AI-T3-01

Tenant-aware な生成 AI デザインパターン

Toshinobu Akazawa
Senior Solutions Architect
Amazon Web Services Japan

アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
4. まとめ

アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. **SaaS と生成 AI の知識整理**
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
4. まとめ

SaaS と生成 AI

テナント体験

テナント A
チャット

テナント B
チャット / 情報検索

テナント C
チャット / 業務自動化

ティア

テナント A
ティア: Basic

テナント B
ティア: Advanced

テナント C
ティア: Premium

オペレーション

- アカウント管理
- 監視・セキュリティ
- コスト効率化
- テナント設定管理
- 評価

オンボーディング

- 支払い管理
- プロビジョニング

SaaS データプレーン
(アプリケーション機能)

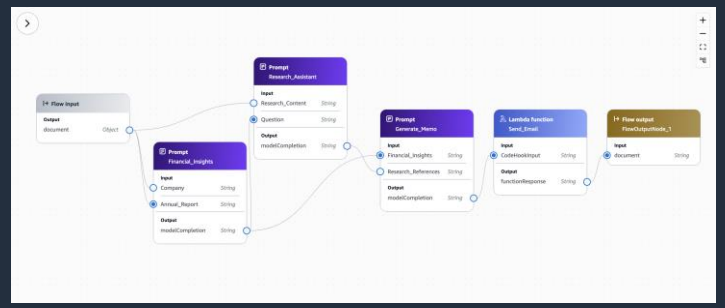
SaaS コントロールプレーン
(管理者向けアプリケーション)

