Software Performance Engineering for FMware (SPE4FMware)

Presented by: Boyuan Chen and Haoxiang Zhang

@ Centre for Software Excellence, Huawei, Canada

How to cite this session?

```
@misc{Zhang2024AIwareTutorial,
author = {Haoxiang Zhang and Boyuan Chen and Ahmed E. Hassan},
title = {Software Performance Engineering for FMware (SPE4FMware)},
howpublished = {Tutorial presented at the AIware Leadership Bootcamp 2024},
month = {November},
year = {2024},
address = {Toronto, Canada},
note = {Part of the AIware Leadership Bootcamp series.},
url = {https://aiwarebootcamp.io/slides/2024_aiwarebootcamp_zhang_software_performance_engineering_for_fmware.pdf}}
```

Check this paper for more information about this session

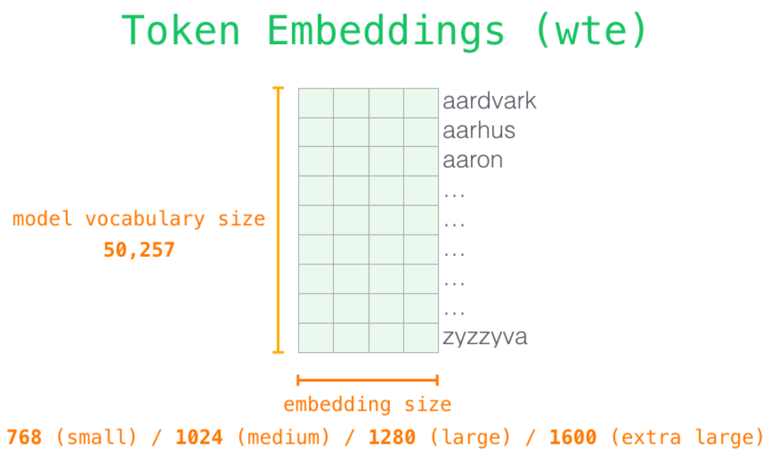
```
@article{zhang2024softwareperformanceengineeringfoundation,
  title={Software Performance Engineering for Foundation Model-Powered Software (FMware)},
  author={Haoxiang Zhang and Shi Chang and Arthur Leung and Kishanthan Thangarajah and Boyuan Chen and Hanan Lutfiyya and Ahmed E. Hassan},
  journal={arXiv preprint arXiv:2411.09580},
  year={2024},
  url={https://arxiv.org/abs/2411.09580},
}
```

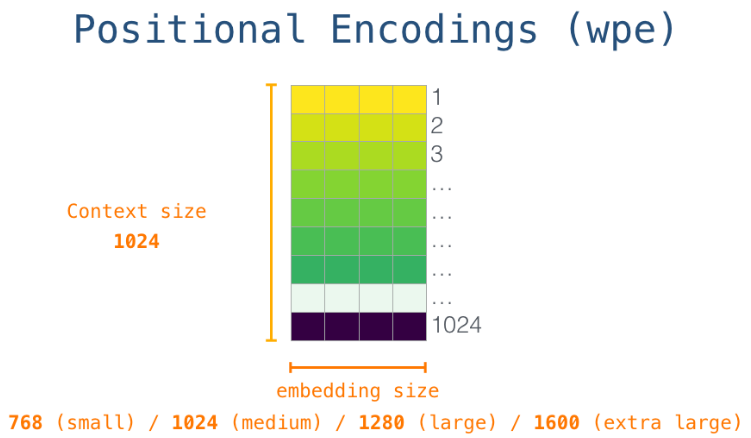
Outlines

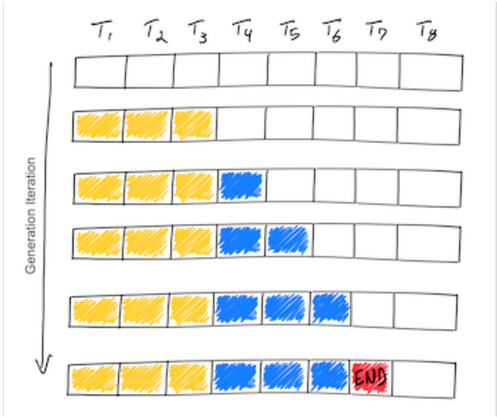
- ❑**Background of FM Serving and Techniques (Boyuan)**
- ❑Challenges and Innovation Paths in SPE for FMware (Haoxiang)

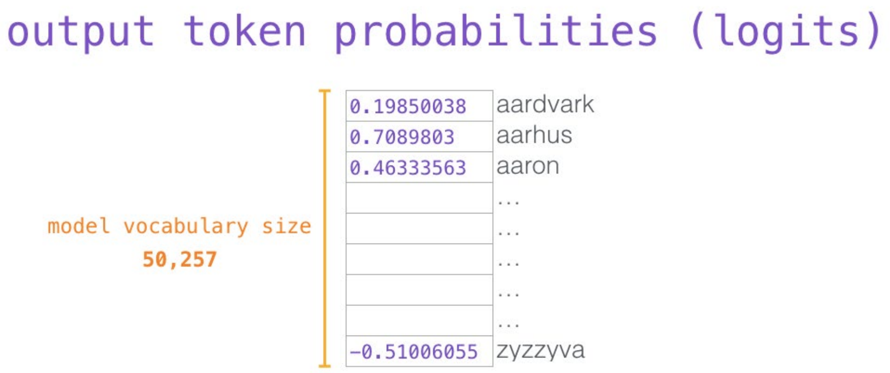
Quick recap on autoregressive decoding

Tokenization -> Token embedding + Positional encodings -> a series of Decoder blocks -> sampling from logits




Positional Encodings (wpe)
Context size
1024
1
2
3
…
…
…
…
…
1024
embedding size
768 (small) / 1024 (medium) / 1280 (large) / 1600 (extra large)





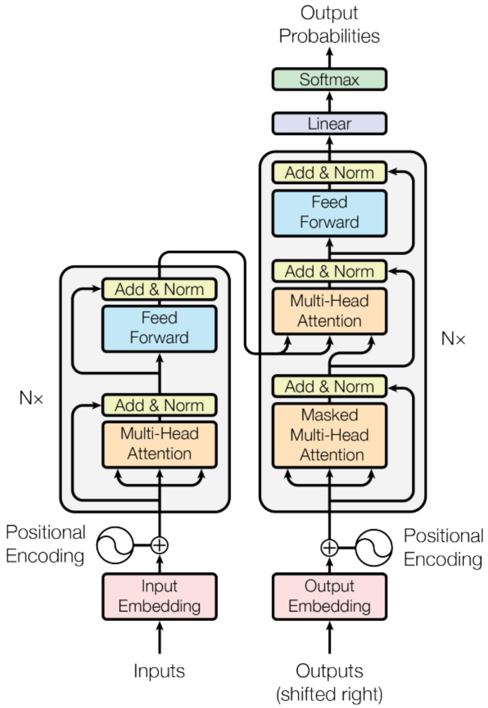
Three matrices in self-attention

- Query: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're **currently** processing.
- Key: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
- Value: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

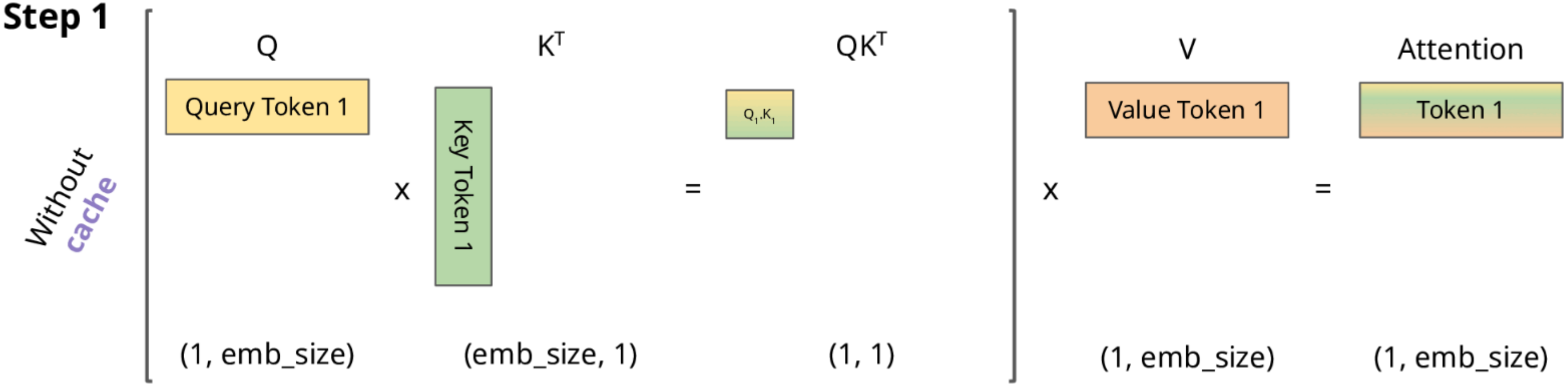
Breakdown in shapes

- T, C = 1, 768 // C represents the embedding size
- x.shape = (1,768) // this input combines token embedding and positional embedding
- head_size = 16 // tunable hyperparameter
- q_weight.shape = (768,16) // fully connected layer, trainable parameters
- k_weight.shape = (768,16) // fully connected layer, trainable parameters
- v_weight.shape = (768,16) // fully connected layer, trainable parameters
- Q = x@q_weight (shape is 1,16)
- K = x@k_weight (shape is 1,16)
- V = x@v_weight (shape is 1,16)
- Attention score shape (1,16)



No KV Cache

Step 1



Without *cache*

Q | Query Token 1 | (1, emb_size)

x

K^T | Key Token 1 | (emb_size, 1)

=

QK^T | $Q_1.K_1$ | (1, 1)

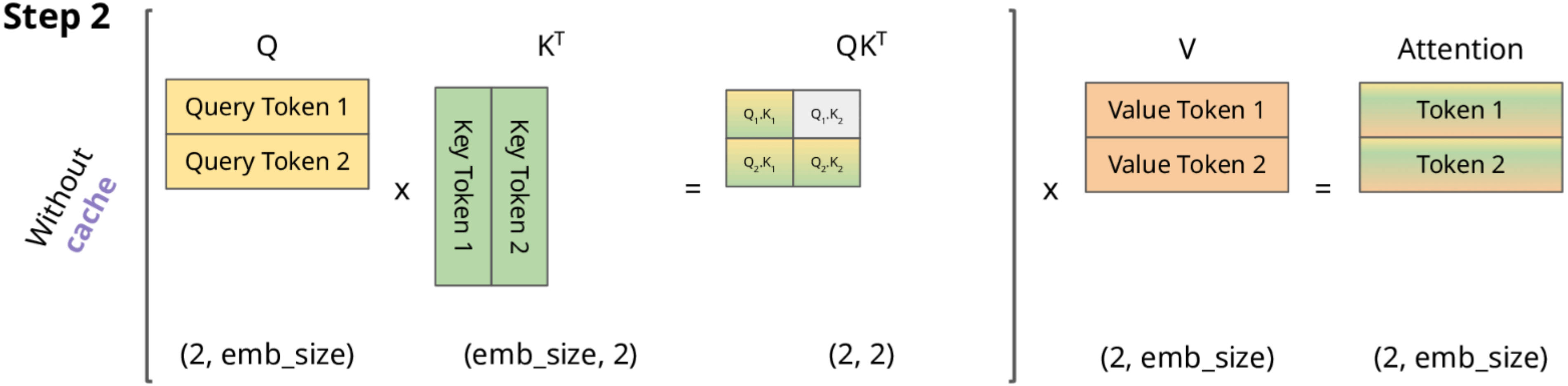
x

V | Value Token 1 | (1, emb_size)

=

Attention | Token 1 | (1, emb_size)

Step 2



Without *cache*

Q | Query Token 1 | Query Token 2 | (2, emb_size)

x

K^T | Key Token 1 | Key Token 2 | (emb_size, 2)

=

QK^T | $Q_1.K_1$ | $Q_1.K_2$ | $Q_2.K_1$ | $Q_2.K_2$ | (2, 2)

x

V | Value Token 1 | Value Token 2 | (2, emb_size)

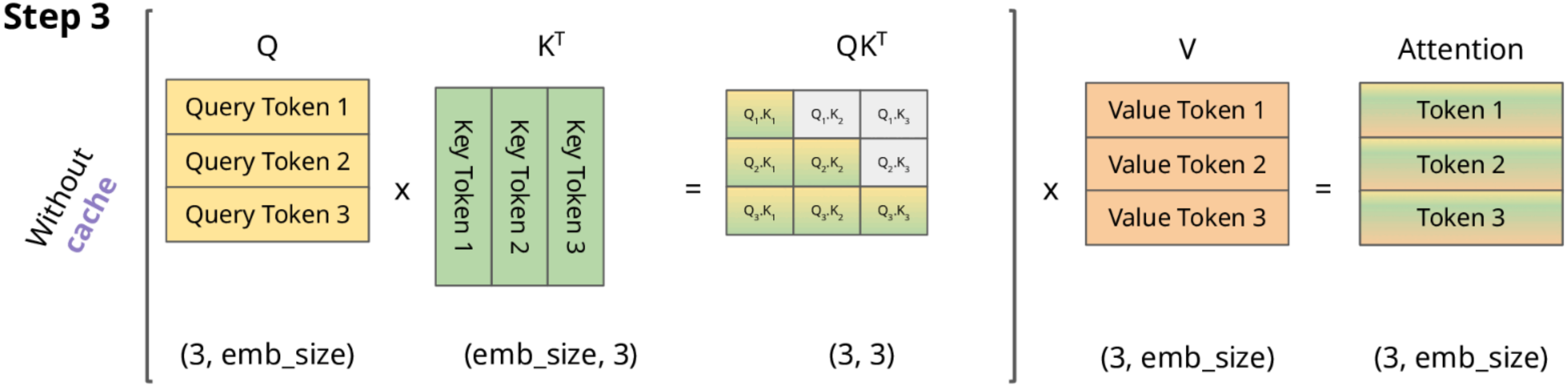
=

Attention | Token 1 | Token 2 | (2, emb_size)

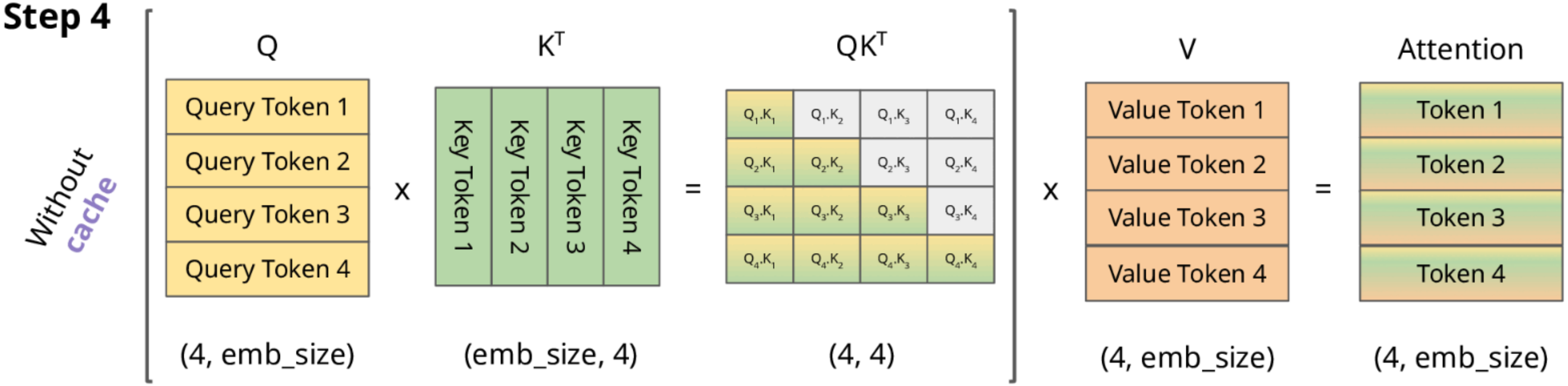
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

