

15-442/15-642: Machine Learning Systems

Enable Model Deployment across Cloud and Edge with ML Compilation

Spring 2026

Tianqi Chen

Carnegie Mellon University

ML Systems Plays a Critical Role in Machine Learning Revolutions

Big Data



Recommendation
Data analytics

Deep Learning



Strong pattern
recognition capabilities

Generative AI



Open ended conversations
Generalist models

Key
Capabilities

ML
Systems



ML Systems plays an
even more central role

2010

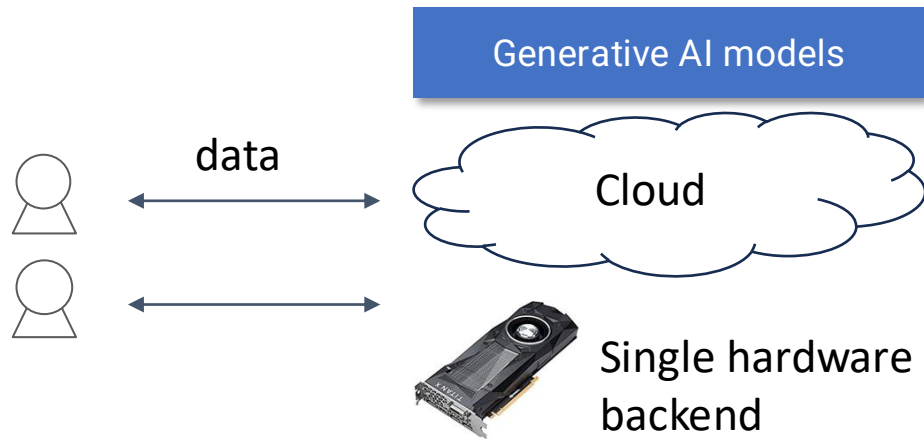
2013

2023

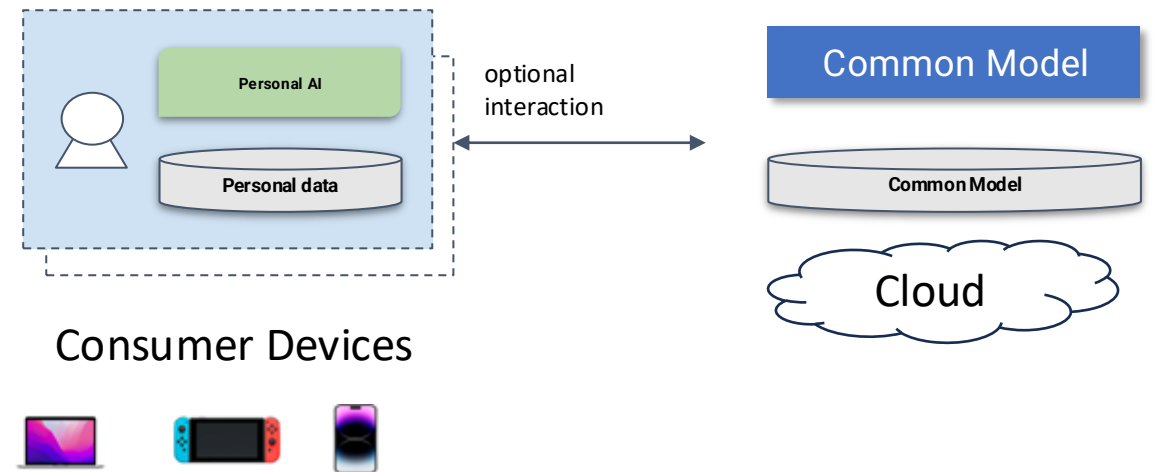


Opportunities in both Cloud and Edge

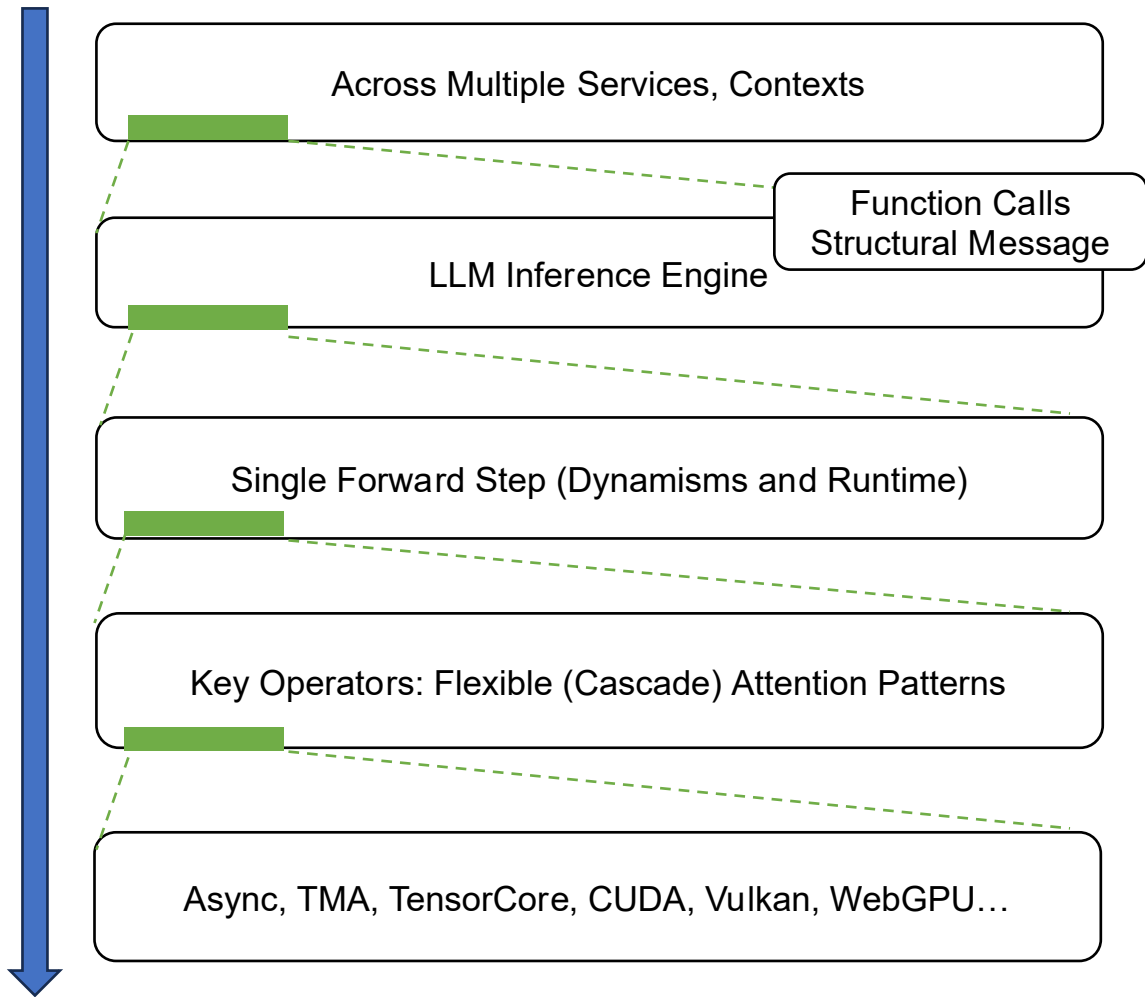
Cloud focused approach



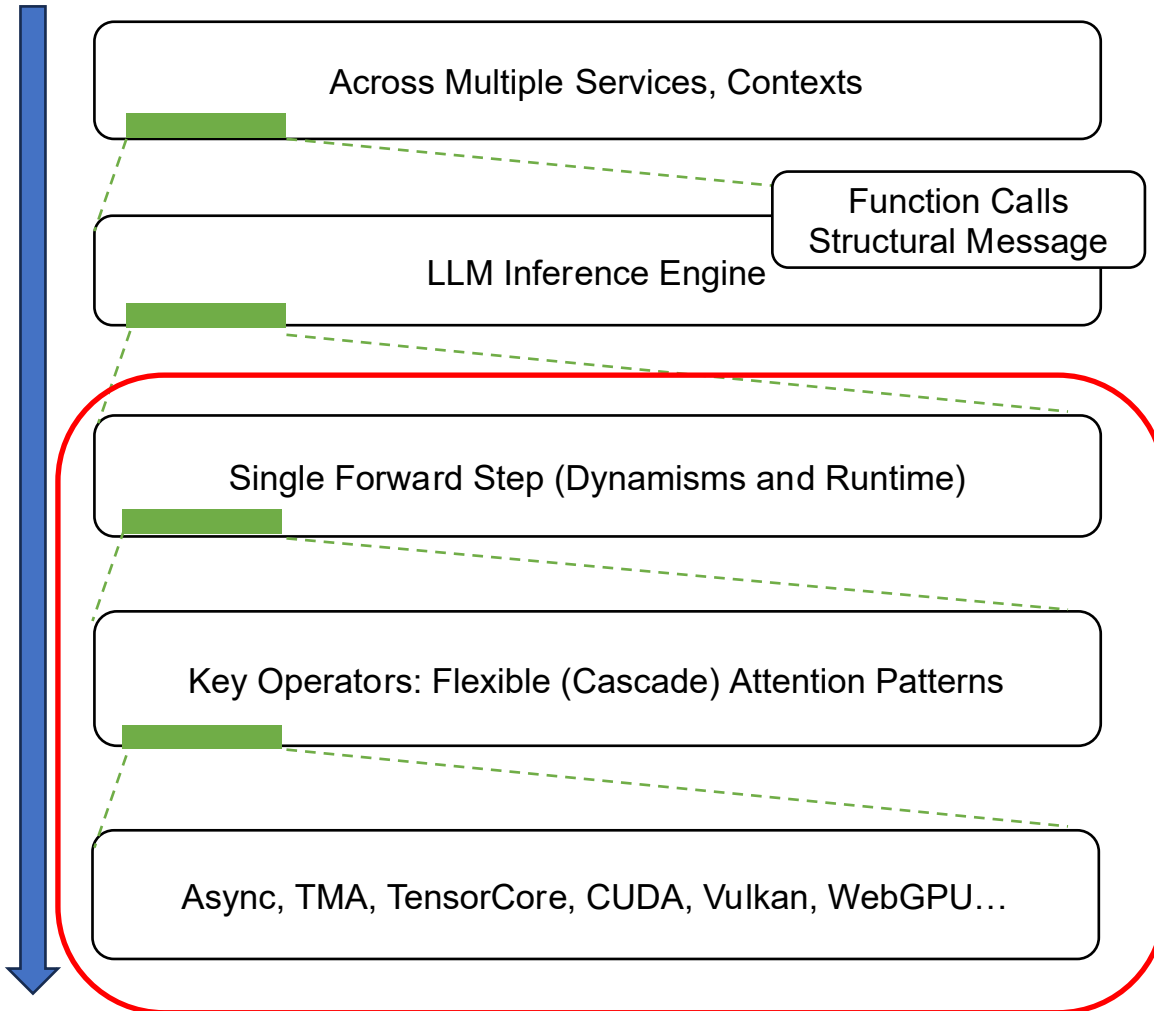
Cloud and Edge



Putting ML Compiler in Context of LLM System



Outline



Cross-level ML compilers for dynamic machine learning models

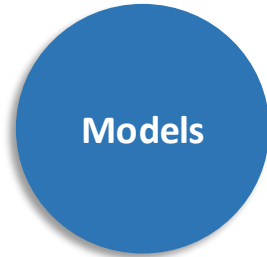
Think about Deployment Spectrum

How to bring a solution across the spectrum?

with minimal amount of engineering effort

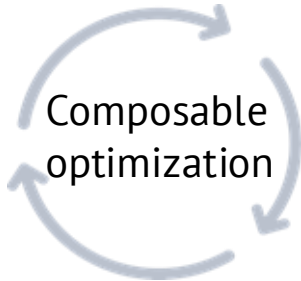


ML Compiler Approaches can help



Llama3, Whisper, CLIP, SAM, ...

ML Program Abstraction



- Memory Planning
- Library Dispatch
- Op Fusion
- Parameter Sharding
- More optimizations

Challenges

Customize to LLM models
Dynamism in model (shape and KV)

Universal Deployment Runtime

Cloud Server



Desktop / Laptop



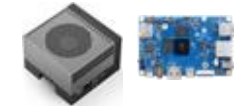
Tablets



Mobile



Robots/Embedded



Web Browsers

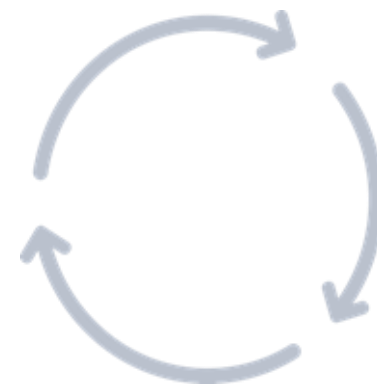


What is the Biggest Challenge?

