Universal Large-language Model Deployment with ML Compilation

Tianqi Chen

Carnegie Mellon University
School of Computer Science

catalyst

OctoAI

tvm

What are the Biggest Challenge in ML Engineering?

ML modeling

- Language Models
- Diffusion
- MultiQuery Attention
- RoPE

ML (Compiler) Engineering

- Memory Planning
- Library Dispatch
- Sparse Weights
- Paged Attention
- Op Fusion
- Parameter Sharding
- Quantized Kernels
- Layout Optimization

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

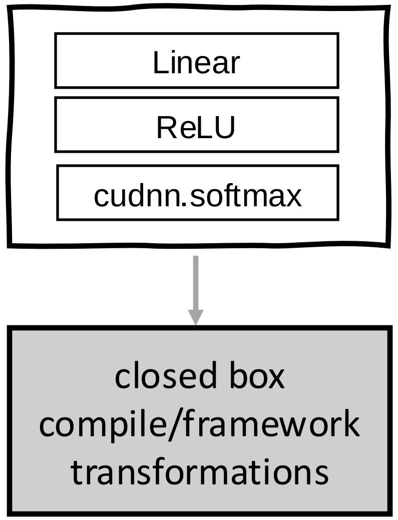
Cross-level abstraction with first class dynamic shape

Enable universal deployment across cloud and edge

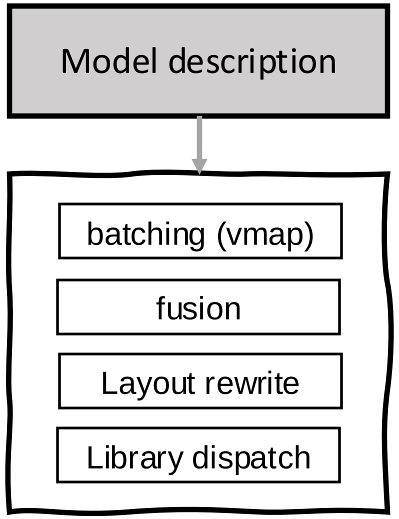
Development Patterns

Normal development assuming a mature framework foundation

Customize function compositions for model

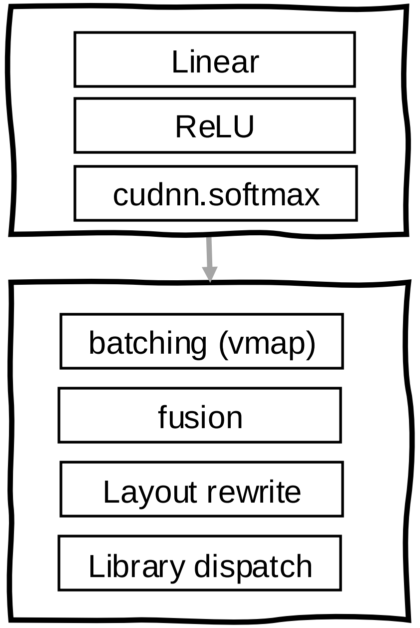


Normal compiler development



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

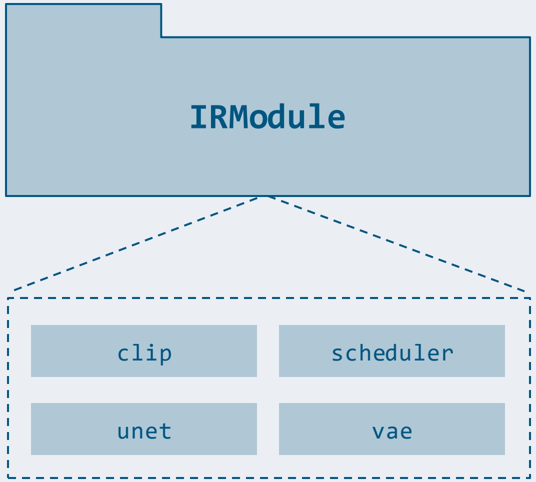
Domain specific ML compilation pipeline development



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.