テナント体験の技術階層

SaaS アプリケーション

テナント A　テナント B　テナント C

生成 AI ワークフロー / エージェント



Premium ティア
- チャット
- 業務自動化機能

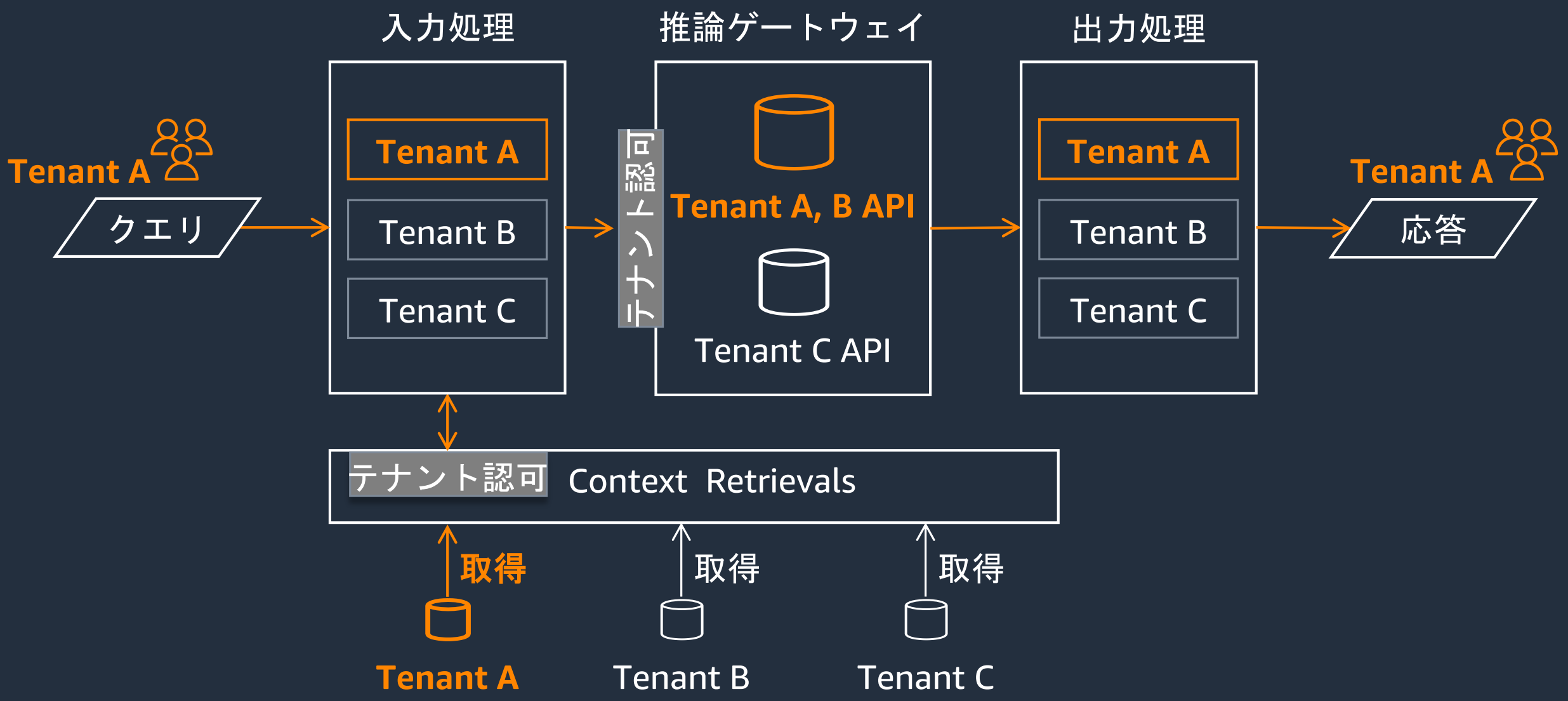
Context Retrievals

テナント C DB

基盤モデル

Hugging Face　amazon　AI21labs　Meta　cohere　ANTHROP\C　stability.ai　MISTRAL AI_

Multi-tenant 生成 AI のデータフロー

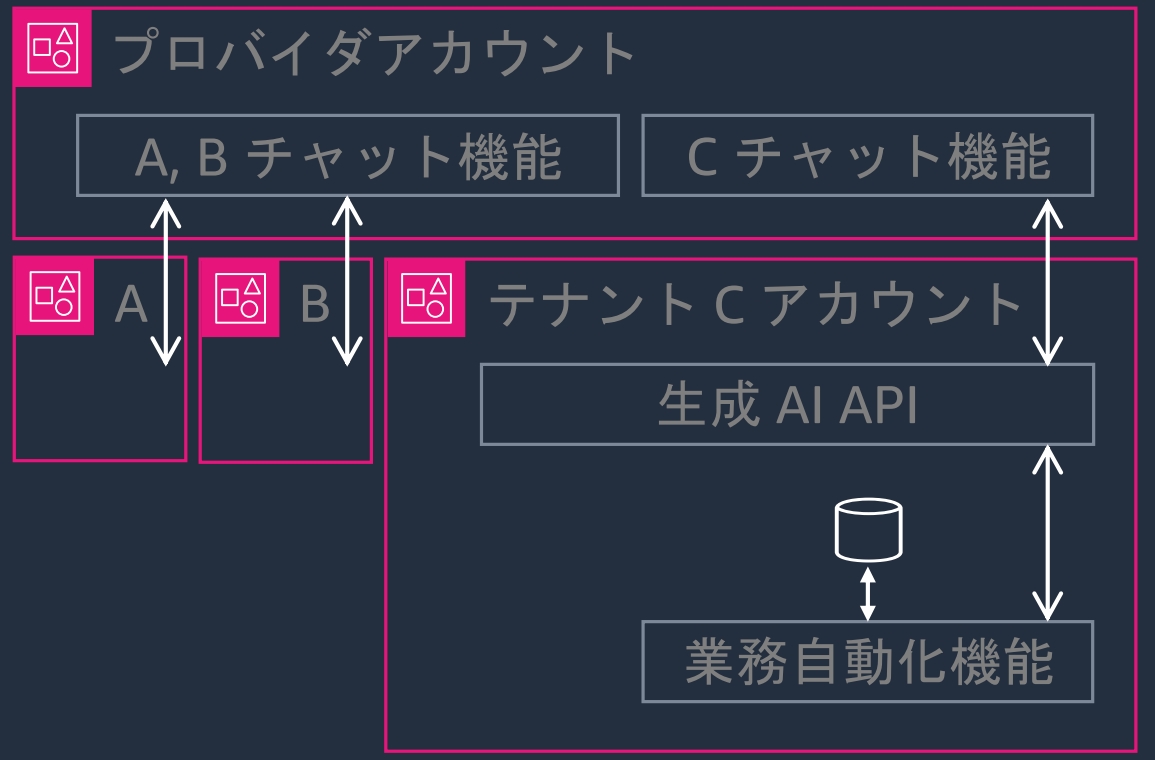


SaaS データプレーンのデプロイメントパターン

パターン 1: 単一アカウントモデル

Pros: 初期構築の容易性
Cons: ノイジーネイバー、情報管理

パターン 2: スプリットプレーンモデル



Pros: クォータ、情報管理
Cons: テナント管理の複雑性

パターン 3: 専用アカウントモデル

Pros: クォータ、テナント費用計算
Cons: アカウント管理の複雑性

SaaS データプレーンのデプロイメントパターン

パターン 1: 単一アカウントモデル

プロバイダアカウント

A, B チャット機能

C チャット機能

A, B, C 生成 AI API (プール)