ML modeling



ML Engineering

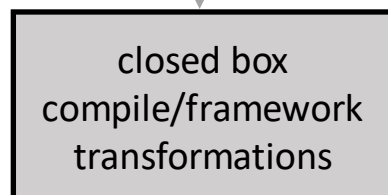
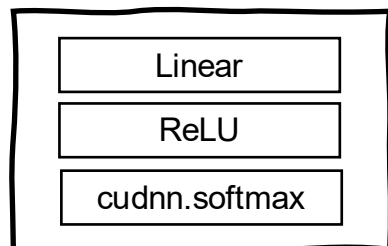


ML engineering now becomes critical and go hand in hand with ML modeling
It is not about build silver bullet once but **continuous improvement and innovations**

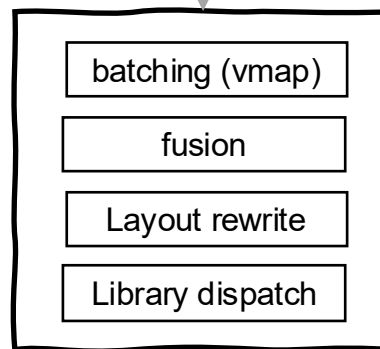
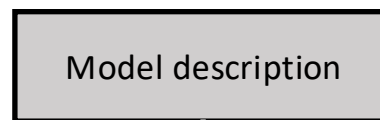
Development Patterns in Age of LLMs

Normal development
assuming a mature
framework foundation

Customize function
compositions for model

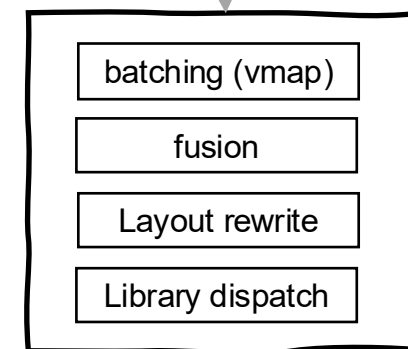
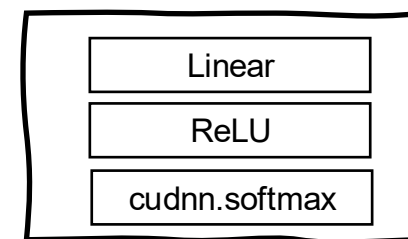


Normal compiler
development



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

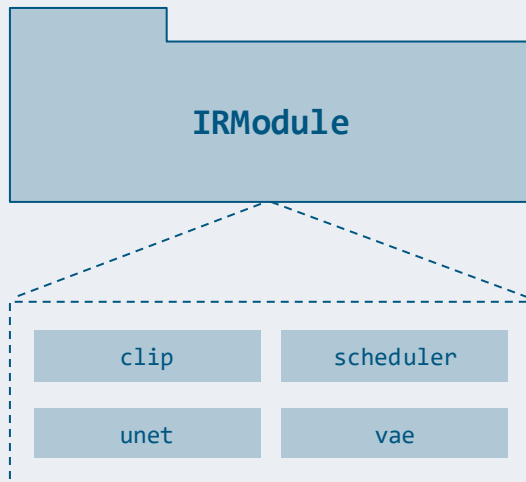
**Domain specific ML
compilation pipeline
development**



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

Relax: Composable Abstractions for End-to-End Dynamic Machine Learning

Centers around one key construct



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

Accessible through python

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

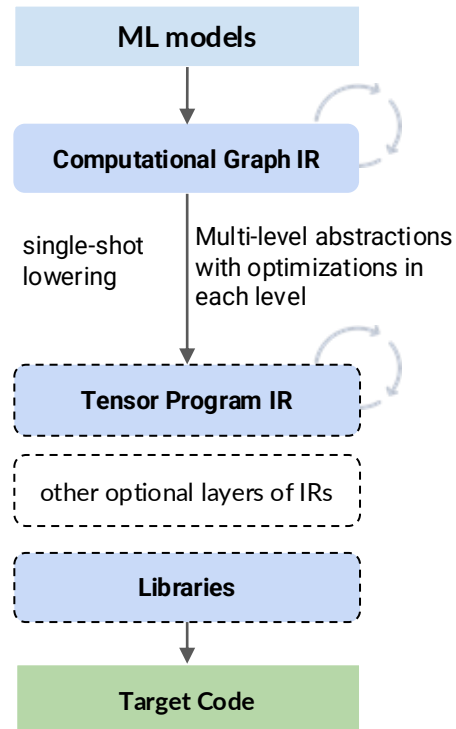
```
import tvm.script
from tvm.script import tir as T, relax as R

@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)
            ...
```

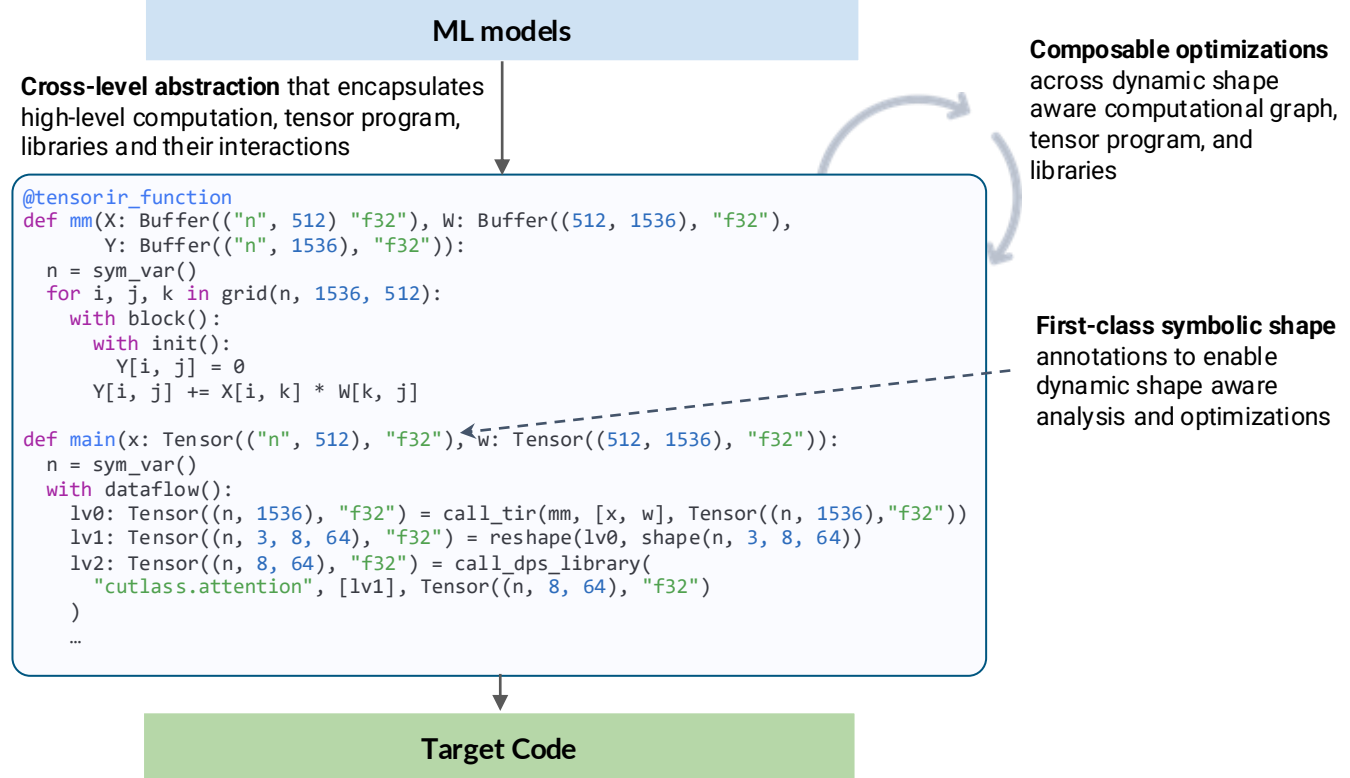
Python-first representation of a ML computation

Relax is a Cross-Level Abstraction

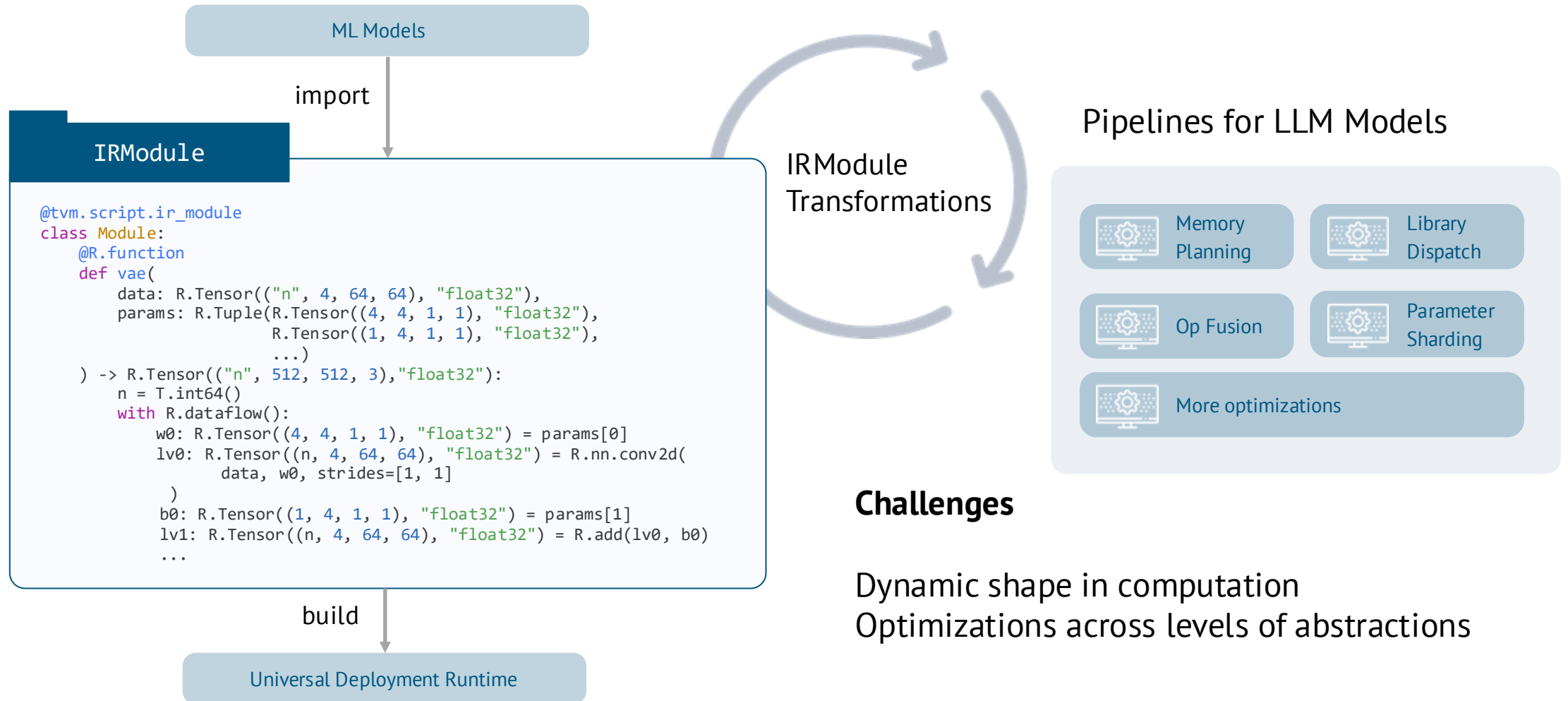
Multi-level ML Compilers



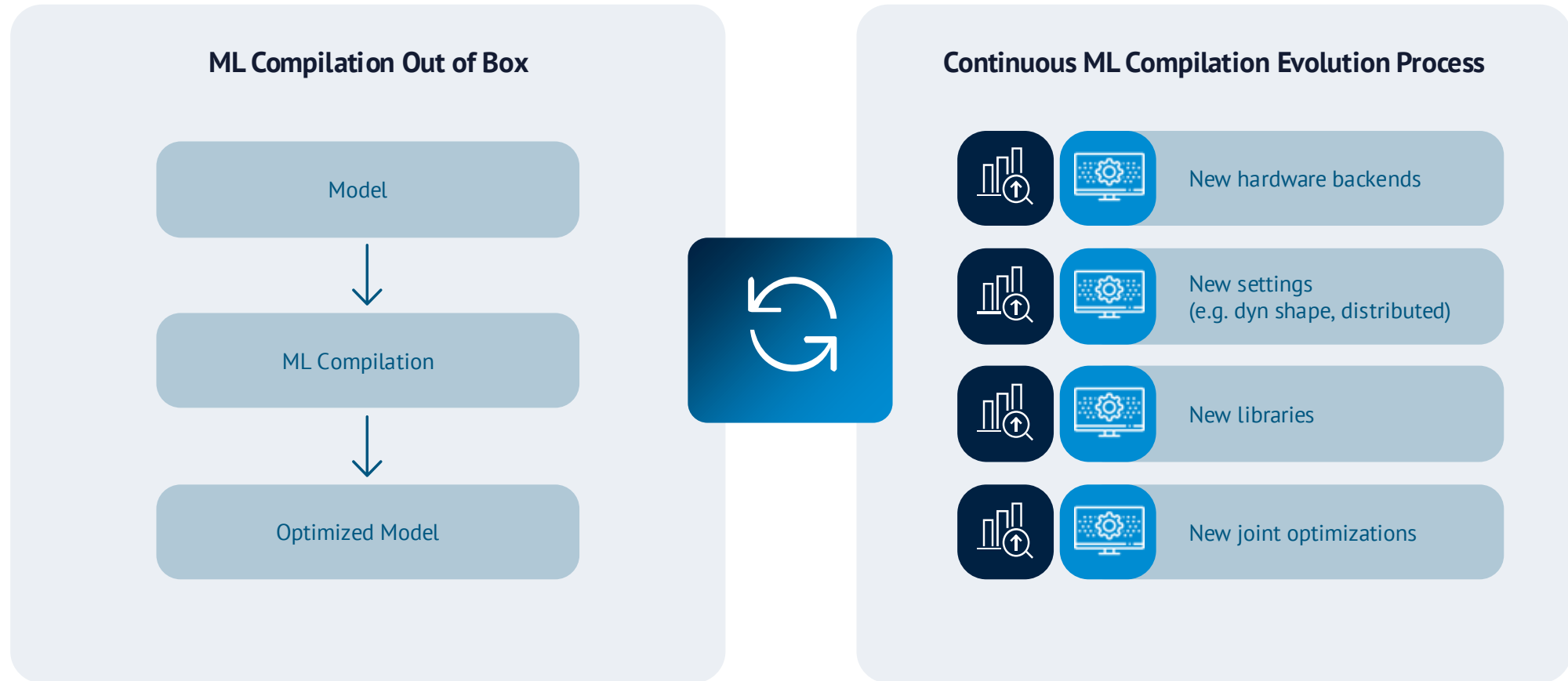
Our Approach



Customizing a Domain Specific Compiler for LLM



Continuous Improvement Process



This is not a one shot game, but continuous ML compilation evolution process for every new model, backend features, new improvements. We can enable more people to do it, together :)

First-class Symbolic Shape

```
def any_shape_fn(x: Tensor((?, 2, 2), "f32")):  
    n = get_shape_value(x, axis=0)  
    lv0: Tensor((?, 4), "f32") = reshape(x, (n, 4))  
    lv1: Tensor((?,), "f32") = flatten(lv0)  
    lv2: Tensor(?, "f32") = unique(lv1)  
  
    lv3: Tensor(?, "f32") = exp(lv2)  
    return lv3
```