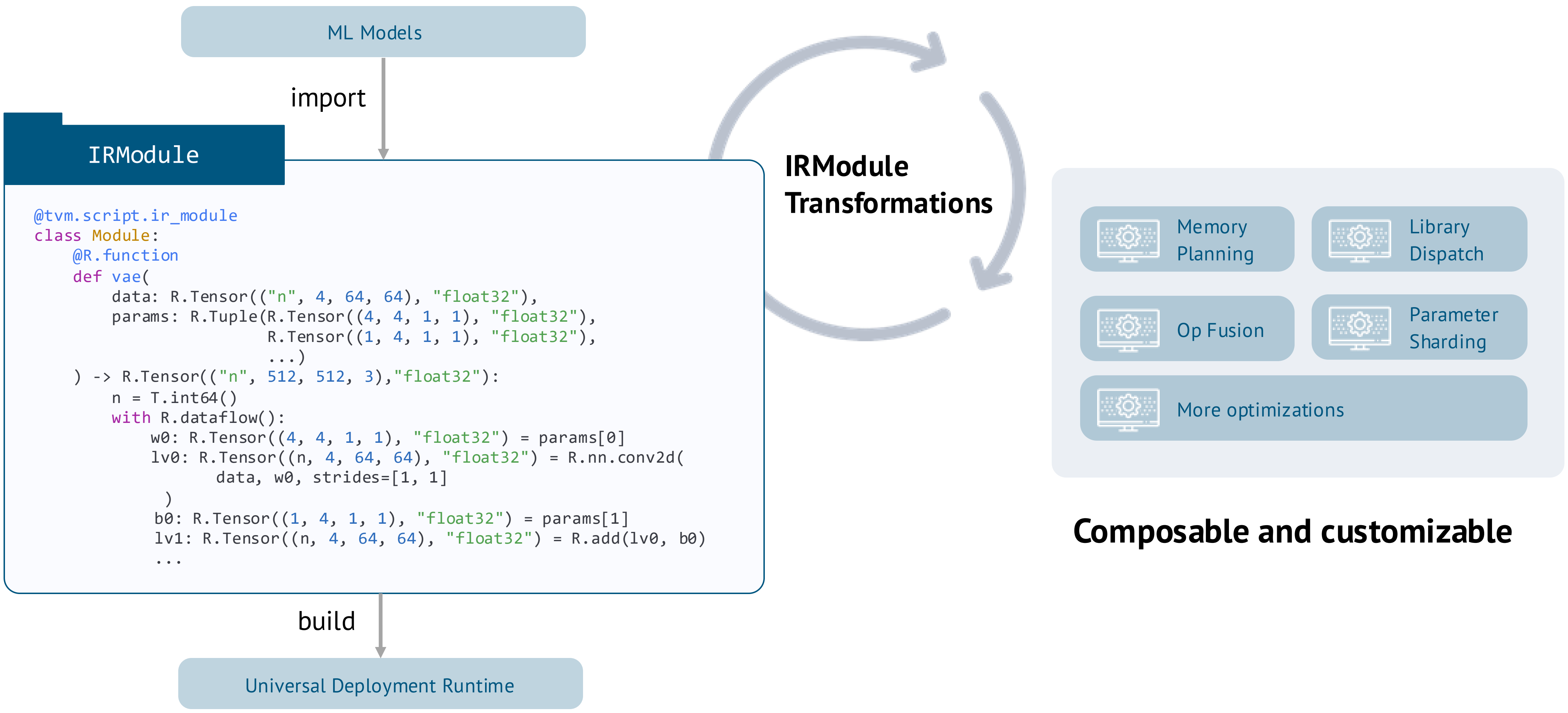
Accessible in python through TVMScript

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

```python
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




```python
@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)
            ...
```

Composable and customizable

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 = DispatchToLibary("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.

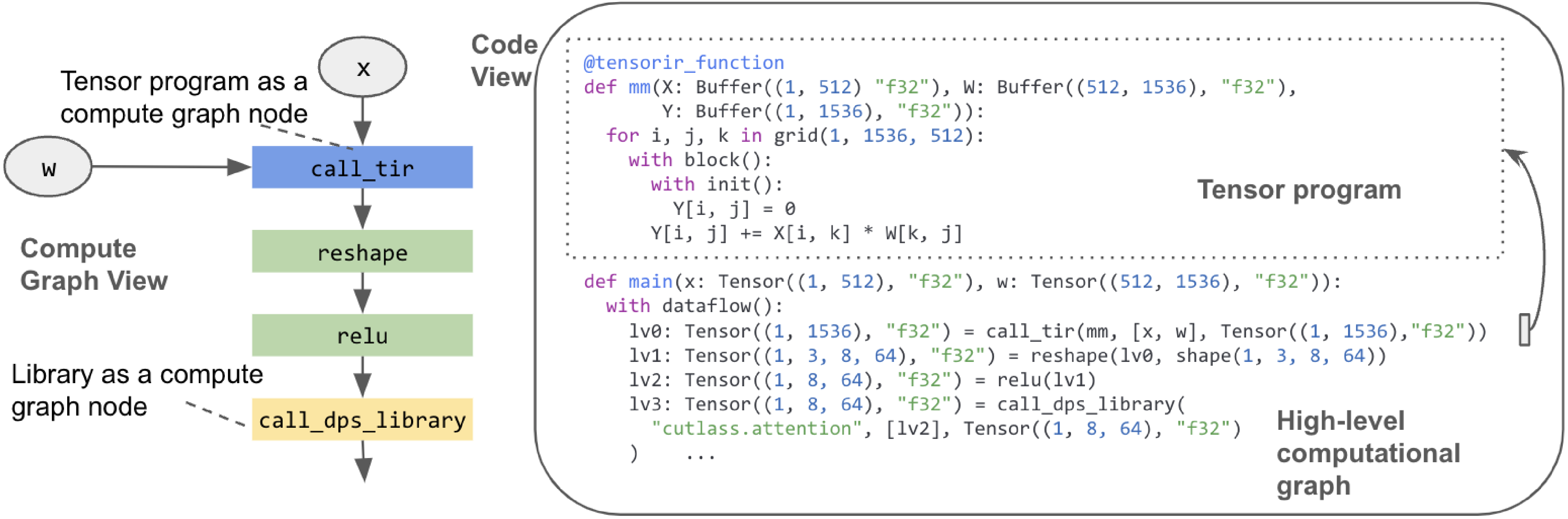
High-level Thoughts

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



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)
    ...
```

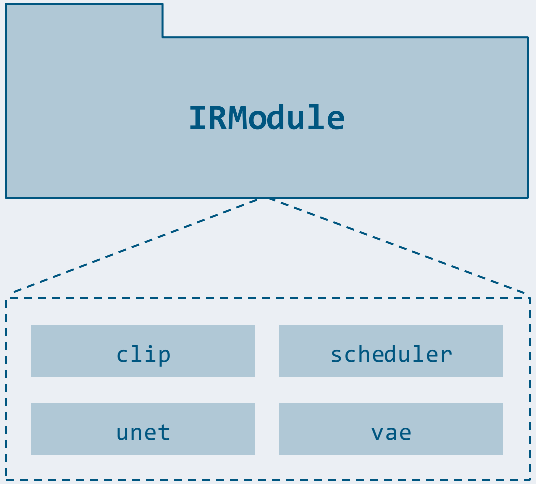
Describe quantization customization in tensor program

Still enables effective analysis and optimizations

```
@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")
    )
    ...
```

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()
```

```python
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(lv3)
        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

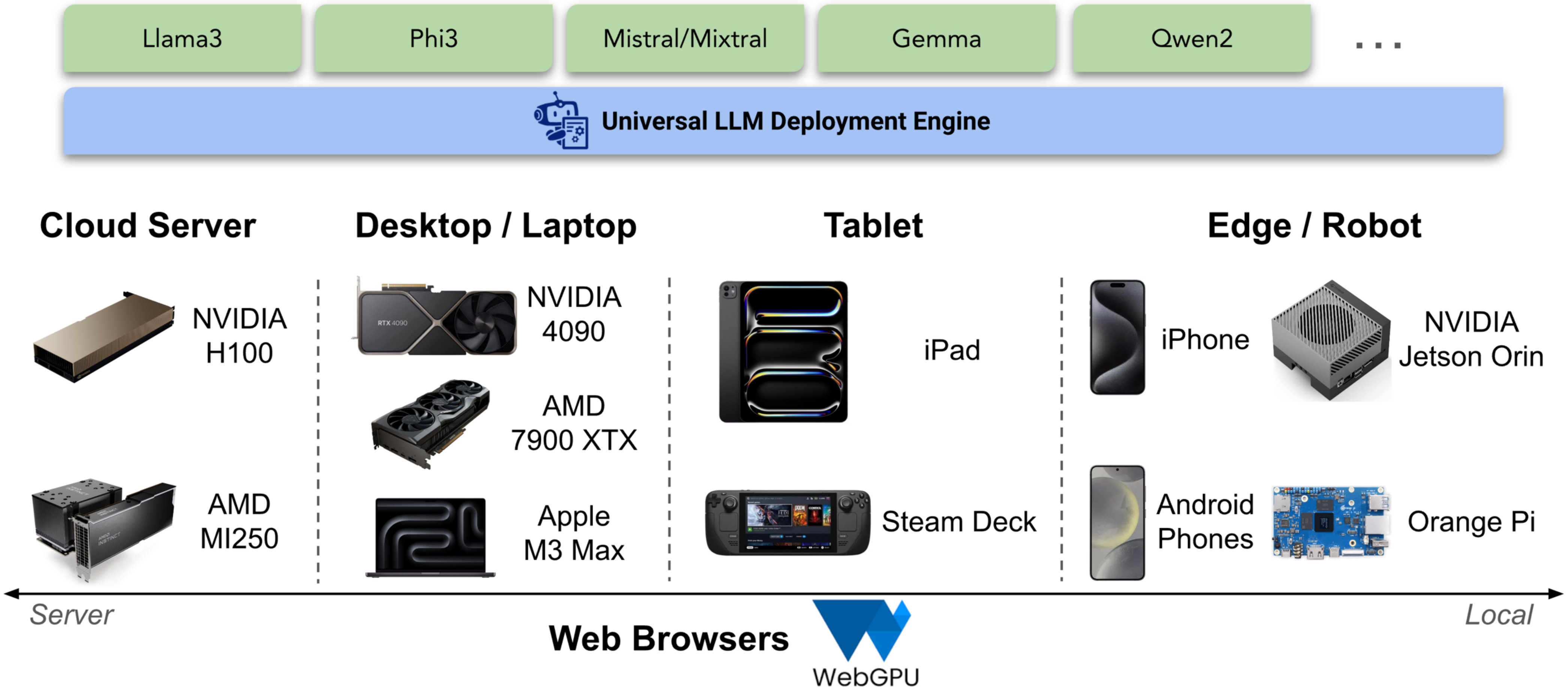
High-level Thoughts

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

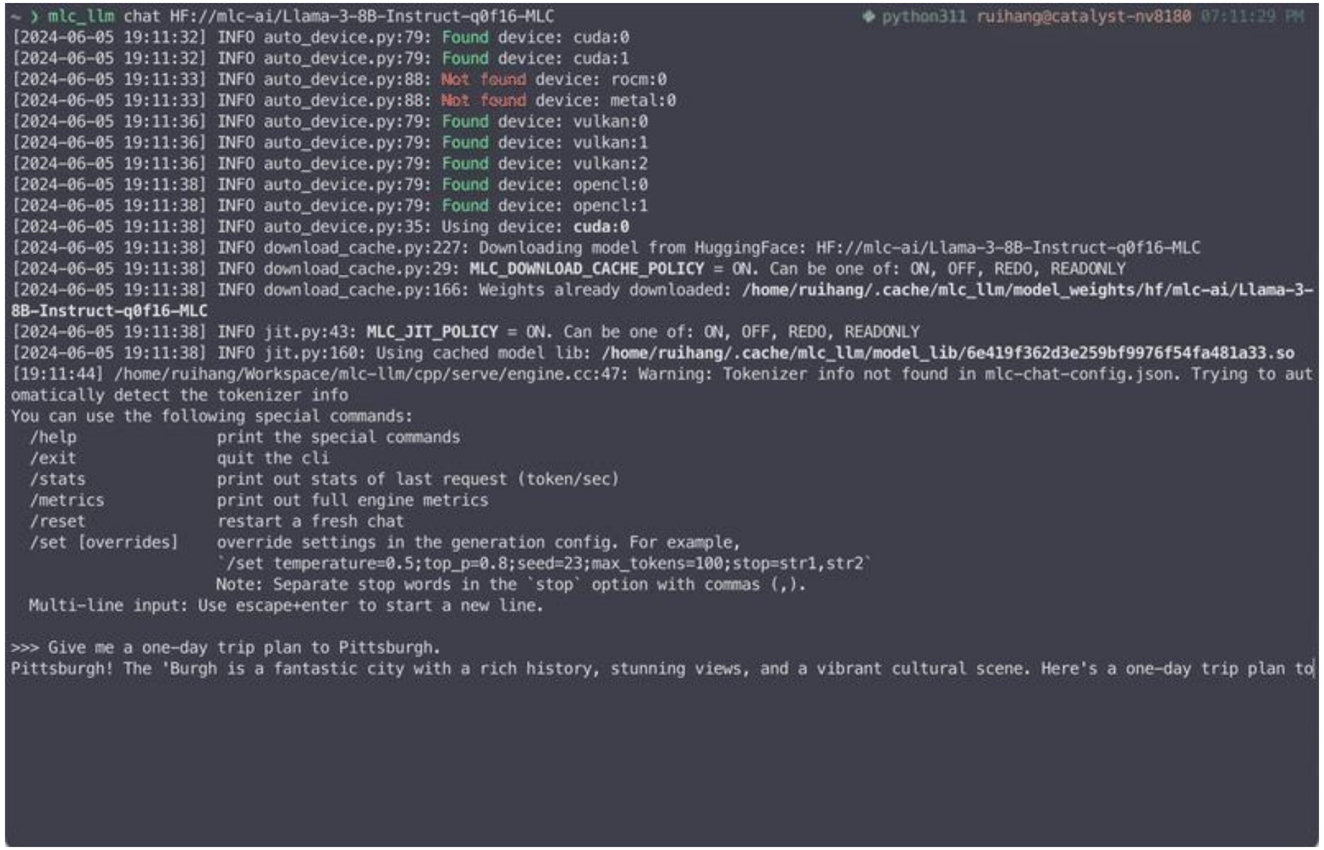
MLCEngine: Universal LLM Deployment



MLCEngine: Windows Linux Mac

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

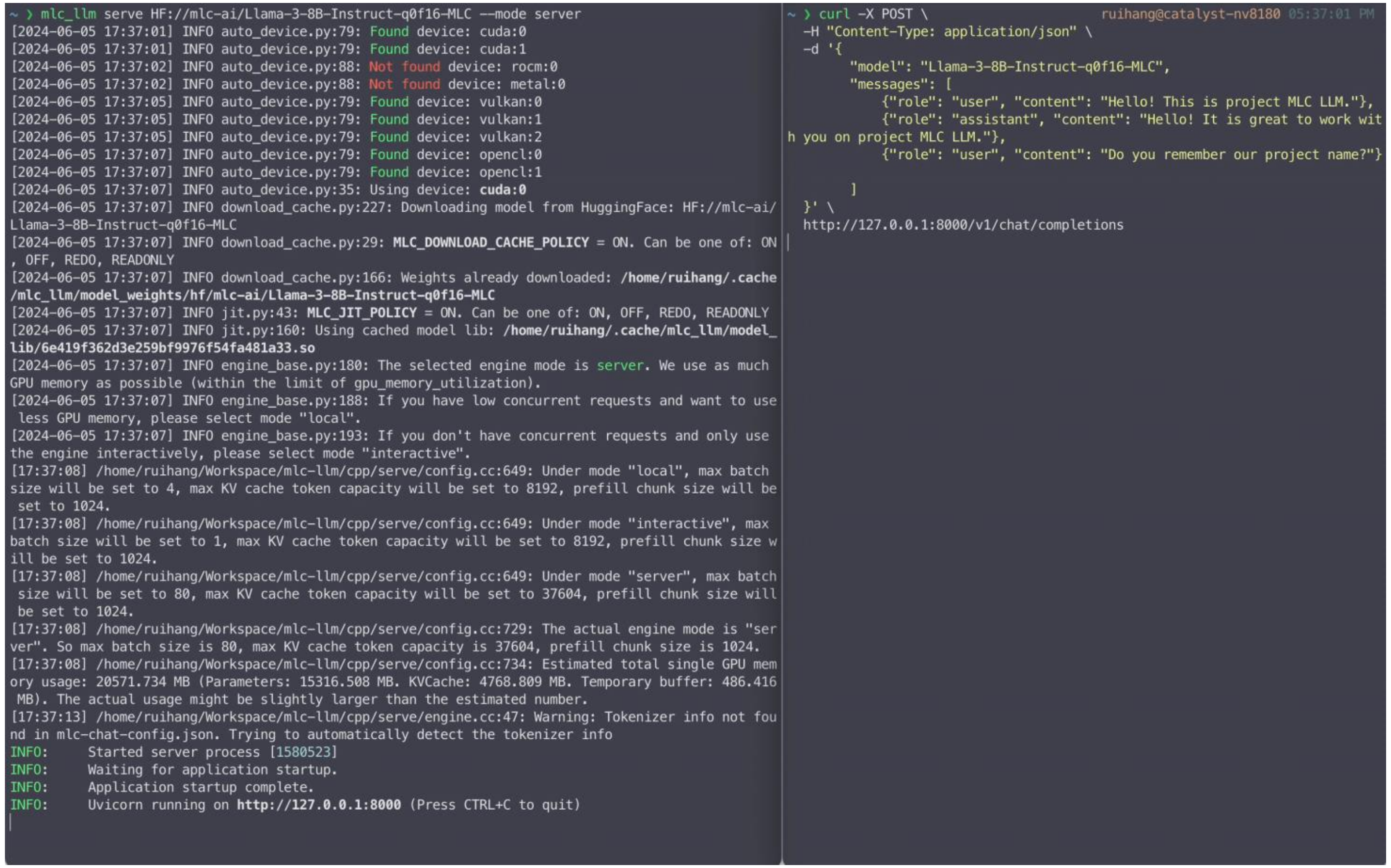
Running across platforms



MLCEngine: OpenAI-Compatible Server

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

Full OpenAI support



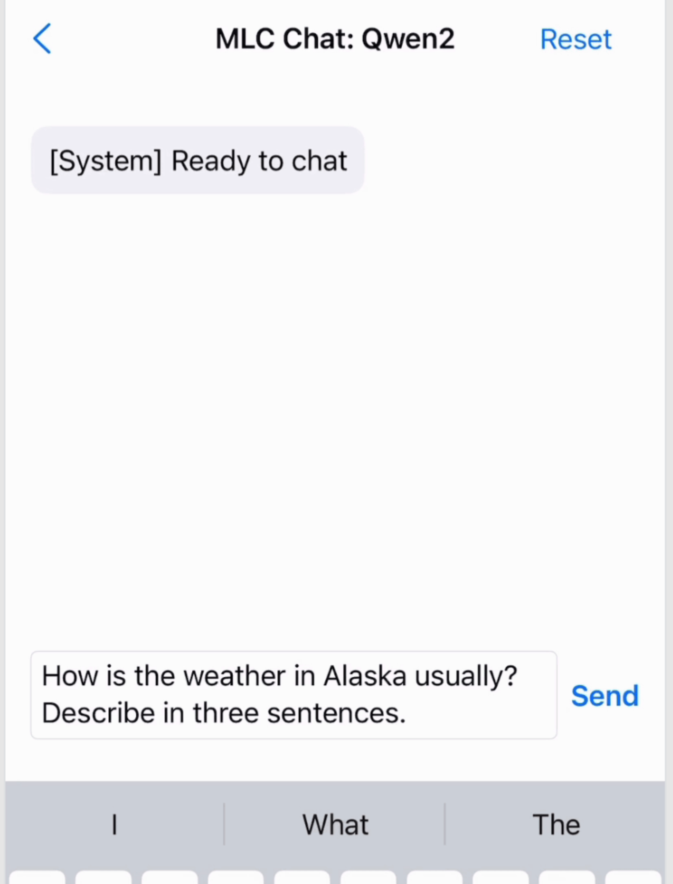
iOS SDK

OpenAI-style swift API

Demo on AppStore

Search for MLC Chat

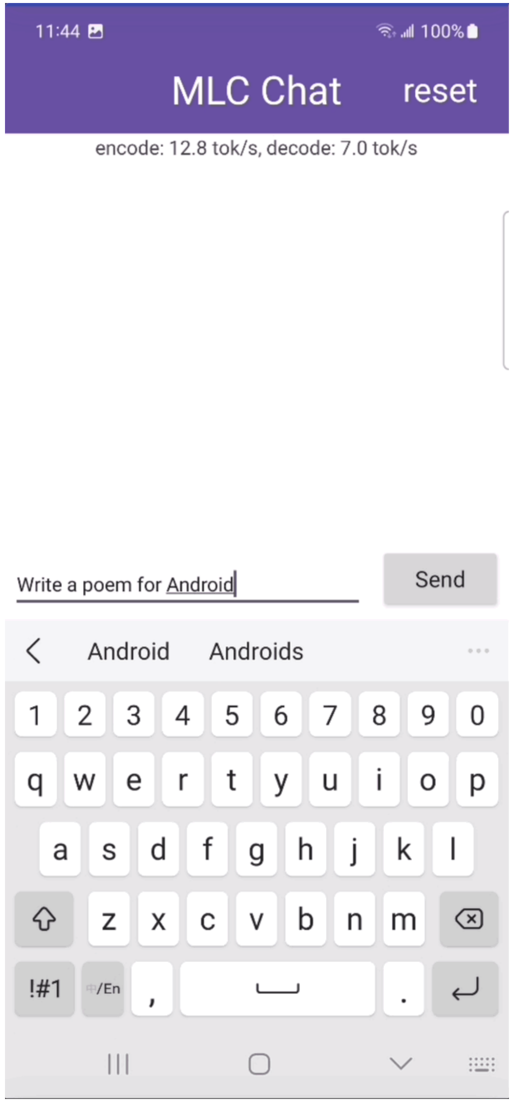
```swift
func requestGenerate(prompt: String) {
    appendMessage(role: .user, message: prompt)
    appendMessage(role: .assistant, message: "")

    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
                    }
                }
            }
        }
    }
}
```