No KV Cache

Step 3

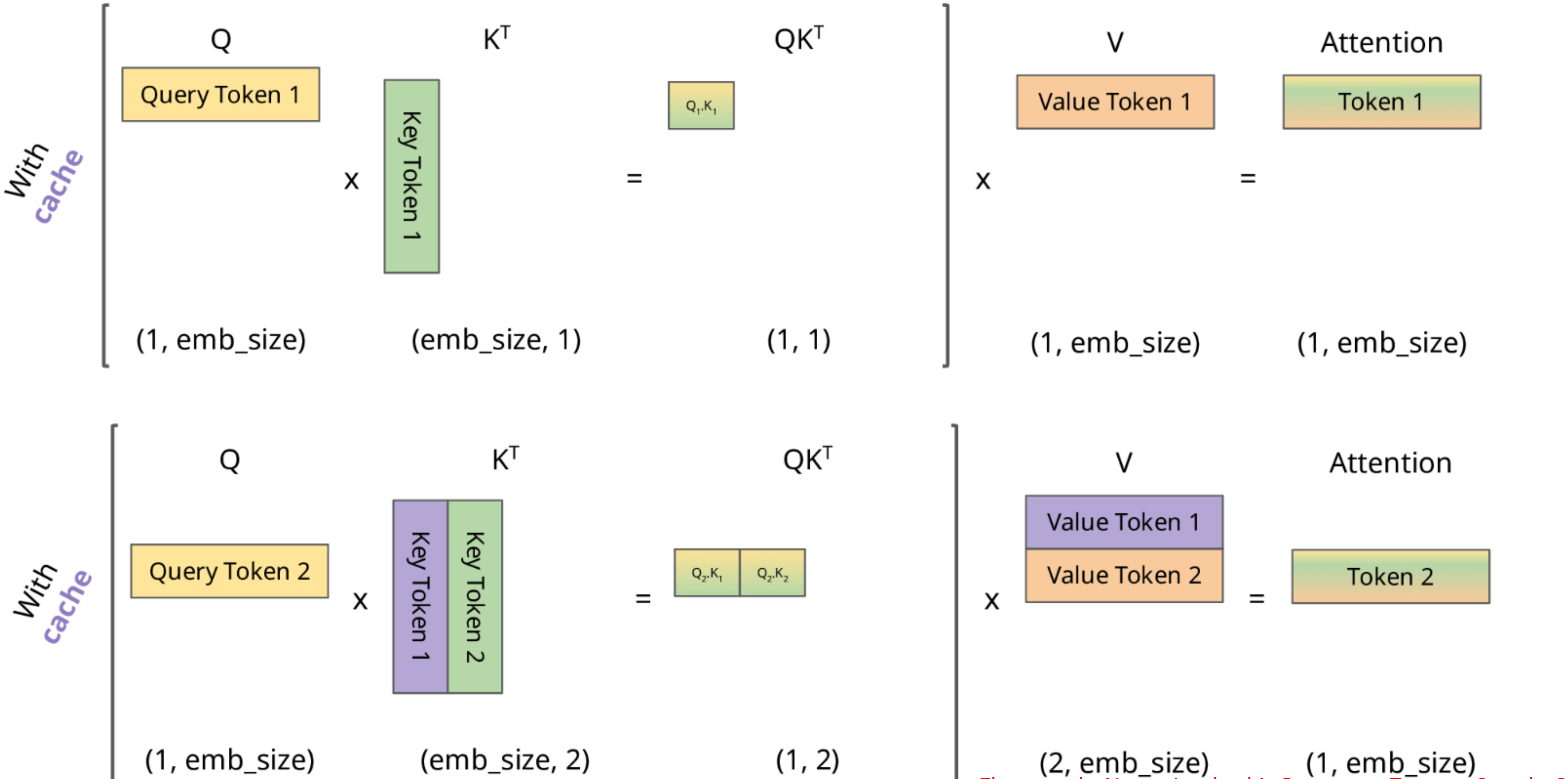


Step 4



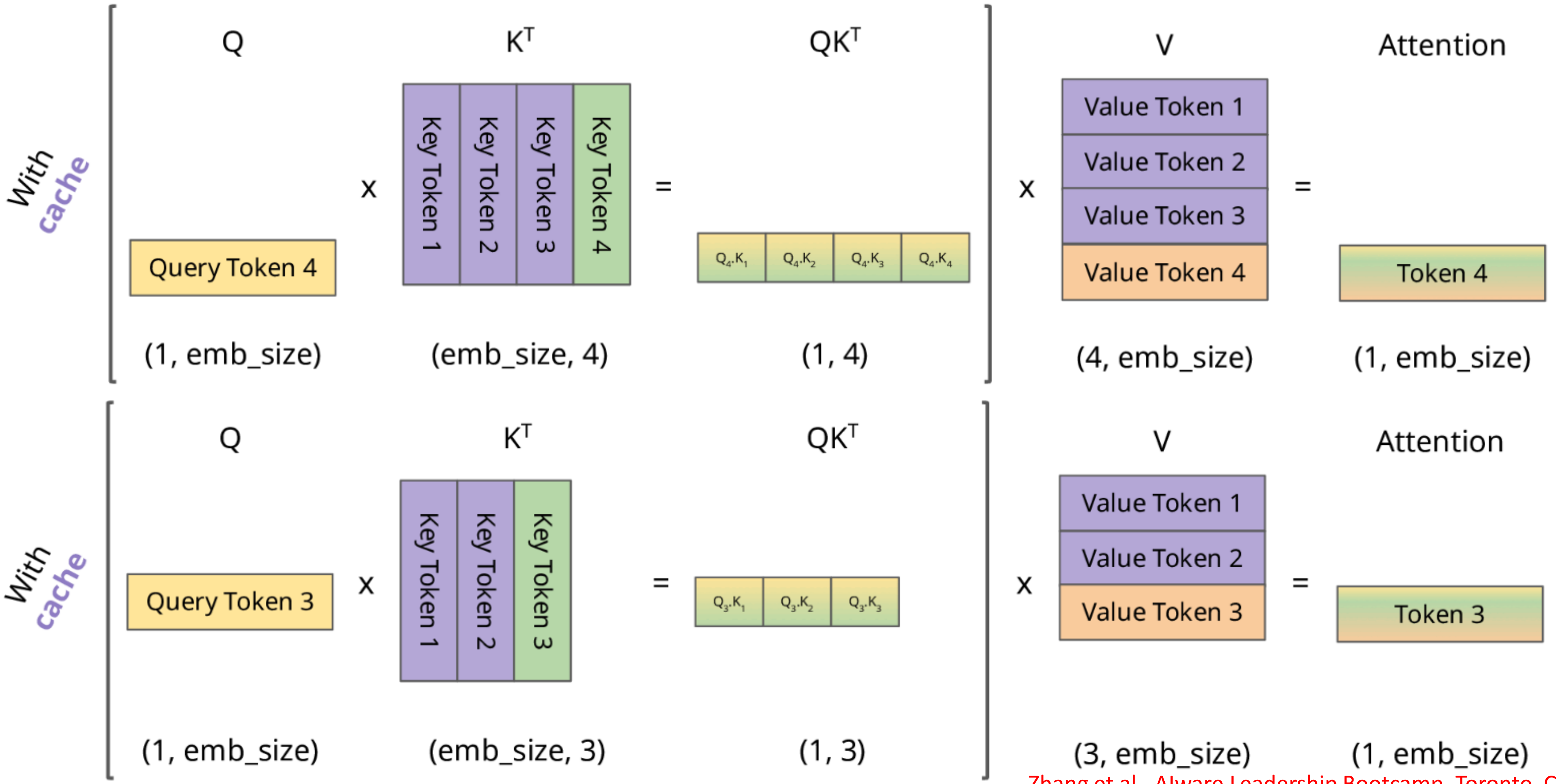
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

KV Cache



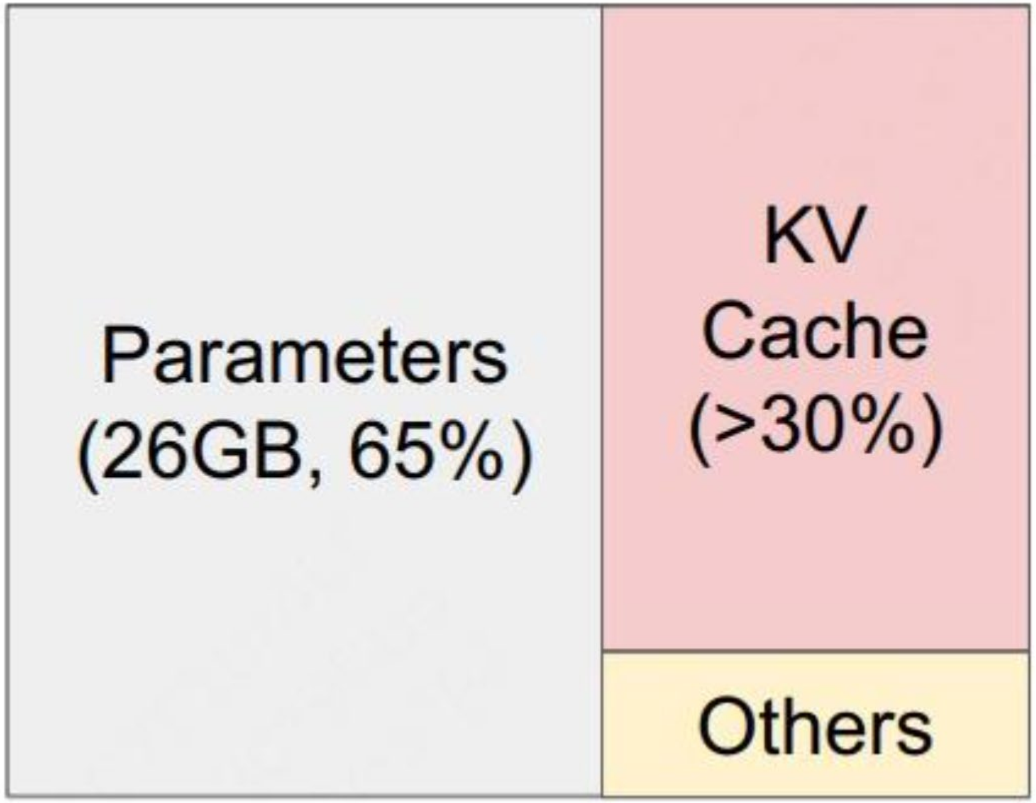
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

KV Cache



Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

KV Cache takes around 30% of VRAM



Parameters (26GB, 65%)

KV Cache (>30%)

Others

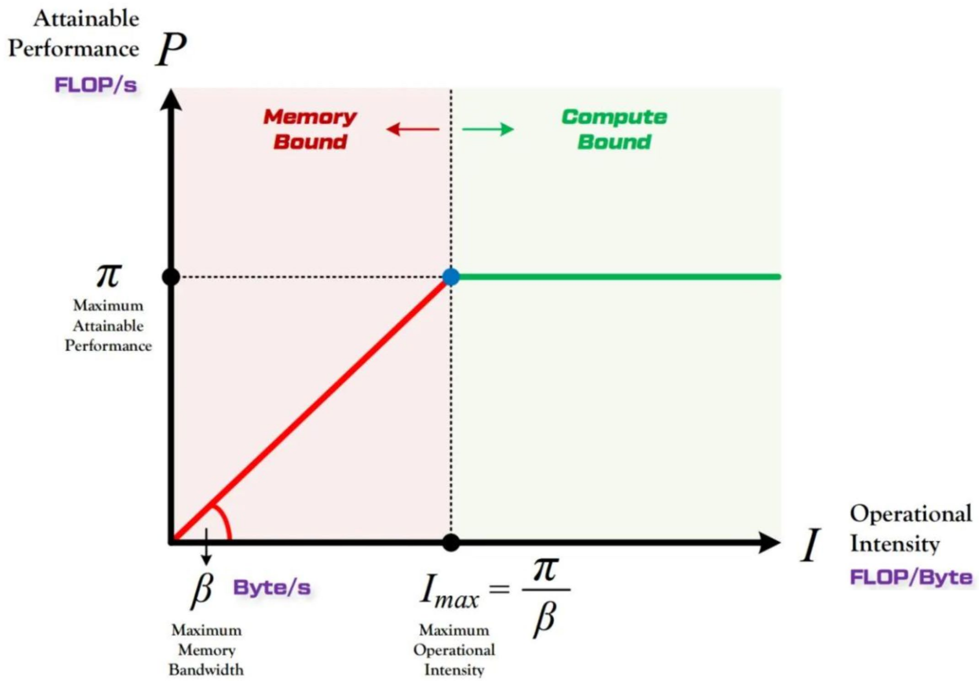
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Compute Bound vs Memory Bound

- Computing power: FLOP/S
- Bandwidth: Bytes/s
- Operational Intensity: I = π/β

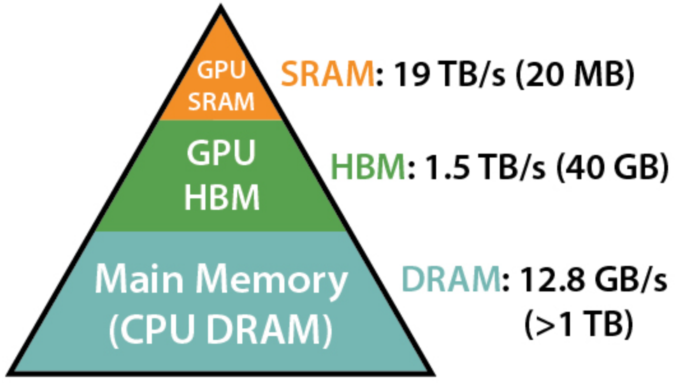
- **Roofline model:**

$$P = \begin{cases} \beta \cdot I, & when\ \ I < I_{max} \quad \text{Memory Bound} \\ \pi, & when\ \ I \geqslant I_{max} \quad \text{Compute Bound} \end{cases}$$

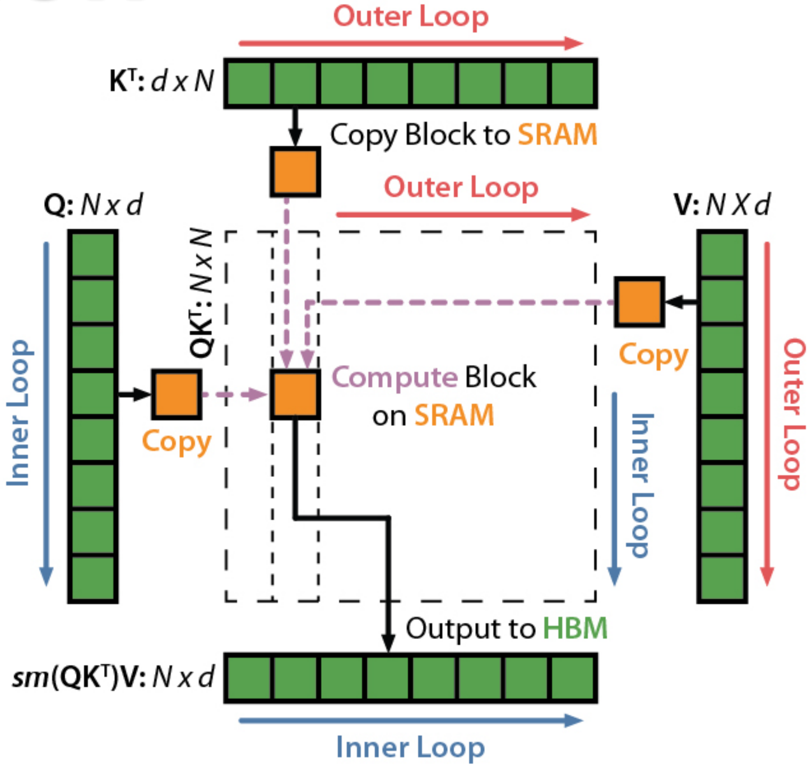


Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

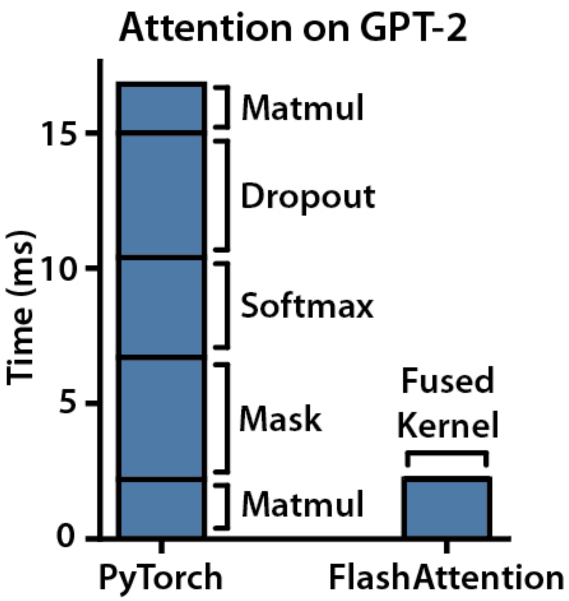
FlashAttention



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention

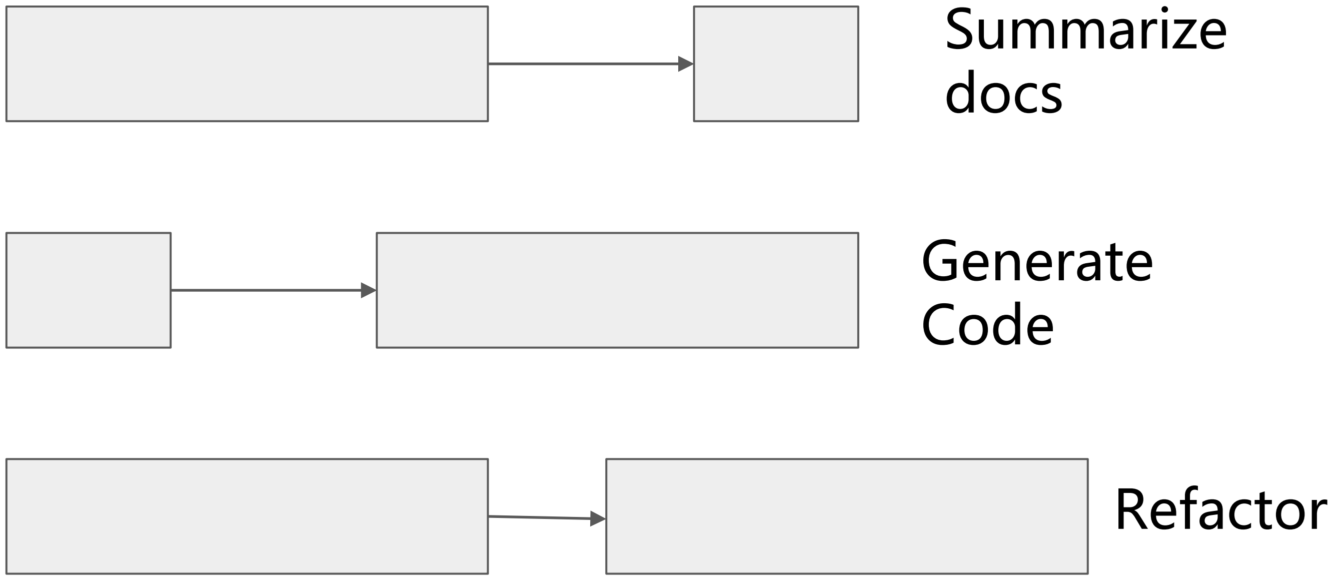


- The key is to minimize the data movement using HBM
- Tiling and re-computation

https://arxiv.org/abs/2205.14135

Characteristics of FM Inference

Task Heterogeneity



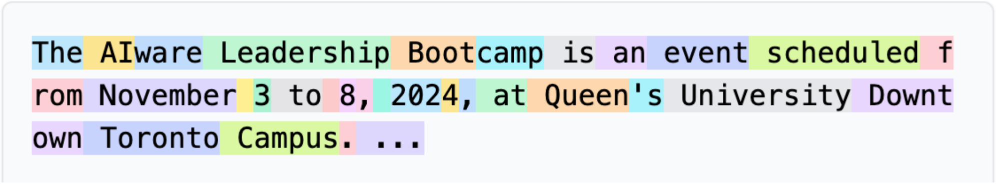
Summarize docs

Generate Code

Refactor

Performance requirements are also different (e.g., Latency requirements are different)

Non-deterministic Execution & Resource Consumption



```
The AIware Leadership Bootcamp is an event scheduled f
rom November 3 to 8, 2024, at Queen's University Downt
own Toronto Campus. ...
```

- Token generation length is not known before the generation
- Lead to unknown GPU memory cost due to KVcache

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

How do you serve FM in production?

