	Mali G715

- When inferring tasks such as Spatial Denoise, ESPCN, Resnet18, and Style Transfer, ShaderNN can save up to 60%, 70%, 45%, and 70% of energy, respectively.

ShaderNN Android Demo App

- A demo app pipeline optimized for throughput over latency, data transfer, and video processing.





Fast Neural Style Transfer described in Perceptual Losses for Real-Time Style Transfer and Super-Resolution along with Instance Normalization

Cooperation between Academia and Industry

IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. XX, NO. XX, XX 2022 3

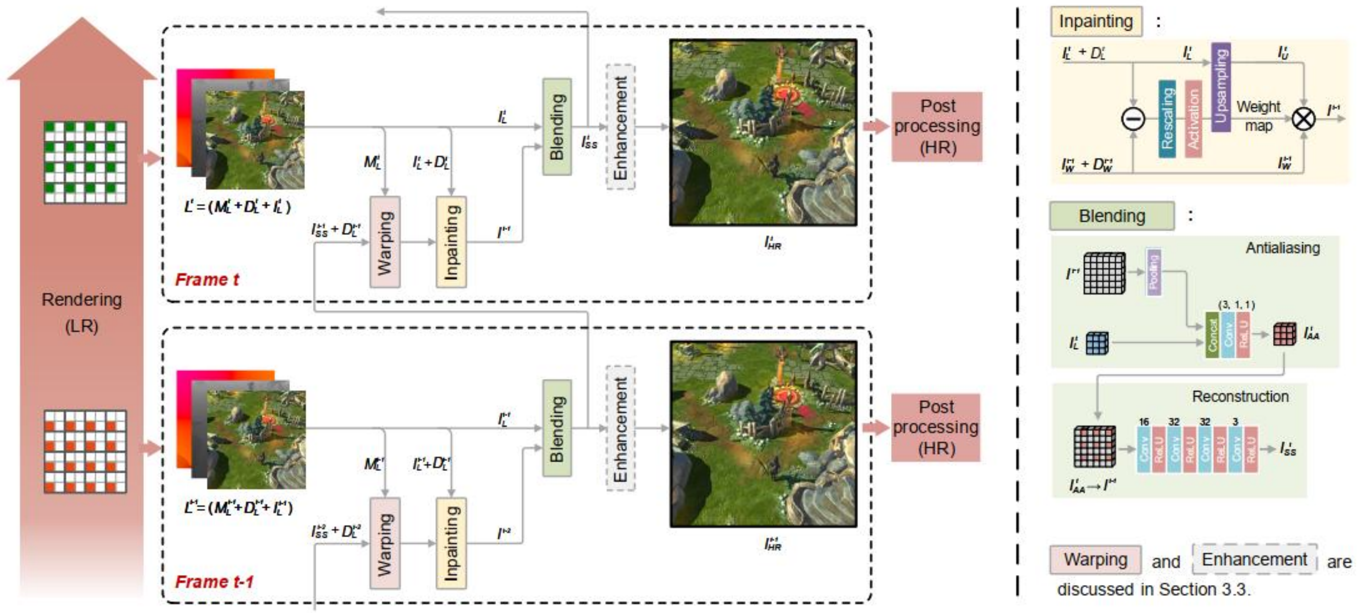


Fig. 2. Overview of our proposed neural supersampling framework. The left shows the pipeline of the method, and the right shows the architecture of sub-networks. For current *Frame t*, we first render the LR data L^t by adding a viewport sub-pixel offset to the camera. Then, the previous reconstructed frame I_{SS}^{t-1} and its depth map D_L^{t-1} are loaded and reprojected to align to the current frame using the motion information M_L^t, following which a weight map is generated by inpainting module to fill in invalid history pixels. After that, the current frame I_L^t and the repaired history frame I^{t-1} are fed into the blending network to generate HR output I_{SS}^t. In addition, the enhancement module can be optionally active by the user to sharpen edges. Lastly, the reconstructed frame is pulled through the post-processing stage of the rendering pipeline.