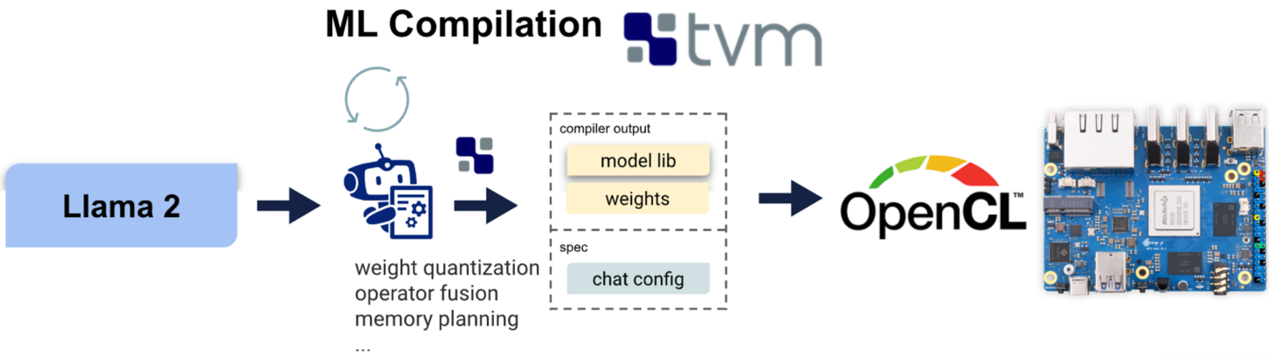
MLC LLM: Android

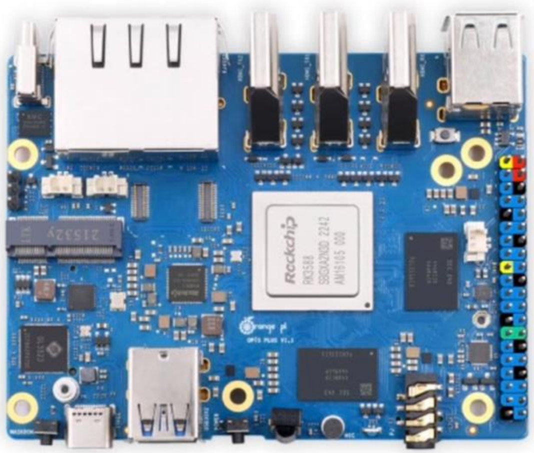
Snapdragon Gen2

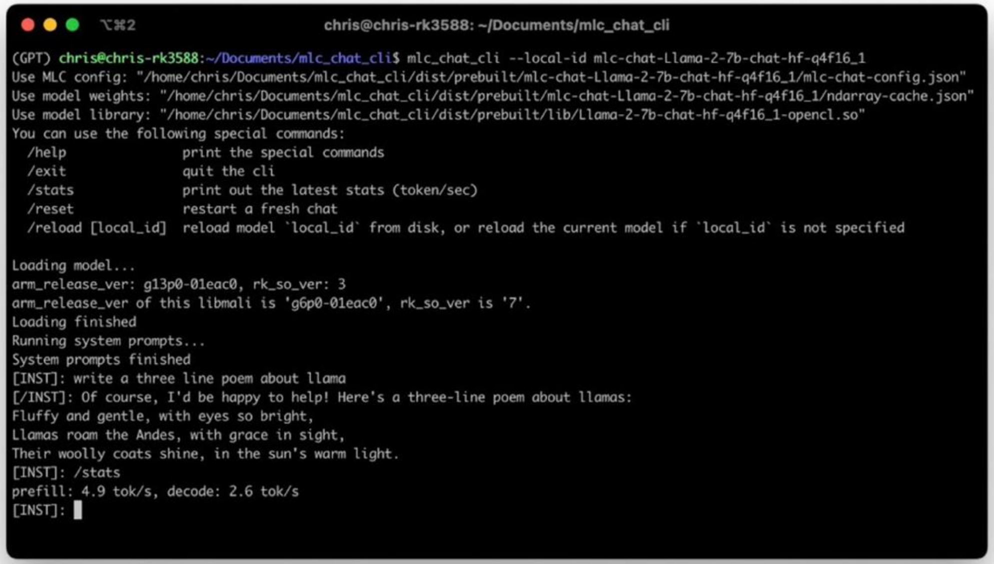
Enables larger models than iPhone



Bringing LLMs to 100$ Orange Pi





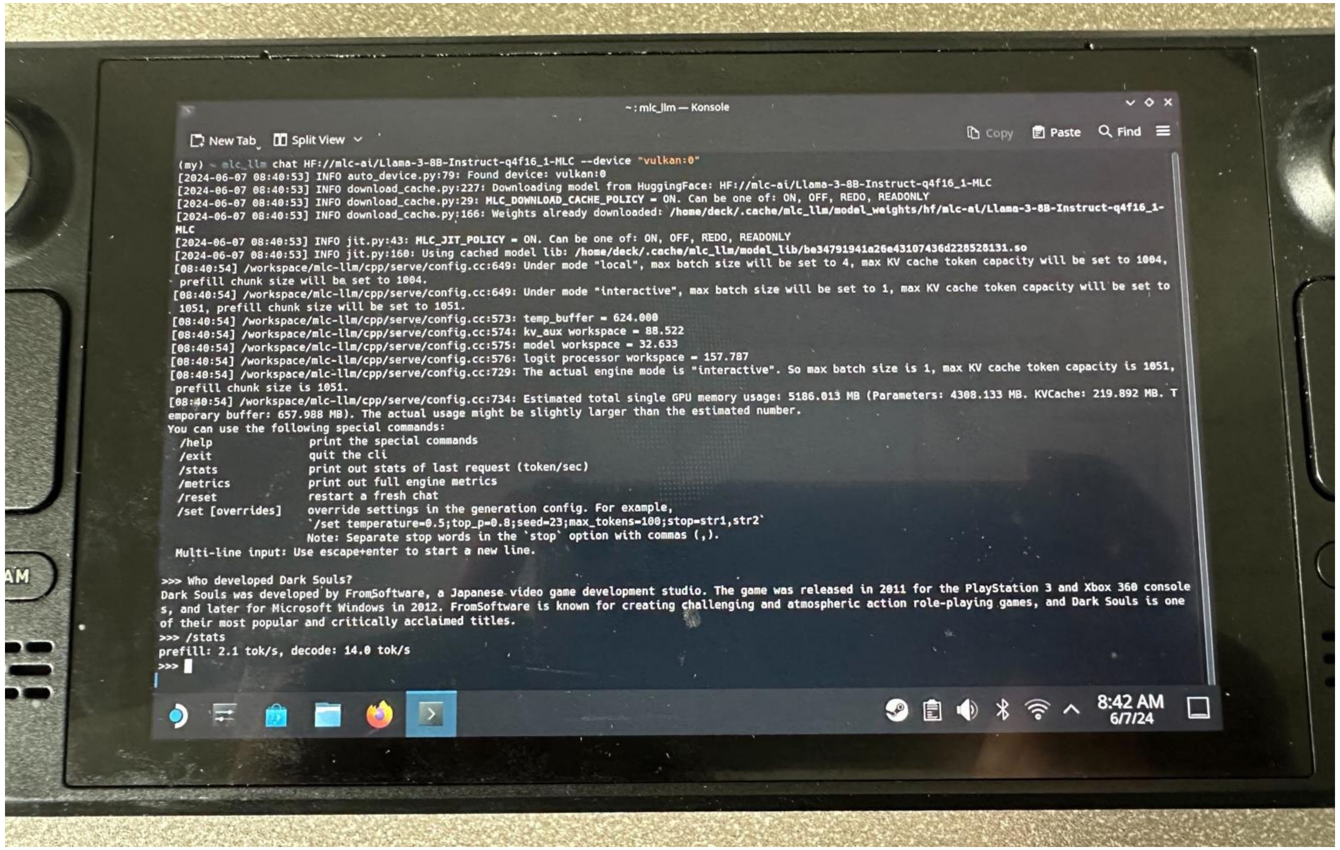


```
chris@chris-rk3588: ~/Documents/mlc_chat_cli