From traditional backend point of view

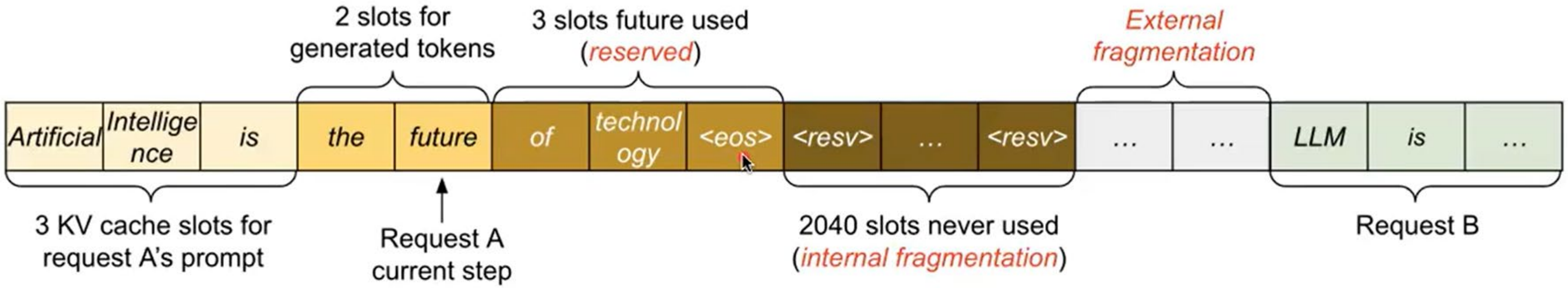
- Traditionally, you need a load balancer
- Then you need to decide the # of replicas and how to dispatch model inference to the engines
- Scheduling techniques matter depending on SLAs

Open Challenges

- [Scheduling] How to design good load balancing techniques?
- [Scheduling] How to deal with request priorities with SLA constraints?
- [Resource] How to avoid memory waste?
- [Resource] Scaling strategies?

vLLM: PagedAttention to manage memory efficiently

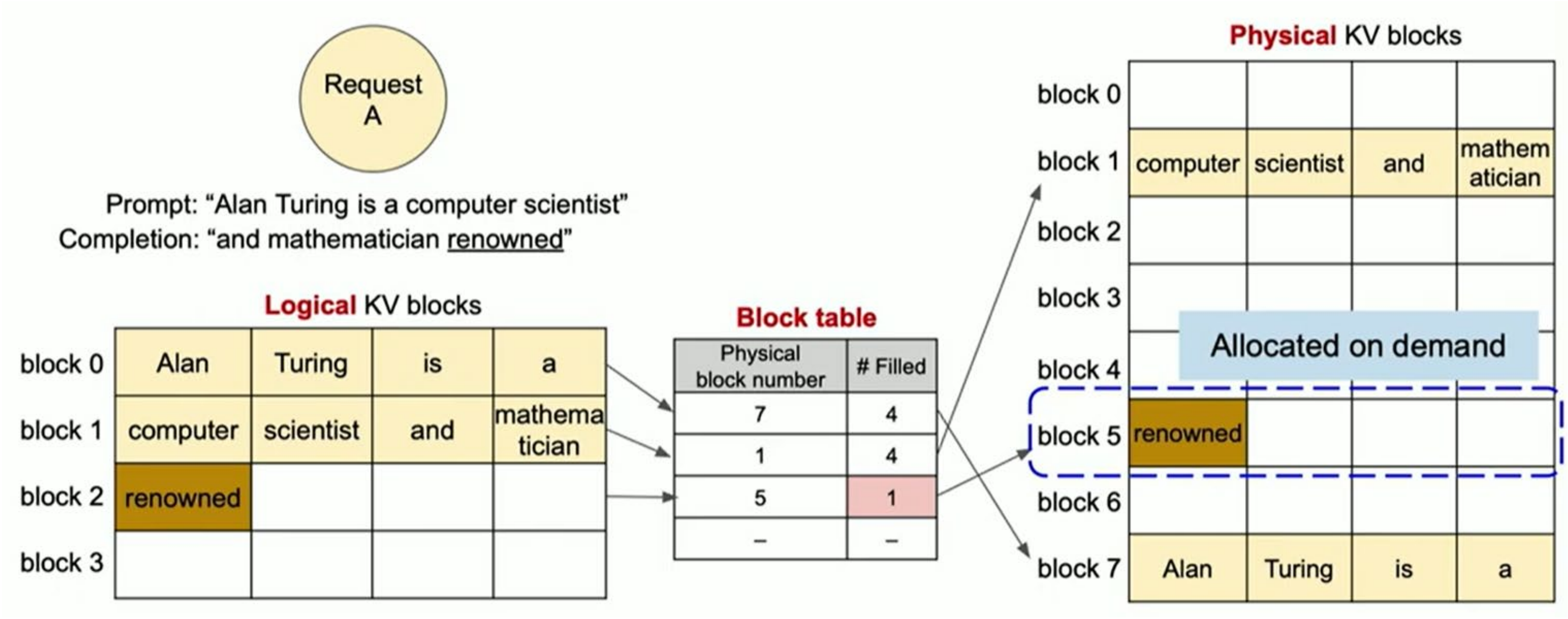
Motivation: kvcache memory are first assigned and the GPU memory is locked (internal/external fragmentation)


2 slots for generated tokens
3 slots future used (reserved)
External fragmentation
Artificial
Intelligence
is
the
future
of
technology
<eos>
<resv>
…
<resv>
…
…
LLM
is
…
3 KV cache slots for request A's prompt
Request A current step
2040 slots never used (internal fragmentation)
Request B

- Internal fragmentation: over-allocated due to the unknown output length.
- Reservation: not used at the current step, but used in the future
- External fragmentation: due to different sequence lengths.

https://arxiv.org/pdf/2309.06180.pdf

Virtual Blocks for storing KV Cache



Request A

Prompt: “Alan Turing is a computer scientist”

Completion: “and mathematician renowned”

Logical KV blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	mathema tician
block 2	renowned			
block 3				

Block table

Physical block number	# Filled
7	4
1	4
5	1
–	–

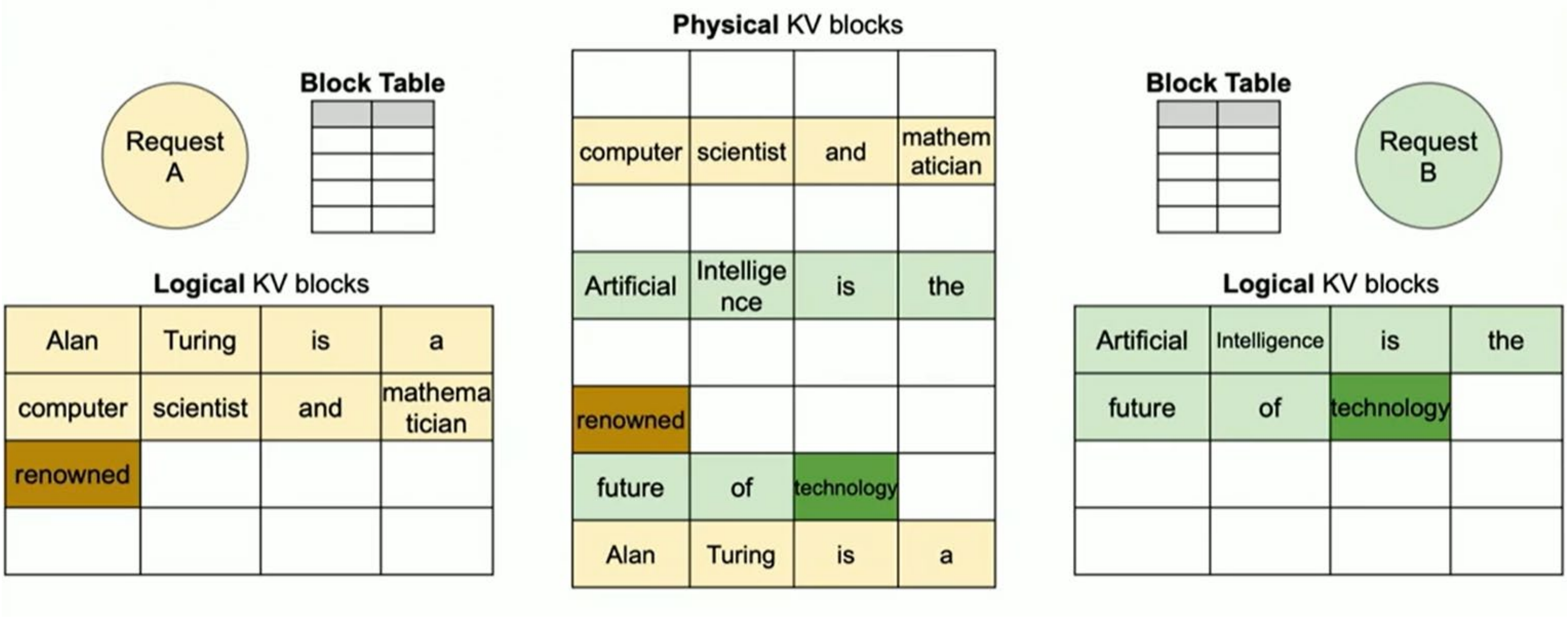
Physical KV blocks

block 0				
block 1	computer	scientist	and	mathem atician
block 2				
block 3				
block 4				
block 5	renowned			
block 6				
block 7	Alan	Turing	is	a

Allocated on demand

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Serving multiple requests, where the memory is allocated on demand



Request A

Block Table

Logical KV blocks

Alan	Turing	is	a
computer	scientist	and	mathema tician
renowned			

Physical KV blocks

computer	scientist	and	mathem atician
Artificial	Intellige nce	is	the
renowned			
future	of	technology	
Alan	Turing	is	a

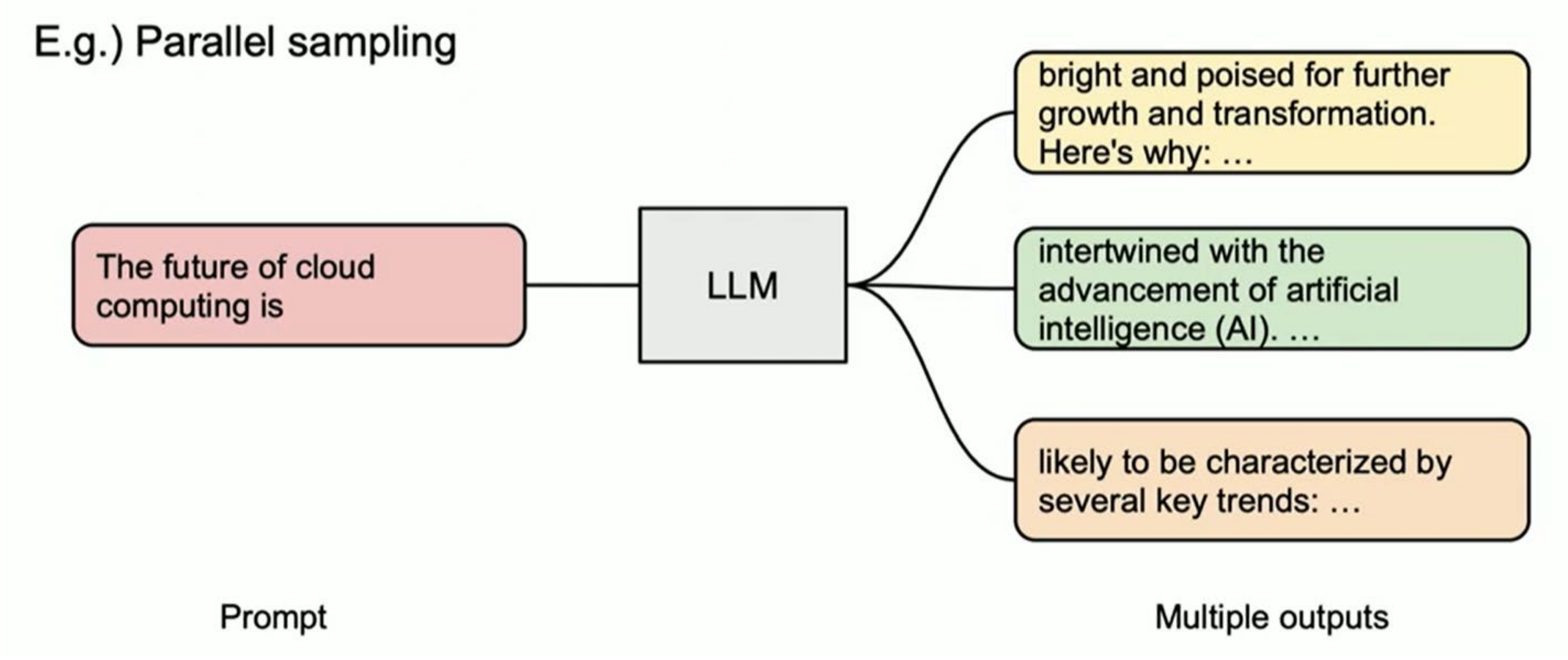
Block Table

Request B

Logical KV blocks

Artificial	Intelligence	is	the
future	of	technology	

With the same input prompts, KV Cache memory can be saved


E.g.) Parallel sampling
The future of cloud computing is
LLM
bright and poised for further growth and transformation. Here's why: …
intertwined with the advancement of artificial intelligence (AI). …
likely to be characterized by several key trends: …
Prompt
Multiple outputs

Speculative Decoding

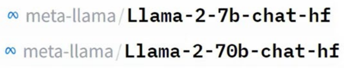
- Observation: difficulty of predicting each token not equal
  - “The city of New _ _ _” vs
  - “This means that _ _ _”
- Idea: use small model on easier tokens
  - 7B model is faster than 70B: ~9ms vs ~90ms on a single GPU
  - But how to know which tokens are easier?
  - Rejection sampling

https://arxiv.org/pdf/2302.01318.pdf

Compare draft model and target model

M_p = draft model

M_q = target model



meta-llama/**Llama-2-7b-chat-hf**

meta-llama/**Llama-2-70b-chat-hf**

pf = prefix, K = 5 tokens

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

$$\ldots$$

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

$$\ldots$$

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Run draft model for K steps

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

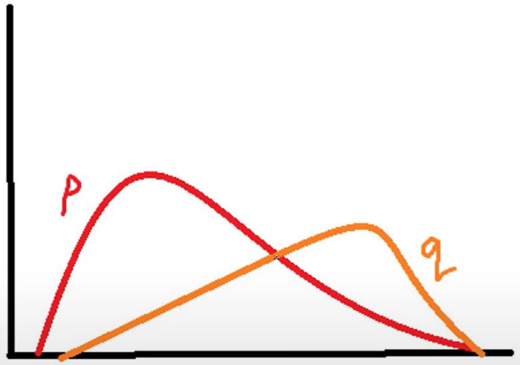
Run target model once

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Goal: Sample a token from $q(x)$