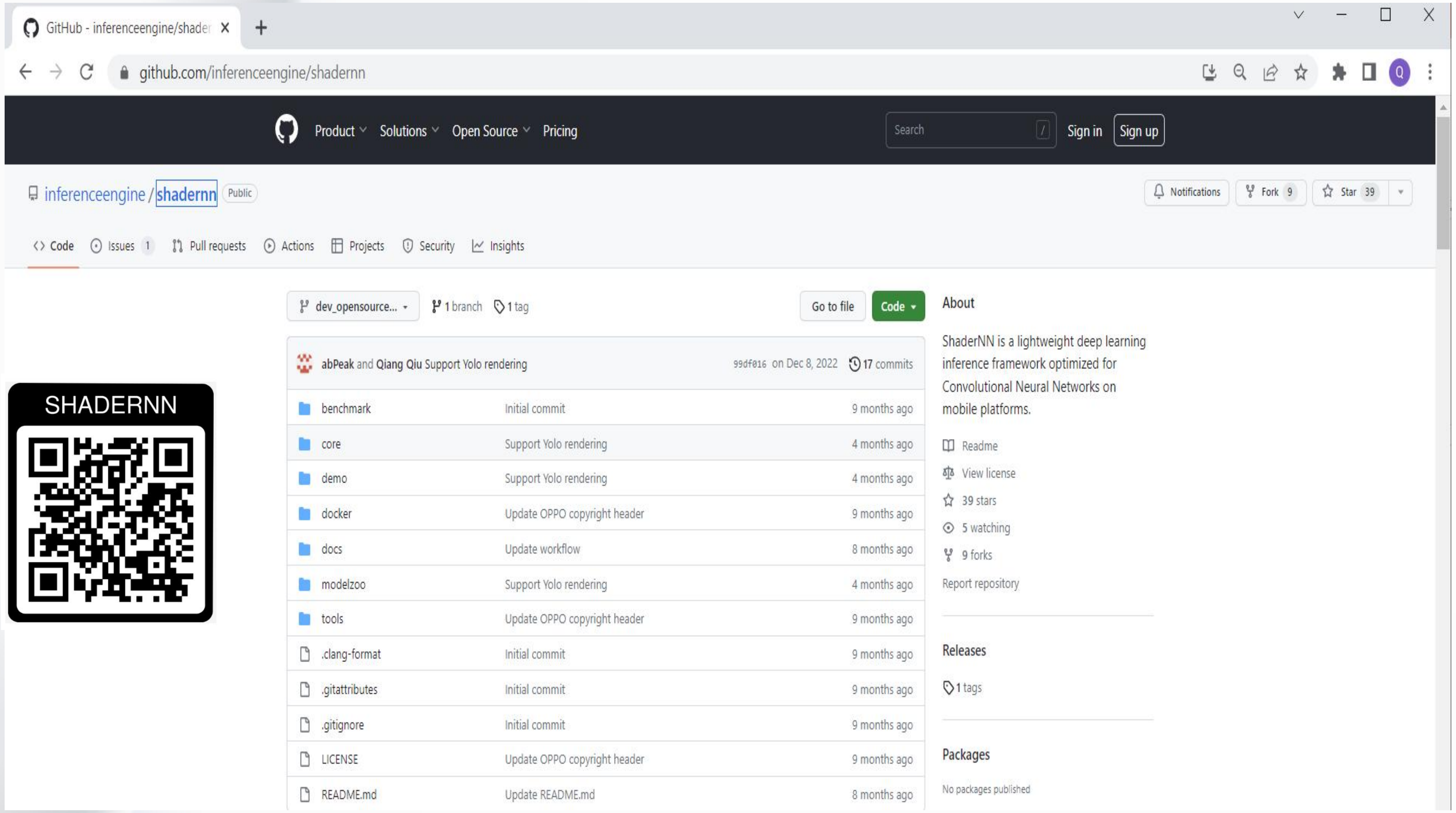
MOBA Game Input

MNSS (×2)

MNSS: Neural Supersampling Framework for Real-Time Rendering on Mobile Devices

by Zhejiang University and OPPO

ShaderNN OpenSource Community



SHADERNN

https://github.com/inferenceengine/shadernn (Apache2.0 License)

- Source Code
  - Standalone inference core that can be easily integrated
- Developer Guide
  - Getting started
  - How to create custom layer
  - How to implement model processor
  - How to load and run model
  - How to validate results
  - How to benchmark
- Tools
  - Tool to covert models from TensorFlow, PyTorch and ONNX
- Demo App
  - Provide Android demo app to show how to integrate ShaderNN
- Model Zoo
  - Provide common CNN models

ShaderNN OpenSource Roadmap

GOTC

2021.10 – 2022.6 ShaderNN Phase I	2022.7-2023.5 ShaderNN Phase II	2023.6-2023.12 ShaderNN Phase III
1. Support OpenGL Fragment Shader backend 2. Support OpenGL Compute Shader backend 3. Open source ShaderNN 1.0 with Apache 2.0 License 4. Demonstrate ShaderNN features at SIGGRAPH 2022	1. Support Vulkan Compute Shader backend 2. Support multiple inputs 3. Open source ShaderNN 2.0 preview release 4. Integrate into OPPO inference platform framework	1. Join LFAI & DATA Sandbox program 2. Demonstrate ShaderNN new features at SIGGRAPH 2023 3. Add new operator support 4. Add new model conversion support 5. Optimize convolution and matrix multiplication 6. Optimize scheduling that automatically selects backend 7. Engage more ShaderNN users

Future Work

- Companies that may be invited as maintainers for the open-source community
  - MediaTek
  - Qualcomm
  - Universities, such as Zhejiang University

- Key technical points for co-construction.
  - New operator and model support
  - ARM optimization
  - OpenGL and Vulkan backend optimization
  - AIGC applications

- Key product demo & implementations
  - Deep learning Super Sampling for mobile game

- Potential target users
  - Mobile GPU providers
  - Android AI app developers
  - University researchers

群聊: ShaderNN开发者交流群

该二维码7天内(5月29日前)有效，重新进入将更新

looking forward to work with you all

hongyu.sun@oppo.com
jingtao.zhang@oppo.com
pengzhouhu@oppo.com

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE