

# Universal Large-language Model Deployment with ML Compilation

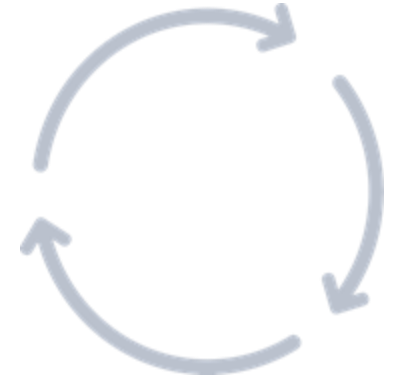
Tianqi Chen

# What are the Biggest Challenge in ML Engineering?

## ML modeling



## ML (Compiler) Engineering



ML engineering now becomes critical and go hand in hand with ML modeling  
It is not about build silver bullet once but **continuous improvement and innovations**

# High-level Thoughts

ML compiler RD can be domain specific, productive, and python first

Cross-level abstraction with first class dynamic shape

Enable universal deployment across cloud and edge

# High-level Thoughts

**ML compiler RD can be domain specific, productive, and python first**

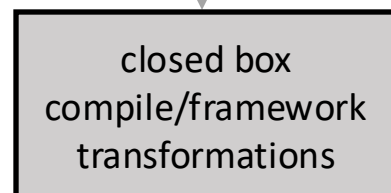
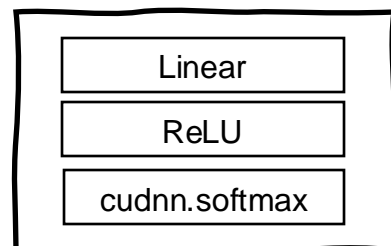
Cross-level abstraction with first class dynamic shape

Enable universal deployment across cloud and edge

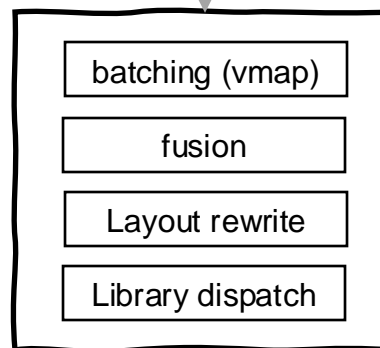
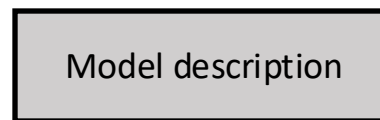
# Development Patterns

Normal development  
assuming a mature  
framework foundation

Customize function  
compositions for model

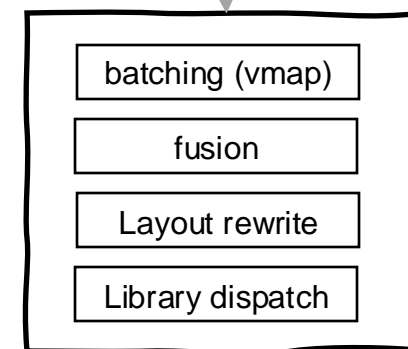
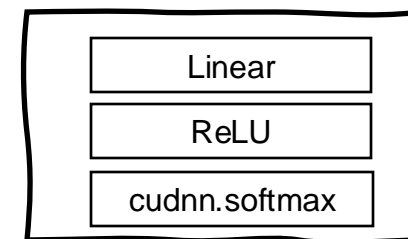


Normal compiler  
development



Customize program  
transformations, need  
to work for all models.  
Slow to change across  
multiple layers.

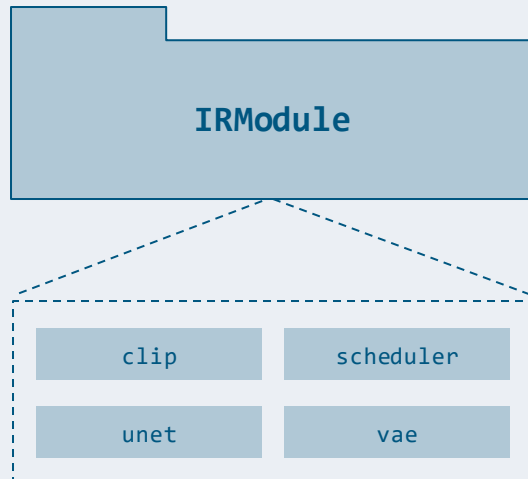
**Domain specific ML  
compilation pipeline  
development**



Customize both initial  
composition and transformation.  
Leverage domain specific  
pipeline if needed.

# IRModule in Python

Centers around one key construct



A collection of (tensor) functions that correspond to model components.

Accessible in python through TVMScript

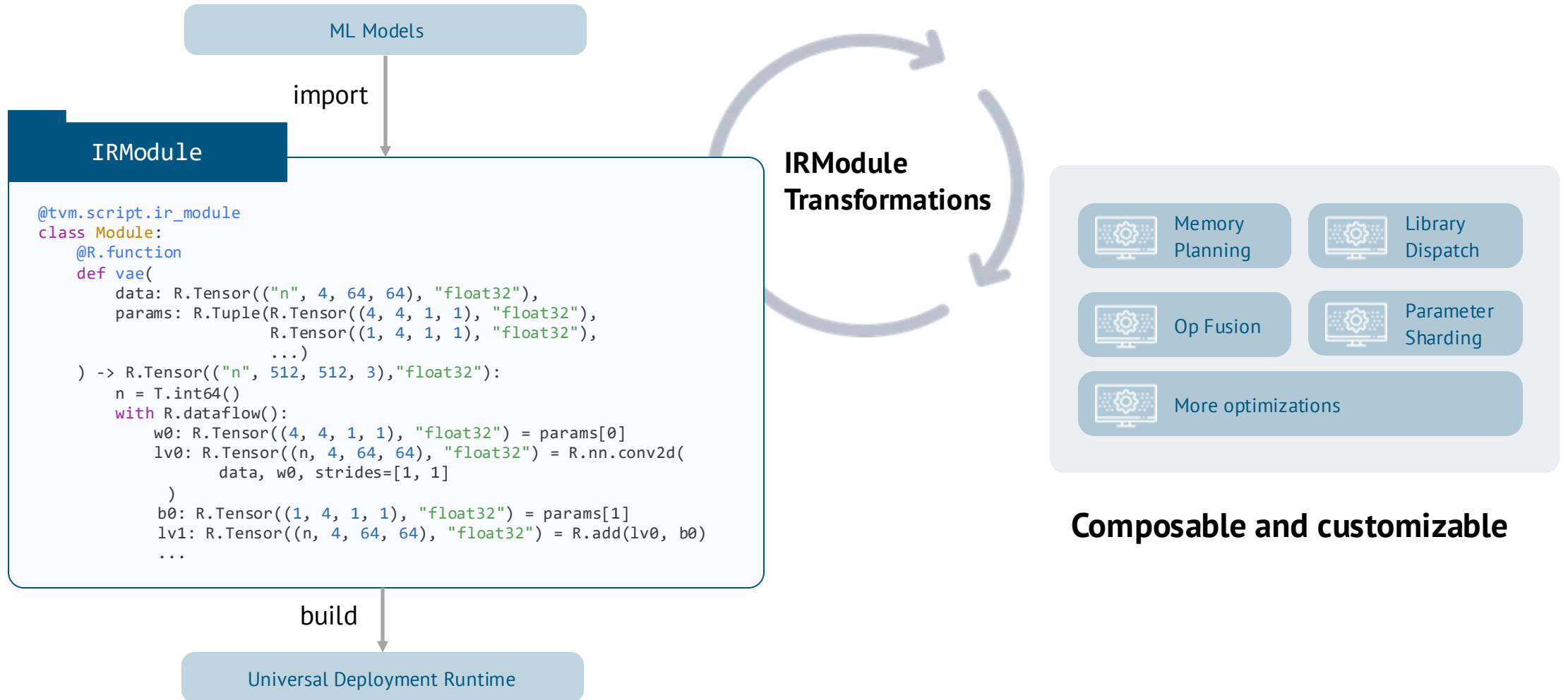
```
>>> mod.show()
```

```
import tvm.script
from tvm.script import tir as T, relax as R

@tvm.script.ir_module
class Module:
    @R.function
    def vae(
        data: R.Tensor(("n", 4, 64, 64), "float32"),
        params: R.Tuple(R.Tensor((4, 4, 1, 1), "float32"),
                        R.Tensor((1, 4, 1, 1), "float32"),
                        ...)
    ) -> R.Tensor(("n", 512, 512, 3), "float32"):
        n = T.int64()
        with R.dataflow():
            w0: R.Tensor((4, 4, 1, 1), "float32") = params[0]
            lv0: R.Tensor((n, 4, 64, 64), "float32") = R.nn.conv2d(
                data, w0, strides=[1, 1]
            )
            b0: R.Tensor((1, 4, 1, 1), "float32") = params[1]
            lv1: R.Tensor((n, 4, 64, 64), "float32") = R.add(lv0, b0)
            ...
```

Unifying abstractions by encapsulation computational graph, tensor program, library, hardware primitives, and their interactions in the same module

# Overall Approach



# Enabling Incremental Developments

## New model or backend

```
mod = frontend.from_fx(model)
mod = relax.get_pipeline()(mod)
```

- ✓ Part of the model accelerated
- ✓ Find room for improvements

## Composable customizations

Mix your own library and compilation

```
mod = DispatchToLibrary("attention")(mod)
mod = DefaultTIRLegalization(mod)
```

Try out new fusion patterns

```
mod = CustomizeFusion()(mod)
mod = transform.Sequential([
    transform.FuseOps(),
    transform.FuseTIR()
])(mod)
```

## Milestones and Feedbacks

- ✓ Feedback to out of box pipelines
- ✓ Full model accelerated and offloaded to target env
- ✓ Deploy ML compilation improvements to prod.



This is not a one shot game, but continuous process for every new model, backend features, new improvements in machine learning compilation.



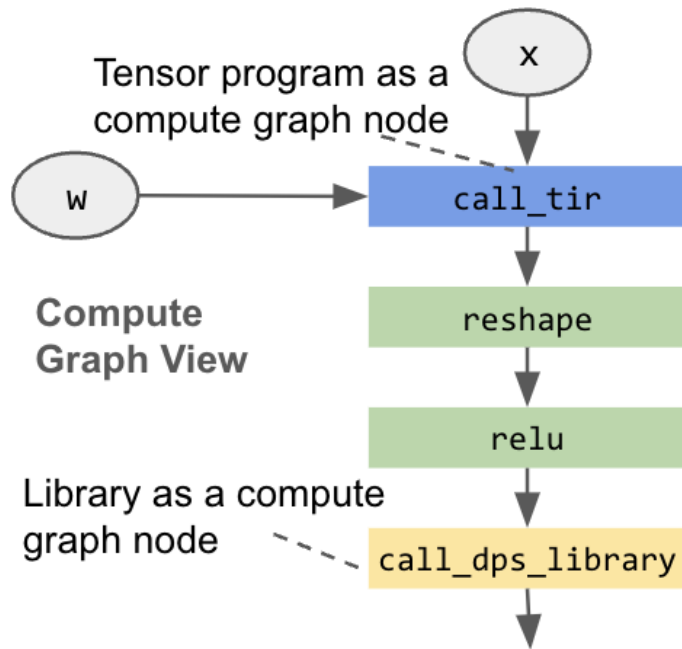
# High-level Thoughts

ML compiler RD can be domain specific, productive, and python first

**Cross-level abstraction with first class dynamic shape**

Enable universal deployment across cloud and edge

# Cross-level Abstraction



Code  
View

```
@tensorir_function
def mm(X: Buffer((1, 512) "f32"), W: Buffer((512, 1536), "f32"),
      Y: Buffer((1, 1536), "f32")):
    for i, j, k in grid(1, 1536, 512):
        with block():
            with init():
                Y[i, j] = 0
            Y[i, j] += X[i, k] * W[k, j]

def main(x: Tensor((1, 512), "f32"), w: Tensor((512, 1536), "f32")):
    with dataflow():
        lv0: Tensor((1, 1536), "f32") = call_tir(mm, [x, w], Tensor((1, 1536), "f32"))
        lv1: Tensor((1, 3, 8, 64), "f32") = reshape(lv0, shape(1, 3, 8, 64))
        lv2: Tensor((1, 8, 64), "f32") = relu(lv1)
        lv3: Tensor((1, 8, 64), "f32") = call_dps_library(
            "cutlass.attention", [lv2], Tensor((1, 8, 64), "f32")
        )
        ...
```

Tensor program

High-level  
computational  
graph

# Why Cross-level: Case Study of Quantized LLM

```
@tensorir_function
def decode_q4(
    Wdata: Buffer((128, 16) "u32"),
    Wscale: Buffer((128, 4), "f16"),
    W: Buffer((128, 128), "f16")
):
    for x, y in grid(128, 128):
        W[x, y] = (
            (data[x, y // 8] >> (y % 8 * 4)) & 15 - 7
        ) * scale[x, y // 32]

def main(
    x: Tensor((1, 128), "f16"),
    Wdata: Tensor((128, 16) "u32"),
    Wscale: Tensor((128, 4), "f16")
):
    with dataflow():
        W0: Tensor((128, 128), "f16") = call_tir(
            decode_q4, [Wdata, Wscale],
            Tensor((128, 128), "f16")
        )
        lv0: Tensor((128, 128), "f16") = matmul(x, W0)
    ...
```



```
@tensorir_function
def fused_decode_q4_mm(
    X: Buffer((1, 128) "f16"),
    Wdata: Buffer((128, 16) "u32"),
    Wscale: Buffer((128, 4), "f16"),
    Y: Buffer((1, 128), "f16")
):
    W = alloc_buffer((128, 128), "f16")
    for x, y in grid(128, 128):
        W[x, y] = (
            (data[x, y // 8] >> (y % 8 * 4)) & 15 - 7
        ) * scale[x, y // 32]

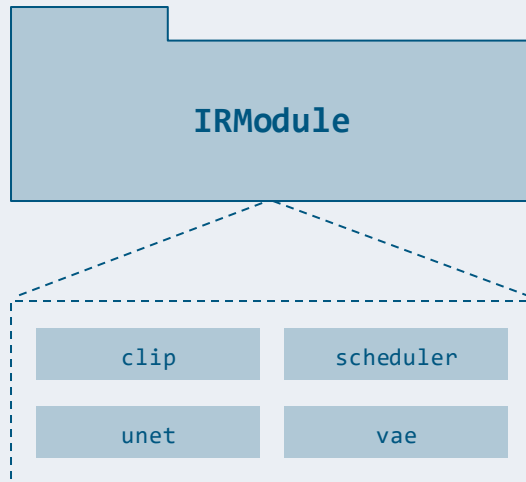
    for i, j, k in grid(1, 128, 128):
        with block():
            with init():
                Y[i, j] = 0
            Y[i, j] += X[i, k] * W[k, j]

def main(
    x: Tensor((1, 128), "f16"),
    Wdata: Tensor((128, 16) "u32"),
    Wscale: Tensor((128, 4), "f16")
):
    with dataflow():
        lv0: Tensor((1, 128), "f16") = call_tir(
            fused_decode_q4_mm, [x, Wdata, Wscale],
            Tensor((1, 128), "f16")
        )
    ...
```

Describe quantization customization in tensor program  
Still enables effective analysis and optimizations

# First class Symbolic Shape

Centers around one key construct



A collection of (tensor) functions that correspond to model components.

Accessible in python through TVMScript

```
>>> mod.show()
```

```
import tvm.script
from tvm.script import tir as T, relax as R

@tvm.script.ir_module
class Module:
    @R.function
    def vae(
        data: R.Tensor(("n", 4, 64, 64), "float32"),
        params: R.Tuple(R.Tensor((4, 4, 1, 1), "float32"),
                        R.Tensor((1, 4, 1, 1), "float32"),
                        ...)
    ) -> R.Tensor(("n", 512, 512, 3), "float32"):
        n = T.int64()
        with R.dataflow():
            w0: R.Tensor((4, 4, 1, 1), "float32") = params[0]
            lv0: R.Tensor((n, 4, 64, 64), "float32") = R.nn.conv2d(
                data, w0, strides=[1, 1]
            )
            b0: R.Tensor((1, 4, 1, 1), "float32") = params[1]
            lv1: R.Tensor((n, 4, 64, 64), "float32") = R.add(lv0, b0)
            ...
```

First-class symbolic shape support to enable dynamic shape compilation.

# Symbolic Shape vs Any Shape

## Symbolic Shape

```
@R.function
def symbolic_shape_fn(x: R.Tensor(("n", 2, 2), "float32")):
    n, m = T.int64(), T.int64()
    with R.dataflow():
        lv0: R.Tensor((n, 4), "float32") = R.reshape(x, R.shape(n, 4))
        lv1: R.Tensor((n * 4, ), "float32") = R.flatten(lv0)
        lv2: R.Tensor(ndim=1, dtype="float32") = R.unique(lv1)
        lv3 = R.match_cast(lv2, R.Tensor((m, ), "float32"))
        gv0: R.Tensor((m, ), "float32") = R.exp(lv3)
        R.output(gv0)
    return gv0
```

- Tracks the shape values (n, n \* 4)
- More optimizations
- Flexible fallback for unknown and rematch
- Shape is part of computation

## Any Shape Dimension

```
@R.function
def any_shape_fn(x: R.Tensor((?, 2, 2), "float32")):
    n = R.get_shape_value(x, axis=0)
    with R.dataflow():
        lv0: R.Tensor((?, 4), "float32") = R.reshape(x, R.shape(n, 4))
        lv1: R.Tensor((?, ), "float32") = R.flatten(lv0)
        lv2: R.Tensor(?, "float32") = R.unique(lv1)

        gv0: R.Tensor((?, ), "float32") = R.exp(lv2)
        R.output(gv0)
    return gv0
```

- Most approaches so far
- ? denotes any shape value
- No relation information: cannot prove shape equivalence by only looking at any dimensions

# Symbolic Shape Notable Elements

Beyond initial checking: need to preserve relations during transformations

Enable AOT: global tracking, as we fold and expand sub functions

```
Callable([Tensor(("n", 4), "f32")], Tensor(("n * 4", ), "f32"))
```

Go across-level: symbolic shape constraints in graph informs shape constraints in Tensor Program and vice versa

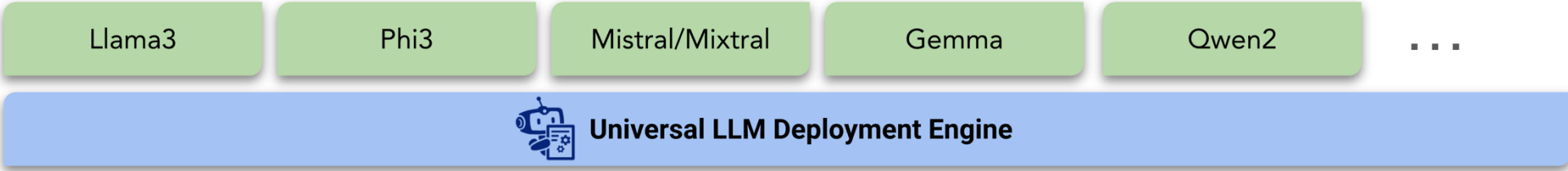
# High-level Thoughts

ML compiler RD can be domain specific, productive, and python first

Cross-level abstraction with first class dynamic shape

**Enable universal deployment across cloud and edge**

# MLCEngine: Universal LLM Deployment



## Cloud Server



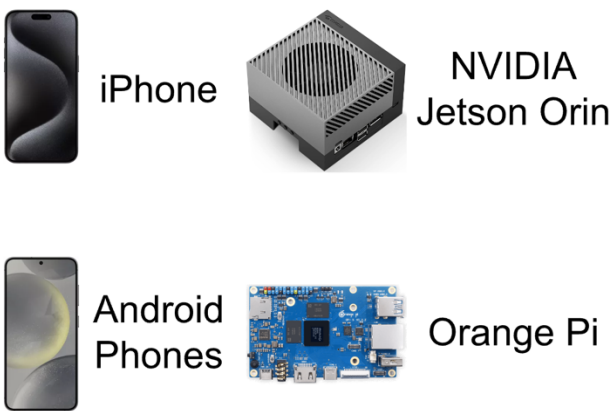
## Desktop / Laptop



## Tablet



## Edge / Robot



Server

Local

Web Browsers





# MLCEngine: Windows Linux Mac

```
>> mlc_llm chat HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC
```

Running across  
platforms

```
~ > mlc_llm chat HF://mlc-ai/Llama-3-8B-Instruct-q0f16-MLC python311 ruihang@catalyst-nv8180 07:11:29 PM
[2024-06-05 19:11:32] INFO auto_device.py:79: Found device: cuda:0
[2024-06-05 19:11:32] INFO auto_device.py:79: Found device: cuda:1
[2024-06-05 19:11:33] INFO auto_device.py:88: Not found device: rocm:0
[2024-06-05 19:11:33] INFO auto_device.py:88: Not found device: metal:0
[2024-06-05 19:11:36] INFO auto_device.py:79: Found device: vulkan:0
[2024-06-05 19:11:36] INFO auto_device.py:79: Found device: vulkan:1
[2024-06-05 19:11:36] INFO auto_device.py:79: Found device: vulkan:2
[2024-06-05 19:11:38] INFO auto_device.py:79: Found device: opencl:0
[2024-06-05 19:11:38] INFO auto_device.py:79: Found device: opencl:1
[2024-06-05 19:11:38] INFO auto_device.py:35: Using device: cuda:0
[2024-06-05 19:11:38] INFO download_cache.py:227: Downloading model from HuggingFace: HF://mlc-ai/Llama-3-8B-Instruct-q0f16-MLC
[2024-06-05 19:11:38] INFO download_cache.py:29: MLC_DOWNLOAD_CACHE_POLICY = ON. Can be one of: ON, OFF, REDO, READONLY
[2024-06-05 19:11:38] INFO download_cache.py:166: Weights already downloaded: /home/ruihang/.cache/mlc_llm/model_weights/hf/mlc-ai/Llama-3-8B-Instruct-q0f16-MLC
[2024-06-05 19:11:38] INFO jit.py:43: MLC_JIT_POLICY = ON. Can be one of: ON, OFF, REDO, READONLY
[2024-06-05 19:11:38] INFO jit.py:160: Using cached model lib: /home/ruihang/.cache/mlc_llm/model_lib/6e419f362d3e259bf9976f54fa481a33.so
[19:11:44] /home/ruihang/Workspace/mlc-llm/cpp/serve/engine.cc:47: Warning: Tokenizer info not found in mlc-chat-config.json. Trying to automatically detect the tokenizer info
You can use the following special commands:
/help          print the special commands
/exit         quit the cli
/stats        print out stats of last request (token/sec)
/metrics      print out full engine metrics
/reset        restart a fresh chat
/set [overrides] override settings in the generation config. For example,
              '/set temperature=0.5;top_p=0.8;seed=23;max_tokens=100;stop=str1,str2'
              Note: Separate stop words in the 'stop' option with commas (,).
Multi-line input: Use escape+enter to start a new line.

>>> Give me a one-day trip plan to Pittsburgh.
Pittsburgh! The 'Burgh is a fantastic city with a rich history, stunning views, and a vibrant cultural scene. Here's a one-day trip plan to
```

# MLCEngine: OpenAI-Compatible Server

```
>> mlc_llm serve HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC
```

Full OpenAI support

```
~ > mlc_llm serve HF://mlc-ai/Llama-3-8B-Instruct-q0f16-MLC --mode server
[2024-06-05 17:37:01] INFO auto_device.py:79: Found device: cuda:0
[2024-06-05 17:37:01] INFO auto_device.py:79: Found device: cuda:1
[2024-06-05 17:37:02] INFO auto_device.py:88: Not found device: rocm:0
[2024-06-05 17:37:02] INFO auto_device.py:88: Not found device: metal:0
[2024-06-05 17:37:05] INFO auto_device.py:79: Found device: vulkan:0
[2024-06-05 17:37:05] INFO auto_device.py:79: Found device: vulkan:1
[2024-06-05 17:37:05] INFO auto_device.py:79: Found device: vulkan:2
[2024-06-05 17:37:07] INFO auto_device.py:79: Found device: opengl:0
[2024-06-05 17:37:07] INFO auto_device.py:79: Found device: opengl:1
[2024-06-05 17:37:07] INFO auto_device.py:35: Using device: cuda:0
[2024-06-05 17:37:07] INFO download_cache.py:227: Downloading model from HuggingFace: HF://mlc-ai/Llama-3-8B-Instruct-q0f16-MLC
[2024-06-05 17:37:07] INFO download_cache.py:29: MLC_DOWNLOAD_CACHE_POLICY = ON. Can be one of: ON, OFF, REDO, READONLY
[2024-06-05 17:37:07] INFO download_cache.py:166: Weights already downloaded: /home/ruihang/.cache/mlc_llm/model_weights/hf/mlc-ai/Llama-3-8B-Instruct-q0f16-MLC
[2024-06-05 17:37:07] INFO jit.py:43: MLC_JIT_POLICY = ON. Can be one of: ON, OFF, REDO, READONLY
[2024-06-05 17:37:07] INFO jit.py:160: Using cached model lib: /home/ruihang/.cache/mlc_llm/model_lib/6e419f362d3e259bf9976f54fa481a33.so
[2024-06-05 17:37:07] INFO engine_base.py:180: The selected engine mode is server. We use as much GPU memory as possible (within the limit of gpu_memory_utilization).
[2024-06-05 17:37:07] INFO engine_base.py:188: If you have low concurrent requests and want to use less GPU memory, please select mode "local".
[2024-06-05 17:37:07] INFO engine_base.py:193: If you don't have concurrent requests and only use the engine interactively, please select mode "interactive".
[17:37:08] /home/ruihang/Workspace/mlc-llm/cpp/serve/config.cc:649: Under mode "local", max batch size will be set to 4, max KV cache token capacity will be set to 8192, prefill chunk size will be set to 1024.
[17:37:08] /home/ruihang/Workspace/mlc-llm/cpp/serve/config.cc:649: Under mode "interactive", max batch size will be set to 1, max KV cache token capacity will be set to 8192, prefill chunk size will be set to 1024.
[17:37:08] /home/ruihang/Workspace/mlc-llm/cpp/serve/config.cc:649: Under mode "server", max batch size will be set to 80, max KV cache token capacity will be set to 37604, prefill chunk size will be set to 1024.
[17:37:08] /home/ruihang/Workspace/mlc-llm/cpp/serve/config.cc:729: The actual engine mode is "server". So max batch size is 80, max KV cache token capacity is 37604, prefill chunk size is 1024.
[17:37:08] /home/ruihang/Workspace/mlc-llm/cpp/serve/config.cc:734: Estimated total single GPU memory usage: 20571.734 MB (Parameters: 15316.508 MB, KVCache: 4768.809 MB, Temporary buffer: 486.416 MB). The actual usage might be slightly larger than the estimated number.
[17:37:13] /home/ruihang/Workspace/mlc-llm/cpp/serve/engine.cc:47: Warning: Tokenizer info not found in mlc-chat-config.json. Trying to automatically detect the tokenizer info
INFO: Started server process [1580523]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

~ > curl -X POST \
-H "Content-Type: application/json" \
-d '{
  "model": "Llama-3-8B-Instruct-q0f16-MLC",
  "messages": [
    {"role": "user", "content": "Hello! This is project MLC LLM."},
    {"role": "assistant", "content": "Hello! It is great to work with you on project MLC LLM."},
    {"role": "user", "content": "Do you remember our project name?"}
  ]
}' \
http://127.0.0.1:8000/v1/chat/completions
```

# iOS SDK

OpenAI-style swift API

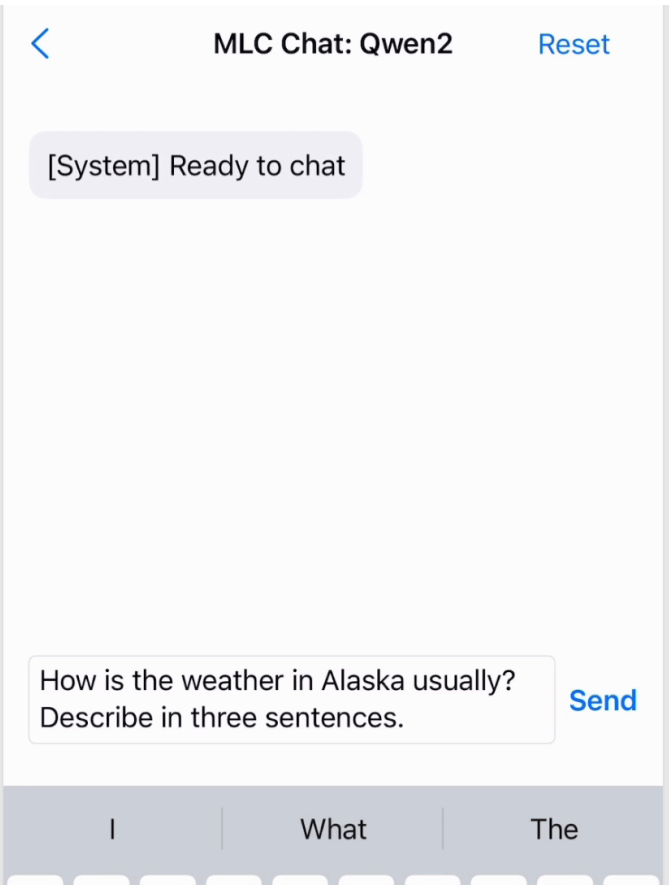
Demo on AppStore

Search for MLC Chat

```
func requestGenerate(prompt: String) {
    appendMessage(role: .user, message: prompt)
    appendMessage(role: .assistant, message: "")

    Task {
        self.historyMessages.append(
            ChatCompletionMessage(role: .user, content: prompt)
        )

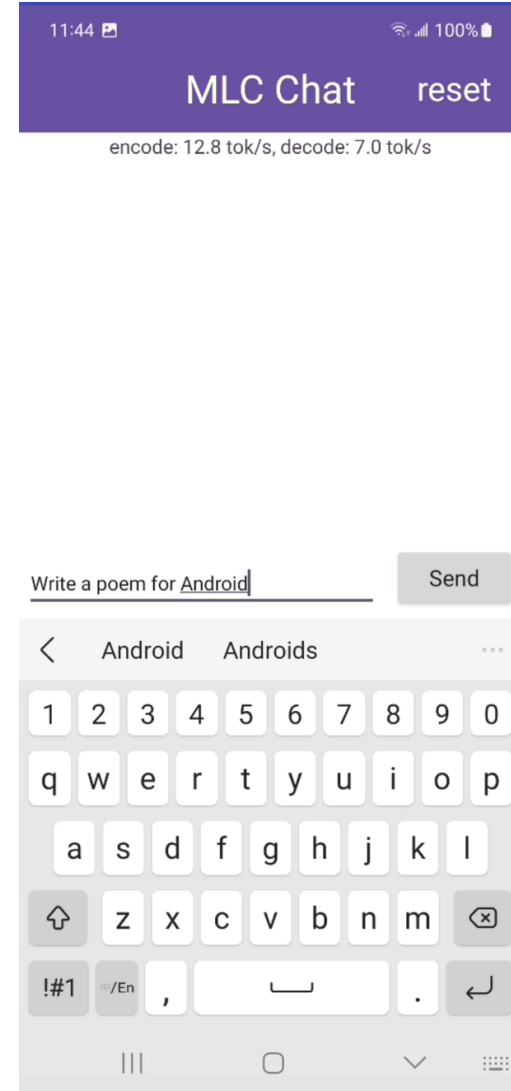
        var finishReasonLength = false
        for await res in await engine.chat.completions.create(
            messages: self.historyMessages,
            stream_options: StreamOptions(include_usage: true)
        ) {
            for choice in res.choices {
                if let content = choice.delta.content {
                    self.streamingText += content.asText()
                }
                if let finish_reason = choice.finish_reason {
                    if finish_reason == "length" {
                        finishReasonLength = true
                    }
                }
            }
        }
    }
}
```



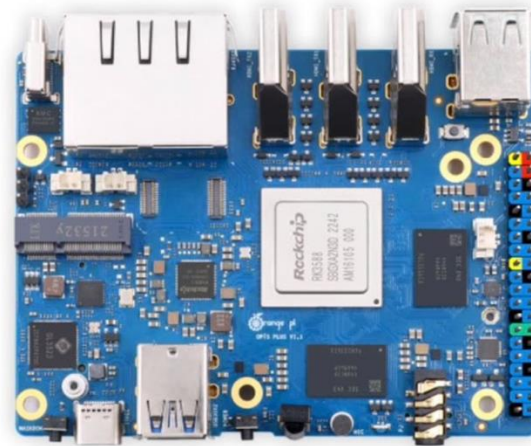
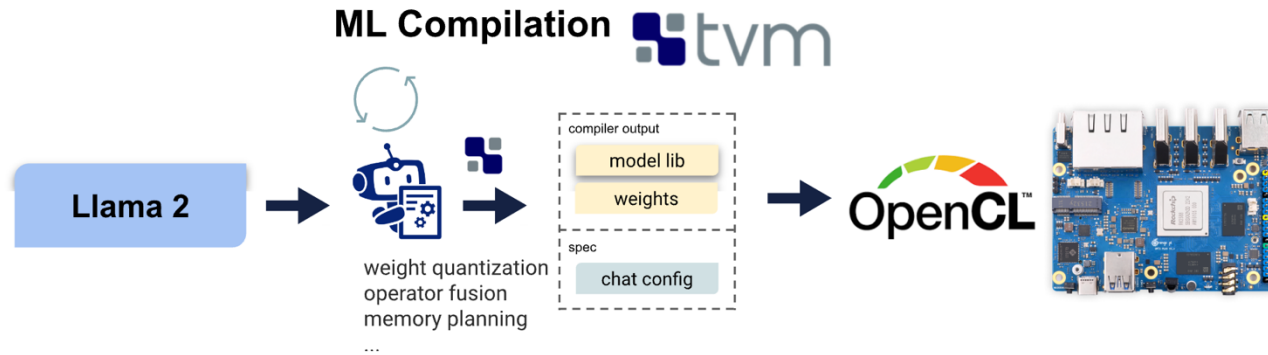
# MLC LLM: Android

Snapdragon Gen2

Enables larger models than iPhone



# Bringing LLMs to 100\$ Orange Pi



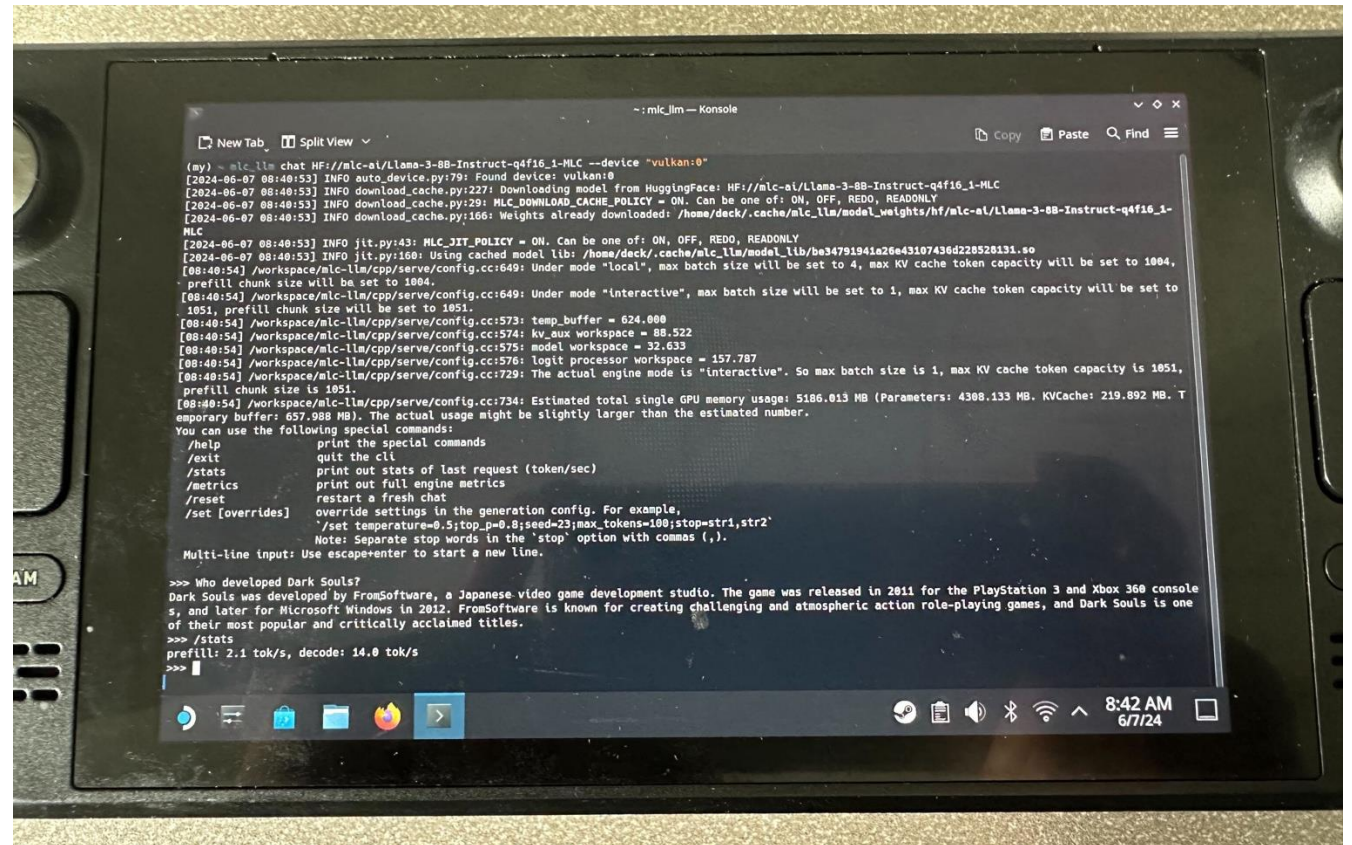
```
chris@chris-rk3588: ~/Documents/mlc_chat_cli
(GPT) chris@chris-rk3588:~/Documents/mlc_chat_cli$ mlc_chat_cli --local-id mlc-chat-Llama-2-7b-chat-hf-q4f16_1
Use MLC config: "/home/chris/Documents/mlc_chat_cli/dist/prebuilt/mlc-chat-Llama-2-7b-chat-hf-q4f16_1/mlc-chat-config.json"
Use model weights: "/home/chris/Documents/mlc_chat_cli/dist/prebuilt/mlc-chat-Llama-2-7b-chat-hf-q4f16_1/ndarray-cache.json"
Use model library: "/home/chris/Documents/mlc_chat_cli/dist/prebuilt/lib/Llama-2-7b-chat-hf-q4f16_1-opencl.so"
You can use the following special commands:
/help          print the special commands
/exit         quit the cli
/stats        print out the latest stats (token/sec)
/reset        restart a fresh chat
/reload [local_id] reload model 'local_id' from disk, or reload the current model if 'local_id' is not specified

Loading model...
arm_release_ver: g13p0-01eac0, rk_so_ver: 3
arm_release_ver of this libmali is 'g6p0-01eac0', rk_so_ver is '7'.
Loading finished
Running system prompts...
System prompts finished
[INST]: write a three line poem about llama
[/INST]: Of course, I'd be happy to help! Here's a three-line poem about llamas:
Fluffy and gentle, with eyes so bright,
Llamas roam the Andes, with grace in sight,
Their woolly coats shine, in the sun's warm light.
[INST]: /stats
prefill: 4.9 tok/s, decode: 2.6 tok/s
[INST]:
```

# LLM on SteamDeck

Leverages vulkan backend

Out of box support



# Efficient Structured Generation

Built-in support

Near zero overhead

Important for agent  
use cases

```
In [6]: class Country(pydantic.BaseModel):
...:     name: str
...:     capital: str
...:

In [7]: class Countries(pydantic.BaseModel):
...:     country: List[Country]
...:

In [8]: prompt = "Randomly list three countries and their capitals in JSON."

In [9]: schema = json.dumps(Countries.model_json_schema())

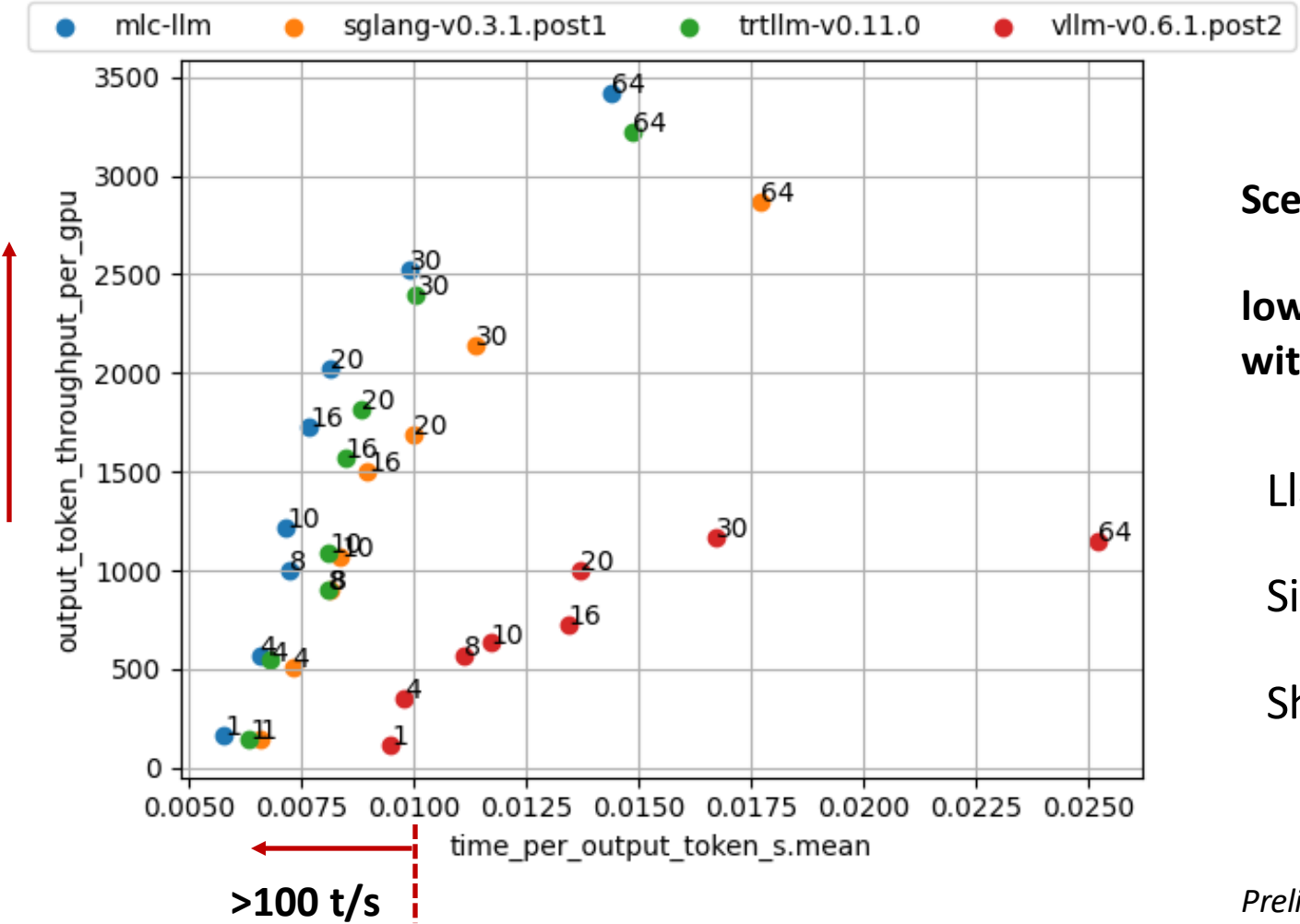
In [10]: response = engine.chat.completions.create(
...:     messages=[{"role": "user", "content": prompt}],
...:     response_format={"type": "json_object", "schema": schema},
...: )

In [11]: print(response.choices[0].message.content)
{"country": [{"name": "Japan", "capital": "Tokyo"}, {"name": "Brazil", "capital": "Brasilia"}, {"name": "India", "capital": "New Delhi"}]}

In [12]: |
```

Try it out via WebLLM: <https://huggingface.co/spaces/mlc-ai/WebLLM-JSON-Playground>

# MLCEngine Low-latency Server Performance



Scenario of interest:

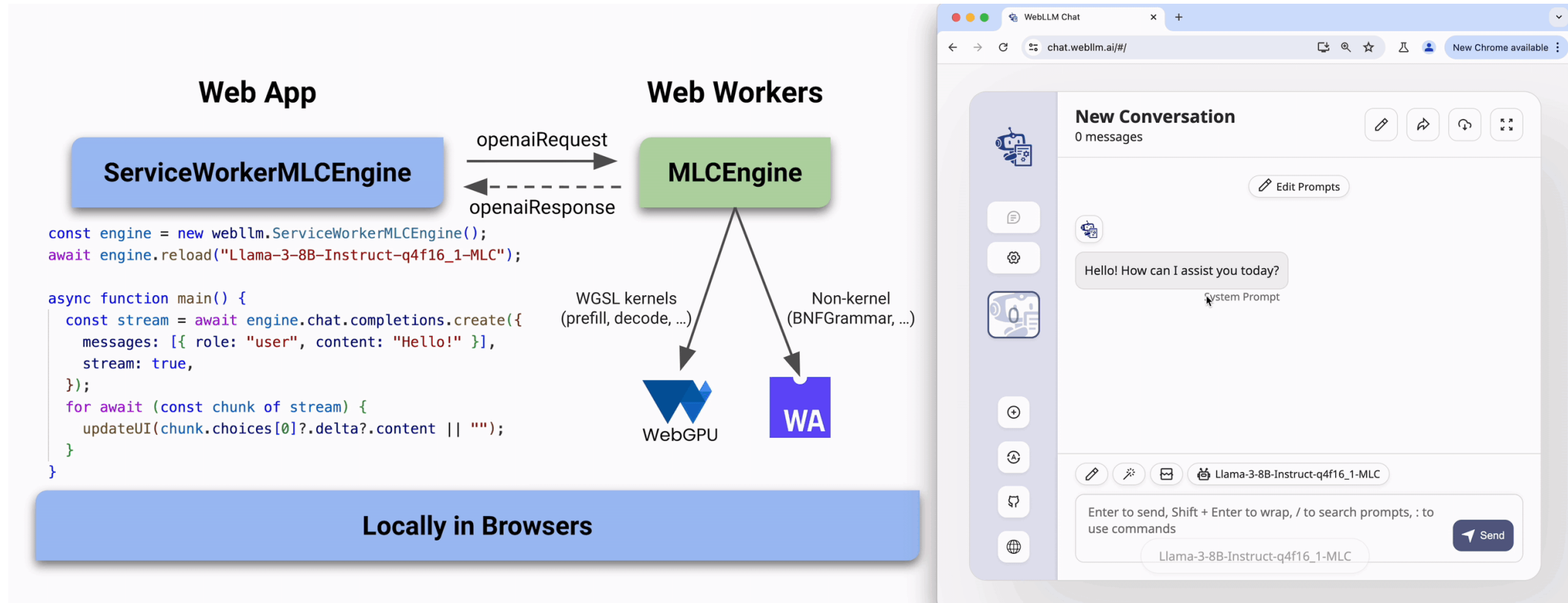
low-latency (fast output token per sec for user)  
with decent throughput(lower cost)

- Llama3.1 8B
- Single H100
- SharedGPT

Preliminary result: his is a rapid developing space



# WebLLM



Runs directly in browser client <https://webllm.mlc.ai/>

# Ongoing directions

More modality

More optimizations and connections with existing ecosystem

Thank you

MLC course: [mlc.ai](https://mlc.ai)

MLC-LLM: [llm.mlc.ai](https://llm.mlc.ai)