(GPT) chris@chris-rk3588:~/Documents/mlc_chat_cli$ mlc_chat_cli --local-id mlc-chat-Llama-2-7b-chat-hf-q4f16_1
Use MLC config: "/home/chris/Documents/mlc_chat_cli/dist/prebuilt/mlc-chat-Llama-2-7b-chat-hf-q4f16_1/mlc-chat-config.json"
Use model weights: "/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

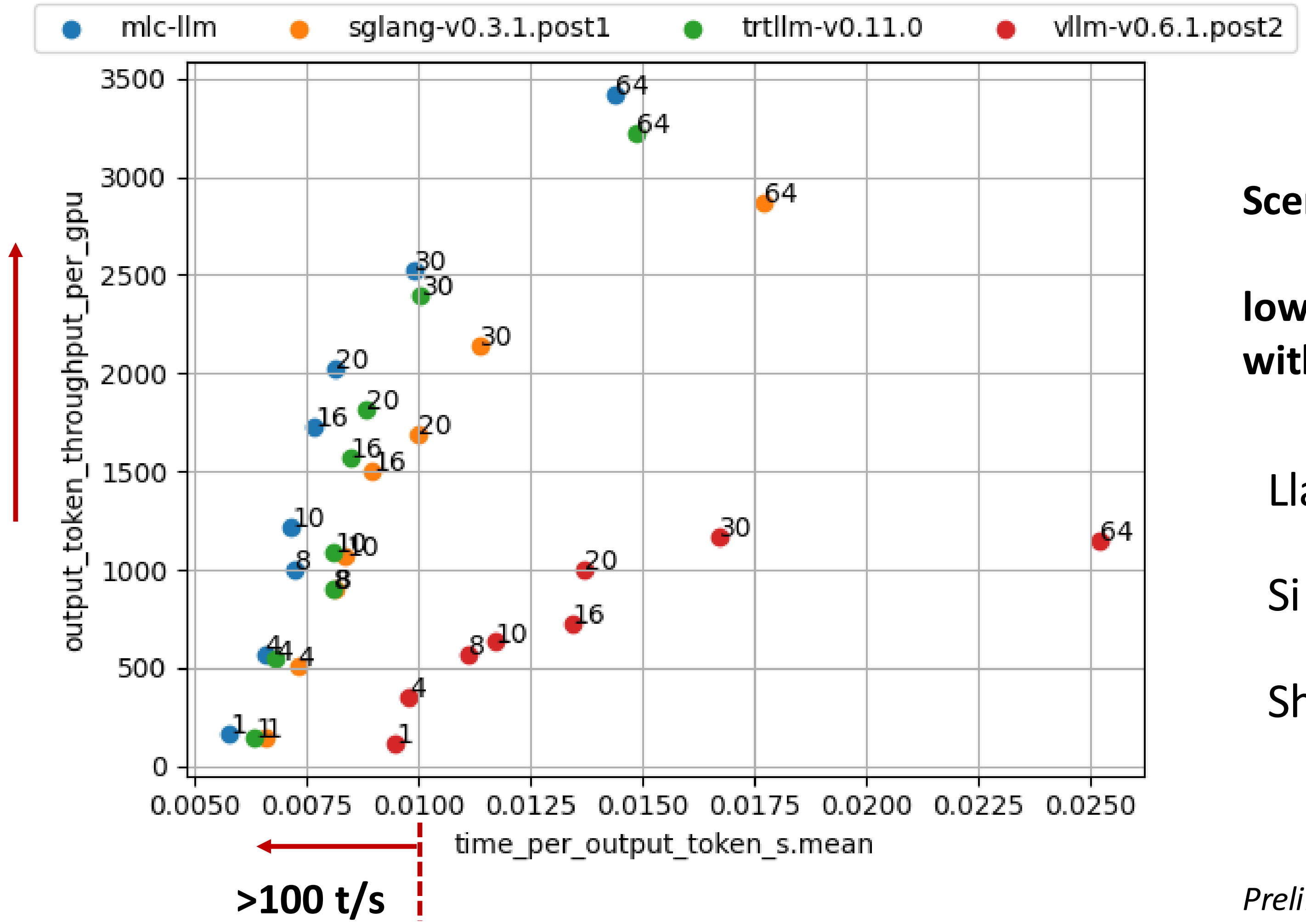
Near zero overhead

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)

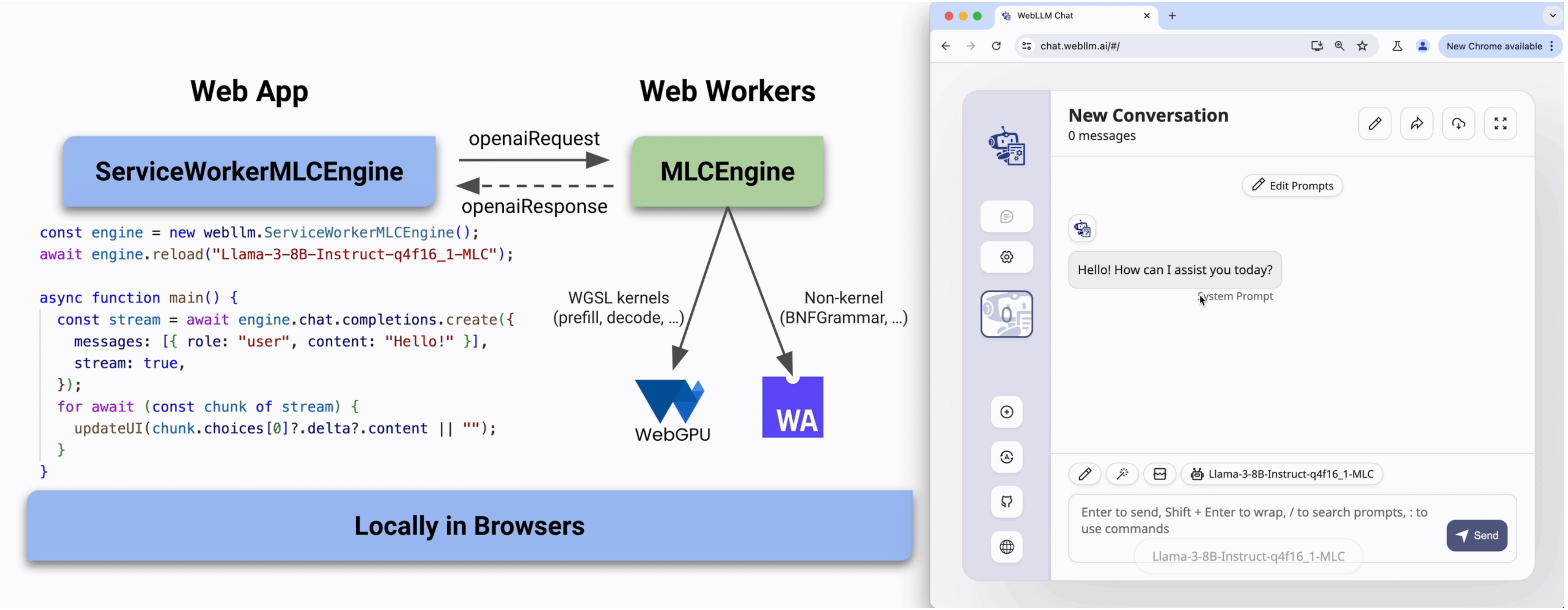
Llama3.1 8B

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

MLC-LLM: llm.mlc.ai

MACHINE LEARNING COMPILATION

tvm