Pros: 初期構築の容易性
Cons: ノイジーネイバー、情報管理

パターン 3: 専用アカウントモデル

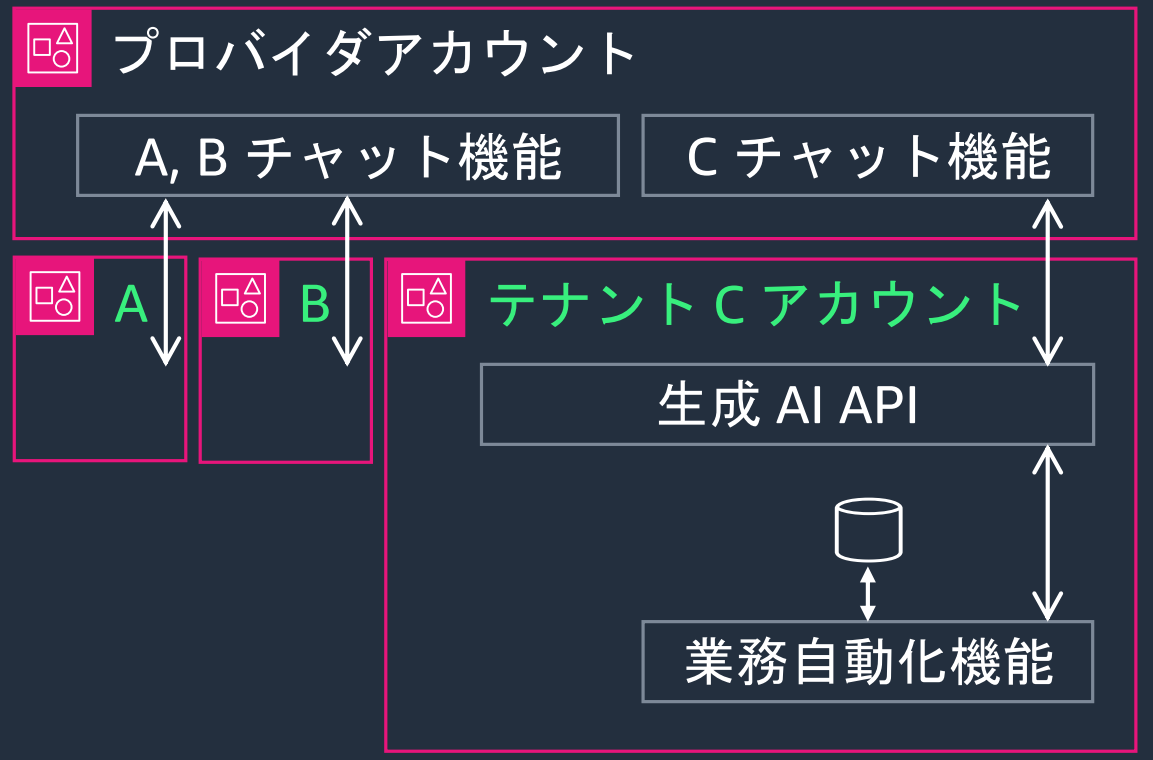
プロバイダアカウント

A, B 用

C 用

Pros: クォータ、テナント費用計算
Cons: アカウント管理の複雑性

パターン 2: スプリットプレーンモデル



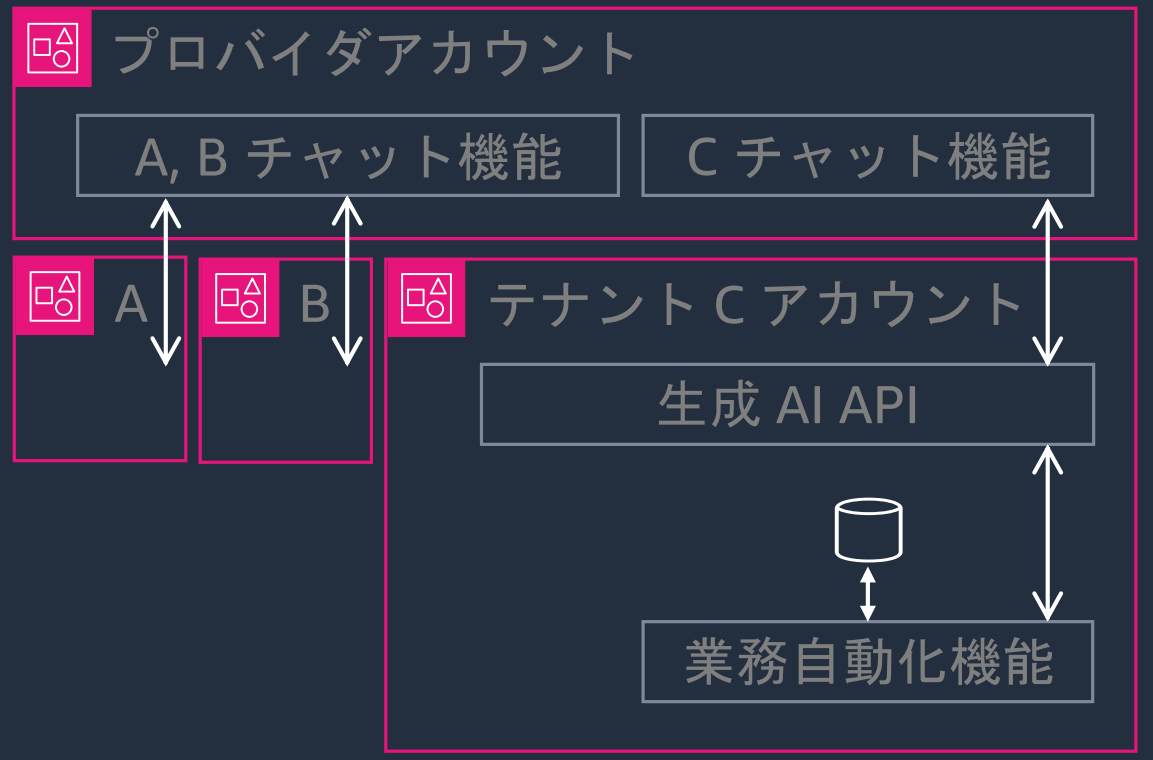
Pros: クォータ、情報管理
Cons: テナント管理の複雑性

SaaS データプレーンのデプロイメントパターン

パターン 1: 単一アカウントモデル

Pros: 初期構築の容易性