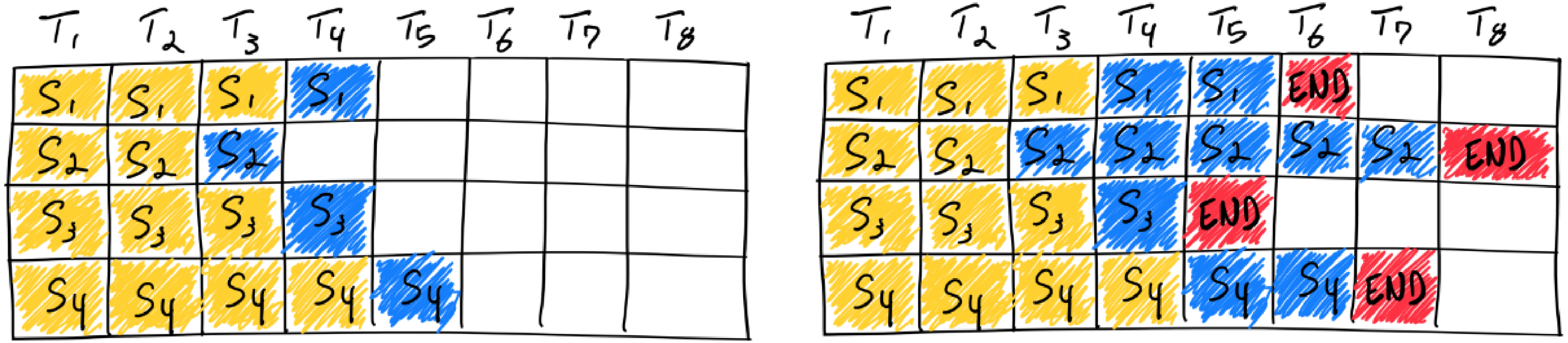
Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$



Continuous batching

- Motivation: static batching wastes GPU resources as sequence length is different



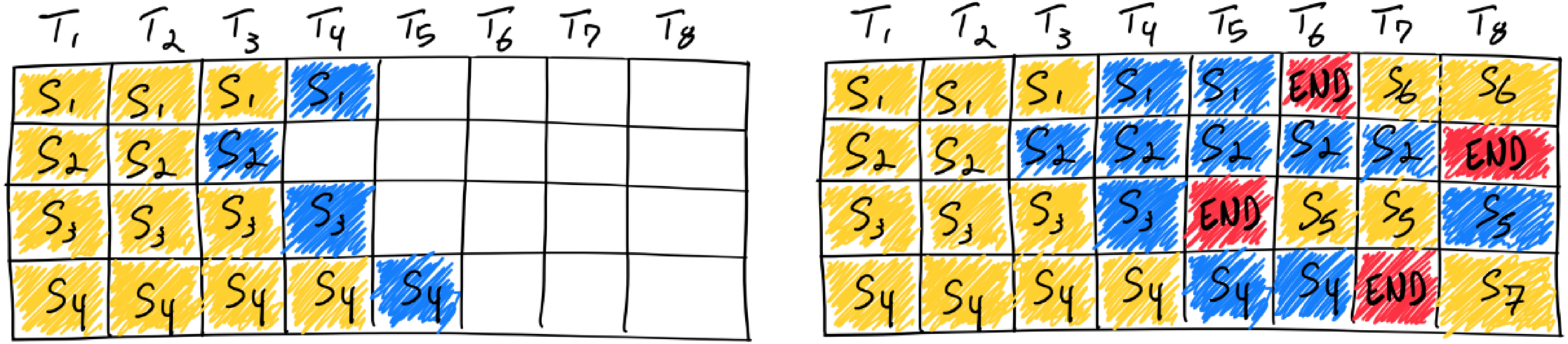
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_1	S_1	S_1	S_1				
S_2	S_2	S_2					
S_3	S_3	S_3	S_3				
S_4	S_4	S_4	S_4	S_4			

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_1	S_1	S_1	S_1	S_1	END		
S_2	S_2	S_2	S_2	S_2	S_2	S_2	END
S_3	S_3	S_3	S_3	END			
S_4	S_4	S_4	S_4	S_4	S_4	END	

Completing four sequences using static batching. On the first iteration (left), each sequence generates one token (blue) from the prompt tokens (yellow). After several iterations (right), the completed sequences each have different sizes because each emits their end-of-sequence-token (red) at different iterations. Even though sequence 3 finished after two iterations, static batching means that the GPU will be underutilized until the last sequence in the batch finishes generation (in this example, sequence 2 after six iterations).

https://arxiv.org/pdf/2309.06180.pdf

Iteration level scheduling



T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_1	S_1	S_1	S_1				
S_2	S_2	S_2					
S_3	S_3	S_3	S_3				
S_4	S_4	S_4	S_4	S_4			

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_1	S_1	S_1	S_1	S_1	END	S_6	S_6
S_2	S_2	S_2	S_2	S_2	S_2	S_2	END
S_3	S_3	S_3	S_3	END	S_5	S_5	S_5
S_4	S_4	S_4	S_4	S_4	S_4	END	S_7

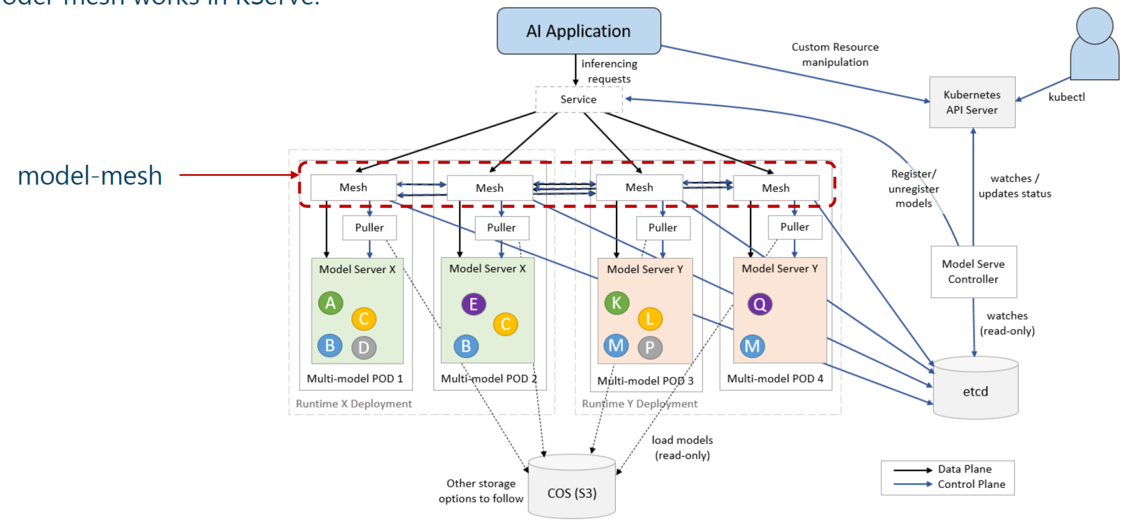
Completing seven sequences using continuous batching. Left shows the batch after a single iteration, right shows the batch after several iterations. Once a sequence emits an end-of-sequence token, we insert a new sequence in its place (i.e. sequences S5, S6, and S7). This achieves higher GPU utilization since the GPU does not wait for all sequences to complete before starting a new one.

Model Swap (Model Multiplex)

- Motivation: serve many models at the same time to provide customized experiences. Not all the models can be served at the same time
- Representative systems:
  - IBM – ModelMesh
  - Alibaba – FaaSwap
- The key idea is to keep models hot to avoid long loading time

KServe ModelMesh Serving Architecture

How model-mesh works in KServe:



https://arxiv.org/abs/2306.03622

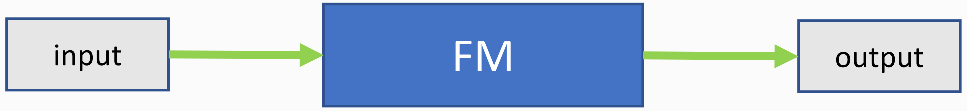
Outlines

- ❑Background of FM Serving and Techniques (Boyuan)
- ❑**Challenges and Innovation Paths in SPE for FMware (Haoxiang)**

Examples of Cognitive Architecture

- Use 1 FM, ask it to extract the name, email, and address from an article
- Output is expected to contain these 3 pieces of info

Cognitive architecture #1: asking FM once for all

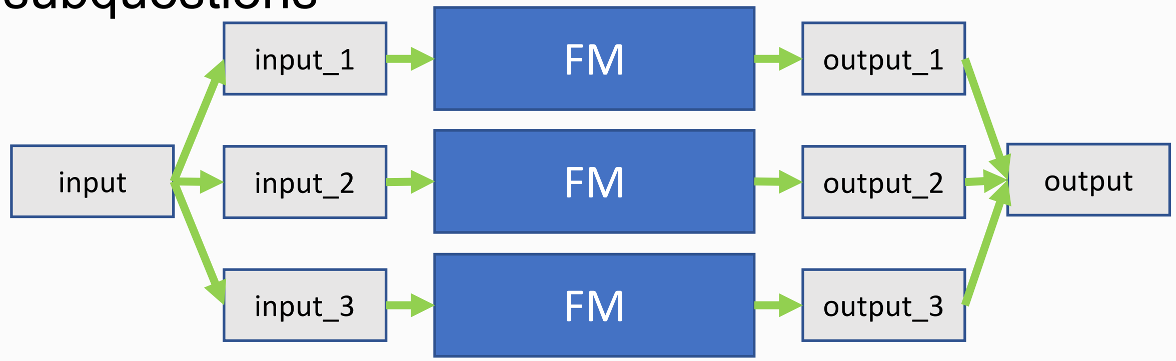


Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Examples of Cognitive Architecture

- The same task as the last example, but use 3 FMs
- The task is broken into 3 subtasks, i.e., extracting name, email, and address
- Eventually 3 pieces of info are aggregated

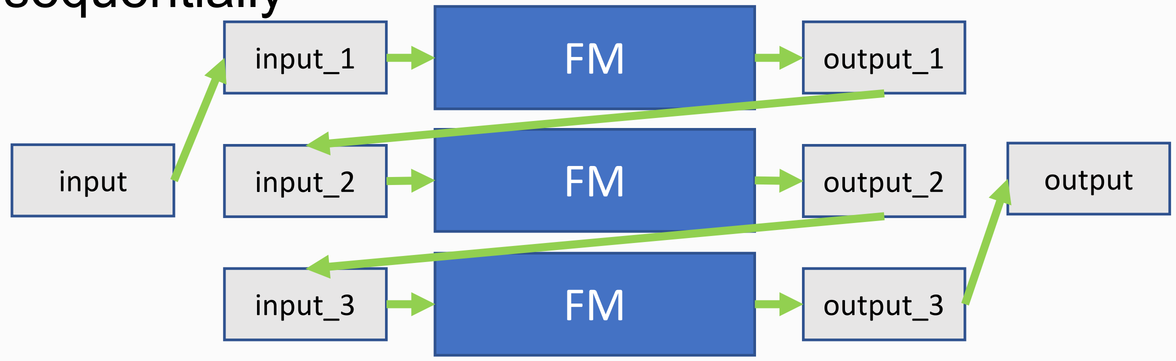
Cognitive architecture #2: asking parallel subquestions



Examples of Cognitive Architecture

- The same task by using 3 FMs
- The task is planned from easy to hard subtasks, i.e., extracting name, email, and address
- Eventually 3 pieces of info are aggregated

Cognitive architecture #3: ask questions sequentially

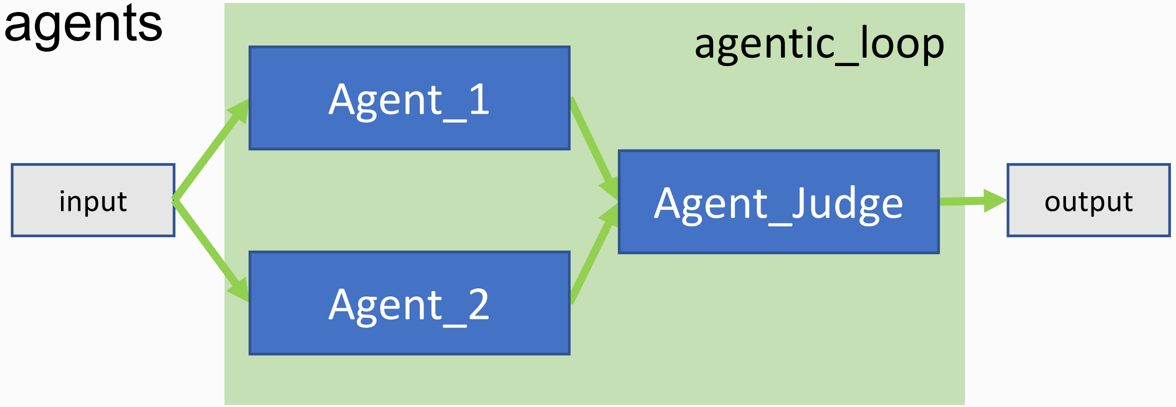


Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Examples of Cognitive Architecture

- The same task, solved by agents
- Two agents can independently solve the same problem
- A third agent as a judge to decide the final result
- The process can be looped and eventually stopped

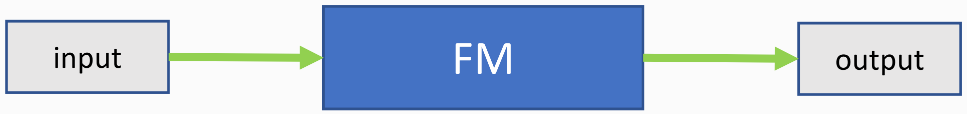
Cognitive architecture #4: solving the problem by agents



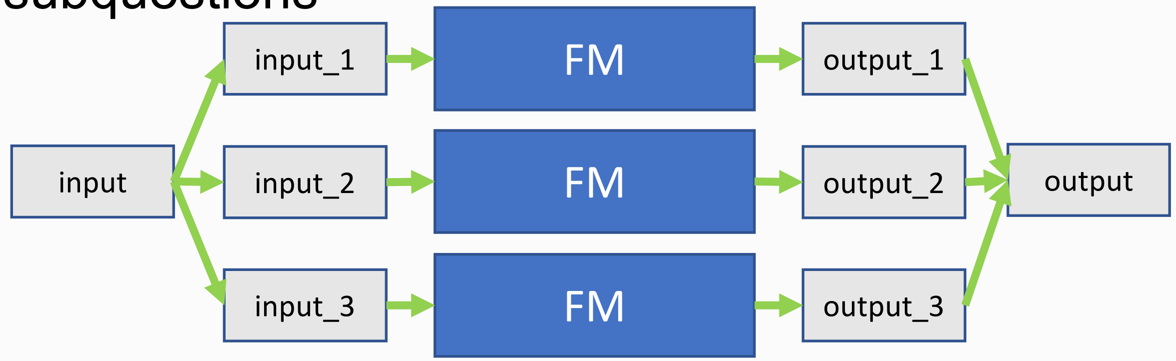
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Different Cognitive Architectures

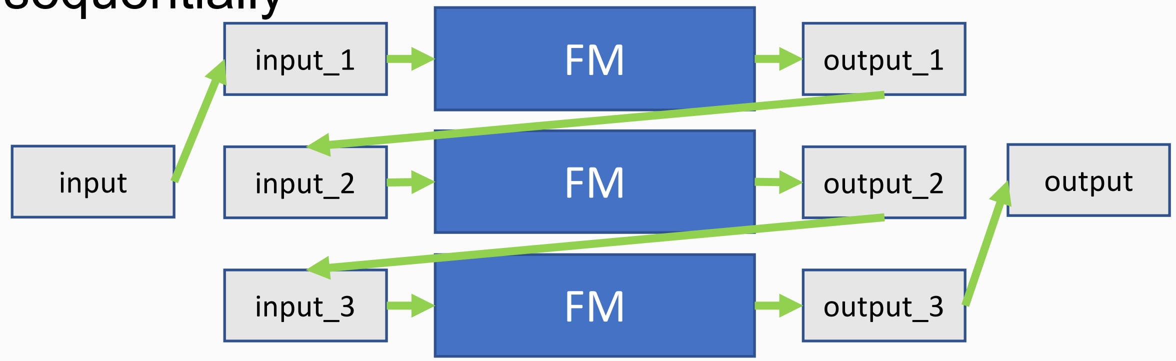
Cognitive architecture #1: asking FM once for all



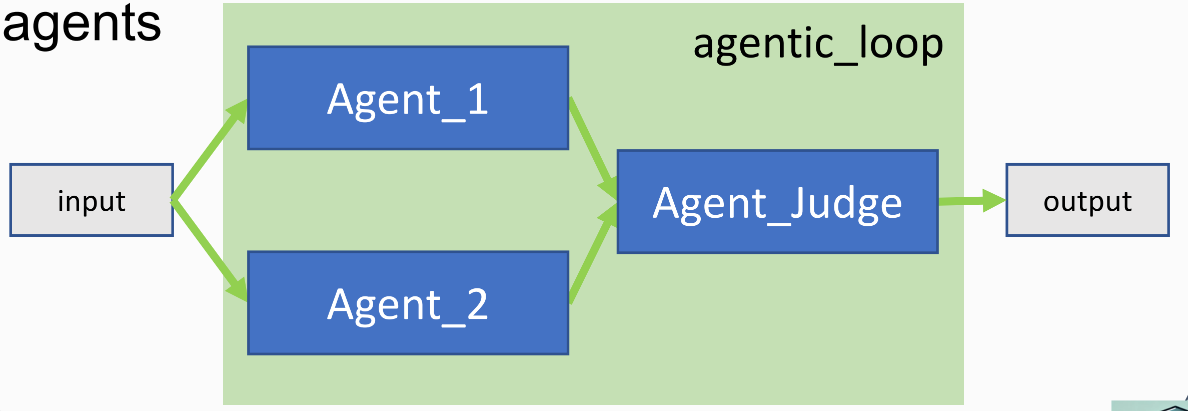
Cognitive architecture #2: asking parallel subquestions



Cognitive architecture #3: ask questions sequentially



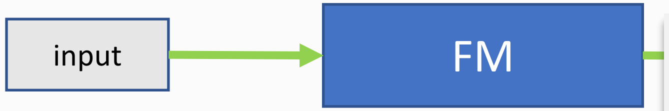
Cognitive architecture #4: solving the problem by agents



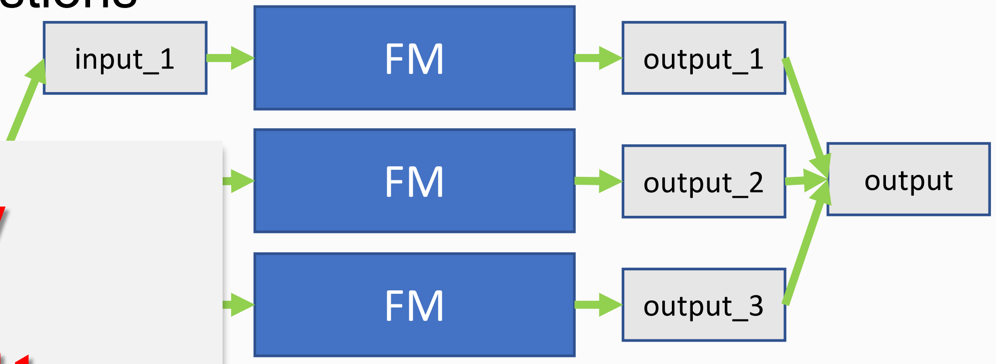
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Different Cognitive Architectures

Cognitive architecture #1: asking FM once for all

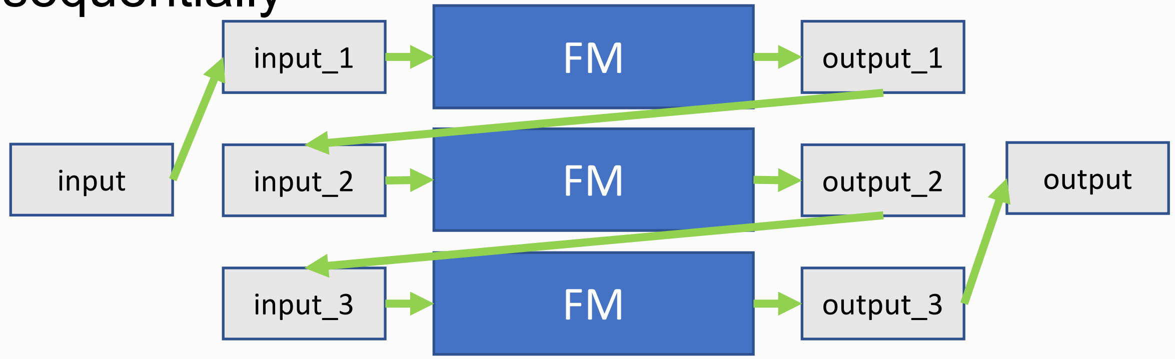


Cognitive architecture #2: asking parallel subquestions

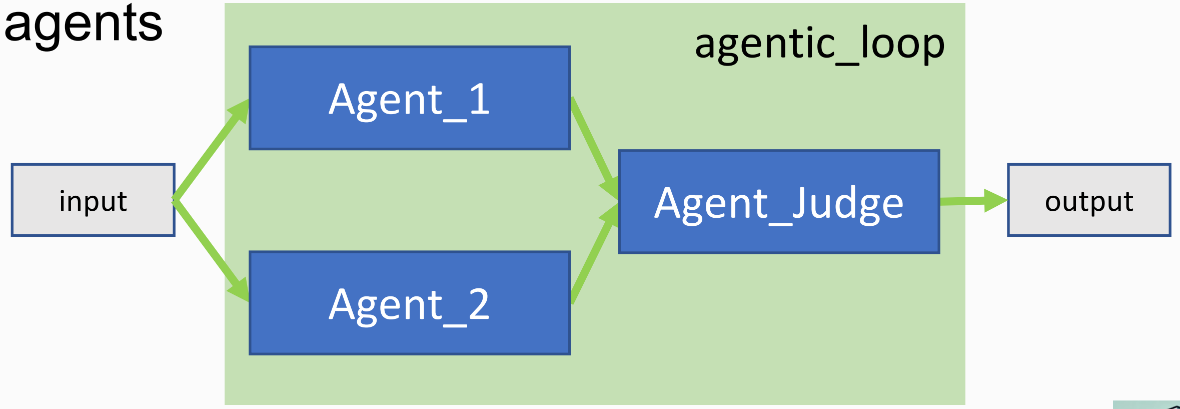


- **Complexity**
- **Cost**
- **Predictability**

Cognitive architecture #3: ask qu
sequentially



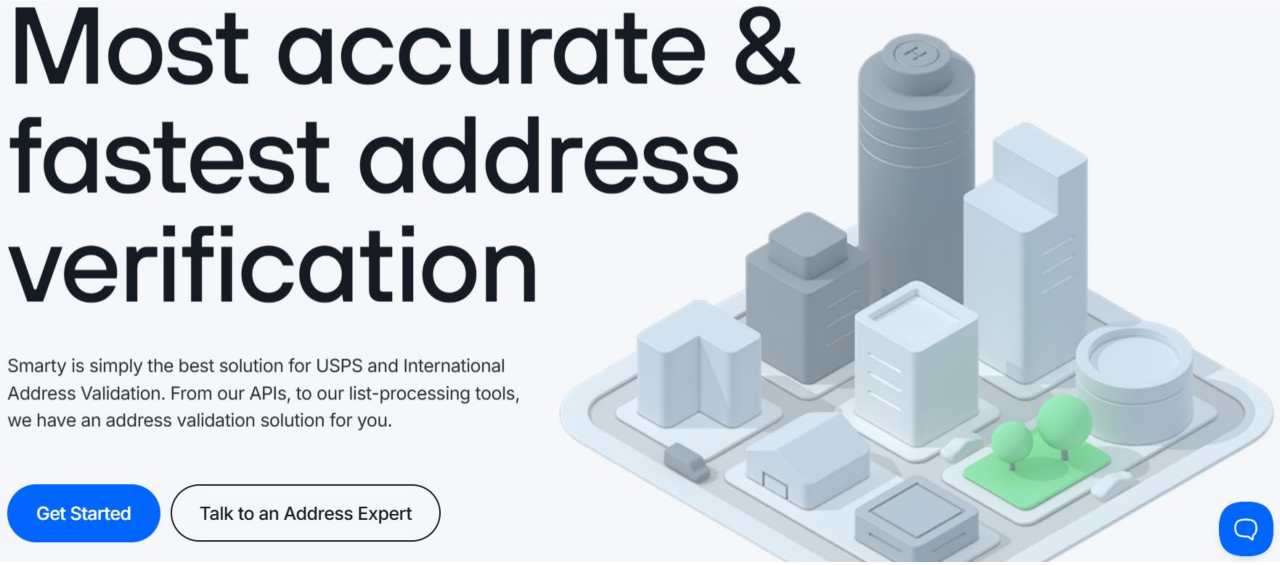
Cognitive architecture #4: solving the problem by agents



Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

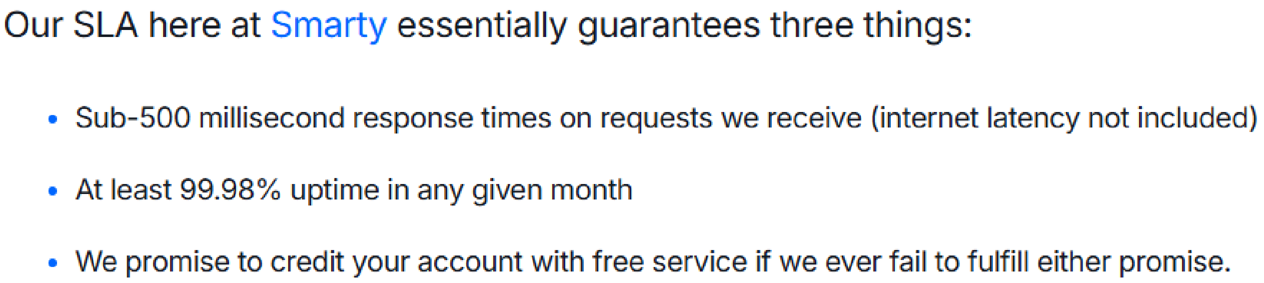
Service Level Agreements

- An SLA (service level agreement) is a part of a standardized service contract in which specific aspects of a service are defined by a service provider.
- “Get your pizza in 20 minutes or less, or it's free.”



Most accurate & fastest address verification

Smarty is simply the best solution for USPS and International Address Validation. From our APIs, to our list-processing tools, we have an address validation solution for you.



Our SLA here at Smarty essentially guarantees three things:

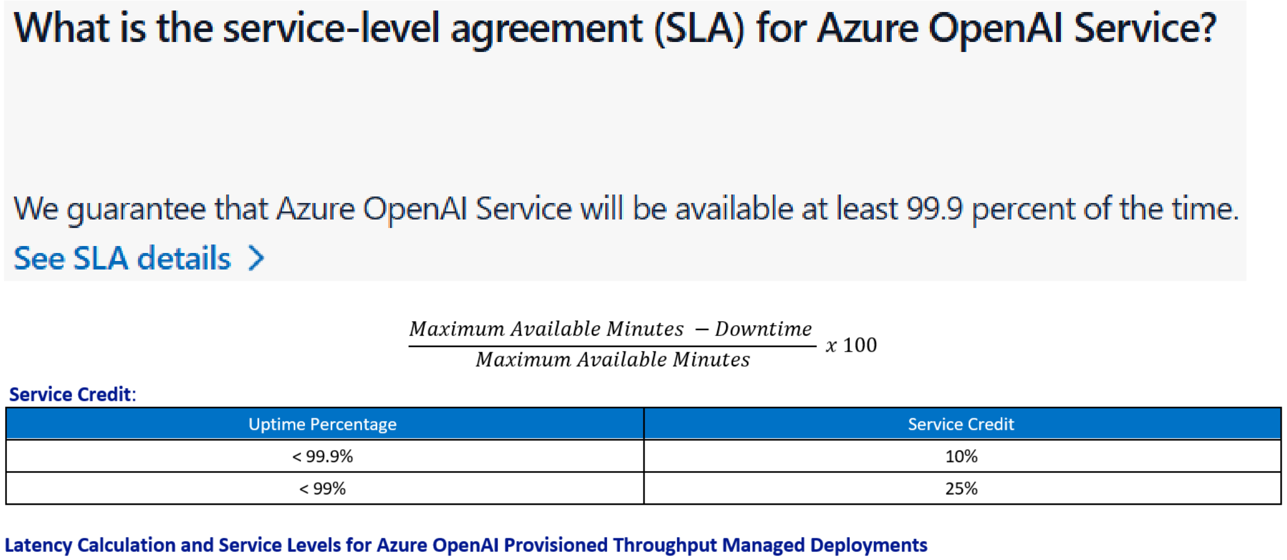
- Sub-500 millisecond response times on requests we receive (internet latency not included)
- At least 99.98% uptime in any given month
- We promise to credit your account with free service if we ever fail to fulfill either promise.

https://www.smarty.com/articles/sla

Service Level Agreements

- Online services expect small latency and high availability.
- Service Level Agreements cover many performance metrics.

Azure OpenAI SLA



What is the service-level agreement (SLA) for Azure OpenAI Service?

We guarantee that Azure OpenAI Service will be available at least 99.9 percent of the time.

See SLA details >

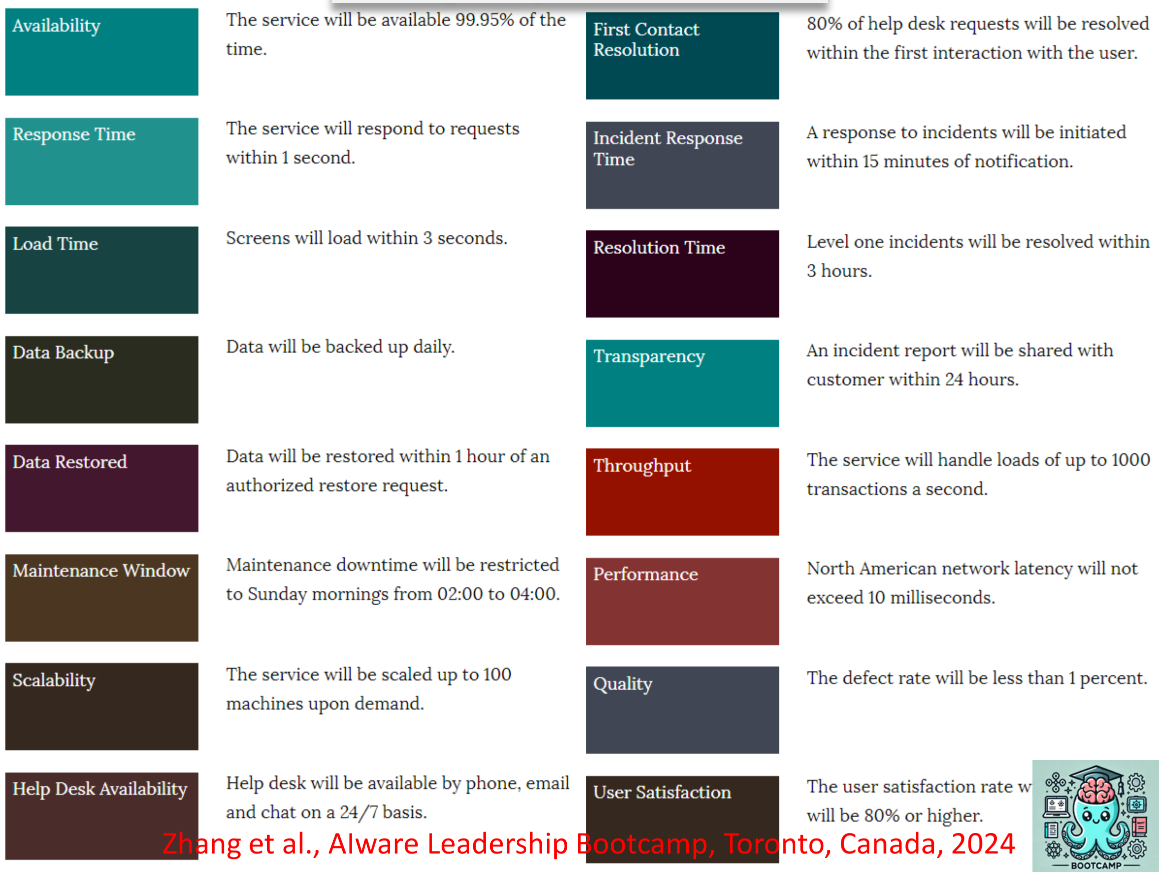
$$\frac{Maximum\ Available\ Minutes\ - Downtime}{Maximum\ Available\ Minutes}\ x\ 100$$

Service Credit:

Uptime Percentage	Service Credit
< 99.9%	10%
< 99%	25%

Latency Calculation and Service Levels for Azure OpenAI Provisioned Throughput Managed Deployments

Many aspects of SLA



Availability	The service will be available 99.95% of the time.	First Contact Resolution	80% of help desk requests will be resolved within the first interaction with the user.
Response Time	The service will respond to requests within 1 second.	Incident Response Time	A response to incidents will be initiated within 15 minutes of notification.
Load Time	Screens will load within 3 seconds.	Resolution Time	Level one incidents will be resolved within 3 hours.
Data Backup	Data will be backed up daily.	Transparency	An incident report will be shared with customer within 24 hours.
Data Restored	Data will be restored within 1 hour of an authorized restore request.	Throughput	The service will handle loads of up to 1000 transactions a second.
Maintenance Window	Maintenance downtime will be restricted to Sunday mornings from 02:00 to 04:00.	Performance	North American network latency will not exceed 10 milliseconds.
Scalability	The service will be scaled up to 100 machines upon demand.	Quality	The defect rate will be less than 1 percent.
Help Desk Availability	Help desk will be available by phone, email and chat on a 24/7 basis.	User Satisfaction	The user satisfaction rate w will be 80% or higher.

https://azure.microsoft.com/en-us/products/ai-services/openai-service

Challenges in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

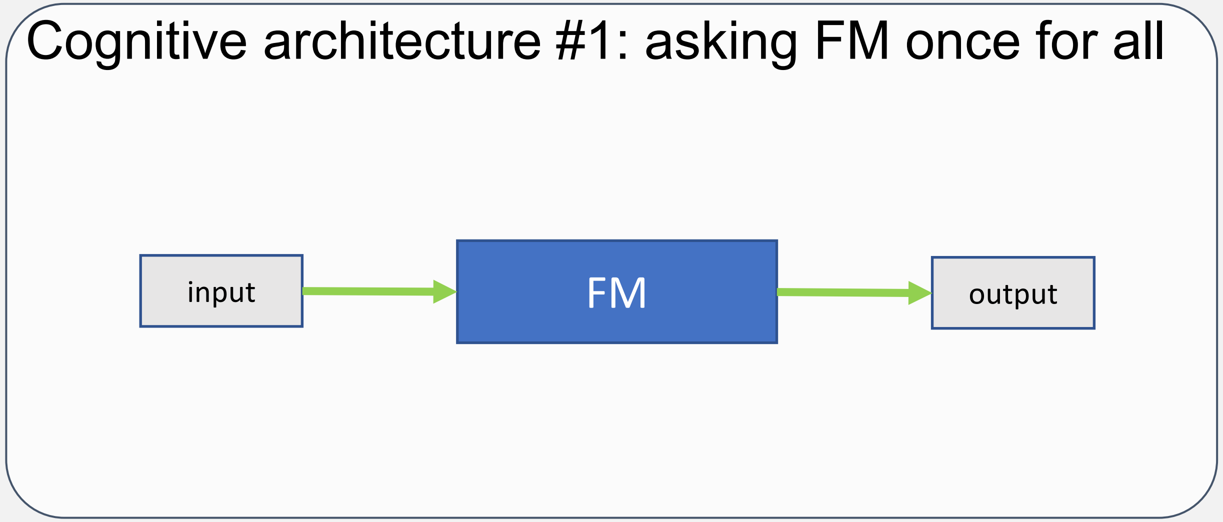
Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