A typical approach:

“unknown ?” dimensions for dynamic shapes

- Misses preserve potentially useful information, such as relations and constraints between shape dimensions
- Hinder optimizations that rely on shape information

First-class Symbolic Shape

```
def any_shape_fn(x: Tensor((?, 2, 2), "f32")):  
    n = get_shape_value(x, axis=0)  
    lv0: Tensor((?, 4), "f32") = reshape(x, (n, 4))  
    lv1: Tensor((?,), "f32") = flatten(lv0)  
    lv2: Tensor(?, "f32") = unique(lv1)  
  
    lv3: Tensor(?, "f32") = exp(lv2)  
    return lv3
```



```
def symbolic_shape_fn(x: Tensor(("n", 2, 2), "f32")):  
    n, m = sym_var(), sym_var()  
    lv0: Tensor((n, 4), "f32") = reshape(x, shape(n, 4))  
    lv1: Tensor((n * 4,), "f32") = flatten(lv0)  
    lv2: Tensor(ndim=1, dtype="f32") = unique(lv1)  
    lv3 = match_cast(lv2, Tensor((m,), "f32"))  
    lv4: Tensor((m,), "f32") = exp(lv3)  
    return lv4
```

A typical approach:

“**unknown ?**” dimensions for dynamic shapes

- Misses preserve potentially useful information, such as relations and constraints between shape dimensions
- Hinder optimizations that rely on shape information

Relax:

first-class symbolic shape

- mixture of integer variables and constants
- forward shape deduction for IR transformations
- allow introducing new shape variables for unknown shapes (such as the results of unique) via `match_cast`

First-class Symbolic Shape

```
def any_shape_fn(x: Tensor((?, 2, 2), "f32")):  
    n = get_shape_value(x, axis=0)  
    lv0: Tensor((?, 4), "f32") = reshape(x, (n, 4))  
    lv1: Tensor((?,), "f32") = flatten(lv0)  
    lv2: Tensor(?, "f32") = unique(lv1)  
  
    lv3: Tensor(?, "f32") = exp(lv2)  
    return lv3
```



```
def symbolic_shape_fn(x: Tensor(("n", 2, 2), "f32")):  
    n, m = sym_var(), sym_var()  
    lv0: Tensor((n, 4), "f32") = reshape(x, shape(n, 4))  
    lv1: Tensor((n * 4,), "f32") = flatten(lv0)  
    lv2: Tensor(ndim=1, dtype="f32") = unique(lv1)  
    lv3 = match_cast(lv2, Tensor((m,), "f32"))  
    lv4: Tensor((m,), "f32") = exp(lv3)  
    return lv4
```

A typical approach:

“**unknown ?**” dimensions for dynamic shapes

- Misses preserve potentially useful information, such as relations and constraints between shape dimensions
- Hinder optimizations that rely on shape information

Relax:

first-class symbolic shape

- mixture of integer variables and constants
- forward shape deduction for IR transformations
- allow introducing new shape variables for unknown shapes (such as the results of unique) via `match_cast`

First-class Symbolic Shape

```
def any_shape_fn(x: Tensor((?, 2, 2), "f32")):  
    n = get_shape_value(x, axis=0)  
    lv0: Tensor((?, 4), "f32") = reshape(x, (n, 4))  
    lv1: Tensor((?,), "f32") = flatten(lv0)  
    lv2: Tensor(?, "f32") = unique(lv1)  
  
    lv3: Tensor(?, "f32") = exp(lv2)  
    return lv3
```



```
def symbolic_shape_fn(x: Tensor(("n", 2, 2), "f32")):  
    n, m = sym_var(), sym_var()  
    lv0: Tensor((n, 4), "f32") = reshape(x, shape(n, 4))  
    lv1: Tensor((n * 4,), "f32") = flatten(lv0)  
    lv2: Tensor(ndim=1, dtype="f32") = unique(lv1)  
    lv3 = match_cast(lv2, Tensor((m,), "f32"))  
    lv4: Tensor((m,), "f32") = exp(lv3)  
    return lv4
```

A typical approach:

“**unknown ?**” dimensions for dynamic shapes

- Misses preserve potentially useful information, such as relations and constraints between shape dimensions
- Hinder optimizations that rely on shape information

Relax:

first-class symbolic shape

- mixture of integer variables and constants
- forward shape deduction for IR transformations
- allow introducing new shape variables for unknown shapes (such as the results of unique) via `match_cast`

First-class Symbolic Shape

```
def any_shape_fn(x: Tensor((?, 2, 2), "f32")):  
    n = get_shape_value(x, axis=0)  
    lv0: Tensor((?, 4), "f32") = reshape(x, (n, 4))  
    lv1: Tensor((?,), "f32") = flatten(lv0)  
    lv2: Tensor(?, "f32") = unique(lv1)  
  
    lv3: Tensor(?, "f32") = exp(lv2)  
    return lv3
```



```
def symbolic_shape_fn(x: Tensor(("n", 2, 2), "f32")):  
    n, m = sym_var(), sym_var()  
    lv0: Tensor((n, 4), "f32") = reshape(x, shape(n, 4))  
    lv1: Tensor((n * 4,), "f32") = flatten(lv0)  
    lv2: Tensor(ndim=1, dtype="f32") = unique(lv1)  
    lv3 = match_cast(lv2, Tensor((m,), "f32"))  
    lv4: Tensor((m,), "f32") = exp(lv3)  
    return lv4
```

A typical approach:

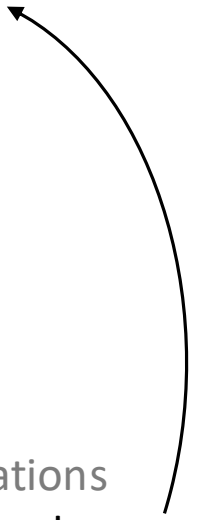
“**unknown ?**” dimensions for dynamic shapes

- Misses preserve potentially useful information, such as relations and constraints between shape dimensions
- Hinder optimizations that rely on shape information

Relax:

first-class symbolic shape

- mixture of integer variables and constants
- forward shape deduction for IR transformations
- allow introducing new shape variables for unknown shapes (such as the results of unique) via **match_cast**



First-class Symbolic Shape – Across Functions

Relax tracks and deduces symbolic shapes globally across functions

```
def subfn(s: Shape(["n", "m"])) -> Tensor(("n * m",), "f32"):
    """Take in a shape of (n, m), return a tensor of (n*m,)"""

def subgraph_func_shape_deduce_example(
    x: Tensor(("n",), "f32"),
    y: Shape(ndim=2)
):
    n = sym_var()
    → lv0: Tensor((n * 4,), "f32") = subfn(shape(n, 4))
```

First-class Symbolic Shape – Across Functions

Relax tracks and deduces symbolic shapes globally across functions

```
def subfn(s: Shape(["n", "m"])) -> Tensor(("n * m",), "f32"):
    """Take in a shape of (n, m), return a tensor of (n*m,)"""
```

```
def subgraph_func_shape_deduce_example(
    x: Tensor(("n",), "f32"),
    y: Shape(ndim=2)
):
    n = sym_var()
    lv0: Tensor((n * 4,), "f32") = subfn(shape(n, 4))
    → lv1: Tensor((12,), "f32") = subfn(shape(3, 4))
```

First-class Symbolic Shape – Across Functions

Relax tracks and deduces symbolic shapes globally across functions

```
def subfn(s: Shape(["n", "m"])) -> Tensor(("n * m",), "f32"):
    """Take in a shape of (n, m), return a tensor of (n*m,)"""
```

```
def subgraph_func_shape_deduce_example(
    x: Tensor(("n",), "f32"),
    y: Shape(ndim=2)
):
```

```
    n = sym_var()
```

```
    lv0: Tensor((n * 4,), "f32") = subfn(shape(n, 4))
```

```
    lv1: Tensor((12,), "f32") = subfn(shape(3, 4))
```

```
     lv2: Tensor(((n + 1) * 4,), "f32") = subfn(shape(n + 1, 4))
```

First-class Symbolic Shape – Across Functions

Relax tracks and deduces symbolic shapes globally across functions

```
def subfn(s: Shape(["n", "m"])) -> Tensor(("n * m",), "f32"):
    """Take in a shape of (n, m), return a tensor of (n*m,)"""
```

```
def subgraph_func_shape_deduce_example(
```

```
    x: Tensor(("n",), "f32"),
```

```
    y: Shape(ndim=2)
```

```
):
```

```
    n = sym_var()
```

```
    lv0: Tensor((n * 4,), "f32") = subfn(shape(n, 4))
```

```
    lv1: Tensor((12,), "f32") = subfn(shape(3, 4))
```

```
    lv2: Tensor(((n + 1) * 4,), "f32") = subfn(shape(n + 1, 4))
```

```
    → lv3: Tensor(ndim=1, dtype="f32") = subfn(y)
```

Tracking Dynamisms Across Levels

Encapsulate everything together

- Computational graphs,
- Loop-level tensor programs
- External library functions

```
# Graph-level end-to-end dynamic ML model
def main(x: Tensor(("n", 128), "f32"), w: Tensor((128, 256), "f32")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f32") = call_tir(
            mm, [x, w], Tensor((n, 256), "f32")
        )
        lv1: Tensor((n, 256), "f32") = relu(lv0)
        lv2: Tensor((n, 256), "f32") = call_dps_library(
            "cutlass.rms_norm", [lv1], Tensor((n, 256), "f32")
        )
    ...
```

```
@tensorir_function
def mm(X: Buffer(("n", 128), "f32"), W: Buffer((128, 256), "f32"),
       Y: Buffer(("n", 256), "f32")):
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        with block():
            with init():
                Y[i, j] = 0
            Y[i, j] += X[i, k] * W[k, j]
```

Loop-level tensor program



Tracking Dynamisms Across Levels

Encapsulate everything together

- Computational graphs,
- Loop-level tensor programs
- External library functions

```
# Graph-level end-to-end dynamic ML model
def main(x: Tensor(("n", 128), "f32"), w: Tensor((128, 256), "f32")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f32") = call_tir(
            mm, [x, w], Tensor((n, 256), "f32")
        )
        lv1: Tensor((n, 256), "f32") = relu(lv0)
        lv2: Tensor((n, 256), "f32") = call_dps_library(
            "cutlass.rms_norm", [lv1], Tensor((n, 256), "f32")
        )
    ...
```

call loop-level
tensor programs
from graph level

```
@tensorir_function
def mm(X: Buffer(("n", 128), "f32"), W: Buffer((128, 256), "f32"),
      Y: Buffer(("n", 256), "f32")):
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        with block():
            with init():
                Y[i, j] = 0
            Y[i, j] += X[i, k] * W[k, j]
```

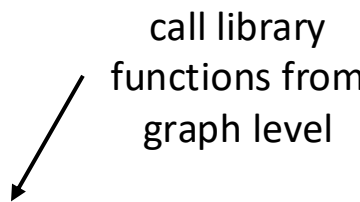
Loop-level tensor program

Tracking Dynamisms Across Levels

Encapsulate everything together

- Computational graphs,
- Loop-level tensor programs
- External library functions

```
# Graph-level end-to-end dynamic ML model
def main(x: Tensor(("n", 128), "f32"), w: Tensor((128, 256), "f32")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f32") = call_tir(
            mm, [x, w], Tensor((n, 256), "f32")
        )
        lv1: Tensor((n, 256), "f32") = relu(lv0)
        lv2: Tensor((n, 256), "f32") = call_dps_library(
            "cutlass.rms_norm", [lv1], Tensor((n, 256), "f32")
        )
    ...
```



```
@tensorir_function
def mm(X: Buffer(("n", 128), "f32"), W: Buffer((128, 256), "f32"),
       Y: Buffer(("n", 256), "f32")):
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        with block():
            with init():
                Y[i, j] = 0
            Y[i, j] += X[i, k] * W[k, j]
```

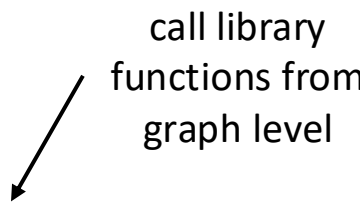
Tracking Dynamisms Across Levels

Encapsulate everything together

- Computational graphs,
- Loop-level tensor programs
- External library functions

**Unlocks more customization
and optimizations across levels**

```
# Graph-level end-to-end dynamic ML model
def main(x: Tensor(("n", 128), "f32"), w: Tensor((128, 256), "f32")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f32") = call_tir(
            mm, [x, w], Tensor((n, 256), "f32")
        )
        lv1: Tensor((n, 256), "f32") = relu(lv0)
        lv2: Tensor((n, 256), "f32") = call_dps_library(
            "cutlass.rms_norm", [lv1], Tensor((n, 256), "f32")
        )
    ...
```



```
@tensorir_function
def mm(X: Buffer(("n", 128), "f32"), W: Buffer((128, 256), "f32"),
      Y: Buffer(("n", 256), "f32")):
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        with block():
            with init():
                Y[i, j] = 0
            Y[i, j] += X[i, k] * W[k, j]
```

Case Study: Operator Fusion for Quantized Model

```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        W: Tensor((128, 256), "f16") = call_tir(decode_q4, [Wdata, Wscale], Tensor((128, 256), "f16"))
        lv0: Tensor((n, 256), "f16") = call_tir(mm, [x, W], Tensor((n, 256), "f16"))
    return lv0
```

@tensorir_function

```
def decode_q4(Wdata: Buffer((128, 32), "u32"), Wscale: Buffer((128, 8), "f16"),
             W: Buffer((128, 128), "f16")):
    for k, j in grid(128, 256):
        W[k, j] = ((data[k, j//8] >> (k%8*4)) & 15 - 7) * scale[i, k // 32]
```

@tensorir_function

```
def mm(X: Buffer(("n", 128), "f16"), W: Buffer((128, 256), "f16"),
      Y: Buffer(("n", 256), "f16")):
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        if k == 0:
            Y[i, j] = 0.0
        Y[i, j] += X[i, k] * W[k, j]
```

Case Study: Operator Fusion for Quantized Model

Pass 1. Analyze
tensor program
compute patterns

```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        W: Tensor((128, 256), "f16") = call_tir(decode_q4, [Wdata, Wscale], Tensor((128, 256), "f16"))
        lv0: Tensor((n, 256), "f16") = call_tir(mm, [x, W], Tensor((n, 256), "f16"))
    return lv0

@tensorir_function
def decode_q4(Wdata: Buffer((128, 32), "u32"), Wscale: Buffer((128, 8), "f16"),
             W: Buffer((128, 256), "f16")):
    for k, j in grid(128, 256):
        W[k, j] = ((data[k, j//8] >> (k%8*4)) & 15 - 7) * scale[i, k // 32]

@tensorir_function
def mm(X: Buffer(("n", 128), "f16"), W: Buffer((128, 256), "f16"),
      Y: Buffer(("n", 256), "f16")):
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        if k == 0:
            Y[i, j] = 0.0
        Y[i, j] += X[i, k] * W[k, j]
```

Case Study: Operator Fusion for Quantized Model

Pass 1. Analyze
tensor program
compute patterns

```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        W: Tensor((128, 256), "f16") = call_tir(decode_q4, [Wdata, Wscale], Tensor((128, 256), "f16"))
        lv0: Tensor((n, 256), "f16") = call_tir(mm, [x, W], Tensor((n, 256), "f16"))
    return lv0
```

@tensorir_function

```
def decode_q4(Wdata: Buffer((128, 32), "u32"), Wscale: Buffer((128, 8), "f16"),
             W: Buffer((128, 128), "f16")):
    func_attr("compute_pattern", "Injective")
    for k, j in grid(128, 256):
        W[k, j] = ((data[k, j//8] >> (k%8*4)) & 15 - 7) * scale[i, k // 32]
```

@tensorir_function

```
def mm(X: Buffer(("n", 128), "f16"), W: Buffer((128, 256), "f16"),
      Y: Buffer(("n", 256), "f16")):
    func_attr("compute_pattern", "OutputEwiseFusible")
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        if k == 0:
            Y[i, j] = 0.0
        Y[i, j] += X[i, k] * W[k, j]
```

Case Study: Operator Fusion for Quantized Model

Pass 2. FuseOps

```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        W: Tensor((128, 256), "f16") = call_tir(decode_q4, [Wdata, Wscale], Tensor((128, 256), "f16"))
        lv0: Tensor((n, 256), "f16") = call_tir(mm, [x, W], Tensor((n, 256), "f16"))
    return lv0
```

@tensorir_function

```
def decode_q4(Wdata: Buffer((128, 32), "u32"), Wscale: Buffer((128, 8), "f16"),
             W: Buffer((128, 128), "f16")):
    func_attr("compute_pattern", "Injective")
    for k, j in grid(128, 256):
        W[k, j] = ((data[k, j//8] >> (k%8*4)) & 15 - 7) * scale[i, k // 32]
```

@tensorir_function

```
def mm(X: Buffer(("n", 128), "f16"), W: Buffer((128, 256), "f16"),
      Y: Buffer(("n", 256), "f16")):
    func_attr("compute_pattern", "OutputEwiseFusible")
    n = sym_var()
    for i, j, k in grid(n, 256, 128):
        if k == 0:
            Y[i, j] = 0.0
        Y[i, j] += X[i, k] * W[k, j]
```

Case Study: Operator Fusion for Quantized Model

Pass 2. FuseOps

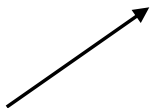
```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f16") = fused_decode_q4_mm(x, Wdata, Wscale)
    return lv0
```

```
def fused_decode_q4_mm(x: Tensor(("n", 128), "f16"),
                       Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        W: Tensor((128, 256), "f16") = call_tir(decode_q4, [Wdata, Wscale], Tensor((128, 256), "f16"))
        lv0: Tensor((n, 256), "f16") = call_tir(mm, [x, W], Tensor((n, 256), "f16"))
    return lv0
```

```
@tensorir_function
def decode_q4(Wdata: Buffer((128, 32), "u32"), Wscale: Buffer((128, 8), "f16"),
             W: Buffer((128, 128), "f16")):
    func_attr("compute_pattern", "Injective")
    ...
```

```
@tensorir_function
def mm(X: Buffer(("n", 128), "f16"), W: Buffer((128, 256), "f16"),
      Y: Buffer(("n", 256), "f16")):
    func_attr("compute_pattern", "OutputEwiseFusible")
    ...
```

Merged subgraph function



Case Study: Operator Fusion for Quantized Model

Pass 3. FuseTIR

```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f16") = fused_decode_q4_mm(x, Wdata, Wscale)
    return lv0
```

```
def fused_decode_q4_mm(x: Tensor(("n", 128), "f16"),
                      Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        W: Tensor((128, 256), "f16") = call_tir(decode_q4, [Wdata, Wscale], Tensor((128, 256), "f16"))
        lv0: Tensor((n, 256), "f16") = call_tir(mm, [x, W], Tensor((n, 256), "f16"))
    return lv0
```

Merged subgraph function



```
@tensorir_function
def decode_q4(Wdata: Buffer((128, 32), "u32"), Wscale: Buffer((128, 8), "f16"),
             W: Buffer((128, 128), "f16")):
    func_attr("compute_pattern", "Injective")
    ...
```

```
@tensorir_function
def mm(X: Buffer(("n", 128), "f16"), W: Buffer((128, 256), "f16"),
      Y: Buffer(("n", 256), "f16")):
    func_attr("compute_pattern", "OutputEwiseFusible")
    ...
```

Case Study: Operator Fusion for Quantized Model

Pass 3. FuseTIR

Modularizes quantize and fusion

more follow ups optimizing fused ops

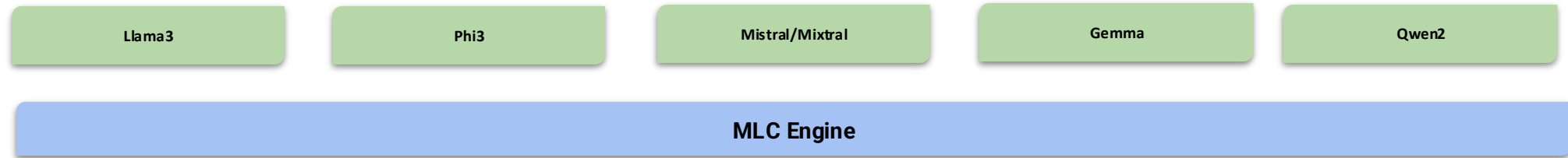
Merged tensor program

```
def main(x: Tensor(("n", 128), "f16"),
         Wdata: Tensor((128, 32), "u32"), Wscale: Tensor((128, 8), "f16")):
    n = sym_var()
    with dataflow():
        lv0: Tensor((n, 256), "f16") = call_tir(
            fused_decode_q4_mm, [x, Wdata, Wscale], Tensor((n, 256), "f16")
        )
    return lv0
```

```
@tensorir_function
def fused_decode_q4_mm(X: Buffer(("n", 128), "f16"), Wdata: Buffer((128, 32), "u32"),
                      Wscale: Buffer((128, 8), "f16"), Y: Buffer(("n", 256), "f16")):
    n = sym_var()
    W = alloc_buffer((128, 256), "f16")
    # decode_q4
    for k, j in grid(128, 256):
        W[k, j] = ((data[k, j//8] >> (k%8*4)) & 15 - 7) * scale[i, k // 32]
    # matmul
    for i, j, k in grid(n, 256, 128):
        if k == 0:
            Y[i, j] = 0.0
        Y[i, j] += X[i, k] * W[k, j]
```

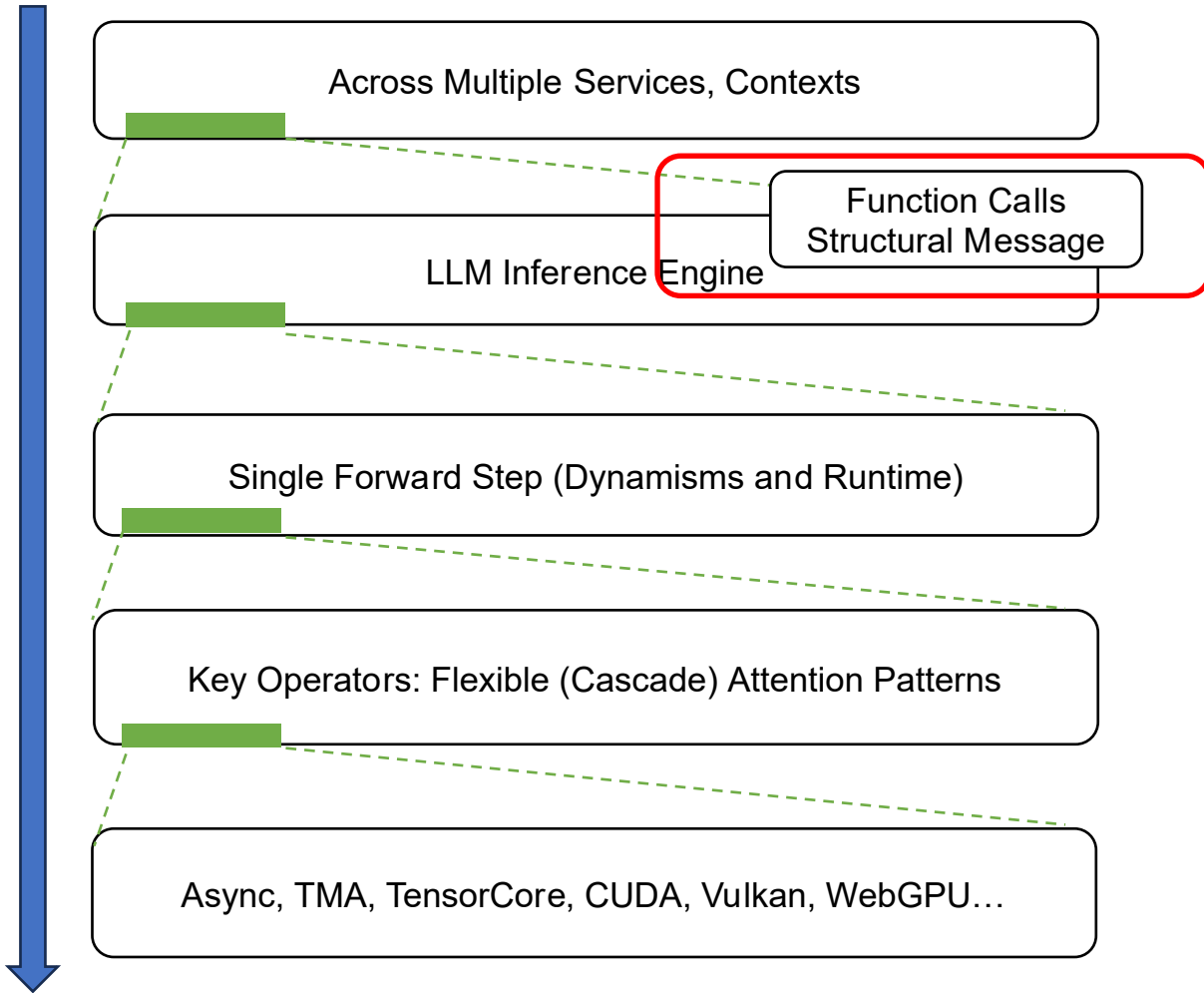
Relax Enables MLC Engine

Universally deploys across cloud and edge



Web Browsers 
WebGPU

Outline



Efficient and Flexible Structured Generation

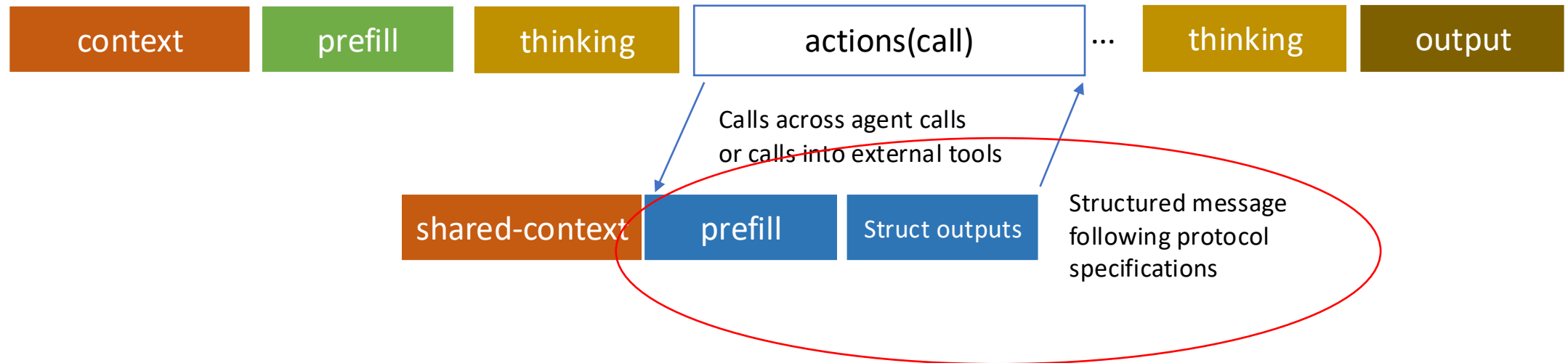
LLM System are evolving toward Agent Centric

Simplified view of inference on a single worker



Simple input / output, running on the same worker (color indicated the same worker)

Agentic Flow



Generating Structured Outputs via Constraint Decoding

JSON Schema

```
class Task(BaseModel):  
    done: bool  
    name: str  
    steps: List[int]
```

Example Valid JSONs

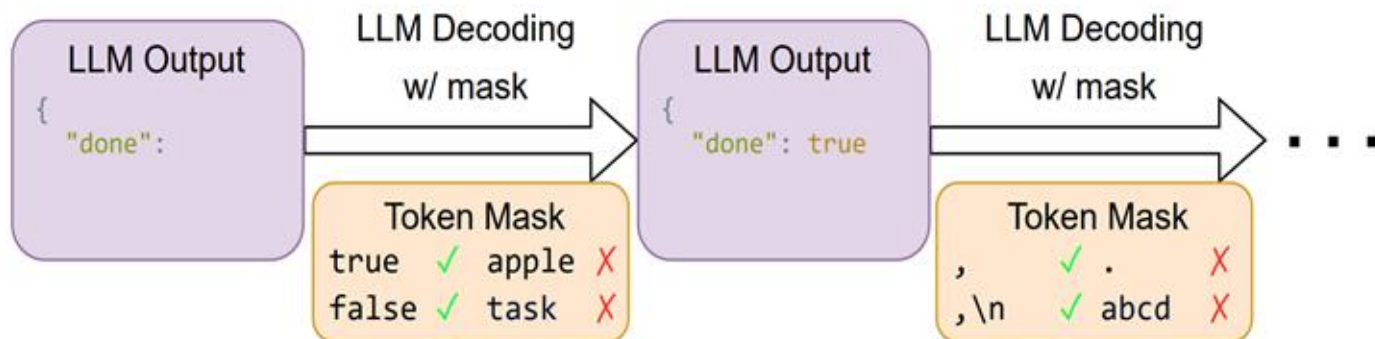
```
{  
  "done": true,  
  "name": "Clean kitchen",  
  "steps": [1, 2, 3, 4]  
}  
  
{  
  "done": false,  
  "name": "Presentation",  
  "steps": [1, 2]  
}
```

Challenges

Large vocabulary size

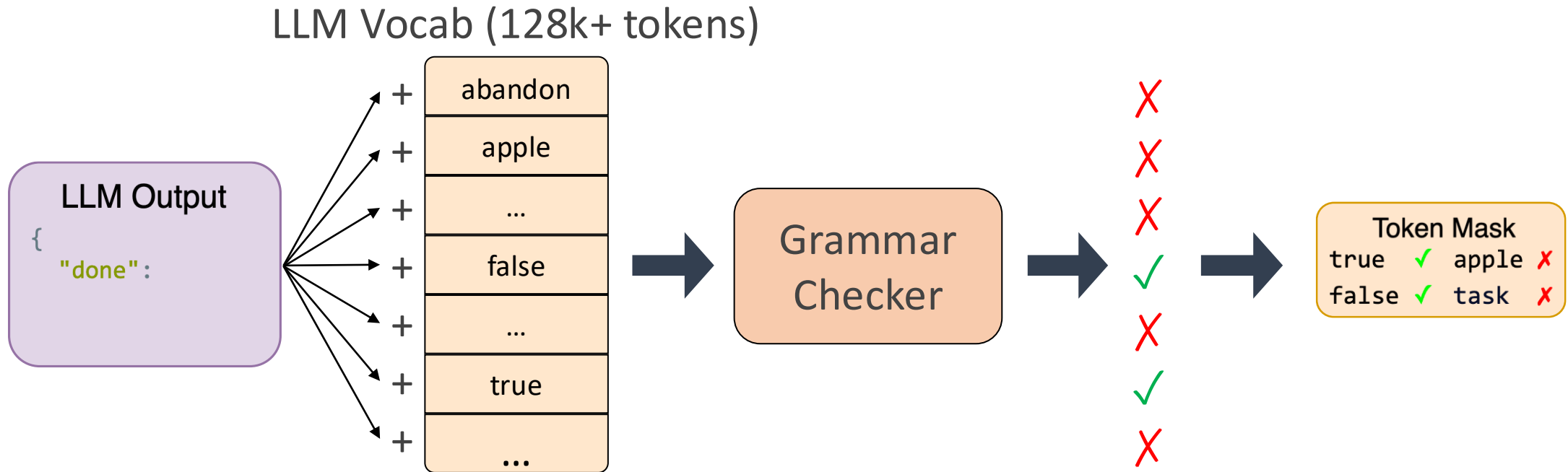
Infinite possible grammar state for complex grammars

GPUs getting faster (and mask generation on CPU cannot keep up with them)



Challenge: Efficient Mask Generation

Mask is generated with a grammar checker

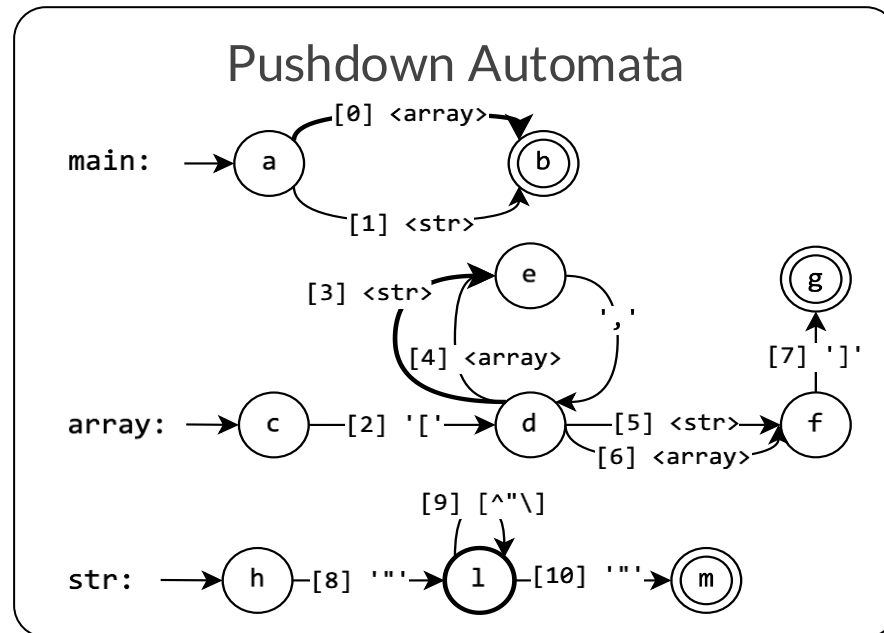


The grammar checker must be very fast!

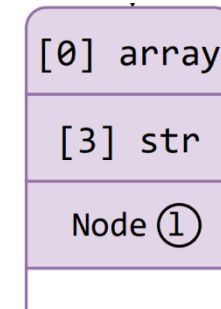
Handling General Grammar

Prior LLM Output

["a



Parsing Stack

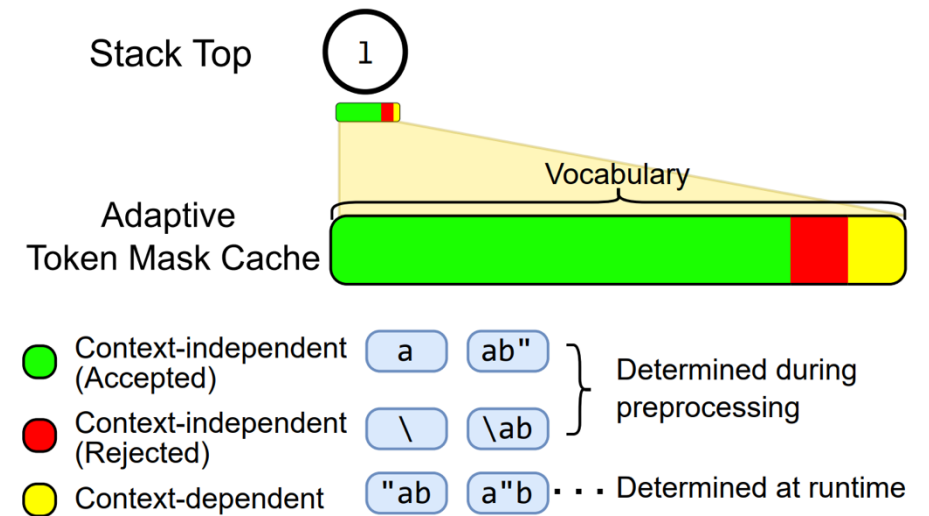


The length of the stack is unbounded \Rightarrow there can be ∞ possible parsing states

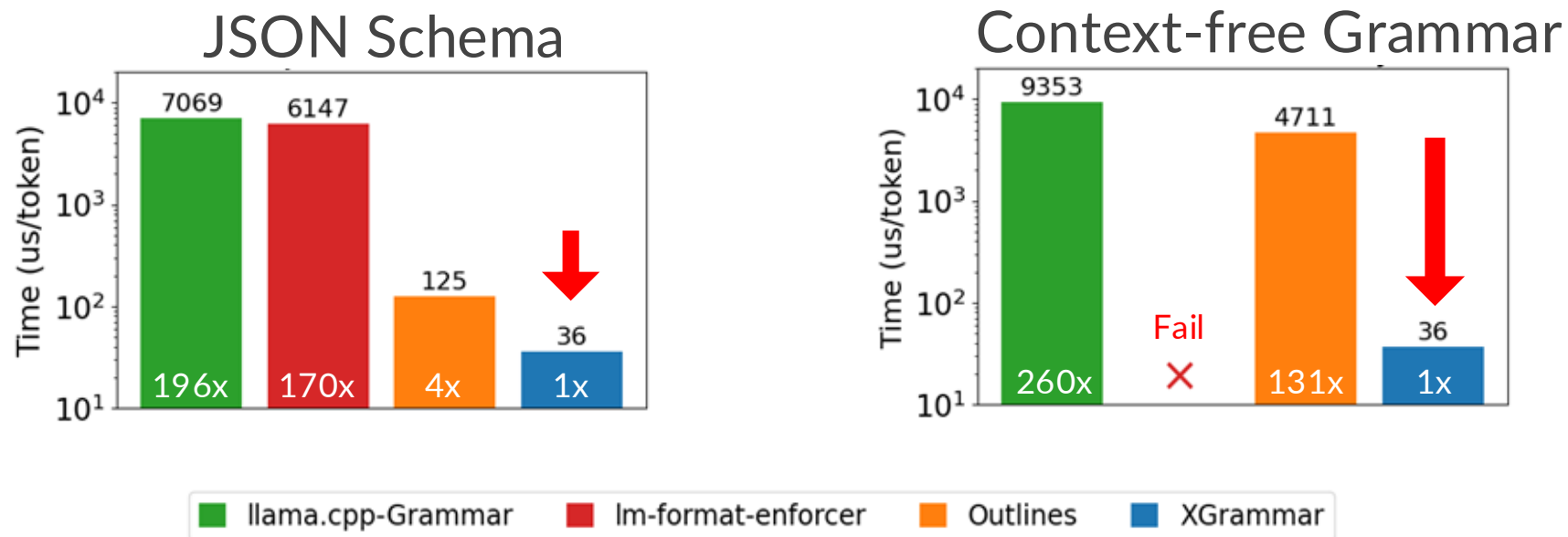
XGrammar: Efficient and Flexible Grammar Engine

Key insight

- **Context-independent tokens:** Most tokens (>99%) can be determined **ahead of time by only looking at top of the stack**
- **Context-dependent tokens** we still check outlier tokens at runtime to ensure full coverage.
- At runtime, we first retrieve the pre-computed token mask from the parsing state, then check the context-dependent tokens efficiently.

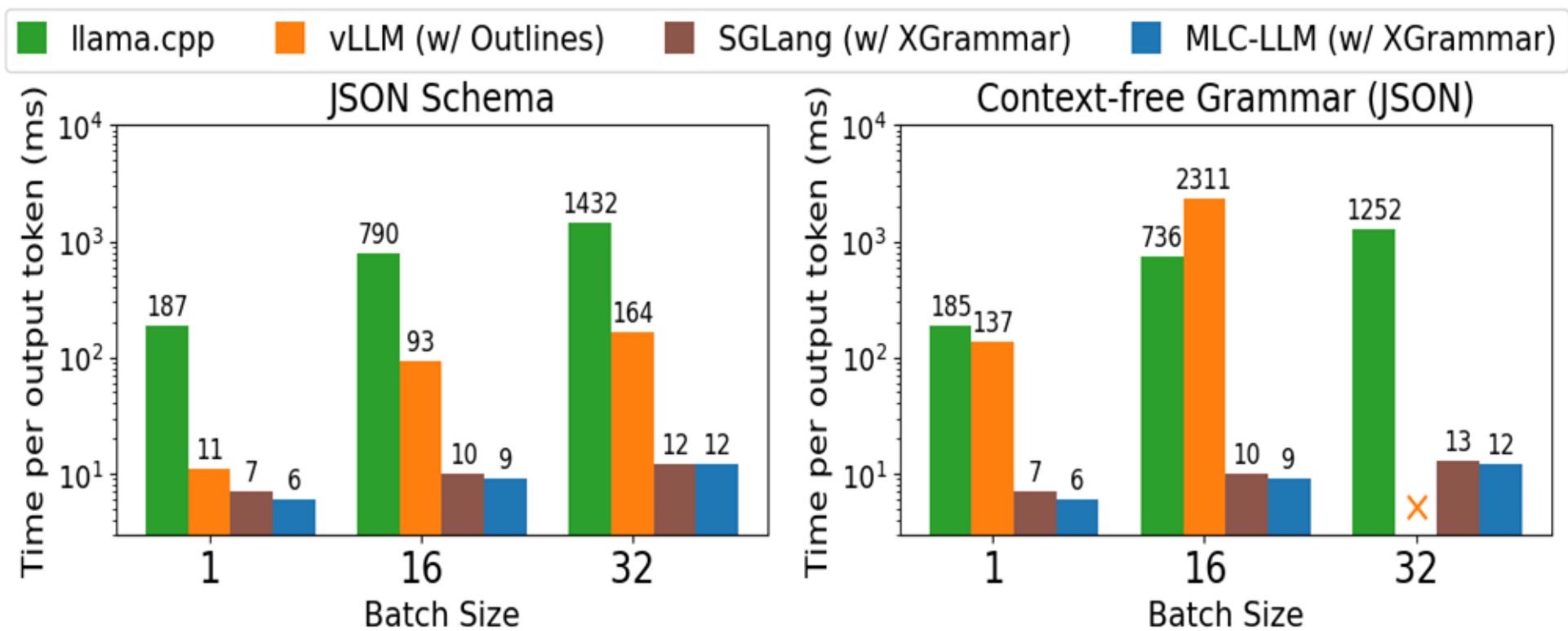


Mask Generation Efficiency



Environment: Llama-3-8B, AMD 7950X CPU, RTX 4090 GPU

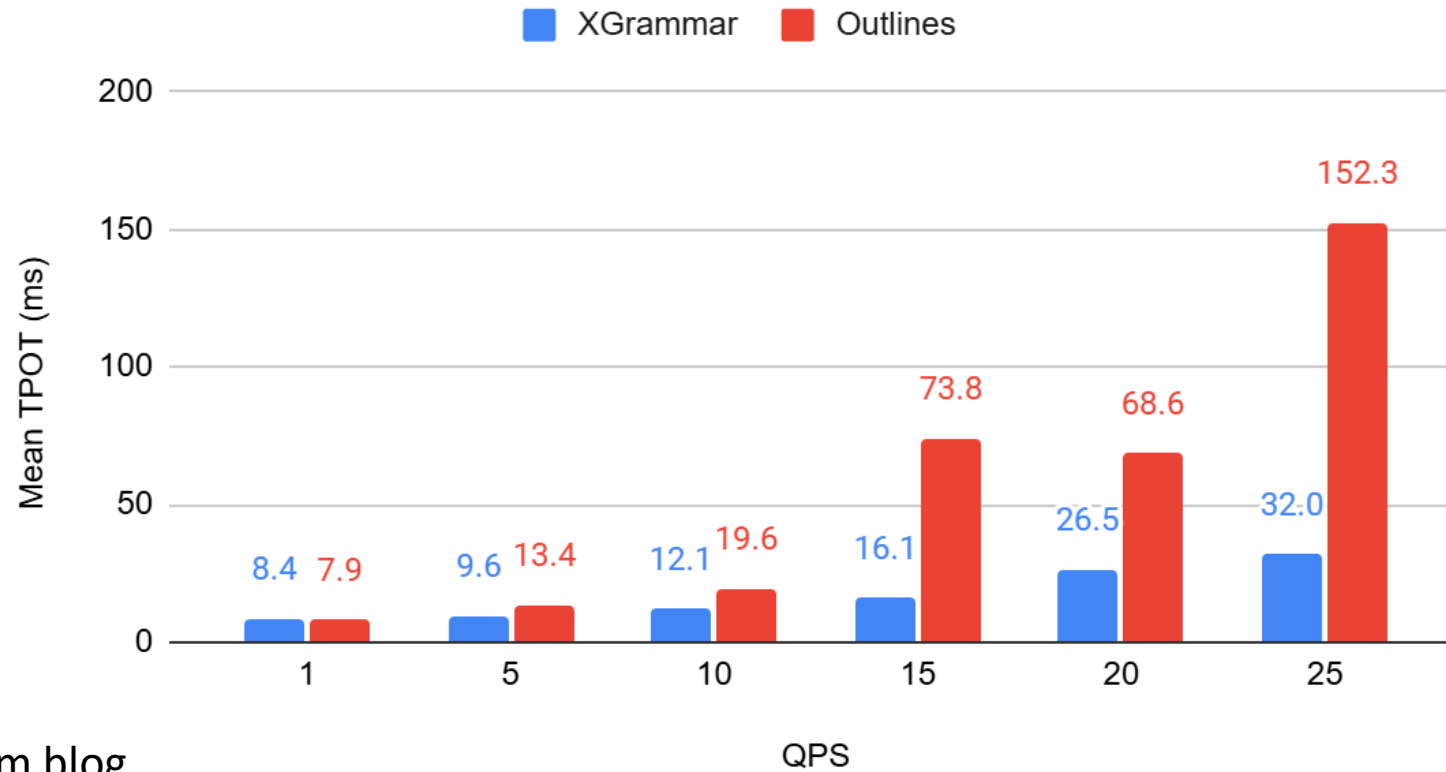
End to end Comparison in XGrammar



XGrammar in vLLM

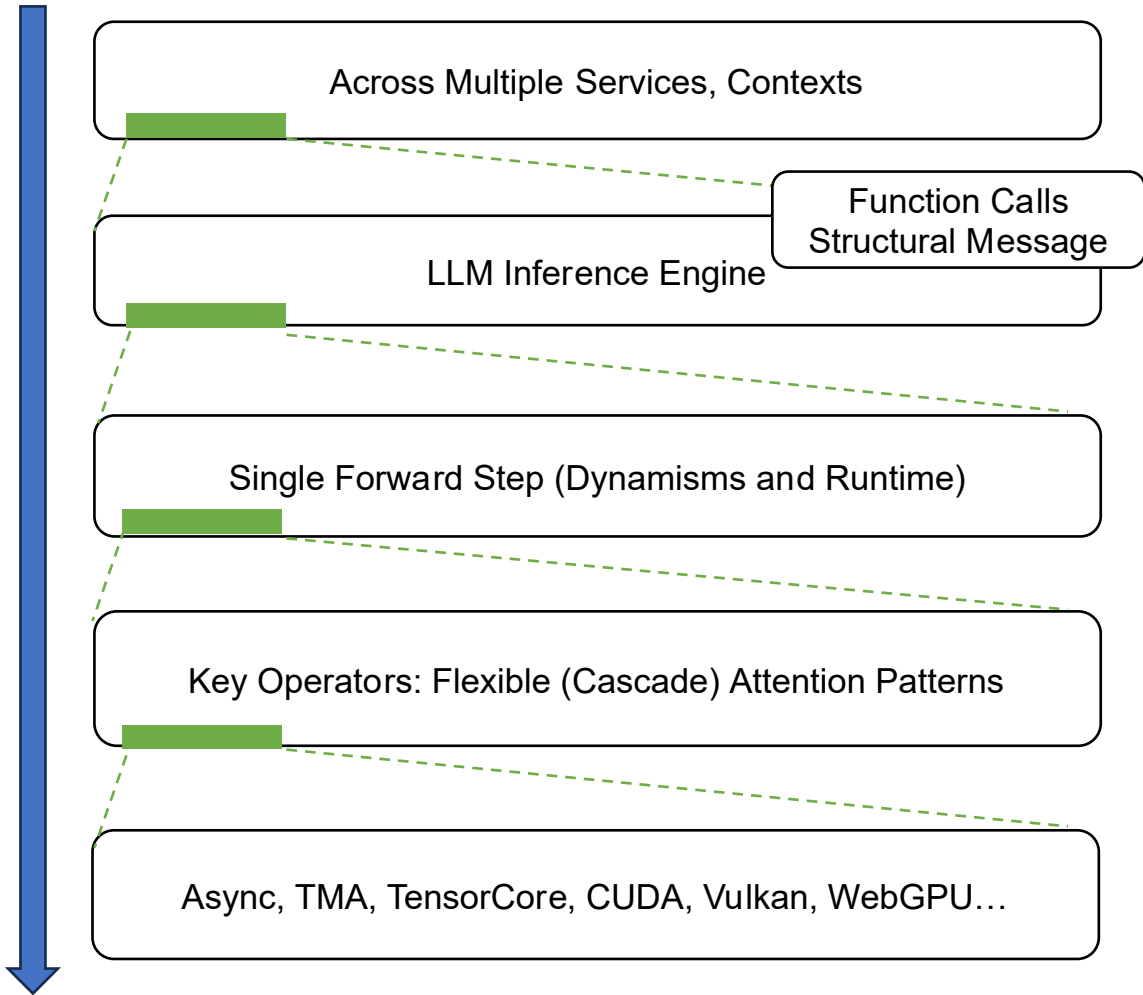
vLLM Guided Decoding Time per output token

Llama-3.1-8B | H100 GPU | NousResearch/json-mode-eval | 50% guided decoding

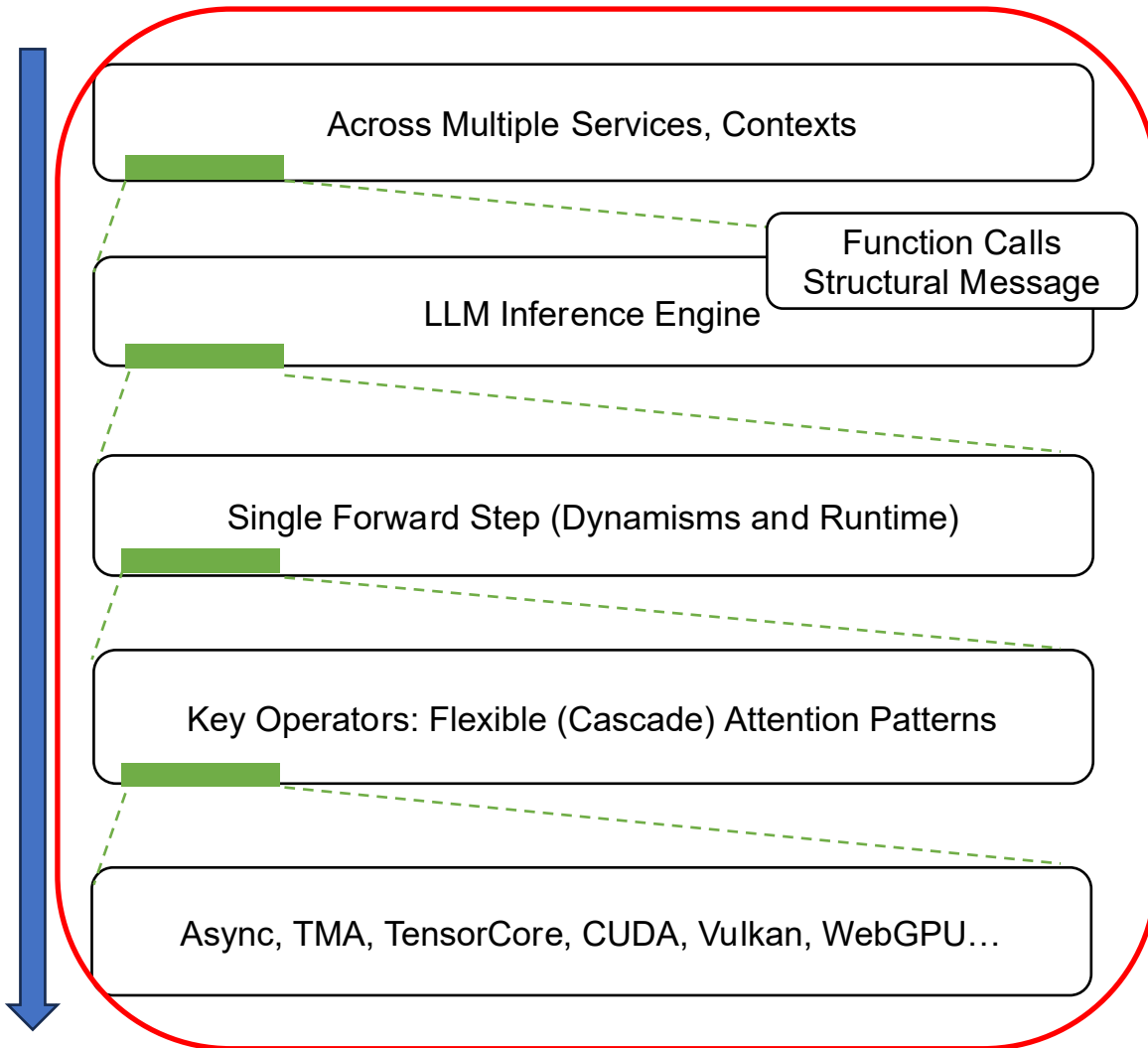


Source: vllm blog

Outline

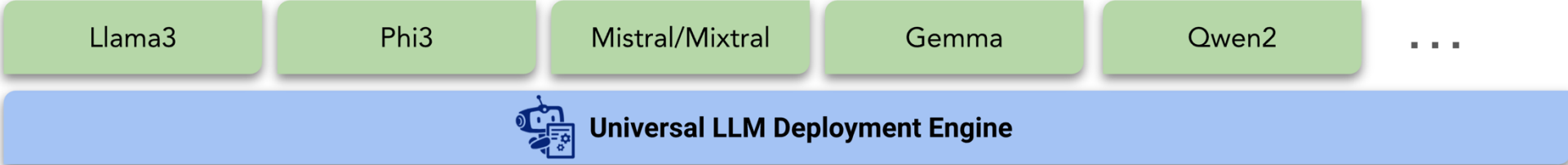


Outline



End to end system solution

MLCEngine: Universal LLM Deployment



Cloud Server



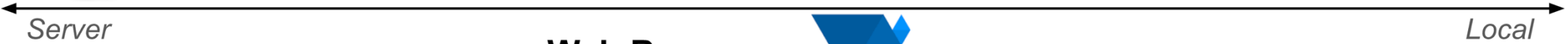
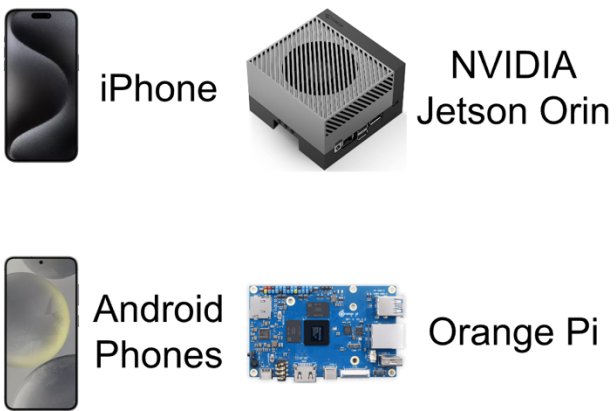
Desktop / Laptop



Tablet



Edge / Robot



Web Browsers  WebGPU

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-q4f16-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-q4f16-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-q4f16-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: opencl:0
[2024-06-05 17:37:07] INFO auto_device.py:79: Found device: opencl: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)