Cons: ノイジーネイバー、情報管理

パターン 2: スプリットプレーンモデル

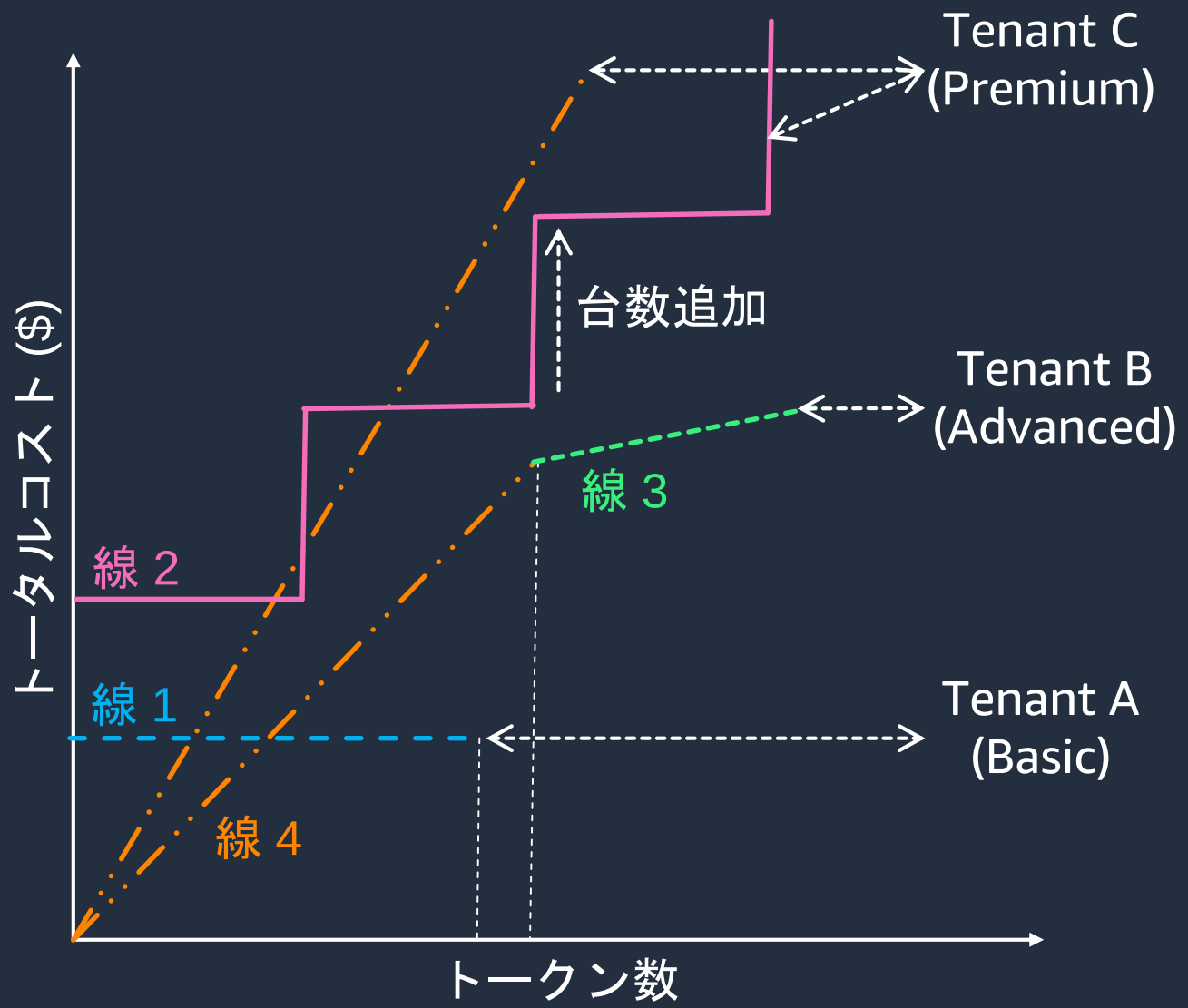


Pros: クォータ、情報管理
Cons: テナント管理の複雑性

パターン 3: 専用アカウントモデル

Pros: クォータ、テナント費用計算
Cons: アカウント管理の複雑性

ティアごとのコスト設計パターン



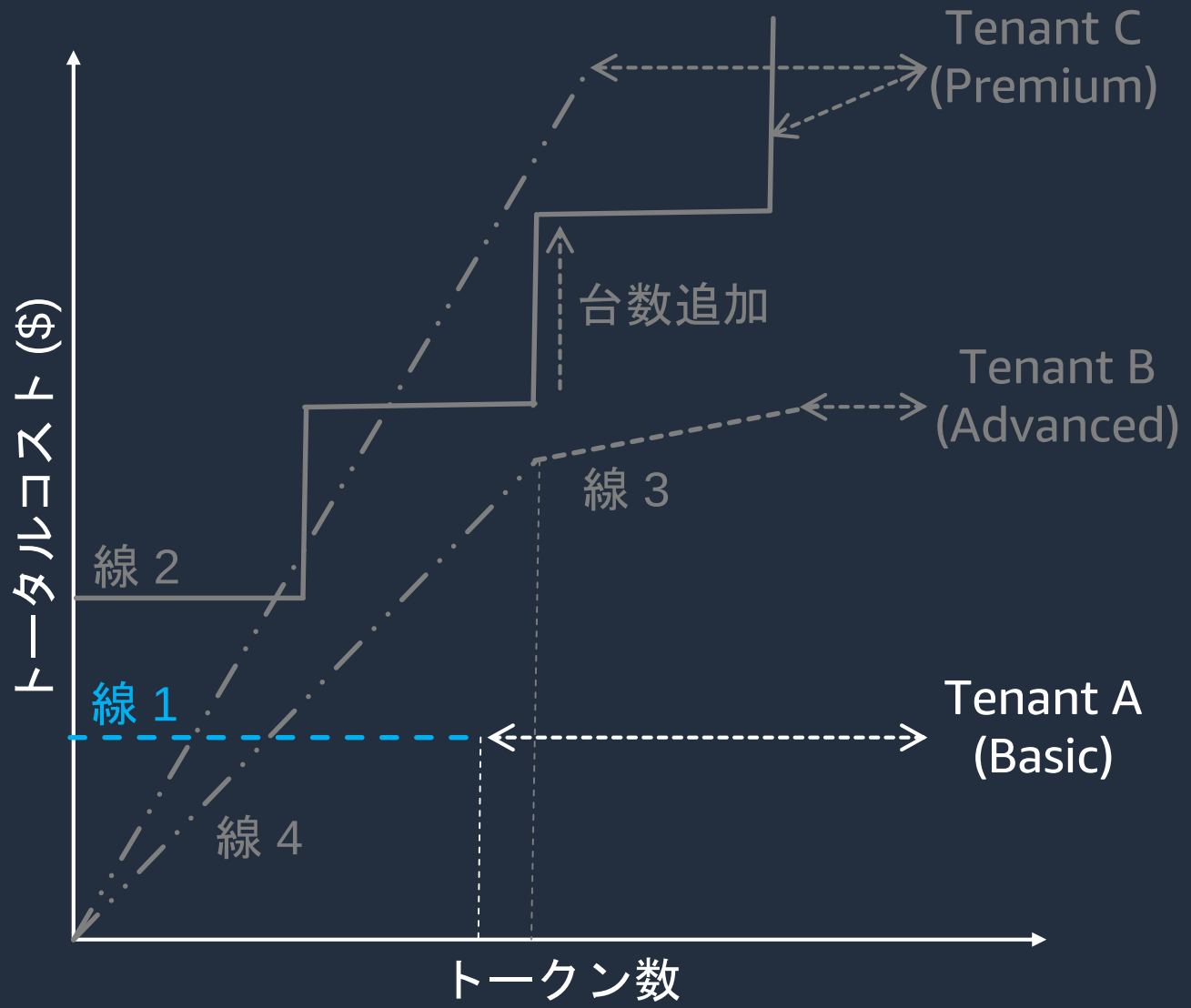
線 1: Basic 向けサービング
線 2: 高性能モデルサービング
線 3: 中性能モデル API
線 4: 高性能モデル API

