Universal LLM deployment and low-latency serving in MLC LLM

Ruihang Lai, CMU

Oct 16, 2024

MACHINE LEARNING
COMPILATION

Outline

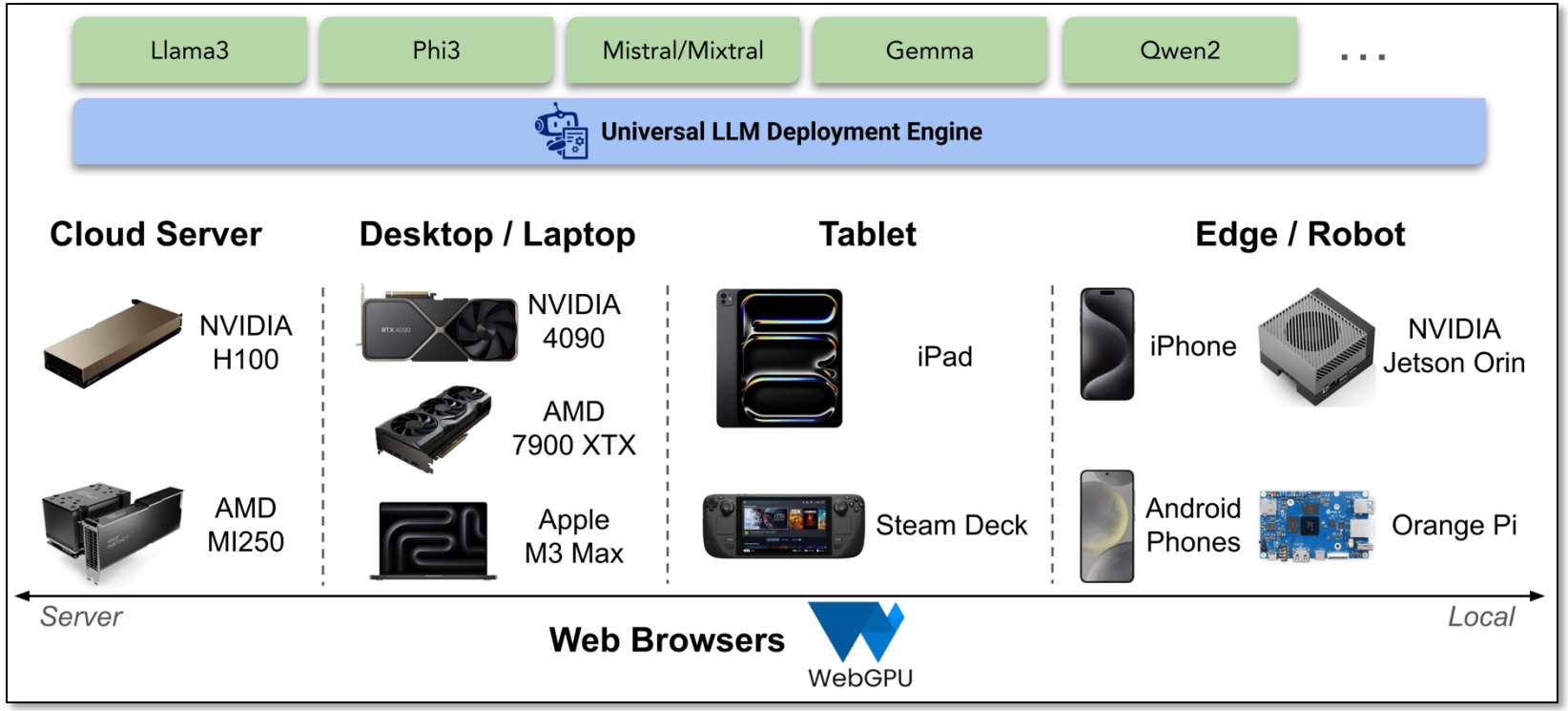
- Part I. Universal LLM deployment in MLCEngine
- Part II. Low-latency serving study in MLCEngine

Outline

- Part I. Universal LLM deployment in MLCEngine
- Part II. Low-latency serving study in MLCEngine

What is MLC LLM?

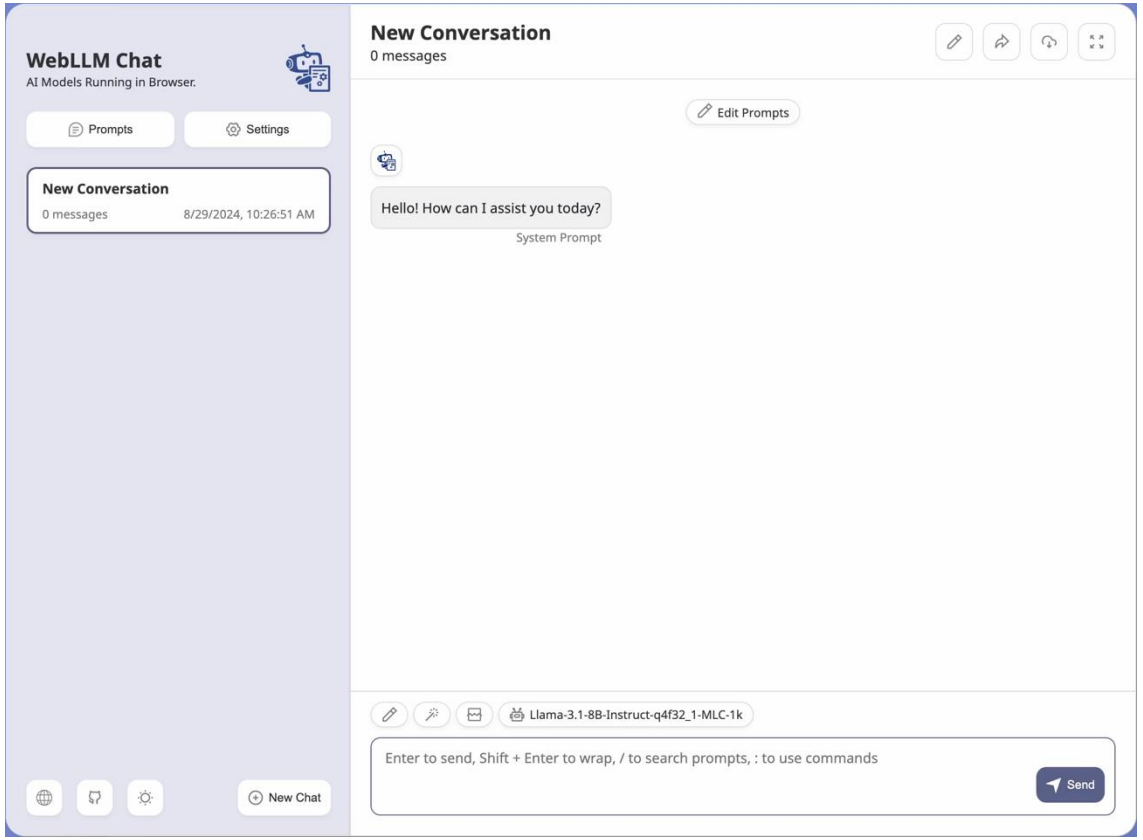
A high-performance universal LLM engine that allows native deployment any large language models with native APIs with compiler acceleration.