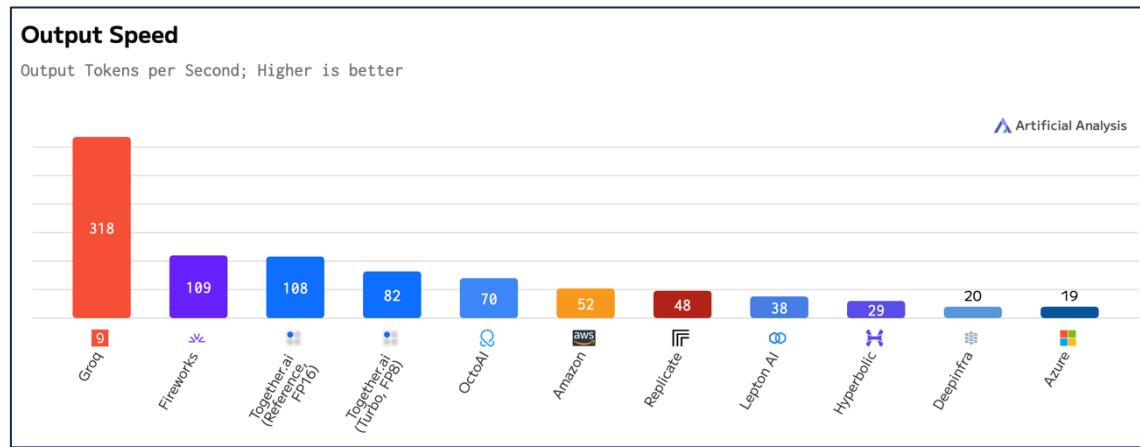
~ > curl -X POST \
-H "Content-Type: application/json" \
-d '{
  "model": "Llama-3-8B-Instruct-q0f16-MLC",
  "messages": [
    {"role": "user", "content": "Hello! This is project MLC LLM."},
    {"role": "assistant", "content": "Hello! It is great to work with you on project MLC LLM."},
    {"role": "user", "content": "Do you remember our project name?"}
  ]
}' \
http://127.0.0.1:8000/v1/chat/completions
ruihang@catalyst-nv8180 05:37:01 PM
```

Low-latency Server GPU Serving

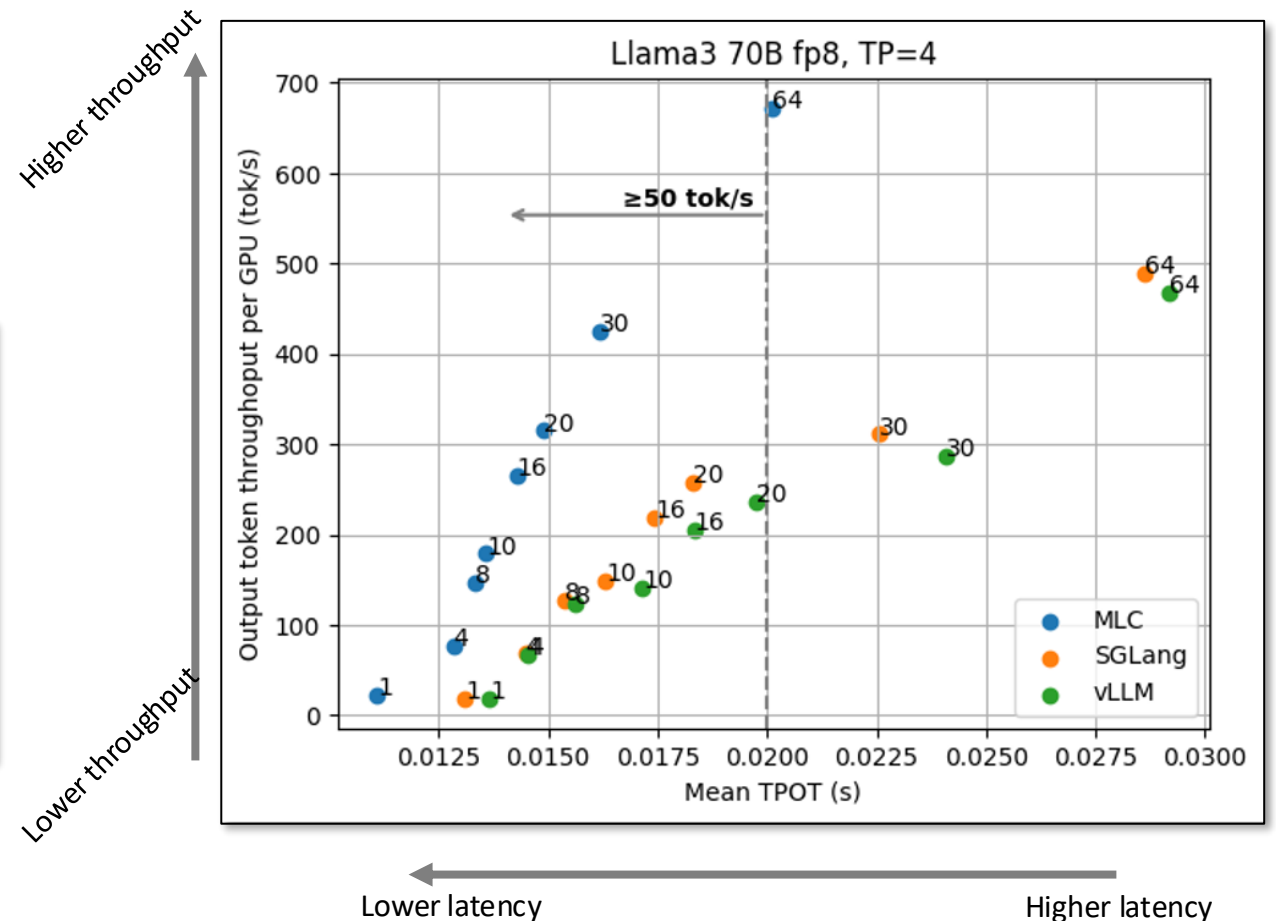
Fix the request concurrency to be 1 to 64

- Dataset: ShareGPT
- GPU: 4 x H100
- 500 requests

Overall 3% of CPU overhead in batch decode



Llama3 70B provider latency leaderboard



iOS SDK

OpenAI-style swift API

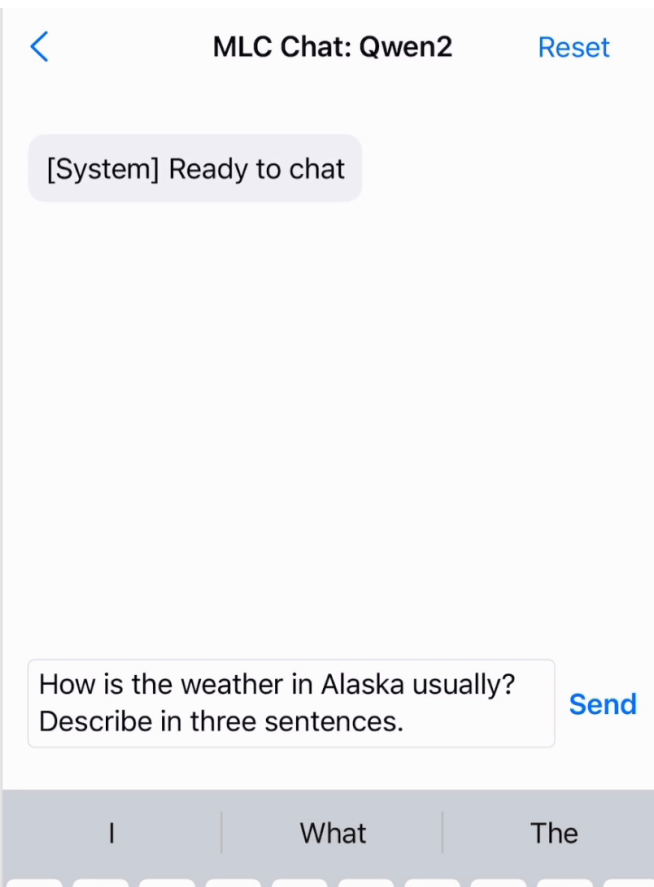
Demo on AppStore

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)
        )

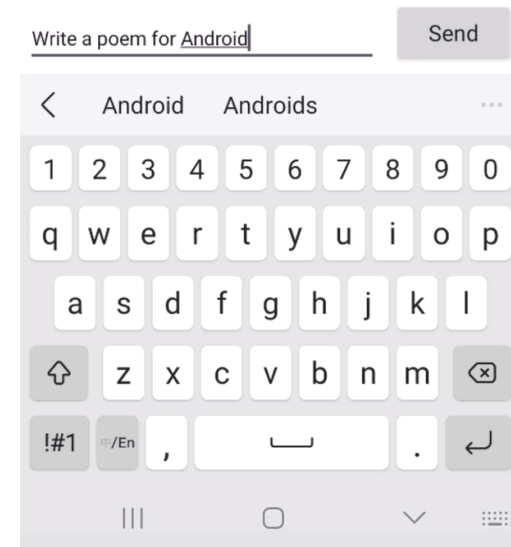
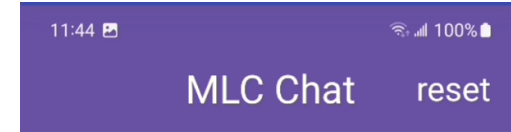
        var finishReasonLength = false
        for await res in await engine.chat.completions.create(
            messages: self.historyMessages,
            stream_options: StreamOptions(include_usage: true)
        ) {
            for choice in res.choices {
                if let content = choice.delta.content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



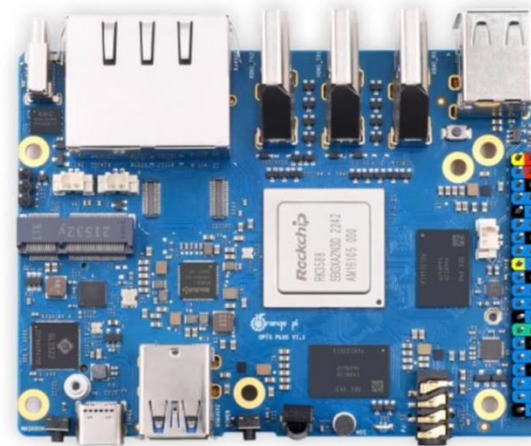
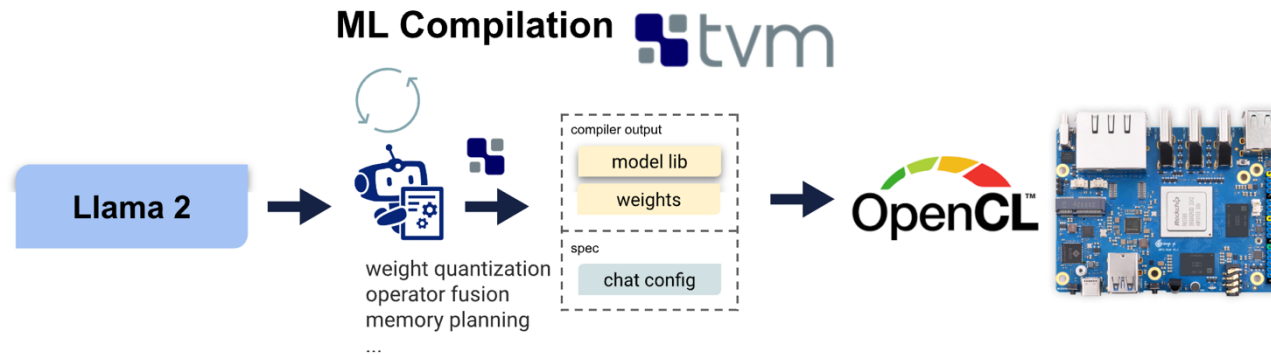
MLC LLM: Android

Snapdragon Gen2

Enables larger models than iPhone



Bringing LLMs to 100\$ Orange Pi



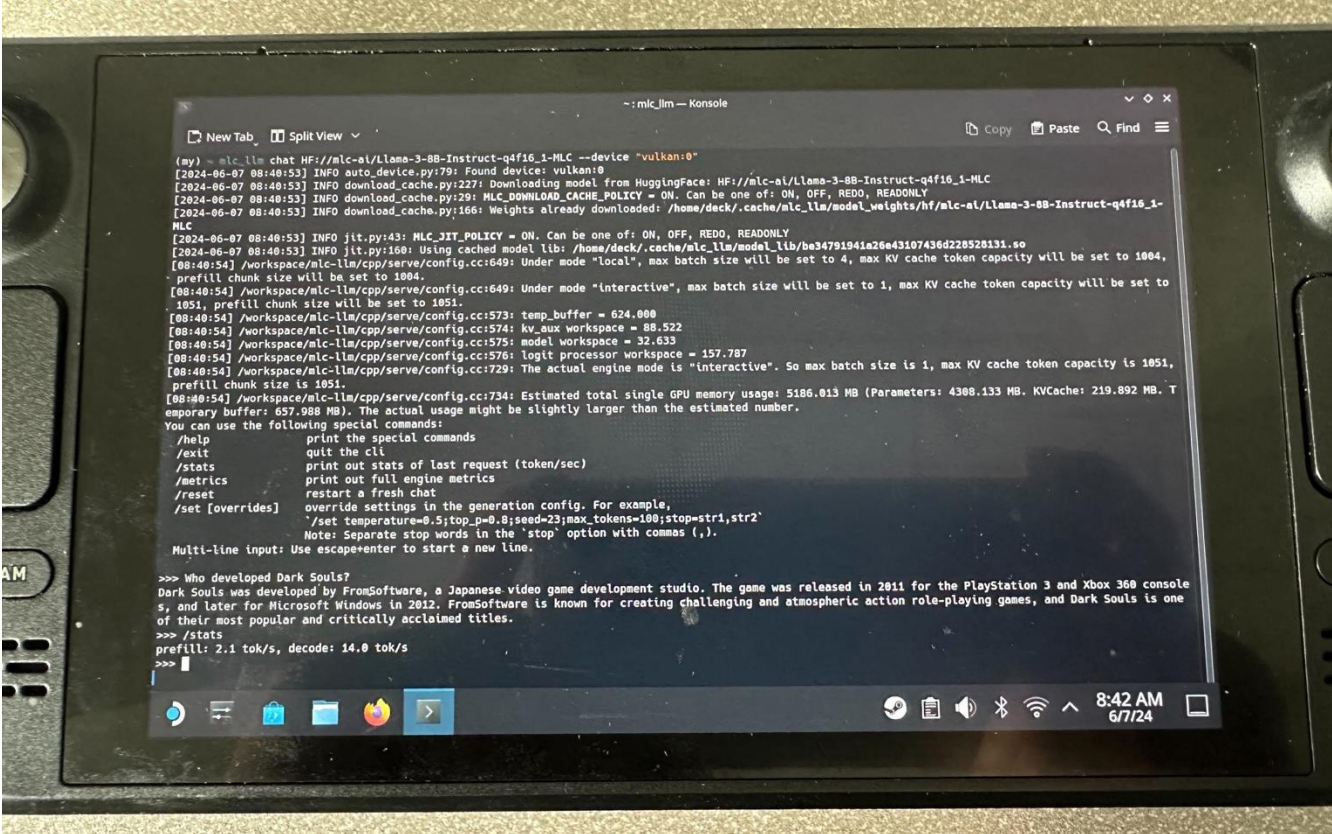
```
chris@chris-rk3588: ~/Documents/mlc_chat_cli
(GPT) chris@chris-rk3588:~/Documents/mlc_chat_cli$ mlc_chat_cli --local-id mlc-chat-Llama-2-7b-chat-hf-q4f16_1
Use MLC config: "/home/chris/Documents/mlc_chat_cli/dist/prebuilt/mlc-chat-Llama-2-7b-chat-hf-q4f16_1/mlc-chat-config.json"
Use model weights: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



```
..:mlc_llm - Konsole
[my] ~ mlc_llm chat HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC --device "vulkan:0"
[2024-06-07 08:40:53] INFO auto_device.py:79: Found device: vulkan:0
[2024-06-07 08:40:53] INFO download_cache.py:227: Downloading model from HuggingFace: HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC
[2024-06-07 08:40:53] INFO download_cache.py:29: MLC_DOWNLOAD_CACHE_POLICY = ON. Can be one of: ON, OFF, REDO, READONLY
[2024-06-07 08:40:53] INFO download_cache.py:166: Weights already downloaded: /home/deck/.cache/mlc_llm/model_weights/hf/mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC
[2024-06-07 08:40:53] INFO jit.py:43: MLC_JIT_POLICY = ON. Can be one of: ON, OFF, REDO, READONLY
[2024-06-07 08:40:53] INFO jit.py:169: Using cached model lib: /home/deck/.cache/mlc_llm/model_lib/be34791941a26e43107436d228528131.so
[08:40:54] /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 1004,
  prefill chunk size will be set to 1004.
[08:40:54] /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
  1051, prefill chunk size will be set to 1051.
[08:40:54] /workspace/mlc-llm/cpp/serve/config.cc:573: temp_buffer = 624.000
[08:40:54] /workspace/mlc-llm/cpp/serve/config.cc:574: kv_aux workspace = 88.522
[08:40:54] /workspace/mlc-llm/cpp/serve/config.cc:575: model workspace = 32.633
[08:40:54] /workspace/mlc-llm/cpp/serve/config.cc:576: logit processor workspace = 157.797
[08:40:54] /workspace/mlc-llm/cpp/serve/config.cc:729: The actual engine mode is "interactive". So max batch size is 1, max KV cache token capacity is 1051,
  prefill chunk size is 1051.
[08:40:54] /workspace/mlc-llm/cpp/serve/config.cc:734: Estimated total single GPU memory usage: 5186.013 MB (Parameters: 4308.133 MB. KVCache: 219.892 MB. T
  emporary buffer: 657.988 MB). The actual usage might be slightly larger than the estimated number.
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.

>>> Who developed Dark Souls?
Dark Souls was developed by FromSoftware, a Japanese video game development studio. The game was released in 2011 for the PlayStation 3 and Xbox 360 console
s, and later for Microsoft Windows in 2012. FromSoftware is known for creating challenging and atmospheric action role-playing games, and Dark Souls is one
of their most popular and critically acclaimed titles.
>>> /stats
prefill: 2.1 tok/s, decode: 14.0 tok/s
>>> |
```