テナント設計がコスト設計で最重要
ティア、デプロイメントパターン

推論エンドポイントのパターン (一例)

- Tenant A (Basic): 線 1
  - コスト固定でサービング
- Tenant B (Advanced): 線 4 to 線 3
  - トークン数の超過時にコスト抑制で 3 利用
- Tenant C (Premium): 線 2 and/or 線 4
  - API・サービングから選択可能、性能保証

ティアごとのコスト設計パターン

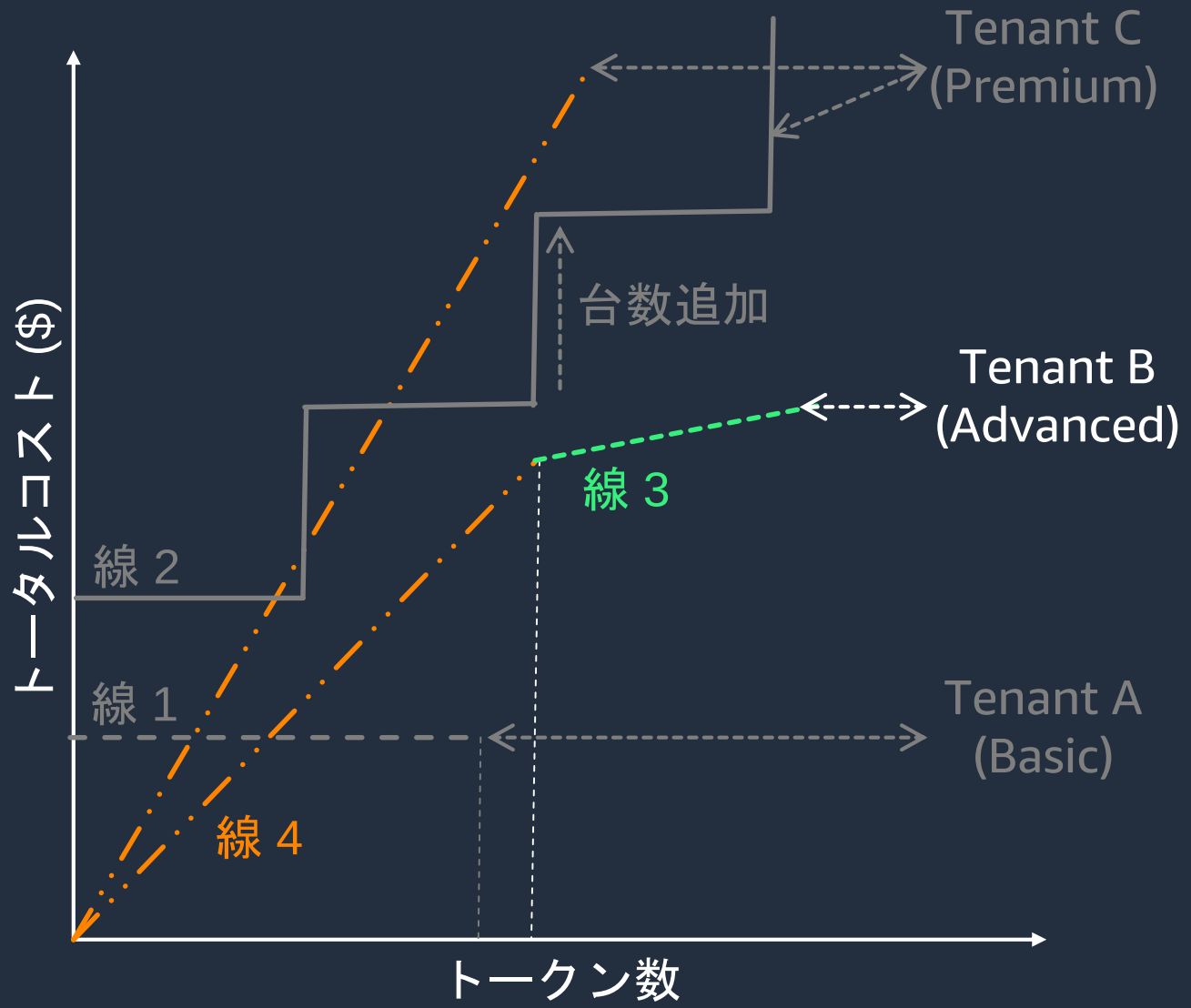


線 1: Basic 向けサービング
線 2: 高性能モデルサービング
線 3: 中性能モデル API
線 4: 高性能モデル API

テナント設計がコスト設計で最重要
ティア、デプロイメントパターン

推論エンドポイントのパターン (一例)
- Tenant A (Basic): 線 1
  - コスト固定でサービング
- Tenant B (Advanced): 線 4 to 線 3
  - トークン数の超過時にコスト抑制で 3 利用
- Tenant C (Premium): 線 2 and/or 線 4
  - API・サービングから選択可能、性能保証

ティアごとのコスト設計パターン



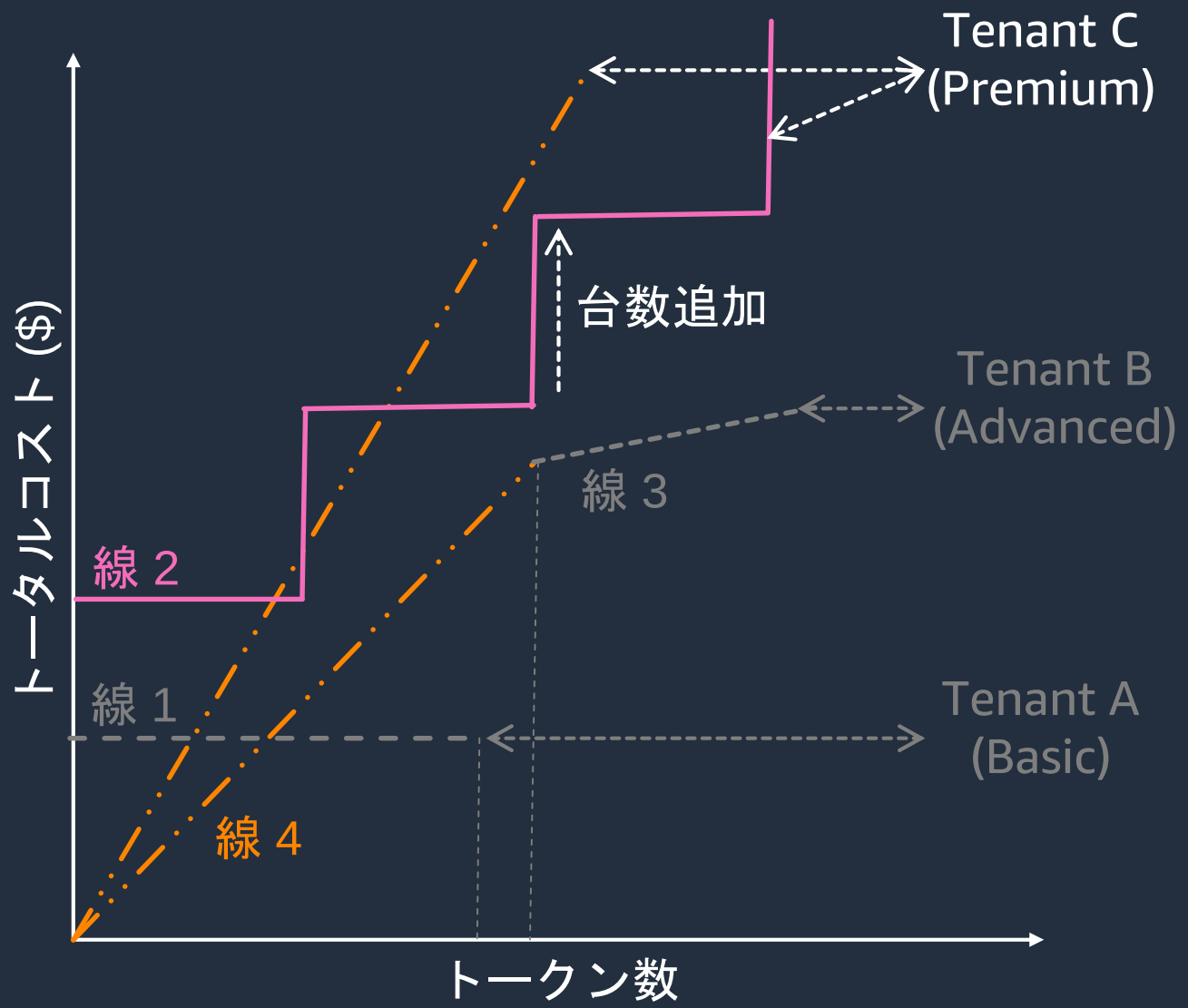
線 1: Basic 向けサービング
線 2: 高性能モデルサービング
線 3: 中性能モデル API
線 4: 高性能モデル API

テナント設計がコスト設計で最重要
ティア、デプロイメントパターン

推論エンドポイントのパターン (一例)

- Tenant A (Basic): 線 1
  - コスト固定でサービング
- Tenant B (Advanced): 線 4 to 線 3
  - トークン数の超過時にコスト抑制で 3 利用
- Tenant C (Premium): 線 2 and/or 線 4
  - API・サービングから選択可能、性能保証

ティアごとのコスト設計パターン



線 1: Basic 向けサービング
線 2: 高性能モデルサービング
線 3: 中性能モデル API
線 4: 高性能モデル API

テナント設計がコスト設計で最重要
ティア、デプロイメントパターン

推論エンドポイントのパターン (一例)

- Tenant A (Basic): 線 1
  - コスト固定でサービング
- Tenant B (Advanced): 線 4 to 線 3
  - トークン数の超過時にコスト抑制で 3 利用
- Tenant C (Premium): 線 2 and/or 線 4
  - API・サービングから選択可能、性能保証

アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. **SaaS サービスへの生成 AI 適用事例紹介**
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
4. まとめ

事例: How Forethought saves over 66% in costs for generative AI models using Amazon SageMaker

事例紹介の意図: コスト設計パターンで紹介した、固定費用での機能提供、サイロでの性能保証、の要求に対応する**選択肢の一つとしてサービング利用の可能性を模索**したい。本事例は Multi-tenant サービング事例であり、以降のスライドでコスト・性能試算のために利用する。

状況: カスタマーサービスを提供するフォアソートでは、顧客ごとに Fine-tuning された大規模言語モデルを活用して、年間 3,000 万件のパーソナライズされた顧客問い合わせサポートを提供

ビジネス課題: 肥大化する運用含むコストの削減
技術課題:
1. 応答の迅速性とコストの両立
2. テナント単位のリクエストを捌くためのモデル配置数の調整コスト
3. モデル配置状況によって発生する GPU メモリの Out of Memory (OOM)

成果: Amazon SageMaker Multi Model Endpoint を利用して 66% コスト削減

https://aws.amazon.com/blogs/machine-learning/how-forethought-saves-over-66-in-costs-for-generative-ai-models-using-amazon-sagemaker/

生成 AI 利用のコスト概算の一例

前提	数値
リージョン	オレゴン
テナント数	300
年間対応件数	3000 万
トークン数 / 対応	In: 307, Out: 533
スループット要求 (RPS)	0.95
平均処理時間 (sec)	4.4
モデル並列数 (0.95 × 4.4)	4.18

Amazon Bedrock (オンデマンド) *2

利用モデル	$/1K トークン	年額 ($)
Claude 3.5 Sonnet	In: 0.003 Out: 0.015	267K
Claude 3 Haiku	In: 0.00025 Out: 0.00125	22K
Llama 3.1 (70B)	In: 0.00099 Out: 0.00099	25K
*Llama 3.1 (70B) サービング	-	?

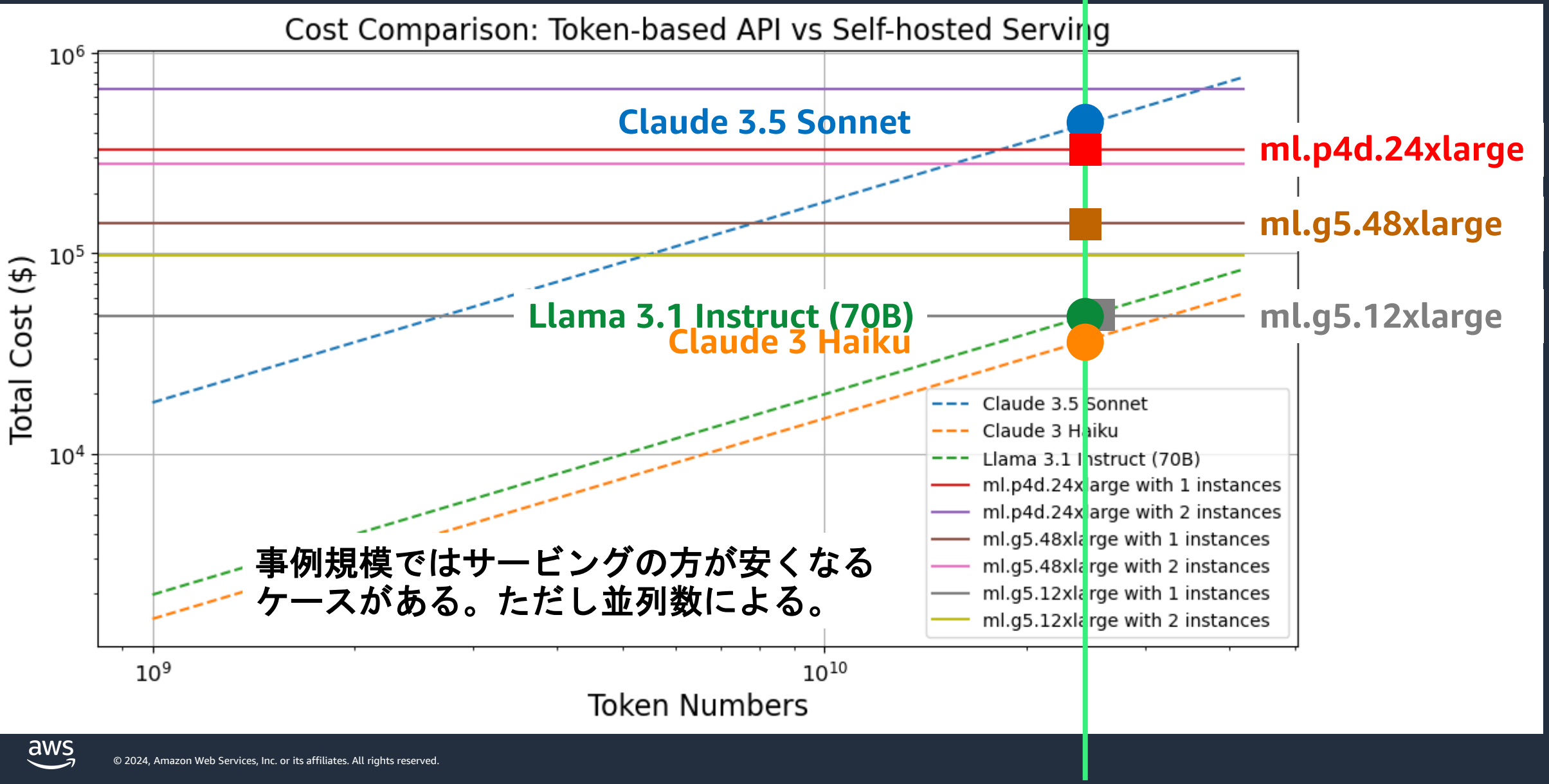
*meta-llama/Llama-3.1-70B-Instruct

概算時の条件 (数値は目安です)
- テナント数は *1 より概算
- スループット要求 = 3000 万 ÷ (365 × 24 × 60 × 60)
- 年額計算例 = 3000 万 × { (307 × $0.003 + 533 × $0.015) / 1000 } = $267K
- トークン数 / 対応、平均処理時間、は顧客対応を想定して生成したテキストでの計測値

*1: https://www.businesswire.com/news/home/20211214006409/en/Forethought-Raises-65-Million-Series-C-to-Accelerate-Digital-Transformation-in-Customer-Service
*2: https://aws.amazon.com/jp/bedrock/pricing/

生成 AI 利用のコスト概算の一例

事例規模のトークン数
In / out token 比 = 1 : 1



SaaS での生成 AI 機能提供ブロッカーの整理

Multi-tenant ではスループット要求とコストが読みづらい

- テナント分離レベル
  - 分離なし
    - ノイジーネイバーの影響がある
  - アカウント単位で分離
    - 利用モデルによってはクォータ上限にかかる可能性が依然存在する
  - リソース単位で分離
    - ノイジーネイバー対策とコスト効率の良い提供方法の検討

多様な日本語オープンモデル活用、スループット要求への対応、Fine-tuned モデル対応、ノイジーネイバー、ティアごとのコスト設計、の**要求に対応するパターンの一つとして**、サービング実装手法について深掘り

アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
    1. **Large Model Inference の難しさ**
    2. 効率的なモデル微調整 LoRA の紹介
    3. Multi-tenant LoRA serving on Amazon SageMaker
    4. サービングのコスト概算
4. まとめ