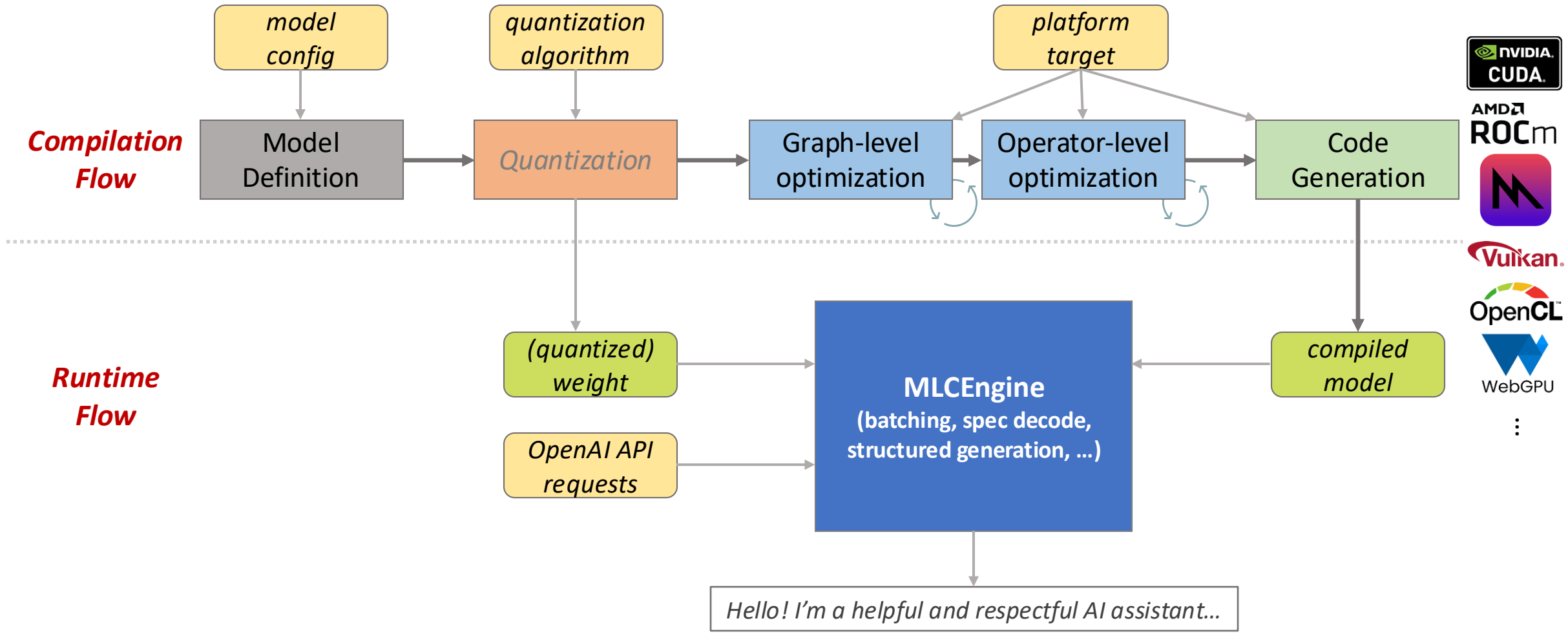
Demo

- iOS & Android App https://llm.mlc.ai

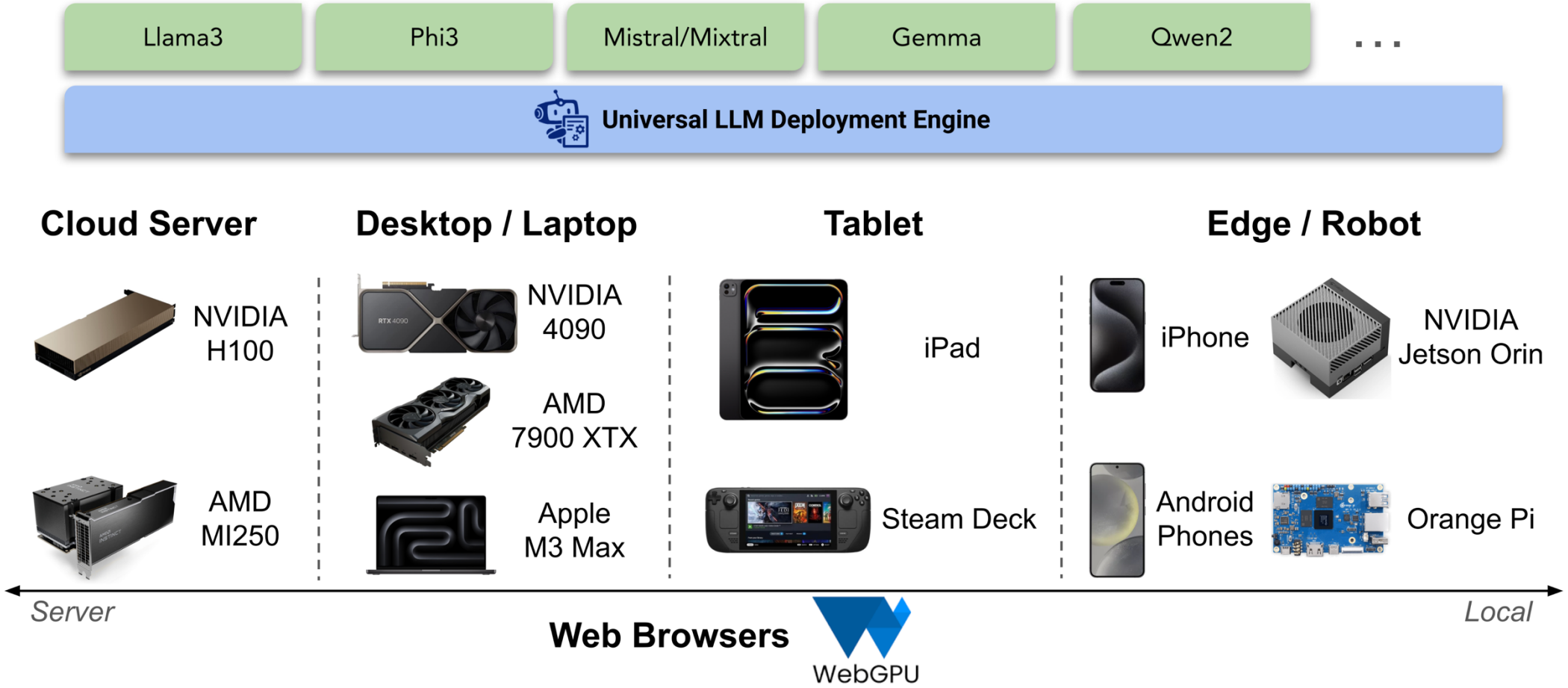
- Web LLM https://webllm.mlc.ai



Architecture of MLC LLM



Why Universal Engine?



APIs of MLCEngine

Core: OpenAI API

```python
class ChatCompletionRequest(BaseModel):
    """OpenAI chat completion request protocol.
    API reference: https://platform.openai.com/docs/api-reference/chat/create
    """

    messages: List[ChatCompletionMessage]
    model: Optional[str] = None
    frequency_penalty: Optional[float] = None
    presence_penalty: Optional[float] = None
    logprobs: bool = False
    top_logprobs: int = 0
    logit_bias: Optional[Dict[int, float]] = None
    max_tokens: Optional[int] = None
    n: int = 1
    seed: Optional[int] = None
    stop: Optional[Union[str, List[str]]] = None
    stream: bool = False
    stream_options: Optional[StreamOptions] = None
    temperature: Optional[float] = None
    top_p: Optional[float] = None
    tools: Optional[List[ChatTool]] = None
    tool_choice: Optional[Union[Literal["none", "auto"], Dict]] = None
    user: Optional[str] = None
    response_format: Optional[RequestResponseFormat] = None
```

OpenAI API ChatCompletion

```
> mlc_llm serve HF://mlc-ai/Llama-3-8B-Instruct-q0f16-MLC
```

Launch Server

```
~ ❯ curl -X POST \
  -H "Content-Type: application/json" \
  -d '{
        "messages": [
            {"role": "user", "content": "When is the recent solar eclipse in the U.S.?"}
        ]
  }' \
  http://127.0.0.1:8000/v1/chat/completions
```

Sending HTTP requests

APIs of MLCEngine

Core: OpenAI API

```python
from openai import OpenAI

# Initialize client
client = OpenAI(base_url=OPENAI_BASE_URL)


# Run chat completion.
prompt = "What is the meaning of life?"
for response in client.chat.completions.create(
    model="gpt-4",
    messages=[{"role": "user", "content": prompt}],
    stream=True,
):
    ...
```

OpenAI Python package

```python
from mlc_llm import MLCEngine

# Create engine
model = "HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC"
engine = MLCEngine(model)

# Run chat completion in OpenAI API.
prompt = "What is the meaning of life?"
for response in engine.chat.completions.create(
    messages=[{"role": "user", "content": prompt}],
    stream=True,
):
    for choice in response.choices:
        print(choice.delta.content, end="", flush=True)
```

MLCEngine Python API

APIs of MLCEngine

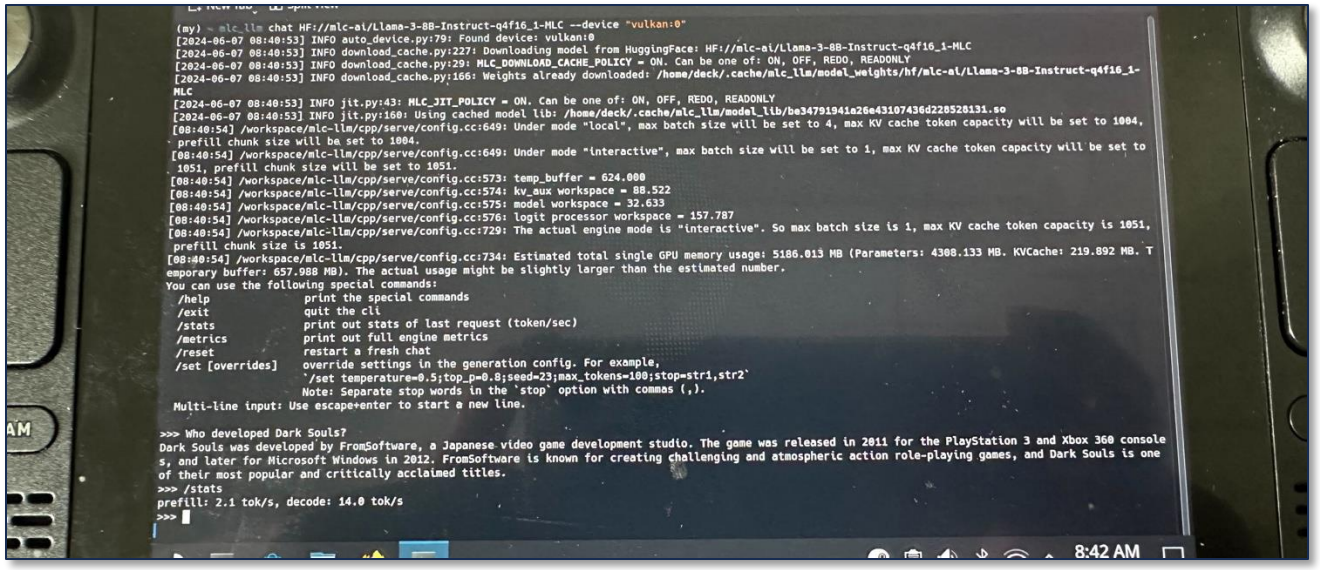
Core: OpenAI API

```python
from mlc_llm import MLCEngine

# Create engine
model = "HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC"
engine = MLCEngine(model)

# Run chat completion in OpenAI API.
prompt = "What is the meaning of life?"
for response in engine.chat.completions.create(
    messages=[{"role": "user", "content": prompt}],
    stream=True,
):
    for choice in response.choices:
        print(choice.delta.content, end="", flush=True)
```

MLCEngine Python API

wrap

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



Interactive chat session

APIs of MLCEngine

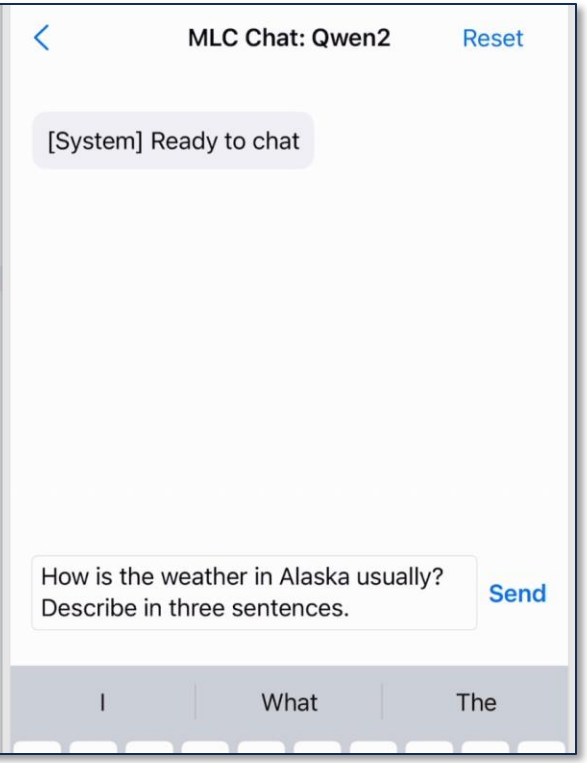
Core: OpenAI API

```python
from mlc_llm import MLCEngine

# Create engine
model = "HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC"
engine = MLCEngine(model)

# Run chat completion in OpenAI API.
prompt = "What is the meaning of life?"
for response in engine.chat.completions.create(
    messages=[{"role": "user", "content": prompt}],
    stream=True,
):
    for choice in response.choices:
        print(choice.delta.content, end="", flush=True)
```

MLCEngine Python API

mimic

```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
                    }
                }
            }
        }
    }
```



iOS SDK in Swift

APIs of MLCEngine

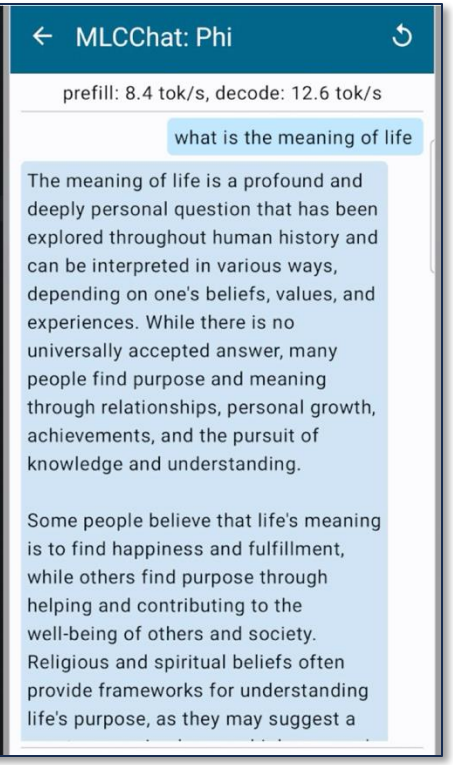
Core: OpenAI API

```python
from mlc_llm import MLCEngine

# Create engine
model = "HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC"
engine = MLCEngine(model)

# Run chat completion in OpenAI API.
prompt = "What is the meaning of life?"
for response in engine.chat.completions.create(
    messages=[{"role": "user", "content": prompt}],
    stream=True,
):
    for choice in response.choices:
        print(choice.delta.content, end="", flush=True)
```

MLCEngine Python API

mimic

```kotlin
fun requestGenerate(prompt: String) {
    require(chatable())
    switchToGenerating()
    executorService.submit {
        appendMessage(MessageRole.User, prompt)
        appendMessage(MessageRole.Assistant, text: "")
        viewModelScope.launch { this: CoroutineScope
            val channel = engine.chat.completions.create(
                messages = listOf(
                    ChatCompletionMessage(
                        role = OpenAIProtocol.ChatCompletionRole.user,
                        content = prompt
                    )
                )
            )
            var texts = ""
            for (response in channel) {
                if (!callBackend {
                    if (response.choices.isNotEmpty()) {
                        texts += response.choices[0].delta.content?.asText().orEmpty()
                    }
                    updateMessage(MessageRole.Assistant, texts)
                });
            }
            if (modelChatState.value == ModelChatState.Generating) switchToReady()
        }
    }
}
```



Android SDK in Kotlin

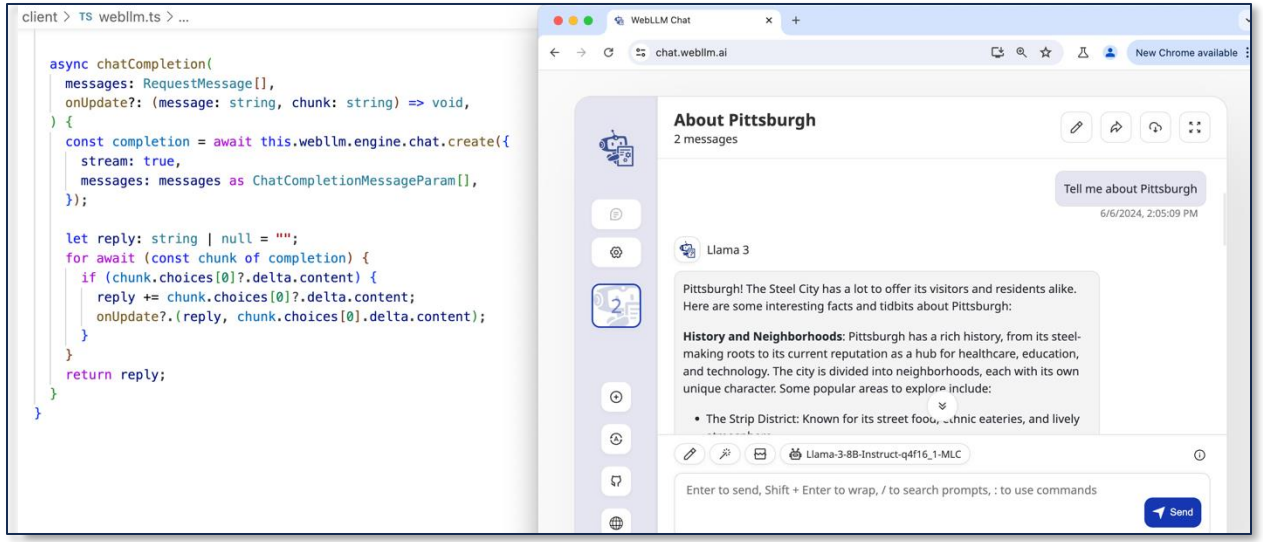
APIs of MLCEngine

Core: OpenAI API

```python
from mlc_llm import MLCEngine

# Create engine
model = "HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-MLC"
engine = MLCEngine(model)

# Run chat completion in OpenAI API.
prompt = "What is the meaning of life?"
for response in engine.chat.completions.create(
    messages=[{"role": "user", "content": prompt}],
    stream=True,
):
    for choice in response.choices:
        print(choice.delta.content, end="", flush=True)
```

MLCEngine Python API

mimic

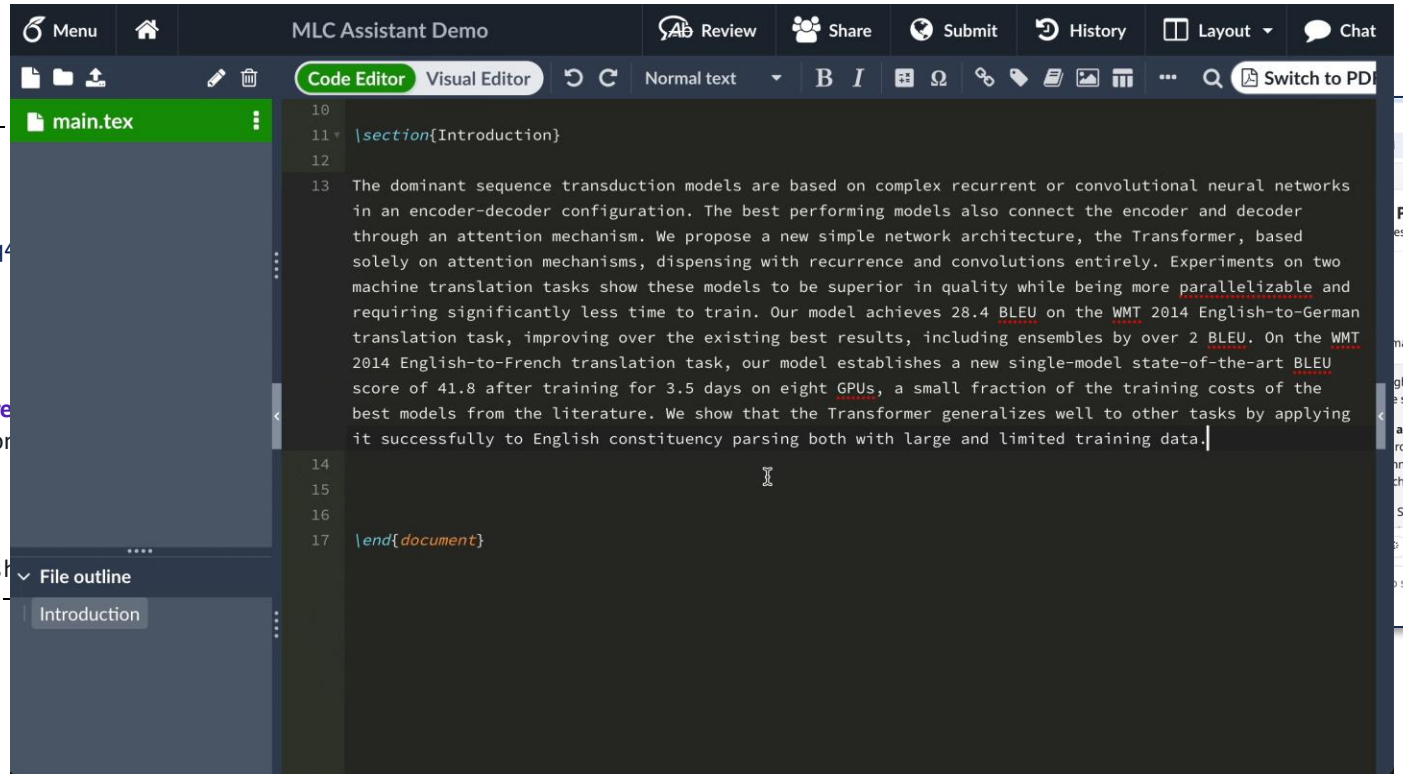


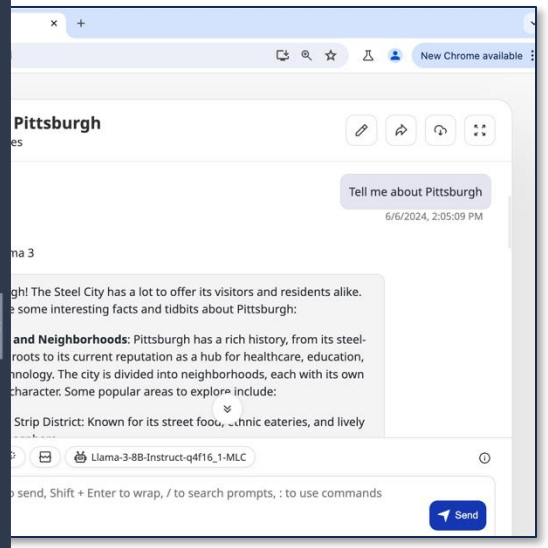
WebLLM SDK in TypeScript

APIs of MLCEngine

Core: OpenAI API

```
from mlc_llm import MLCEngine

# Create engine
model = "HF://mlc-ai/Llama-3-8B-Instruct-q4
engine = MLCEngine(model)

# Run chat completion in OpenAI API.
prompt = "What is the meaning of life?"
for response in engine.chat.completions.cre
    messages=[{"role": "user", "content": pro
    stream=True,
):
    for choice in response.choices:
        print(choice.delta.content, end="", flush
```





https://github.com/mlc-ai/Web-LLM-Assistant

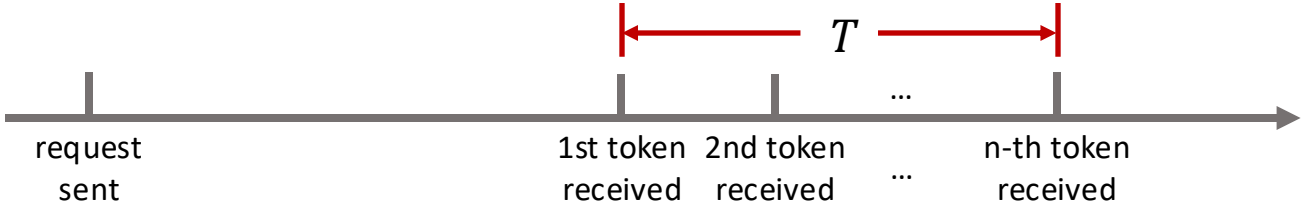
Outline

- Part I. Universal LLM deployment in MLCEngine
- Part II. Low-latency serving study in MLCEngine

Low-latency Serving

Latency is the Top Priority of MLC.

Reflected by TPOT (time per output token, the average number of tokens received per second after the first token is received)



$$\text{TPOT} = \frac{T}{n-1}$$

Complicated tradeoffs between latency and throughput:

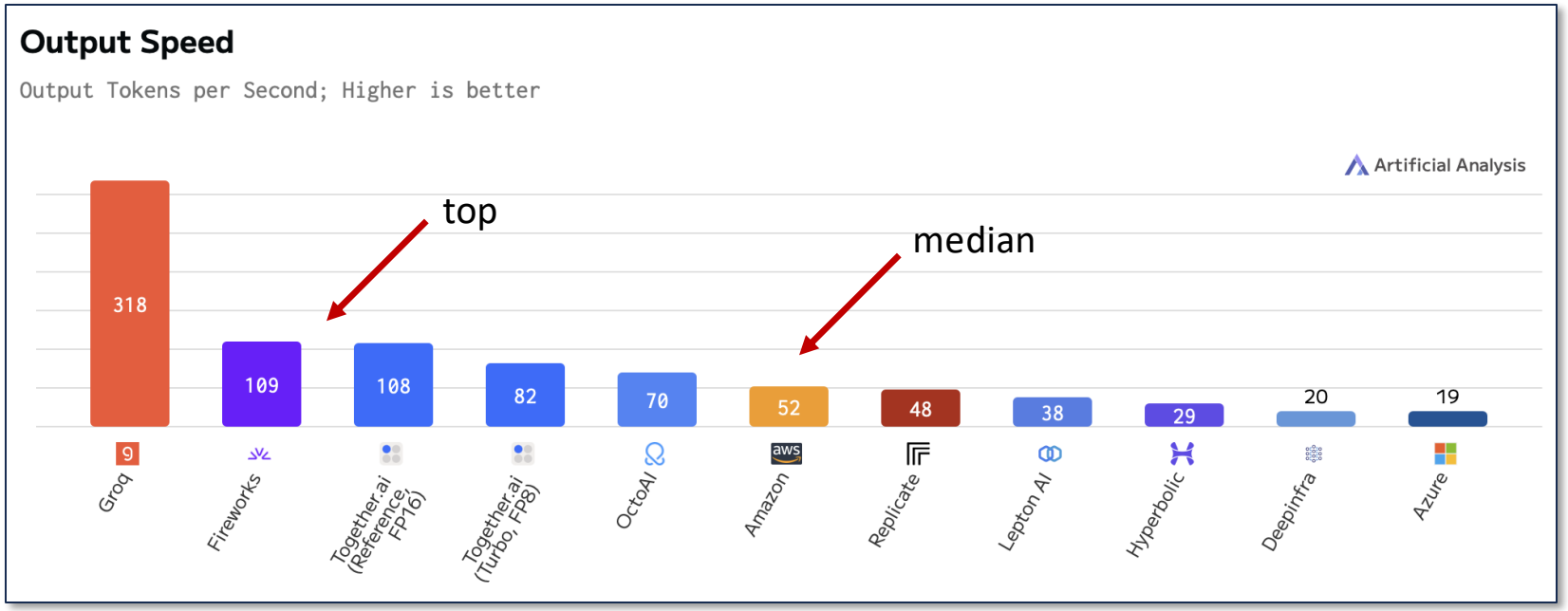
As an endpoint user:

- generation as fast as possible

As an endpoint provider:

- low latency for competitiveness
- reasonable throughput, or waste resource

Low-latency Serving



Llama3 70B provider latency leaderboard

https://artificialanalysis.ai/models/llama-3-instruct-70b/providers#speed

Low-latency Serving

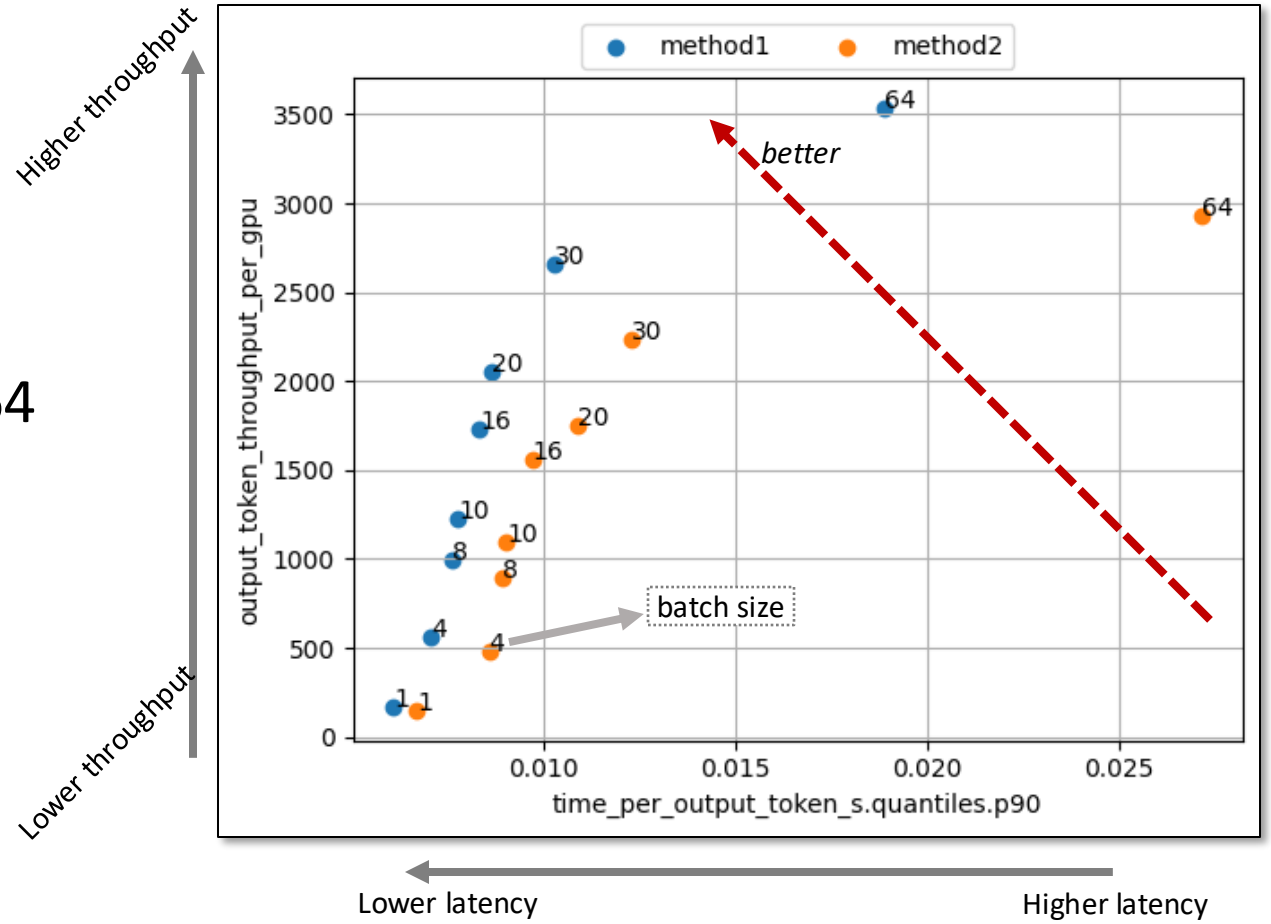
To understand the complication of latency and throughput…

Metrics

- Time-per-output-token (TPOT)
- Output throughput (tok/s)

Fix the request concurrency to be 1 to 64

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

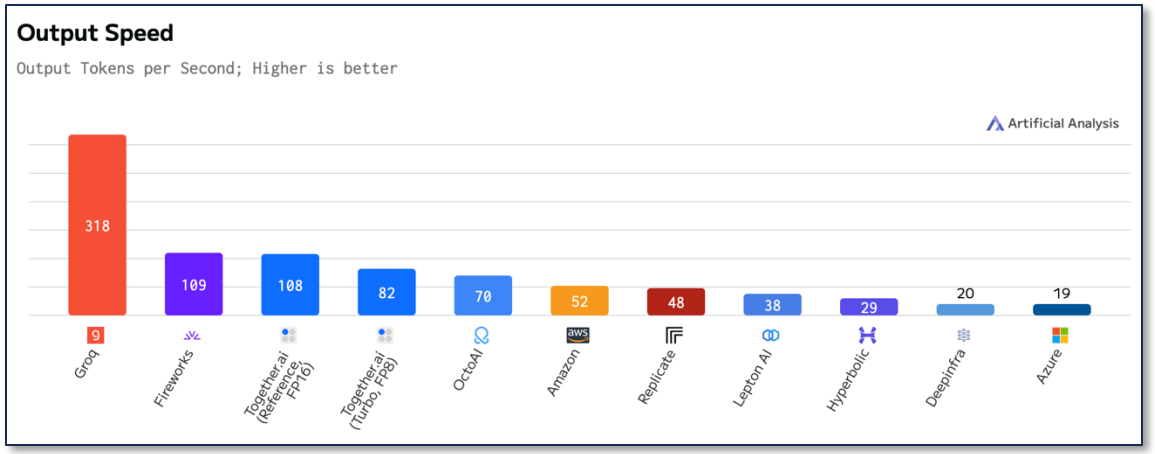


Low-latency Serving

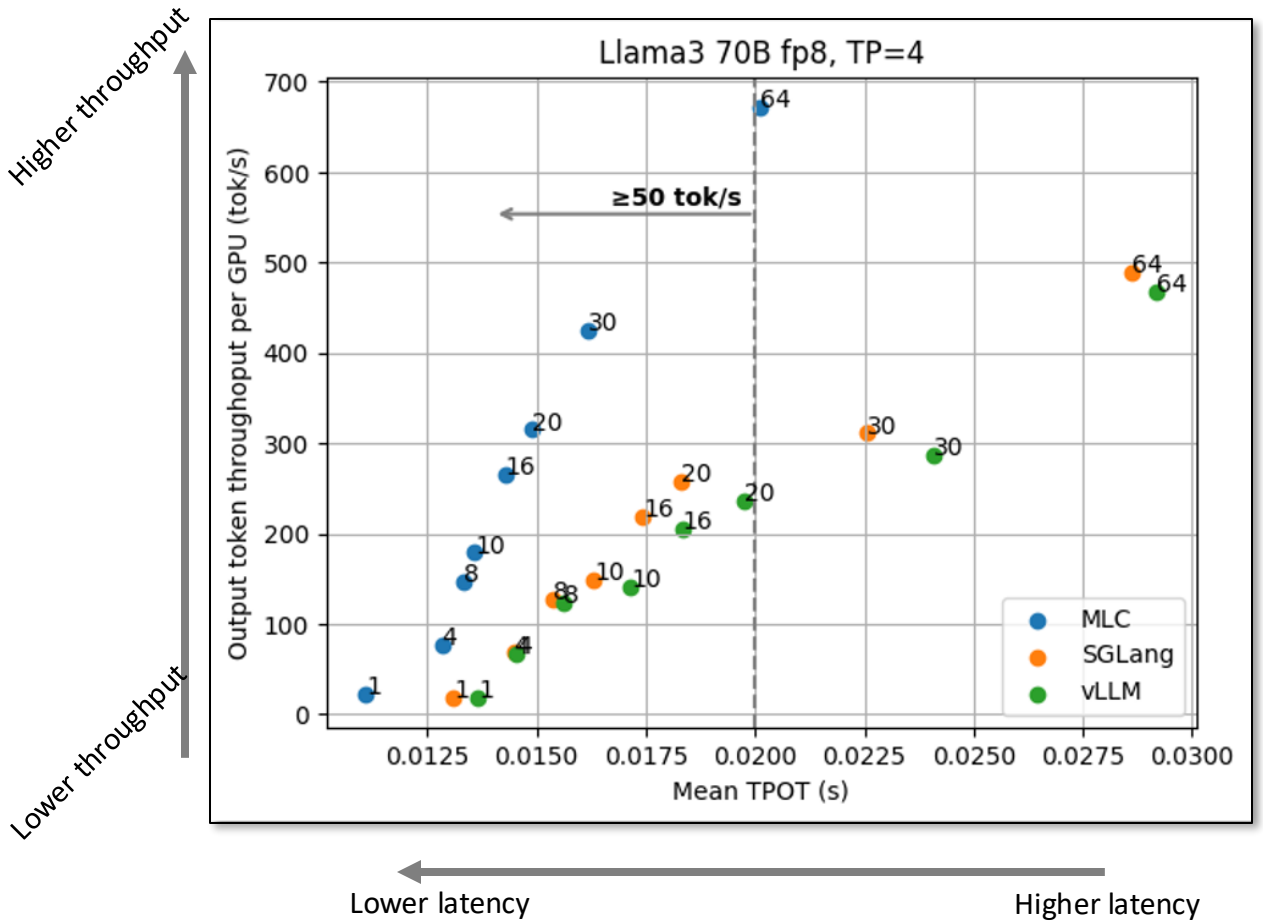
Fix the request concurrency to be 1 to 64

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

Overall 3% of CPU overhead in batch decode



Llama3 70B provider latency leaderboard

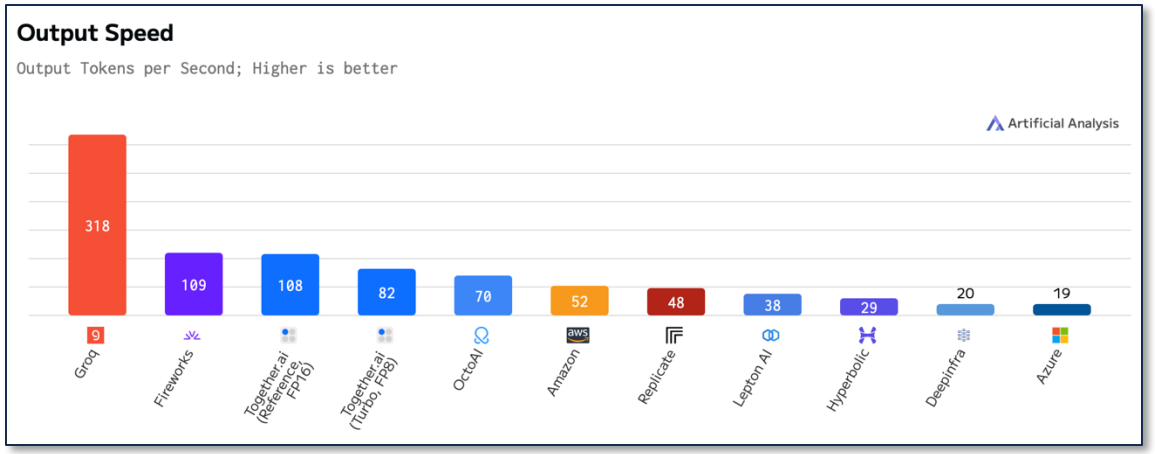


https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference, SGLang v0.3.1.post2, vLLM v0.6.1.post2

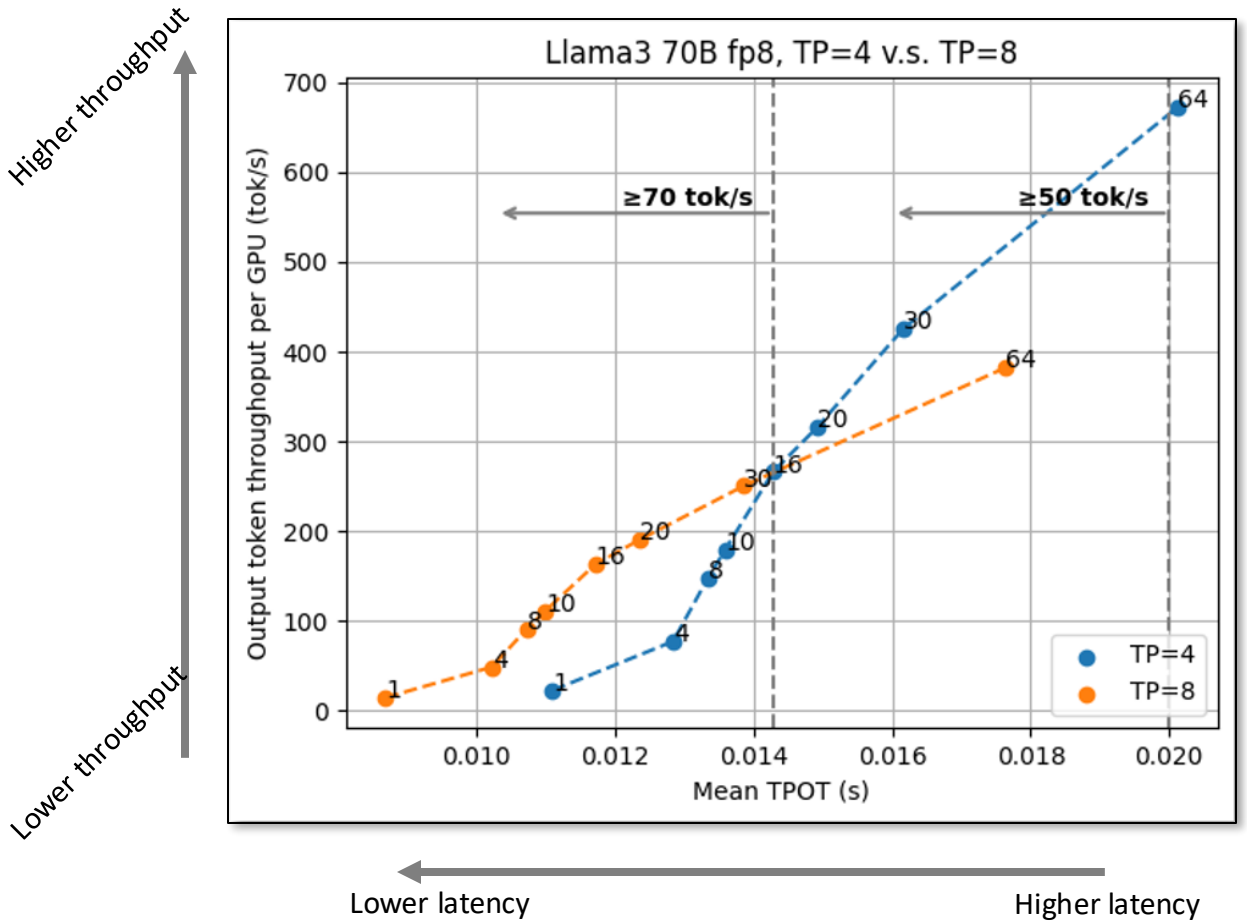
Impact of Tensor Parallelism

Running on 4 GPUs v.s. running on 8 GPUs

1. Reaching 100 tok/s
- But with poor throughput



Llama3 70B provider latency leaderboard



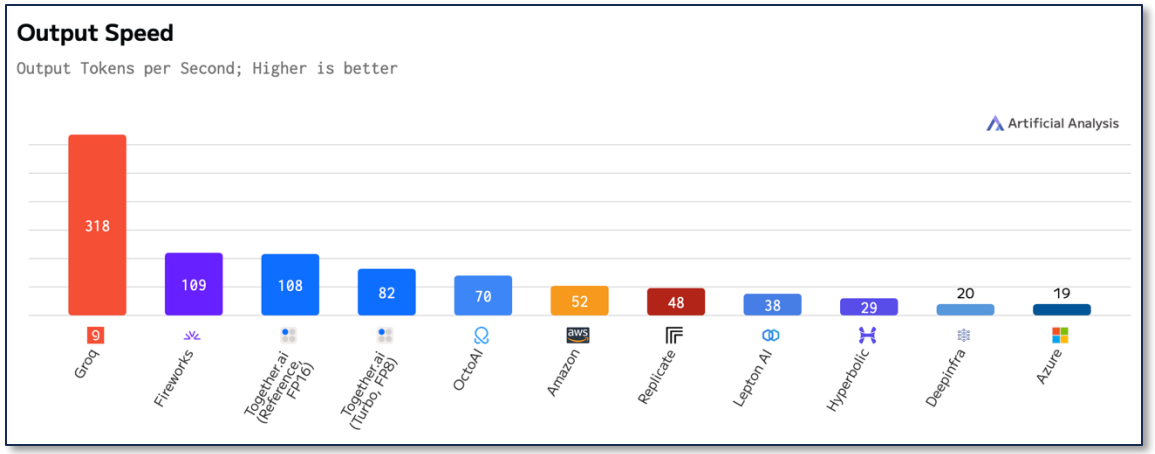
https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference

Impact of Tensor Parallelism

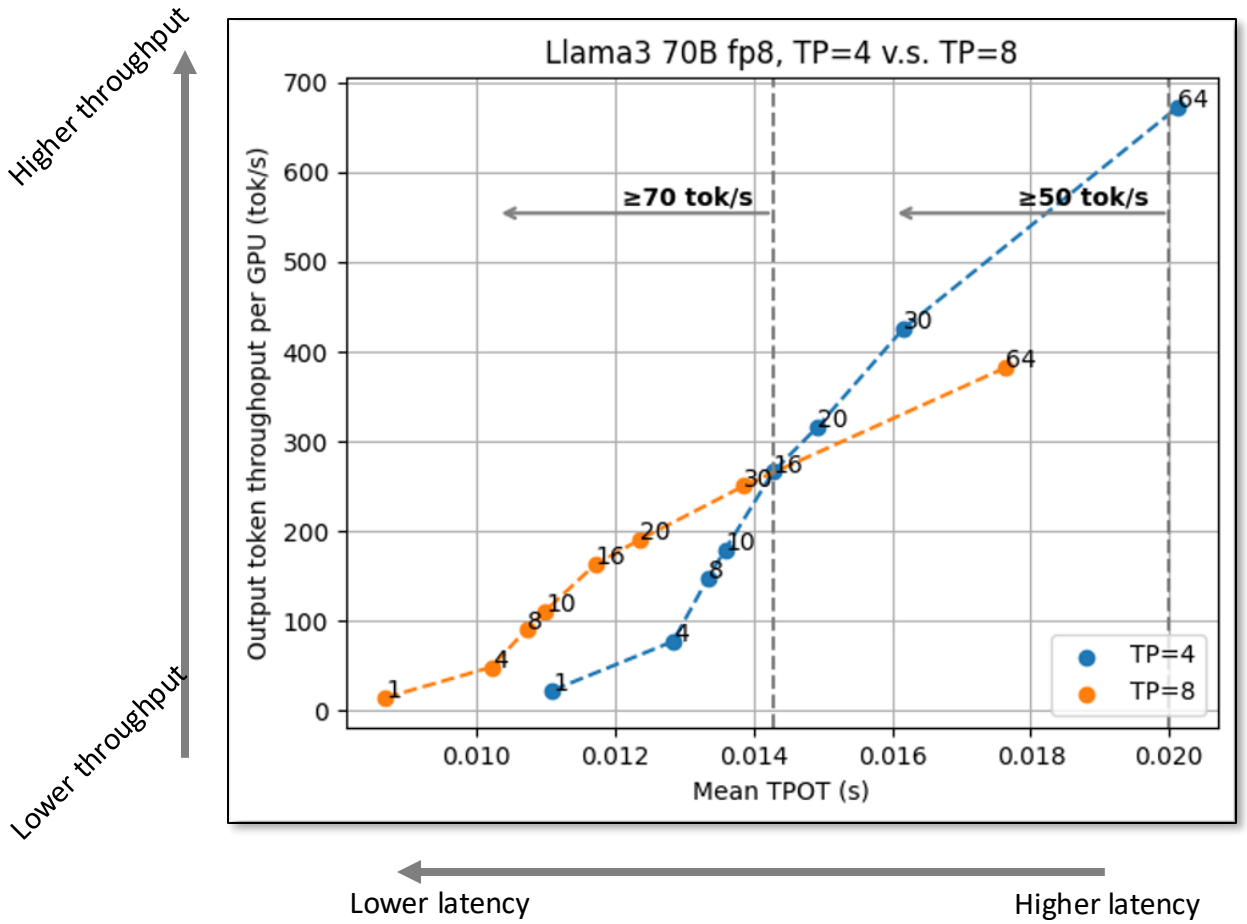
Running on 4 GPUs v.s. running on 8 GPUs

2. Choice is driven by the goal.

- ≥ 70 tok/s → choose TP=8
- < 70 tok/s → choose TP=4



Llama3 70B provider latency leaderboard



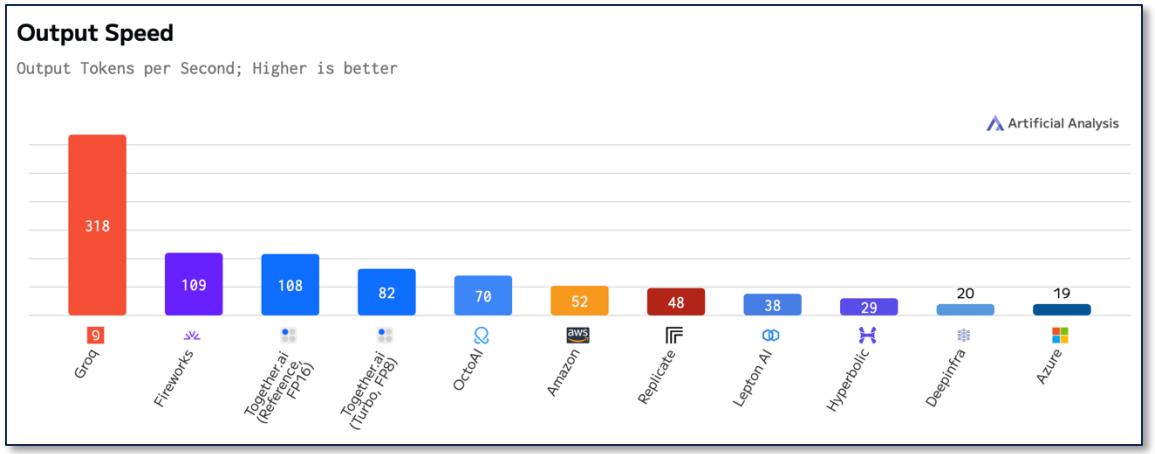
https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference

Impact of Tensor Parallelism

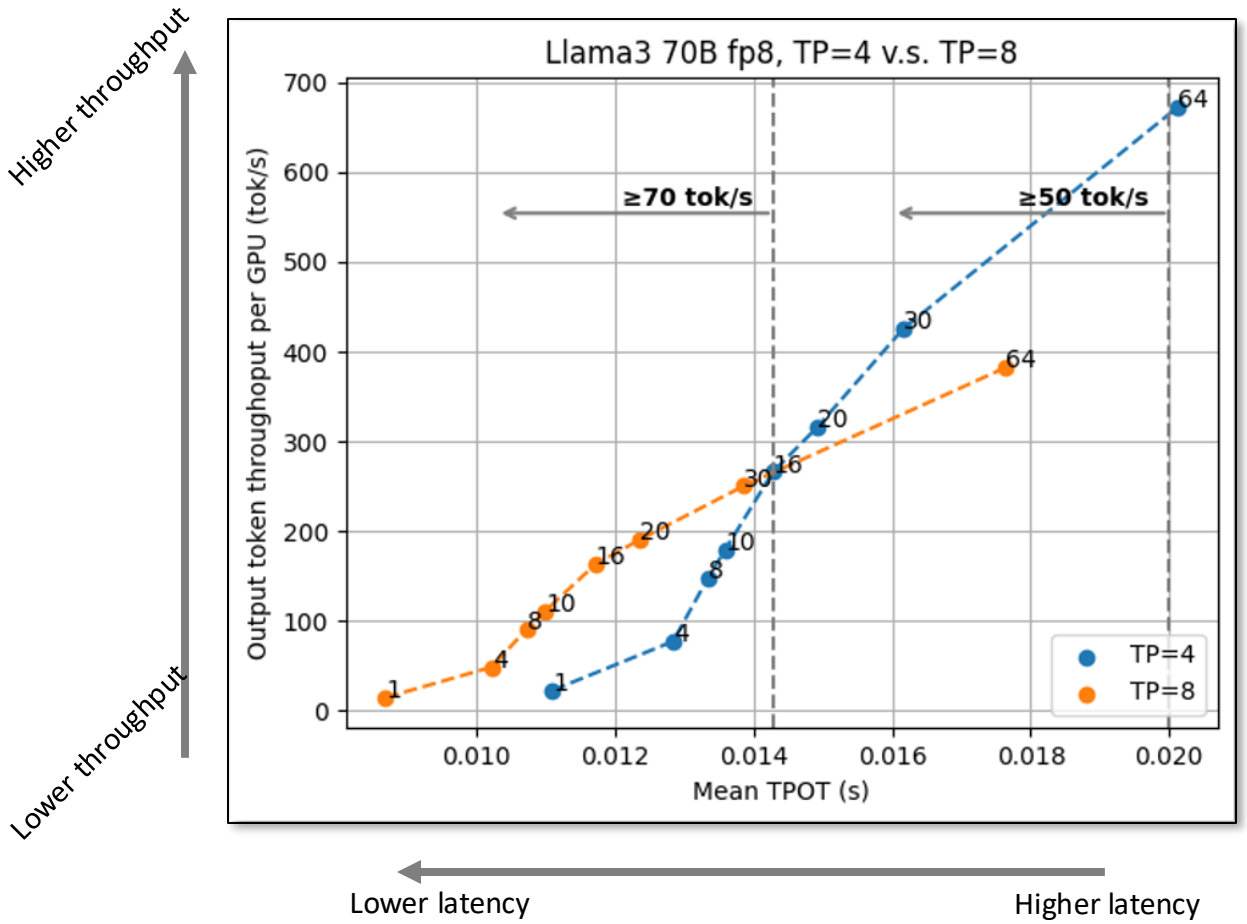
Running on 4 GPUs v.s. running on 8 GPUs

3. TP doesn't scale

- Some GeMM/attention GPU kernels don't scale well linearly



Llama3 70B provider latency leaderboard



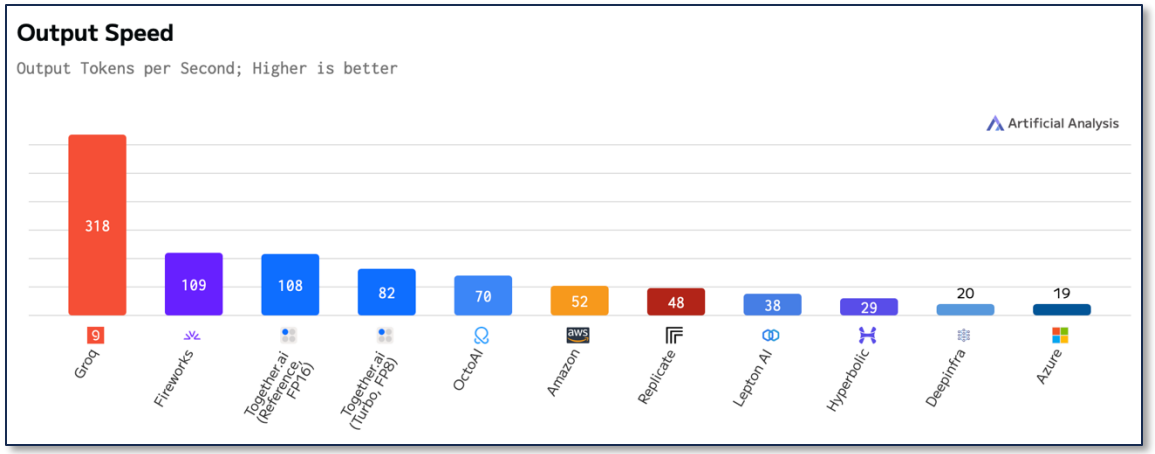
https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference

Impact of Speculative Decoding

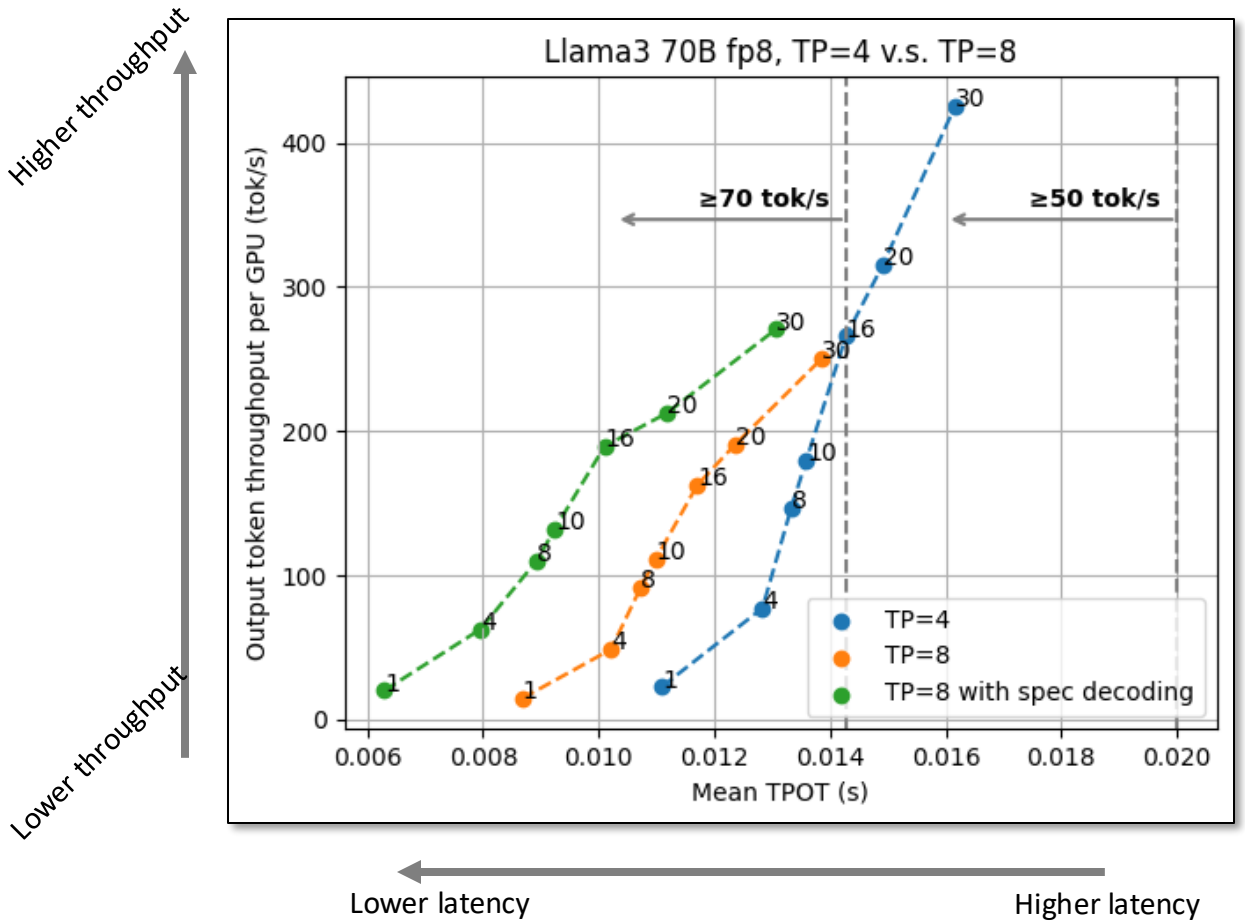
Use Llama3 8B fp8 to speculate

1. Speculative decoding helps further push to the extremity

- Improvement marginalized by batch size



Llama3 70B provider latency leaderboard

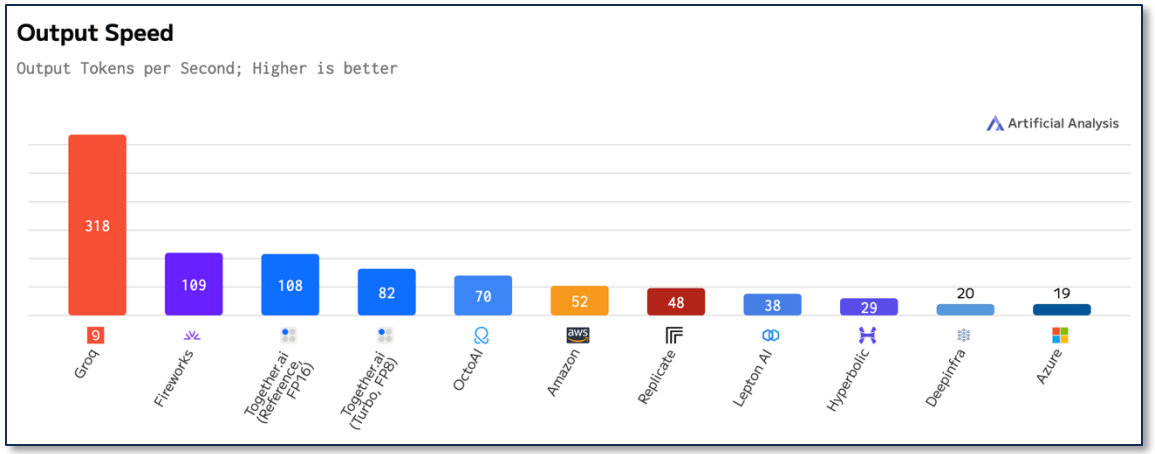


https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference

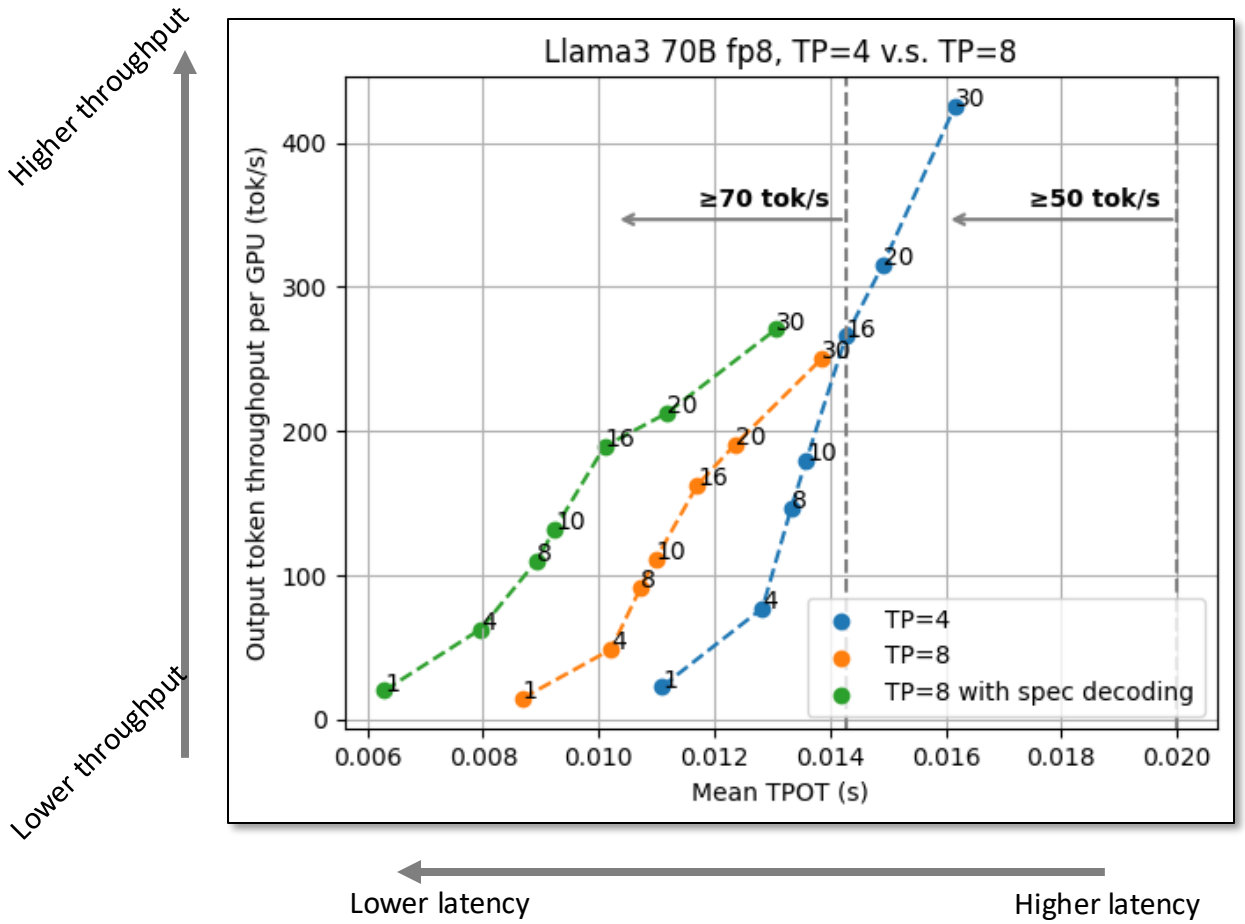
Impact of Speculative Decoding

Use Llama3 8B fp8 to speculate

2. System complexity for spec decoding
- Draft probability distribution management
- Efficient GPU verification kernel
- …



Llama3 70B provider latency leaderboard



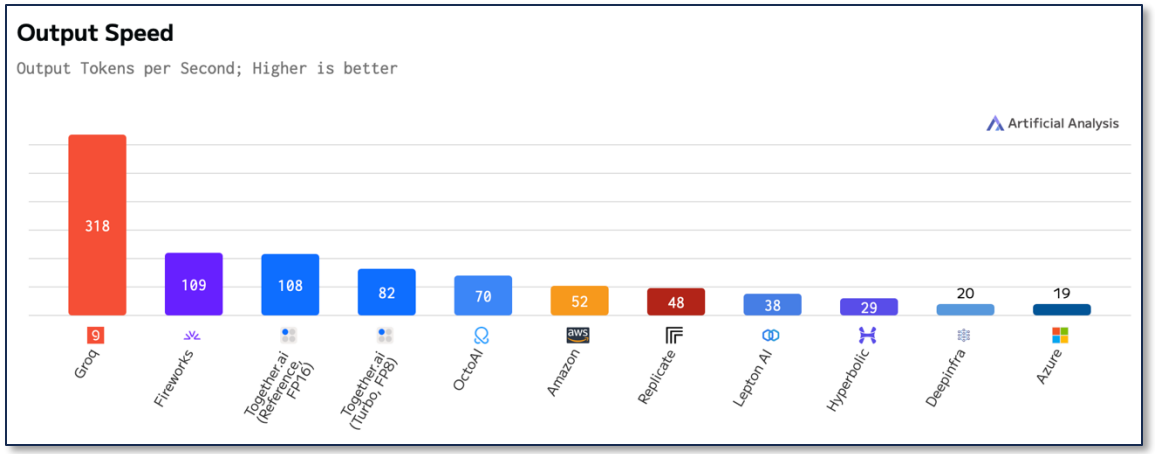
https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference

Impact of Speculative Decoding

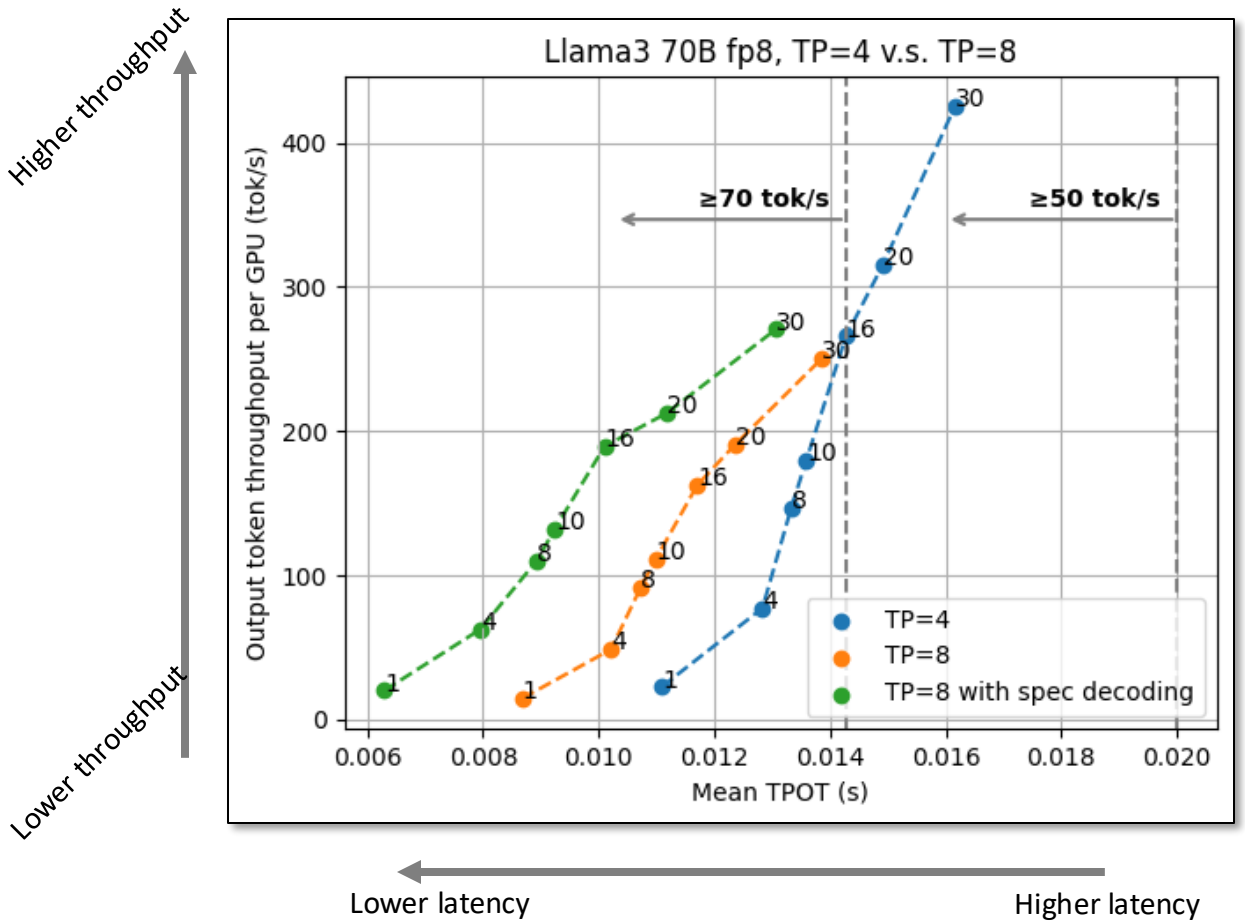
Use Llama3 8B fp8 to speculate

3. The potential of being applied on mobile/edge platforms

- Llama3 8B speculated with 1B: 80% 1-step acceptance on ShareGPT



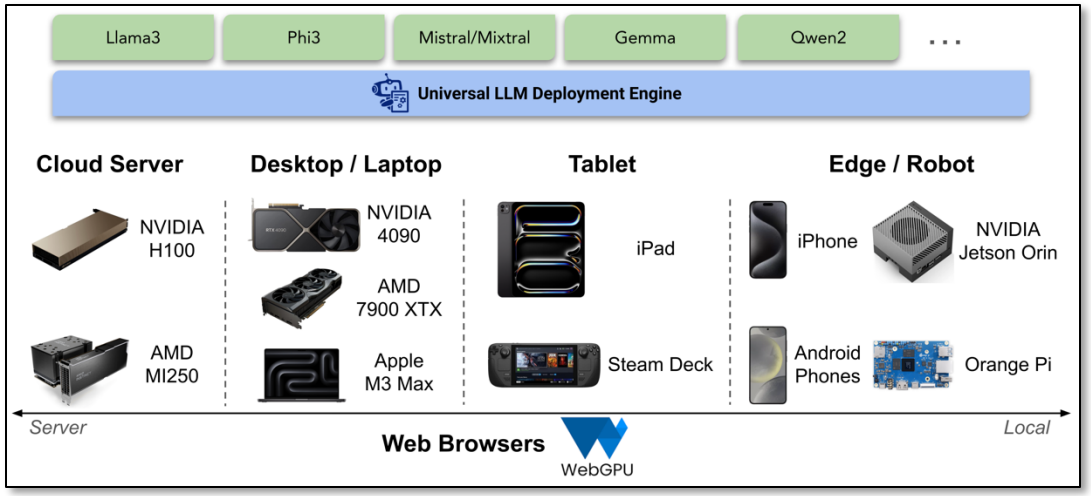
Llama3 70B provider latency leaderboard

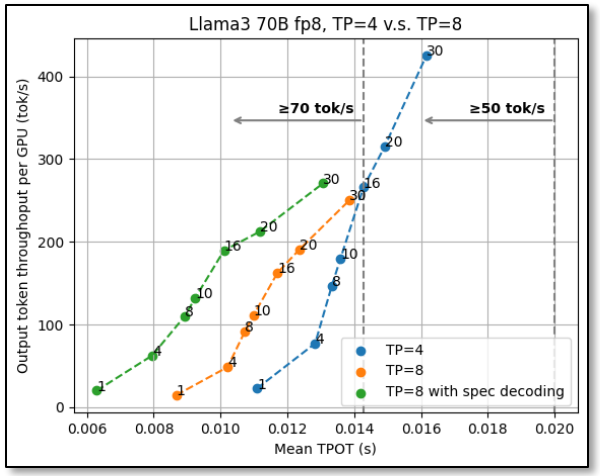


https://blog.mlc.ai/2024/10/10/optimizing-and-characterizing-high-throughput-low-latency-llm-inference

MLC LLM

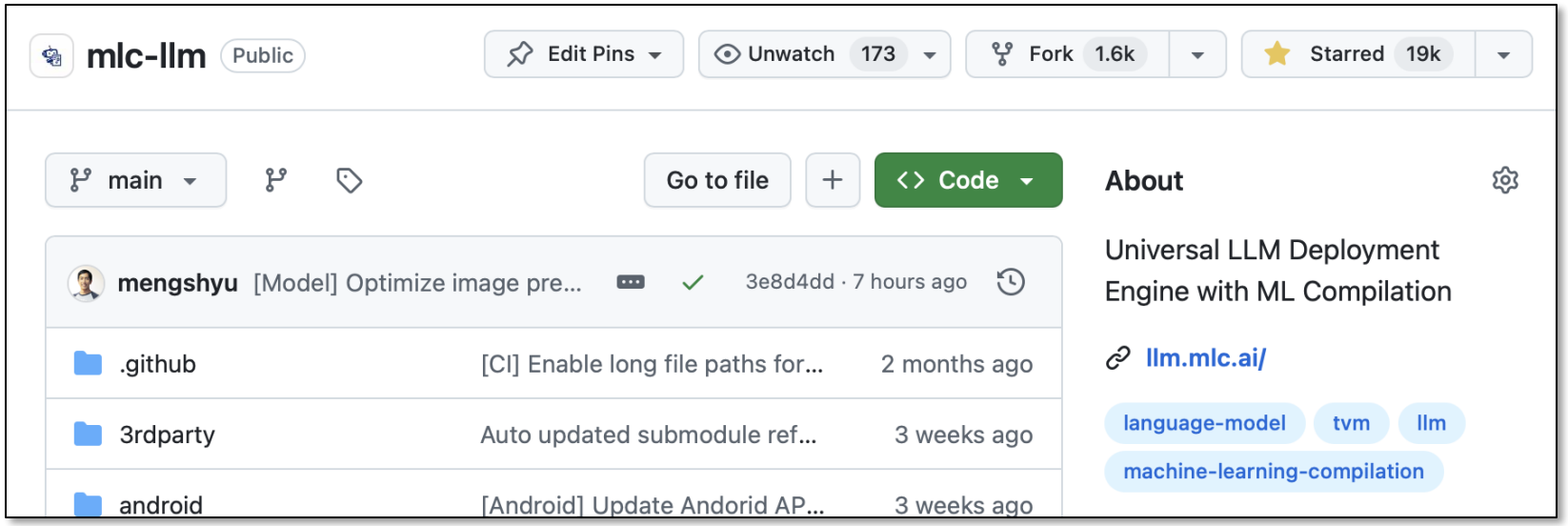
- Today: Universal deployment and low-latency serving in MLCEngine





- Future roadmap
  - Multi-modality completeness across platforms
  - Other speculative decoding methods
  - And more…

MLC LLM



- GitHub: https://github.com/mlc-ai/mlc-llm
- Documentation: https://llm.mlc.ai/docs/
- HuggingFace: https://huggingface.co/mlc-ai
- Blogs: https://blog.mlc.ai
- Discord: https://discord.com/invite/9Xpy2HGBuD