Challenge #3: Complexity of performance tuning and optimization of FMware

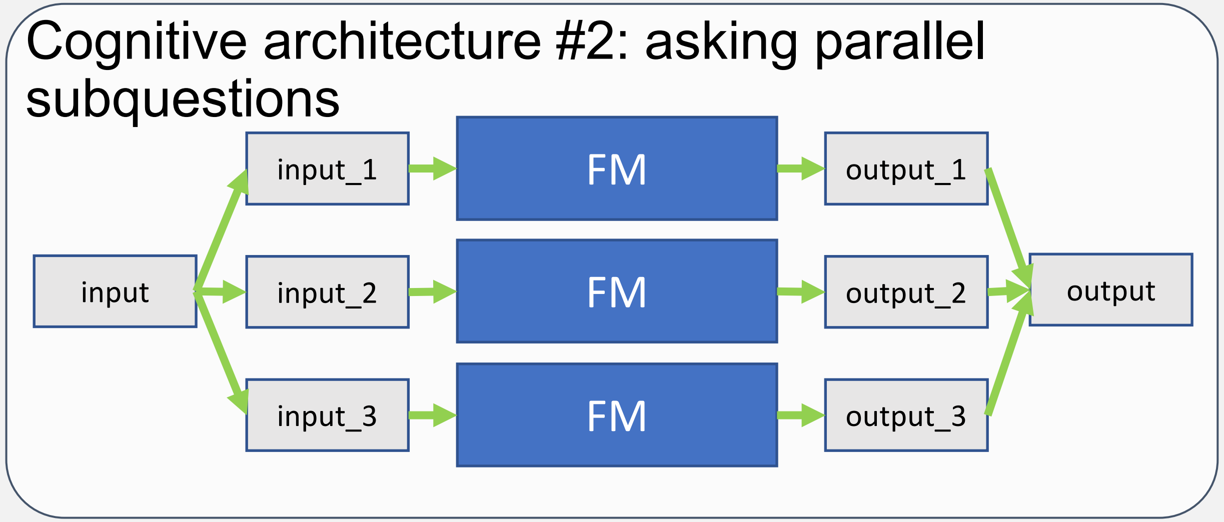
Challenge #4: Complexity of deploying FMware

Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture

Simple cognitive architecture, with 1 FM that is powerful enough to handle multiple aspects (retrieving name, email, address), but high cost per execution.



Complex cognitive architecture with multiple but simpler FMs, lower cost per inference request, but more requests.

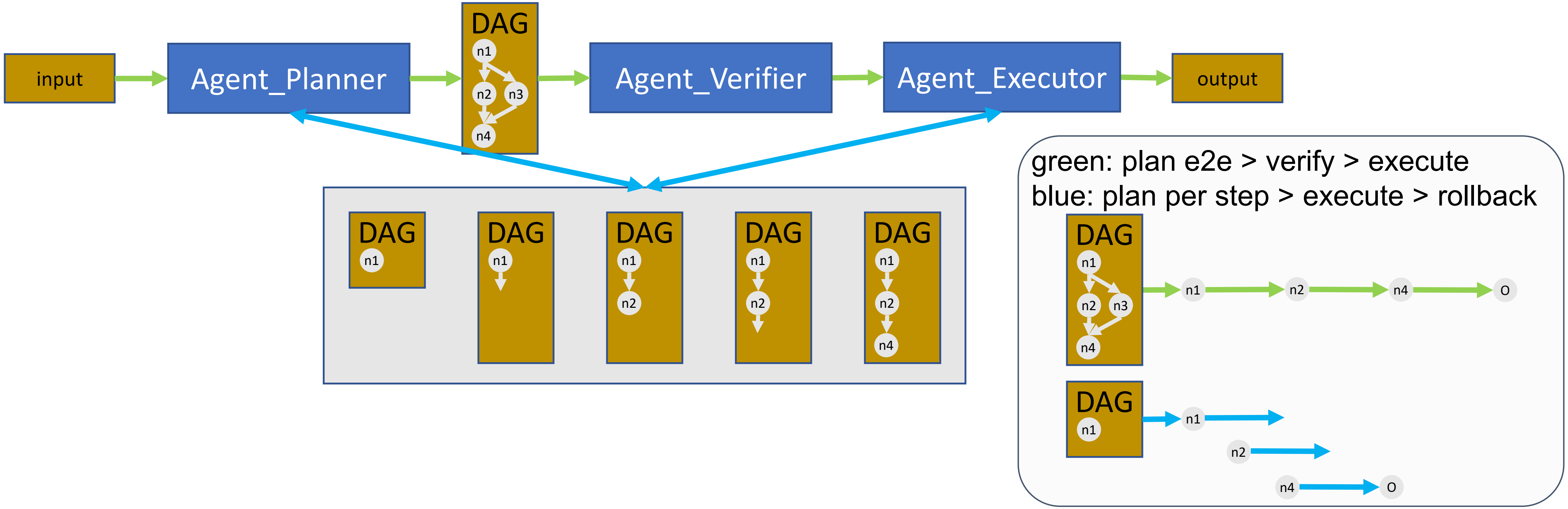


- Some FMs are 10x more expensive per request.
- First token generation significantly more expensive.

- FMware quality: smaller FMs in complex architectures can match or exceed performance of larger FMs.
- FMware latency: more complex architectures may result in higher end-to-end latency.

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code



green: plan e2e > verify > execute

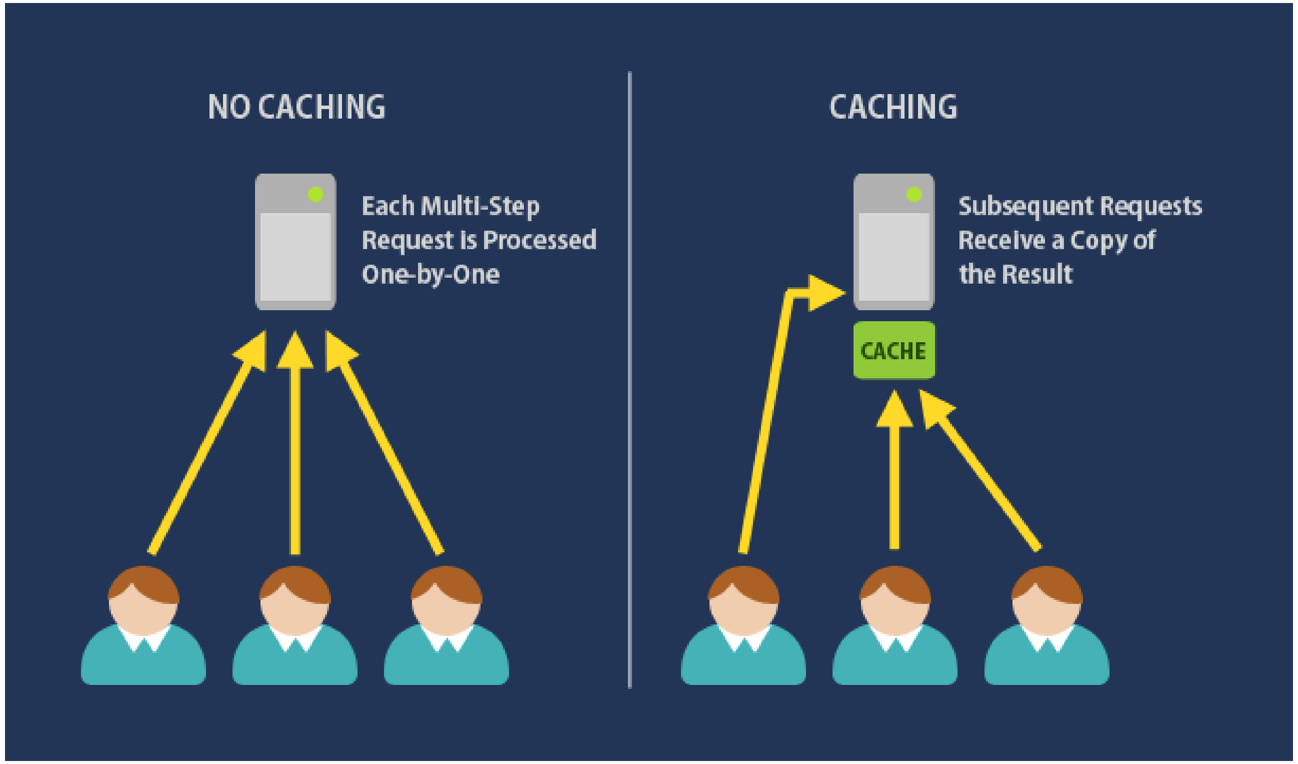
blue: plan per step > execute > rollback

- FMware generates “source code” on the fly.
- Substantial delay in waiting for the complete “source code”, then verifying correctness before executing the plan.

- Pipeline the execution while the plan is still being generated, offers better responsiveness, but risks complex rollbacks if plan fails midway.

The addition of semantic caching throughout the cognitive architecture

- Semantic caching identify similar requests or previously produced content, minimizing FM calls, reducing redundant processing, thus lowering FMware latency.
- How to: efficient storage and quick retrieval of cached results, effective memory management, data retrieval speed optimization.


NO CACHING
Each Multi-Step Request Is Processed One-by-One
CACHING
Subsequent Requests Receive a Copy of the Result
CACHE

Innovation Path in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Design complex cognitive architectures, not only focusing on accuracy, but also tuning the FMware performance
- Develop techniques to help architects balance complex cognitive architectures with performance considerations (e.g., latency, cost)
- Systematically assist architects to reason about FMware design choices about pipelining and rollbacks

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

Challenge #3: Complexity of performance tuning and optimization of FMware

Challenge #4: Complexity of deploying FMware

Challenges in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

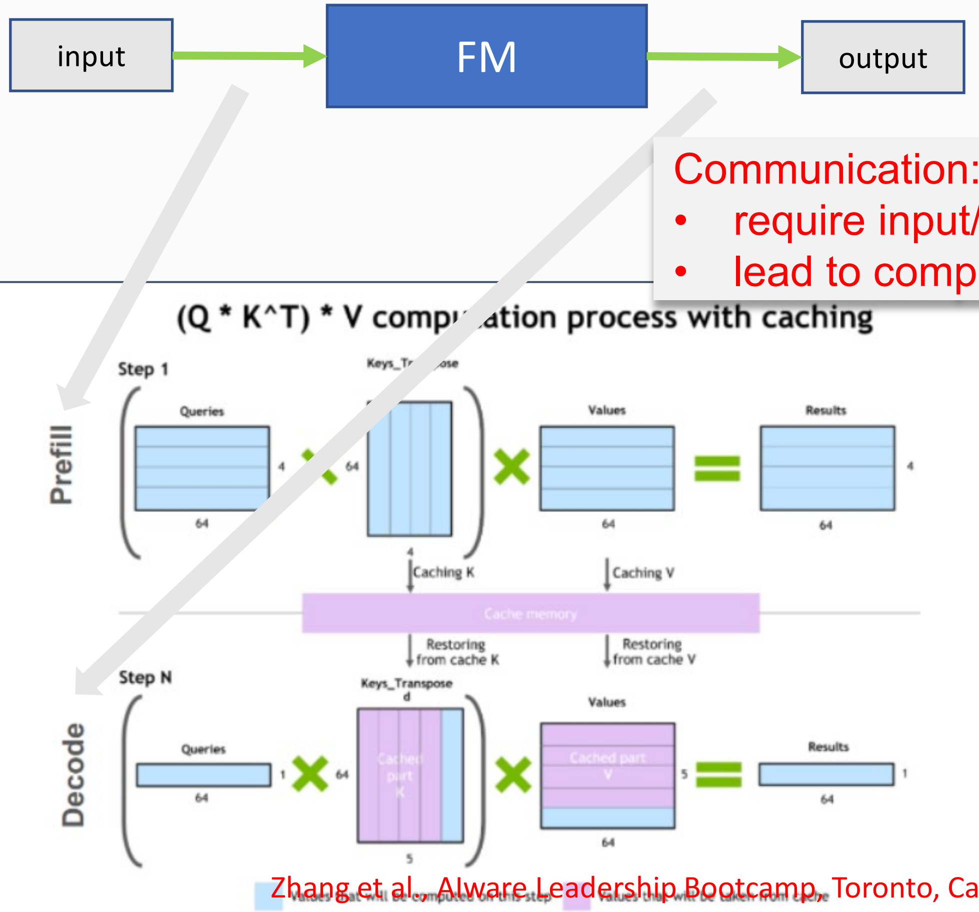
Challenge #3: Complexity of performance tuning and optimization of FMware

Challenge #4: Complexity of deploying FMware

Communication

- Natural language vs function calls:
  - more verbose
  - more complex
  - more variable

Cognitive architecture #1: asking FM once for all



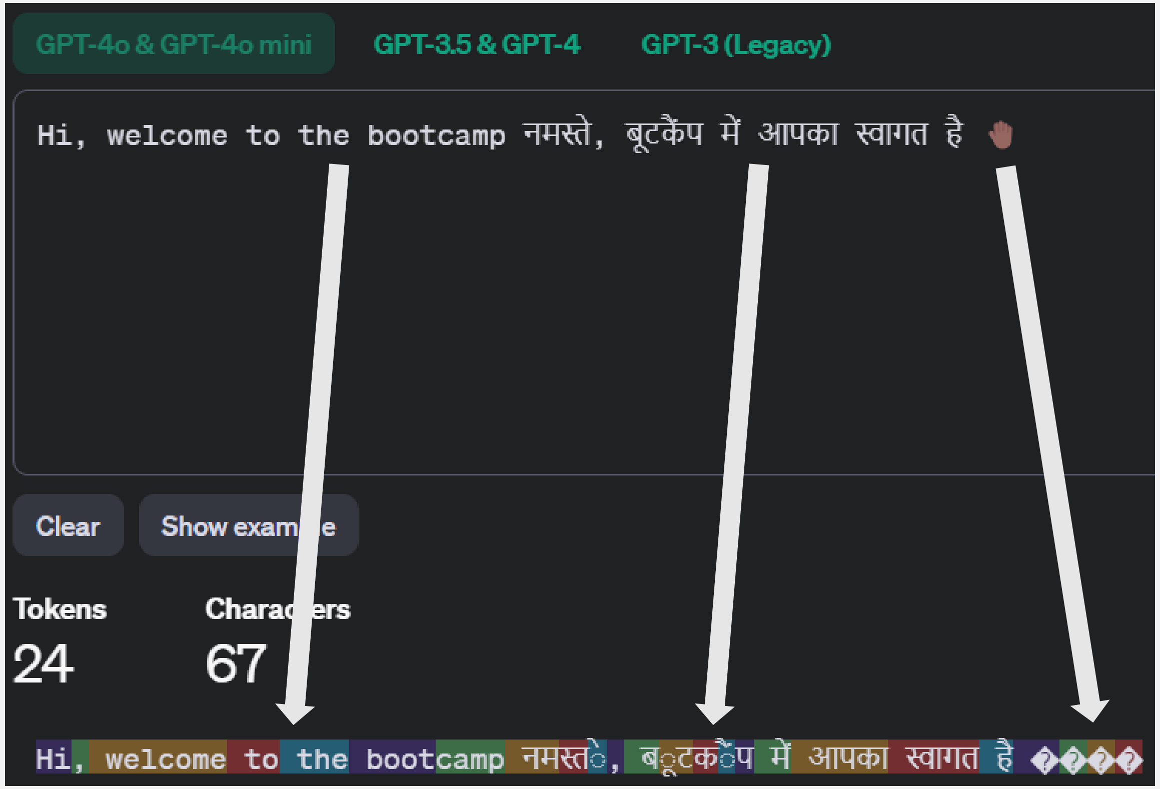
Communication:
- require input/output tokens
- lead to compute overhead

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Deciding the communication language

- More tokens > longer processing time
- Language choices affect AI components in FMware meeting performance requirements
  - In API-hosted models, higher costs & longer response times with more tokens
  - In self-hosted models, although language-specific fine-tuning may help, it can cost substantial compute

Different languages have efficiency and density variations, causing word-to-token ratio varies across languages (English: 7, Hindi: 14, Emoji: 4)



Hi, welcome to the bootcamp नमस्ते, बूटकैंप में आपका स्वागत है 🖐

Tokens 24

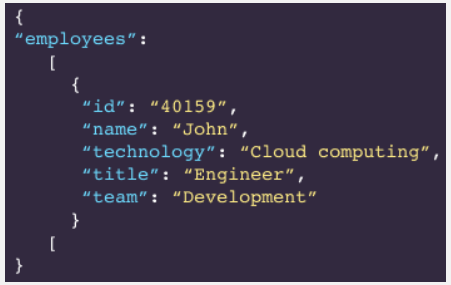
Characters 67

Hi, welcome to the bootcamp नमस्त े, ब ूटक ैंप में आपका स्वागत है ����

Defining the communication format

JSON

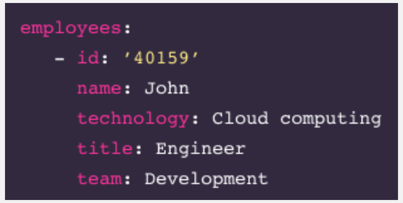
- FMs fine-tuned, more reliably
- more tokens than necessary



```
{
“employees”:
   [
     {
       “id”: “40159”,
       “name”: “John”,
       “technology”: “Cloud computing”,
       “title”: “Engineer”,
       “team”: “Development”
     }
   [
}
```