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())

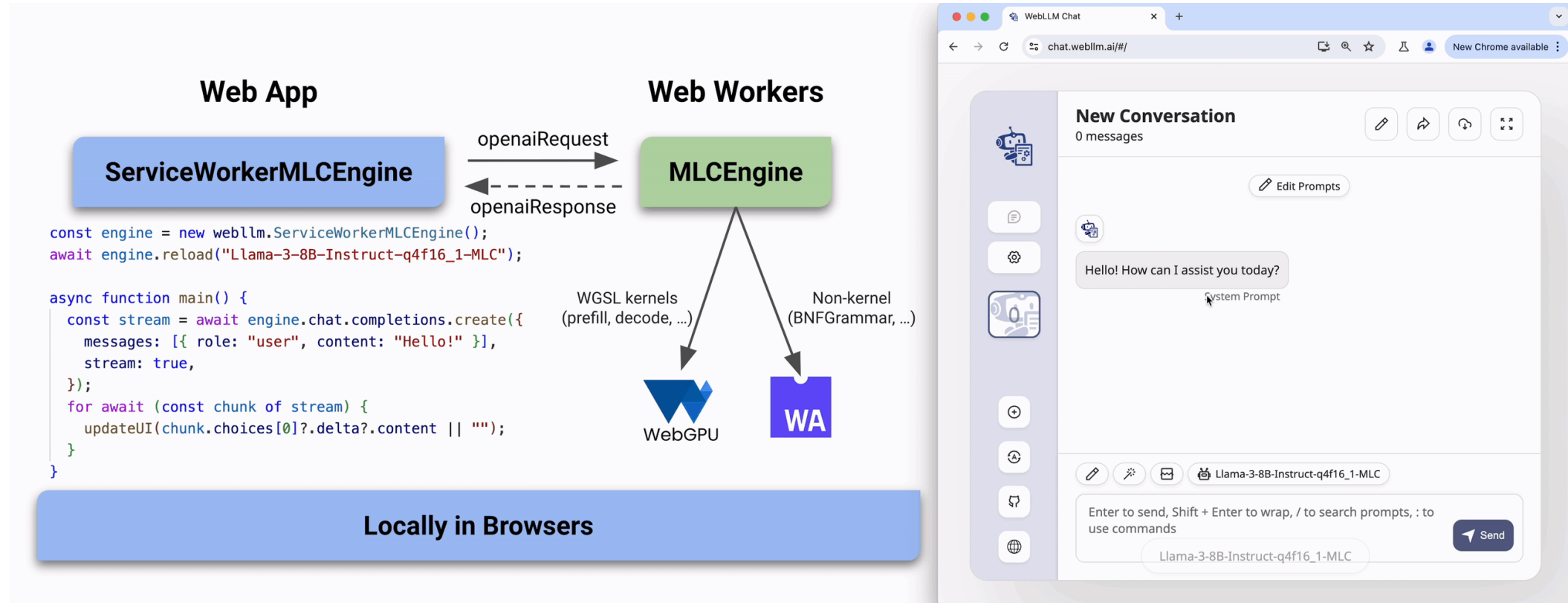
In [10]: response = engine.chat.completions.create(
...:     messages=[{"role": "user", "content": prompt}],
...:     response_format={"type": "json_object", "schema": schema},
...: )

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>

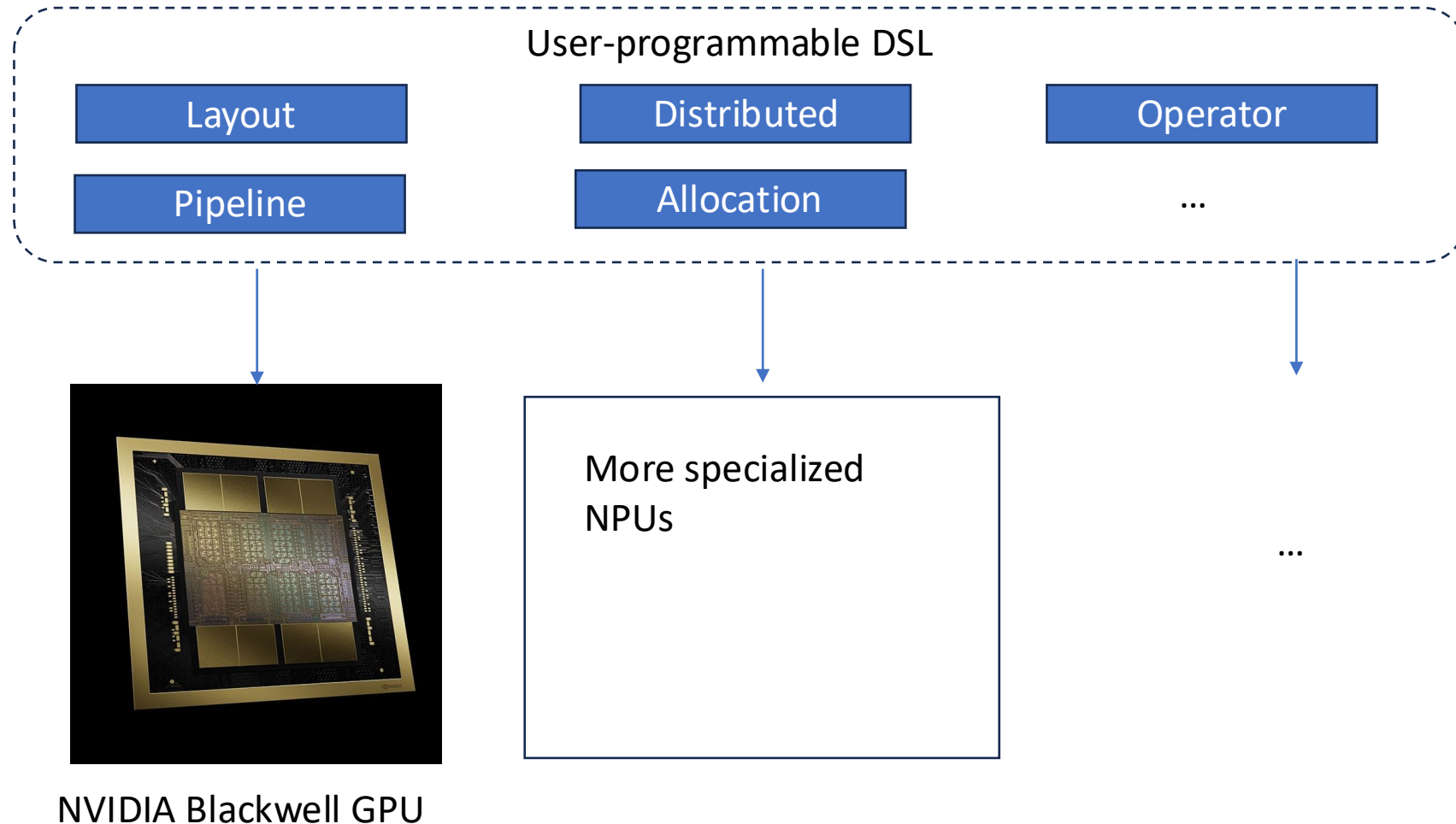
WebLLM



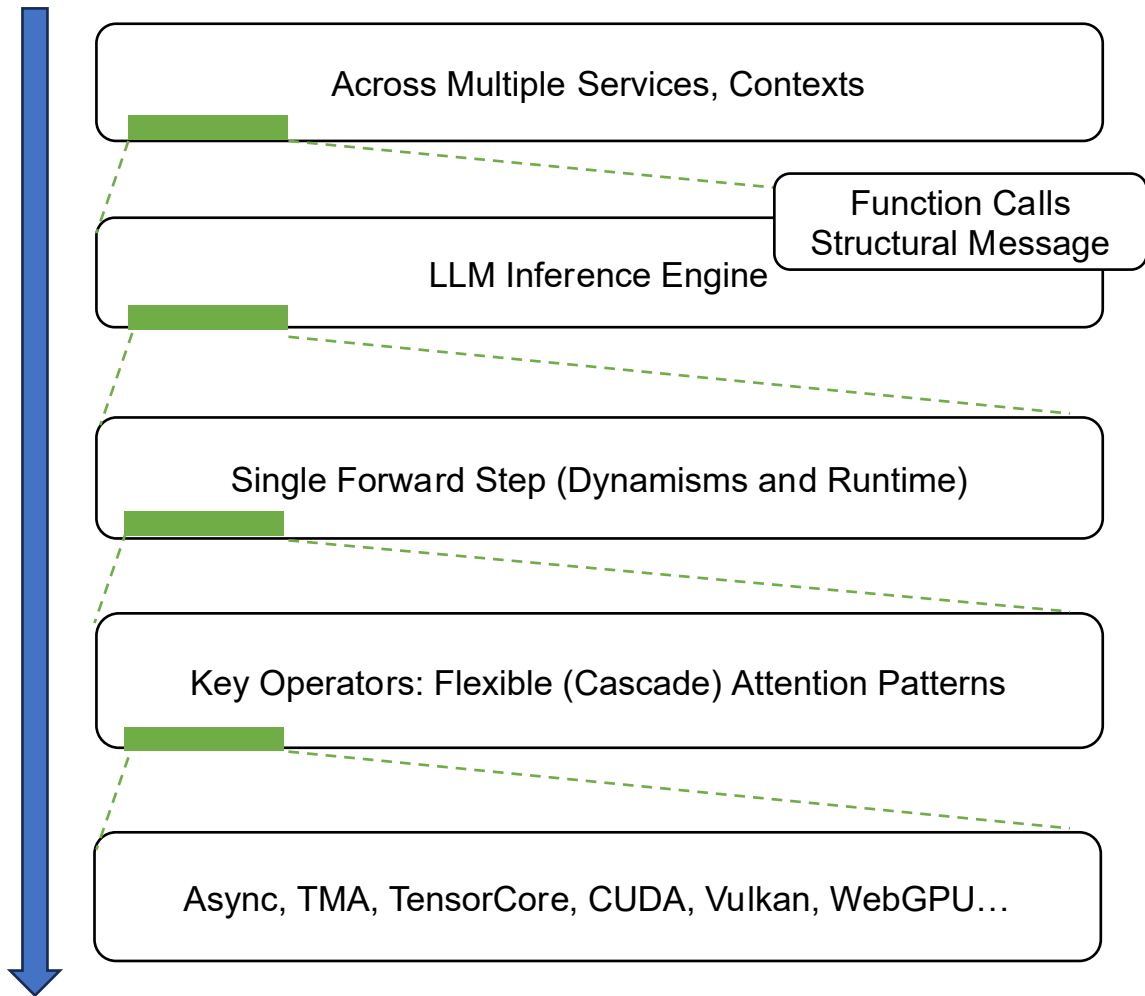
Runs directly in browser client <https://webllm.mlc.ai/>

Ongoing: Unified Abstraction for Future GPUs and Accelerators

Allow both fine-grained and coarse-grained user control



Ongoing directions



- Efficient tensor compiler abstractions for new generations of GPUs and accelerators
- Compiler runtime techniques for scaling up (across devices, nodes)
- Putting things together: end to end systems for serving/tuning large models
- Open source