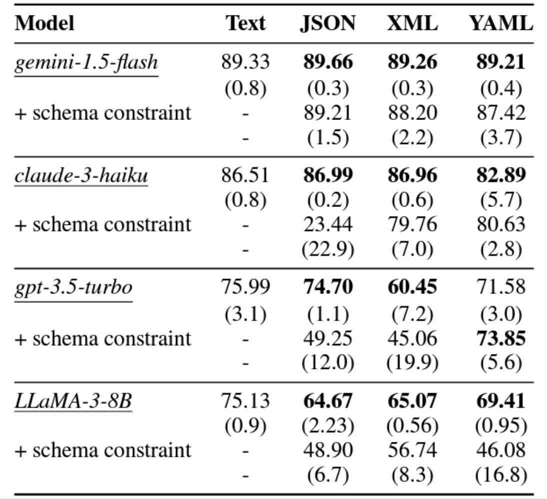
YAML

- less reliable output format
- fewer tokens



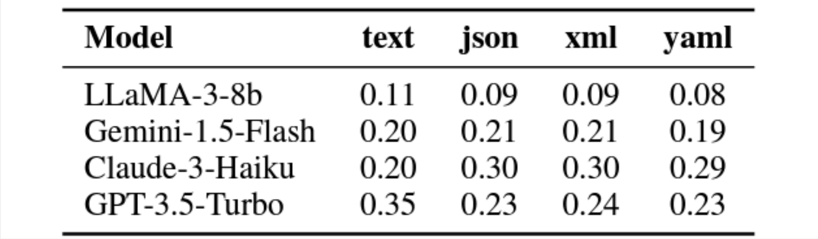
```
employees:
   - id: ‘40159’
     name: John
     technology: Cloud computing
     title: Engineer
     team: Development
```

Adding schema degrades the exact match in the GSM8K dataset



Model	Text	JSON	XML	YAML
gemini-1.5-flash	89.33	**89.66**	**89.26**	**89.21**
	(0.8)	(0.3)	(0.3)	(0.4)
+ schema constraint	-	89.21	88.20	87.42
	-	(1.5)	(2.2)	(3.7)
claude-3-haiku	86.51	**86.99**	**86.96**	**82.89**
	(0.8)	(0.2)	(0.6)	(5.7)
+ schema constraint	-	23.44	79.76	80.63
	-	(22.9)	(7.0)	(2.8)
gpt-3.5-turbo	75.99	**74.70**	**60.45**	71.58
	(3.1)	(1.1)	(7.2)	(3.0)
+ schema constraint	-	49.25	45.06	**73.85**
	-	(12.0)	(19.9)	(5.6)
LLaMA-3-8B	75.13	**64.67**	**65.07**	**69.41**
	(0.9)	(2.23)	(0.56)	(0.95)
+ schema constraint	-	48.90	56.74	46.08
	-	(6.7)	(8.3)	(16.8)

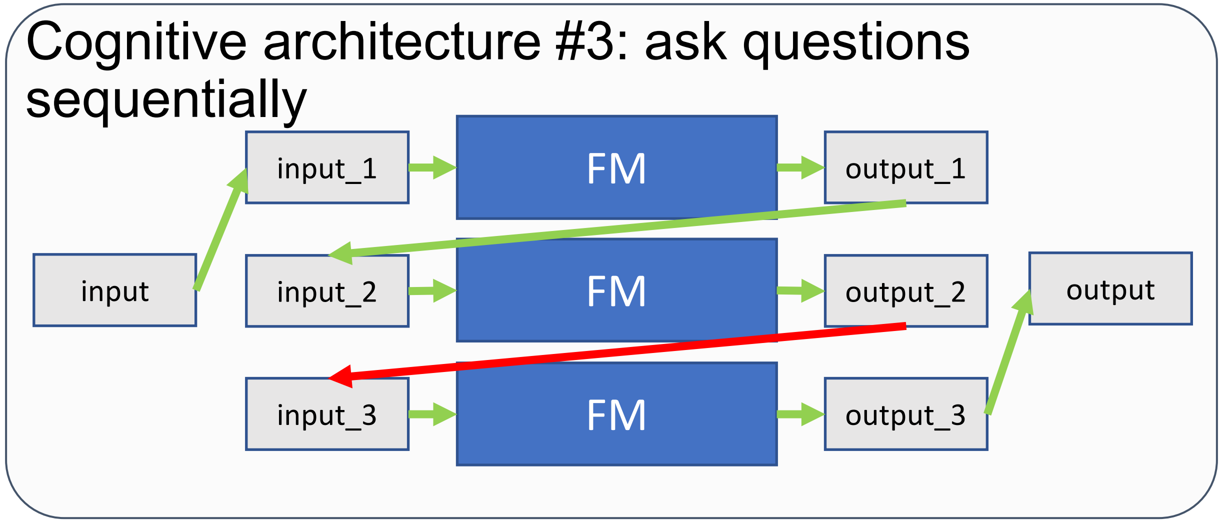
Cost for different models and output formats over 6 dataset



Model	text	json	xml	yaml
LLaMA-3-8b	0.11	0.09	0.09	0.08
Gemini-1.5-Flash	0.20	0.21	0.21	0.19
Claude-3-Haiku	0.20	0.30	0.30	0.29
GPT-3.5-Turbo	0.35	0.23	0.24	0.23

Let Me Speak Freely? A Study on the Impact of Format Restrictions on Performance of Large Language Models,
Tam et al., https://arxiv.org/abs/2408.02442

Correcting communication messages



- Incorrectly structured language/format may cause downstream FMware components to fail or misunderstand input

- Correct communication, e.g., defining communication rules, including examples
- Increasing token count, leading to higher latency and resource consumption

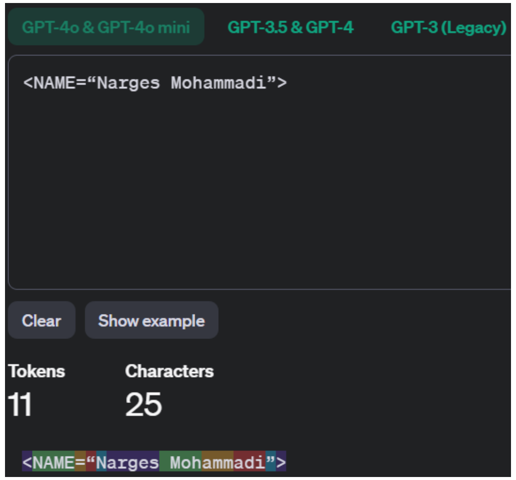
Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Optimizing communication messages

```
format rule:
<NAME=“xxx”>

question:
What is the name of the Nobel prize
winner for peace in 2023?

generation:
<NAME=“Narges Mohammadi”>
```

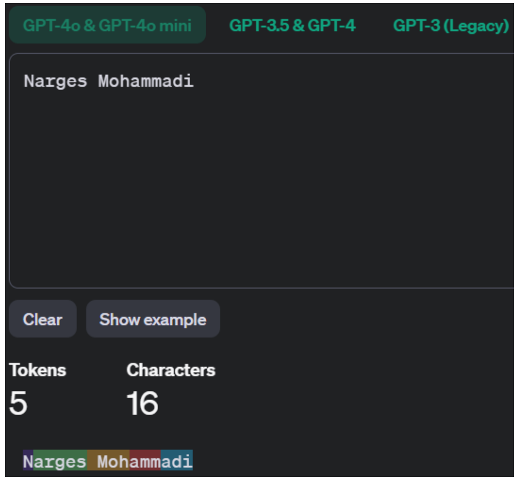


```
<NAME=“Narges Mohammadi”>
```

Tokens 11

Characters 25

```
<NAME=“Narges Mohammadi”>
```



```
Narges Mohammadi
```

Tokens 5

Characters 16

```
Narges Mohammadi
```

- Inefficient token generation in predictable or structurally consistent communication messages

- Difficult to dynamically identify such structures, while accurately extracting and generating only the variable components

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Expressiveness vs Efficiency

Innovation Path in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Design complex cognitive architectures, not only focusing on accuracy, but also tuning the FMware performance
- Develop techniques to help architects balance complex cognitive architectures with performance considerations (e.g., latency, cost)
- Systematically assist architects to reason about FMware design choices about pipelining and rollbacks

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Address token count disparities across languages, e.g., assigning more powerful accelerators to LRLs, prompt compression, fine-tuning
- Reduce invalid outputs across different communication schemas
- Offload communication correction to CPU for reducing cost

Challenge #3: Complexity of performance tuning and optimization of FMware

Challenge #4: Complexity of deploying FMware

Challenges in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

Challenge #3: Complexity of performance tuning and optimization of FMware

- Complex model-level optimization
- Excessive amount of performance configuration knobs
- Evolving and moving target

Challenge #4: Complexity of deploying FMware

Complex model-level optimization

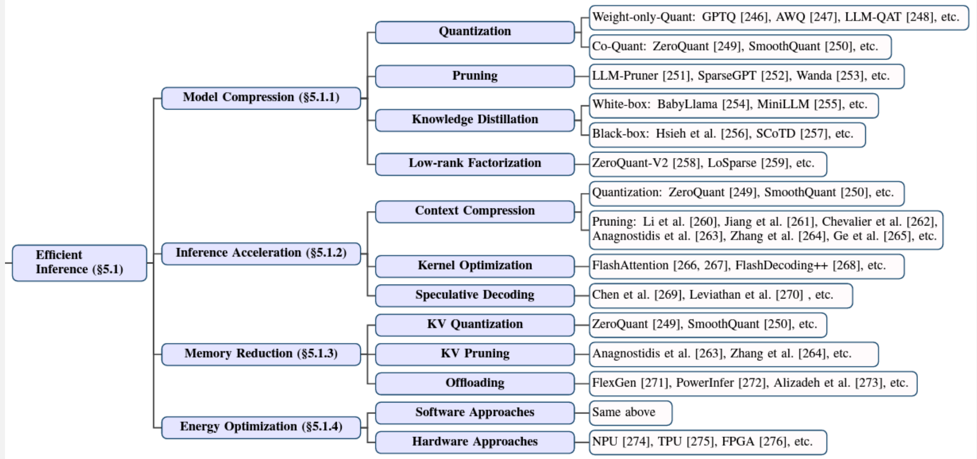
Various model-level performance optimization techniques:

- e.g., multi-query attention, knowledge distillation, quantization

focusing on various performance considerations:

- hardware utilization, latency, throughput, or cost

More details are be referred from this survey, and actually there are 10+ surveys like so about FM inference performance optimization



- Efficient Inference (§5.1)
  - Model Compression (§5.1.1)
    - Quantization
      - Weight-only-Quant: GPTQ [246], AWQ [247], LLM-QAT [248], etc.
      - Co-Quant: ZeroQuant [249], SmoothQuant [250], etc.
    - Pruning
      - LLM-Pruner [251], SparseGPT [252], Wanda [253], etc.
    - Knowledge Distillation
      - White-box: BabyLlama [254], MiniLLM [255], etc.
      - Black-box: Hsieh et al. [256], SCoTD [257], etc.
    - Low-rank Factorization
      - ZeroQuant-V2 [258], LoSparse [259], etc.
  - Inference Acceleration (§5.1.2)
    - Context Compression
      - Quantization: ZeroQuant [249], SmoothQuant [250], etc.
      - Pruning: Li et al. [260], Jiang et al. [261], Chevalier et al. [262], Anagnostidis et al. [263], Zhang et al. [264], Ge et al. [265], etc.
    - Kernel Optimization
      - FlashAttention [266, 267], FlashDecoding++ [268], etc.
    - Speculative Decoding
      - Chen et al. [269], Leviathan et al. [270] , etc.
  - Memory Reduction (§5.1.3)
    - KV Quantization
      - ZeroQuant [249], SmoothQuant [250], etc.
    - KV Pruning
      - Anagnostidis et al. [263], Zhang et al. [264], etc.
    - Offloading
      - FlexGen [271], PowerInfer [272], Alizadeh et al. [273], etc.
  - Energy Optimization (§5.1.4)
    - Software Approaches
      - Same above
    - Hardware Approaches
      - NPU [274], TPU [275], FPGA [276], etc.

question:
what tools do you use that have something to do with any of these things?

- Lack of techniques and tools directly impacting FMware developers, who interact with models through prompting

- Complex prompt decomposition increases FM calls
- Explainability add output token count overhead
- Chained FM calls sequential waiting dependencies

Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security, Li et al., https://arxiv.org/abs/2401.05459

Excessive amount of performance configuration knobs

- Multiple dimensions of configuration space for FMware performance optimization:
  - cognitive architectures, prompt design, model selection, quantization, fine-tuning, communication protocols, etc.

- Model selection: trade-offs among quality metrics, generation speed, memory usage, and SLA compliance
- Inference engine selection
- System level configuration:
  - resource orchestration across model loading/unloading, multi-team environments, heterogeneous workloads, etc.

Evolving and moving target

- FMware has live evolution nature, e.g., a single agent execution can lead to system-wide adjustments, agent has self-evolving capabilities.

- Reproducible issue due to token sampling in inference, also due to dynamic data flywheel or self-exploration.
- Although lower temperature or seed configuration can be applied for reproducibility, sacrifice autonomous exploration capability.

Innovation Path in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

Challenge #3: Complexity of performance tuning and optimization of FMware

- Systematic performance optimization for FMware, e.g., multi-objective, search-based
- Comprehensive benchmarking, evaluation of configuration efficiency
- Develop best practices and anti-patterns
- Automated performance tuning to reduce manual effort, e.g., simulation for configuration testing
- Decide on the optimal frequency of model updates, fine-tuning with new datasets vs improving prompting/post-processing techniques, e.g., using cost-benefit analysis

Challenge #4: Complexity of deploying FMware

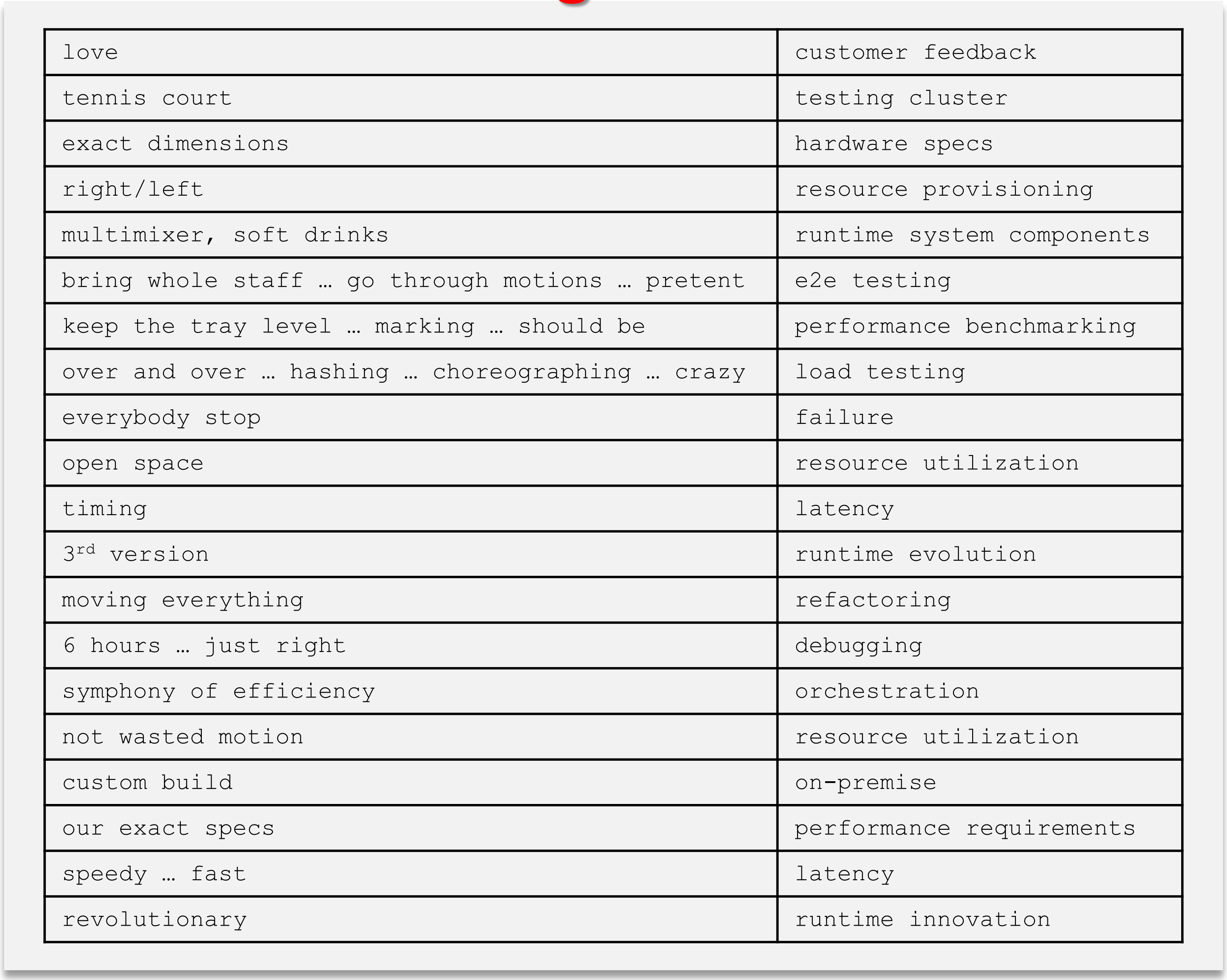
McDonald's Speedy Service System

you are gonna **love** how we did it ... brings me out to this **tennis court** ... drawn this line the **exact dimenions** of our kitchen ... Sink on the **right**, extruder on the **left** … **Multimixer, soft drinks** ... **bring out our whole staff** and we have 'em **go through the motions** ... making **pretend** burgers and fries … **keep the tray level** … with a stick **marking** where all the equipment **should be** … do it **over and over**, **hashing it out**, **choreographing** it like some **crazy** burger ballet … No! NO! **Everybody stop**! … See all this **open space** here now? … This is the **timing** if the Lazy Susan ... I still think there's a **3rd version** … I want to **move everything** … after about **6 hours** of this, we get it **just right** … It's a **symphony of efficiency**, **not** a **wasted motion** … **custom build** the kitchen to **our exact specs** … The **speedy** system is born. The world's first ever system to deliver food **fast**. It is totally **revolutionary**

https://www.youtube.com/watch?v=jTageuhPfAM

Runtime: environment or infrastructure that supports the execution of a program while it is running

you are gonna **love** how we did it ... brings me out to this **tennis court** ... drawn this line the **exact dimenions** of our kitchen ... Sink on the **right**, extruder on the **left** … **Multimixer, soft drinks** ... **bring out our whole staff** and we have 'em **go through the motions** ... making **pretend** burgers and fries … **keep the tray level** … with a stick **marking** where all the equipment **should be** … do it **over and over**, **hashing it out**, **choreographing** it like some **crazy** burger ballet … No! NO! **Everybody stop**! … See all this **open space** here now? … This is the **timing** if the Lazy Susan ... I still think there's a **3rd version** … I want to **move everything** … after about **6 hours** of this, we get it **just right** … It's a **symphony of efficiency**, **not** a **wasted motion** … **custom build** the kitchen to **our exact specs** … The **speedy** system is born. The world's first ever system to deliver food **fast**. It is totally **revolutionary**



love	customer feedback
tennis court	testing cluster
exact dimensions	hardware specs
right/left	resource provisioning
multimixer, soft drinks	runtime system components
bring whole staff … go through motions … pretent	e2e testing
keep the tray level … marking … should be	performance benchmarking
over and over … hashing … choreographing … crazy	load testing
everybody stop	failure
open space	resource utilization
timing	latency
3rd version	runtime evolution
moving everything	refactoring
6 hours … just right	debugging
symphony of efficiency	orchestration
not wasted motion	resource utilization
custom build	on-premise
our exact specs	performance requirements
speedy … fast	latency
revolutionary	runtime innovation

Challenges in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

Challenge #3: Complexity of performance tuning and optimization of FMware

- Complex model-level optimization
- Excessive amount of performance configuration knobs
- Evolving and moving target

Challenge #4: Complexity of deploying FMware

- Selecting optimal deployment options when hosting FMware
- Deploying multi-process FMware efficiently
- Deploying multi-tenant FMware efficiently

Selecting optimal deployment options when hosting FMware

- High computation cost of running FMs with accelerators
- API limitation: performance unpredictability and unreliability, while simple to set up
- Cloud inefficiency: potentially low hardware utilization, unnecessary costs if not managed effectively, while offering flexibility and scalability
- On-premise complexity: require substantial engineering effort to optimize hardware usage and ensure muiti-tenancy, while providing maximum control
- Diverse deployment options with pros and cons
- Complexity of a balanced act, to carefully weigh control, cost, performance, and hardware utilization

Deploying multi-process FMware efficiently

- Multiple concurrent processes, share compute and bandwidth resources, on a unified cluster: inference, fine-tuning, agent autonomous planning and execution.
- Current "Model-as-a-Service" paradigm insufficient for multi-process FMware runtime architecture

- Complexity of scheduling: due to cross-process interference, subprocess interference (prefill vs decode), as processes are inertial, costly to preempt or revert the state (e.g., model weight loading, data loading).
- Complexity of memory management: variable prompt/output lengths affect available memory, dynamic KV cache memory, making memory allocation unpredictable.
- Complexity of agent: autonomous agent on the fly, lacking dynamic resource allocation, also resource contention between these agents.

Deploying multi-tenant FMware efficiently

- High hardware costs force multi-tenant deployment as a shared cluster.

- Complexity to balance competing SLAs by tenants.
- Complexity to maximize cluster-wide hardware utilization, while meeting performance goals of many tenants' numerous FMware deployment.
- Optimize and balance latency/throughput for cross-FMware model sharing to avoid loading costs.

Innovation Path in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

Challenge #3: Complexity of performance tuning and optimization of FMware

- Systematic performance optimization for FMware, e.g., multi-objective, search-based
- Comprehensive benchmarking, evaluation of configuration efficiency
- Develop best practices and anti-patterns
- Automated performance tuning to reduce manual effort, e.g., simulation for configuration testing
- Decide on the optimal frequency of model updates, fine-tuning with new datasets vs improving prompting/post-processing techniques, e.g., using cost-benefit analysis

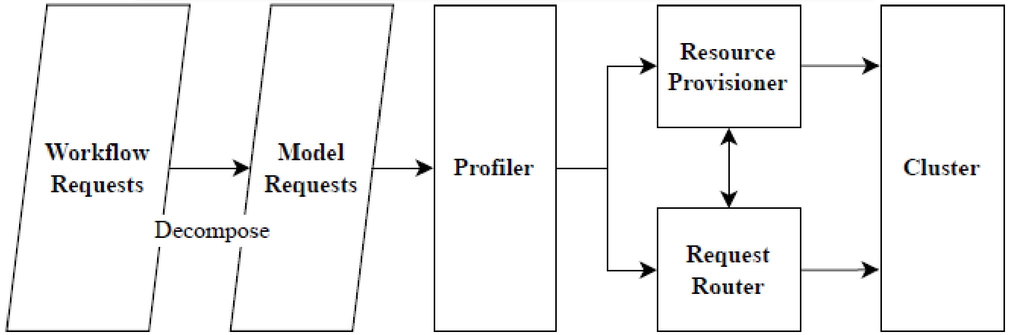
Challenge #4: Complexity of deploying FMware

- Hybrid hosting approaches for FMware, leveraging the benefits of different deployment options
- Integrate SLA-aware scheduling and scaling for optimized performance or cost efficiency, to align with application level performance targets
- Determine optimal granularity for inter-process separation (training vs inference), and intra-process separation (prefill/decode)
- From siloed approaches, to FMware-level, multi-tenant, and cluster optimization

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Our vision: SLA-aware FMware Runtime

FMware Runtime: a simplified architecture and components



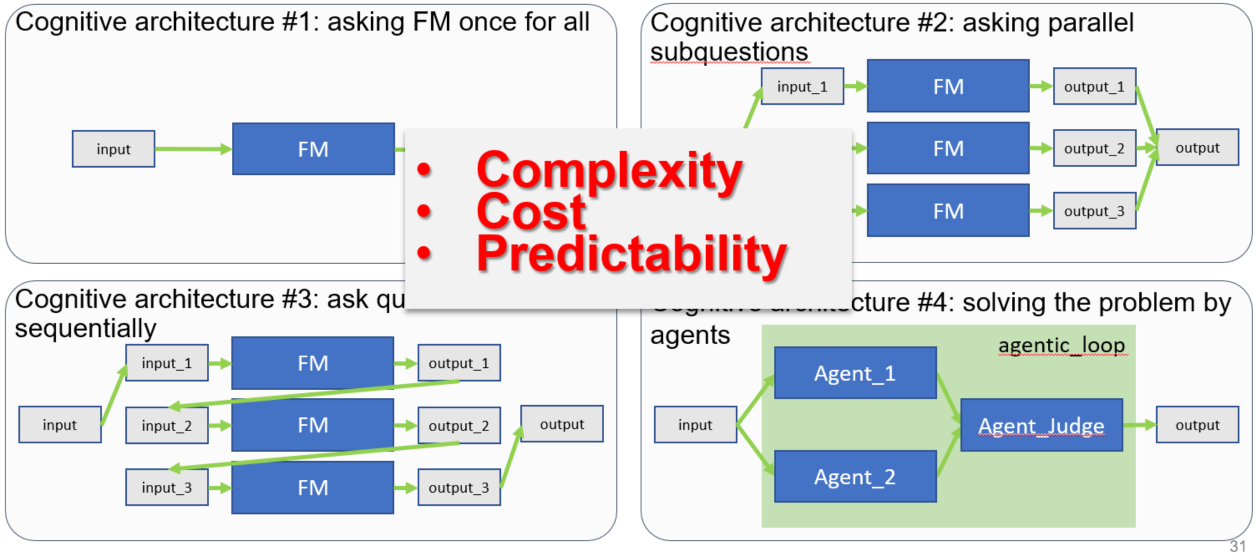
FMware Runtime is a runtime system in production. It significantly reduces SLA violations, and saves costs.

- currently writing a paper with more details and show our evaluation results
- continuously extending FMware Runtime, as an effort to tackle other aforementioned challenges
- the benefit of this system is mentioned in Data Flywheel, in a way that we prototype, observe, improve efficiency, iterate, all in a centrally designed runtime system

- Goal: SLAs as a first citizen in runtime design, guarantee SLA of each FMware while optimizing cluster-level hardware utilization.
- DAG (node: task, e.g., FM inference, traditional code execution, external API call, edge: control flow dependency, e.g., sequential or conditional branching).
- #1. DAG decomposition.
- #2. Profiler. Estimate performance measures (e.g., latency, memory) for individual nodes; done once only.
- #3: Resource Provisioner. Decide to allocate/release hardware resources, by a risk-aware SLA violation algorithm.
- #4: Request Router. Decide to route requests into model replicas. It is crucial to route so to decide which FM replica should serve a request, based on how likely it can avoid SLA at risk. Resource Provisioner and Request Router work together to monitor SLA compliance, to make joint decisions together.
- #5: Cluster. Handle the execution of actions that are fired from Resource Provisioner and Request Router. Cross-node communication and data movement (e.g., loading model weights) are also managed.

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Different Cognitive Architectures


Cognitive architecture #1: asking FM once for all
input
FM
Cognitive architecture #2: asking parallel subquestions
input_1
FM
output_1
FM
output_2
output
FM
output_3
Cognitive architecture #3: ask qu
sequentially
input_1
FM
output_1
input
input_2
FM
output_2
output
input_3
FM
output_3
...cture #4: solving the problem by agents
agentic_loop
Agent_1
input
Agent_Judge
output
Agent_2

- **Complexity**
- **Cost**
- **Predictability**

Challenges in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

Challenge #3: Complexity of performance tuning and optimization of FMware

- Complex model-level optimization
- Excessive amount of performance configuration knobs
- Evolving and moving target

Challenge #4: Complexity of deploying FMware

- Selecting optimal deployment options when hosting FMware
- Deploying multi-process FMware efficiently
- Deploying multi-tenant FMware efficiently

Innovation Path in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token-efficient communication language between the AI components of FMware

- Deciding the communication language
- Defining the communication format
- Correcting communication messages
- Optimizing communication messages

Challenge #3: Complexity of performance tuning and optimization of FMware

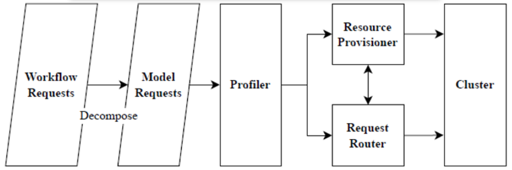
- Systematic performance optimization for FMware, e.g., multi-objective, search-based
- Comprehensive benchmarking, evaluation of configuration efficiency
- Develop best practices and anti-patterns
- Automated performance tuning to reduce manual effort, e.g., simulation for configuration testing
- Decide on the optimal frequency of model updates, fine-tuning with new datasets vs improving prompting/post-processing techniques, e.g., using cost-benefit analysis

Challenge #4: Complexity of deploying FMware

- Hybrid hosting approaches for FMware, leveraging the benefits of different deployment options
- Integrate SLA-aware scheduling and scaling for optimized performance or cost efficiency, to align with application level performance targets
- Determine optimal granularity for inter-process separation (training vs inference), and intra-process separation (prefill/decode)
- From siloed approaches, to FMware-level, multi-tenant, and cluster optimization

Our vision: SLA-aware FMware Runtime

FMware Runtime: a simplified architecture and components



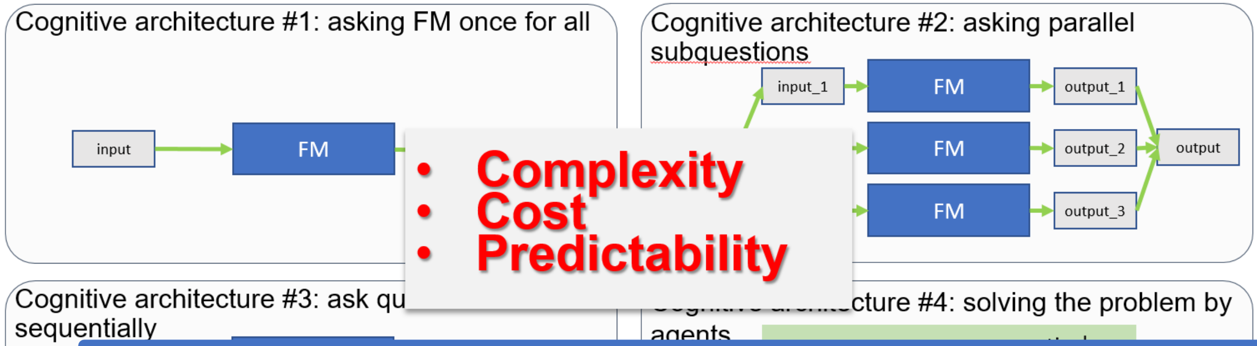
FMware Runtime is a runtime system in production. It significantly reduces SLA violations, and saves costs.

- currently writing a paper with more details and show our evaluation results
- continuously extending FMware Runtime, as an effort to tackle other aforementioned challenges
- the benefit of this system is mentioned in Data Flywheel, in a way that we prototype, observe, improve efficiency, iterate, all in a centrally designed runtime system

- Goal: SLAs as a first citizen in runtime design, guarantee SLA of each FMware while optimizing cluster-level hardware utilization.
- DAG (node: task, e.g., FM inference, traditional code execution, external API call, edge: control flow dependency, e.g., sequential or conditional branching).
- #1. DAG decomposition.
- #2. Profiler. Estimate performance measures (e.g., latency, memory) for individual nodes; done once only.
- #3: Resource Provisioner. Decide to allocate/release hardware resources, by a risk-aware SLA violation algorithm.
- #4: Request Router. Decide to route requests into model replicas. It is crucial to route so to decide which FM replica should serve a request, based on how likely it can avoid SLA at risk. Resource Provisioner and Request Router work together to monitor SLA compliance, to make joint decisions together.
- #5: Cluster. Handle the execution of actions that are fired from Resource Provisioner and Request Router. Cross-node communication and data movement (e.g., loading model weights) are also managed.

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Different Cognitive Architectures


Cognitive architecture #1: asking FM once for all
input
FM
Cognitive architecture #2: asking parallel subquestions
input_1
FM
FM
FM
output_1
output_2
output_3
output
Cognitive architecture #3: ask qu
sequentially
input
Cognitive architecture #4: solving the problem by agents

- **Complexity**
- **Cost**
- **Predictability**

Challenges in SPE4FMware

Challenge #1: Complexity of creating high-performance cognitive architectures

- Picking more powerful FMs within a simple cognitive architecture versus simpler FMs within a more complex cognitive architecture
- Pipelining the execution of cognitive code as it is being generated versus waiting for the full generation and verification of such code
- The addition of semantic caching throughout the cognitive architecture

Challenge #2: Complexity of creating token communication language between the AI c

Challenge #3: Complexity of performance tuning and optimization of FMware

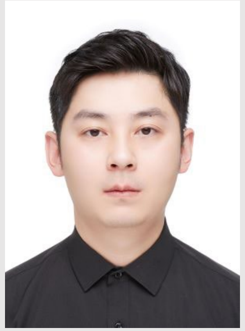
- Complex model-level optimization
- Excessive amount of performance configuration knobs
- Evolving and moving target

https://arxiv.org/abs/2411.09580

Software Performance Engineering for Foundation Model-Powered Software (FMware)

Innovation Path in SPE4FMware

Our vision: SLA-aware FMware Runtime



Haoxiang Zhang

Shi Chang

Arthur Leung

Kishanthan Thangarajah

Boyuan Chen

Hanan Lutfiyya

Ahmed E. Hassan

- Correcting communication messages
- Optimizing communication messages

- Determine optimal granularity for inter-process separation (training vs inference), and intra-process separation (prefill/decode)
- From siloed approaches, to FMware-level, multi-tenant, and cluster optimization

- #1. DAG decomposition.
- #2. Profiler. Estimate performance measures (e.g., latency, memory) for individual nodes; done once only.
- #3: Resource Provisioner. Decide to allocate/release hardware resources, by a risk-aware SLA violation algorithm.

Provisioner and Request Router. Cross-node communication and data movement (e.g., loading model weights) are also managed.

Zhang et al., AIware Leadership Bootcamp, Toronto, Canada, 2024