コスト効率化のためにサービング並列台数を減らしたいが可能か？

Large Model Inference の難しさ – GPU メモリ要求

モデルが GPU メモリ (VRAM) に乗り切らない (OOM)
→ 量子化・低精度化

FP16, BF16

70B パラメータ
140 GB

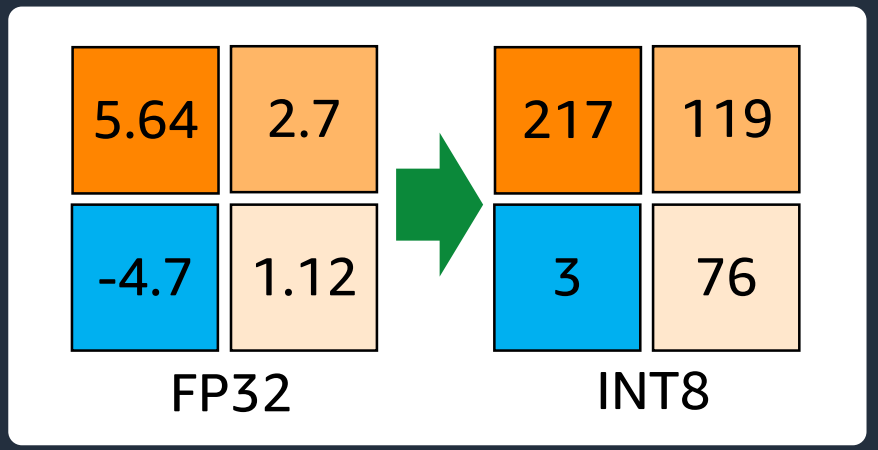
ml.p4d.24xlarge
320GB VRAM

FP32

OOM

70B パラメータ
280 GB

ml.g5.48xlarge
192GB VRAM



INT8, FP8

70B パラメータ
70 GB

ml.g5.12xlarge
96GB VRAM

INT4, NF4

70B パラメータ
35 GB

ml.g5.12xlarge
96GB VRAM

Large Model Inference の難しさ – GPU メモリ要求

KV cache

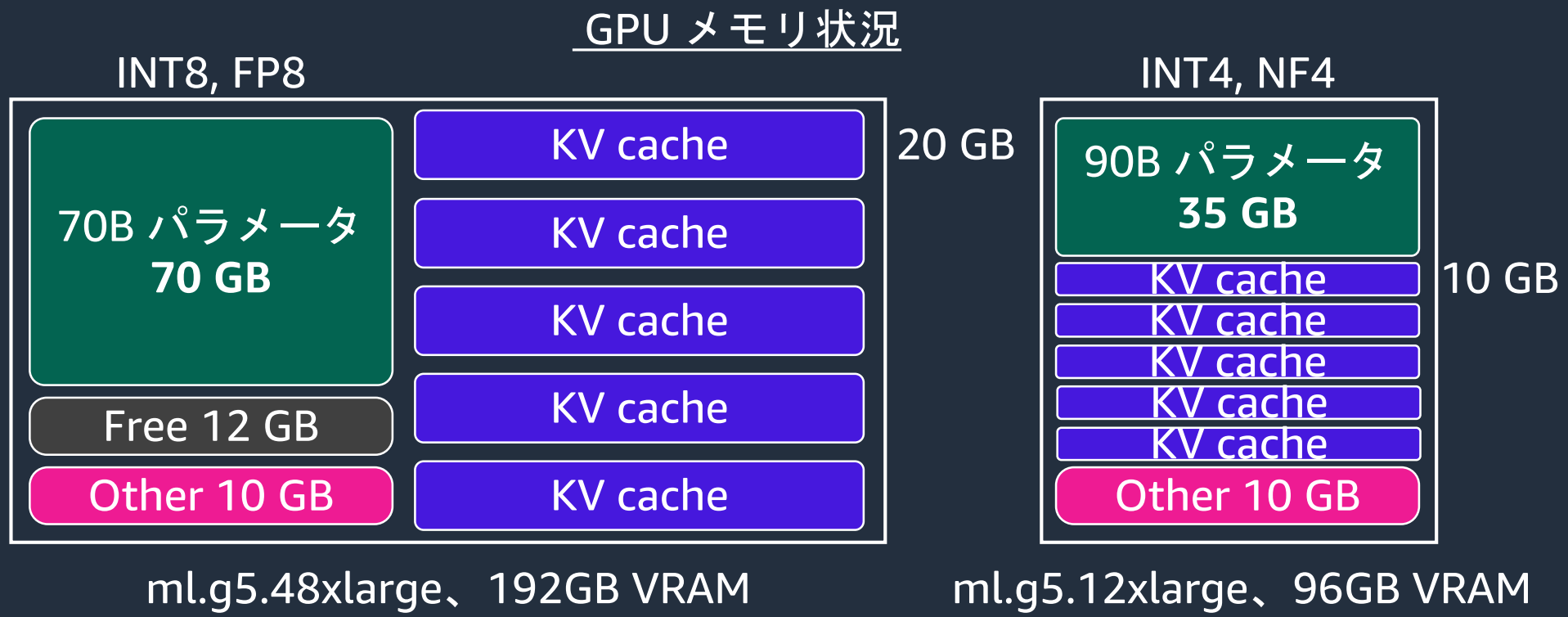
モデルサイズ	最大トークン数	FP32 (GB)	FP16 (GB)	INT8 (GB)	INT4 (GB)
70B	4K	2.50	1.25	0.63	0.31
	16K	10.00	5.00	2.50	1.25
	128K	80.00	40.00	**20.00**	**10.00**

GPU メモリ要求から処理の並列可能数を確認する

- 仮定: Llama 3.1 Instruct 70B のモデル、全リクエスト最大トークン長 128K で処理

✓ INT8 → ml.g5.48xlarge × 1 で約 5 並列可能
✓ INT4 → ml.g5.12xlarge × 1 で約 5 並列可能

→ モデル並列数要求 4.19 を満たす



スループット効率化手法: Dynamic Batching

アジェンダ

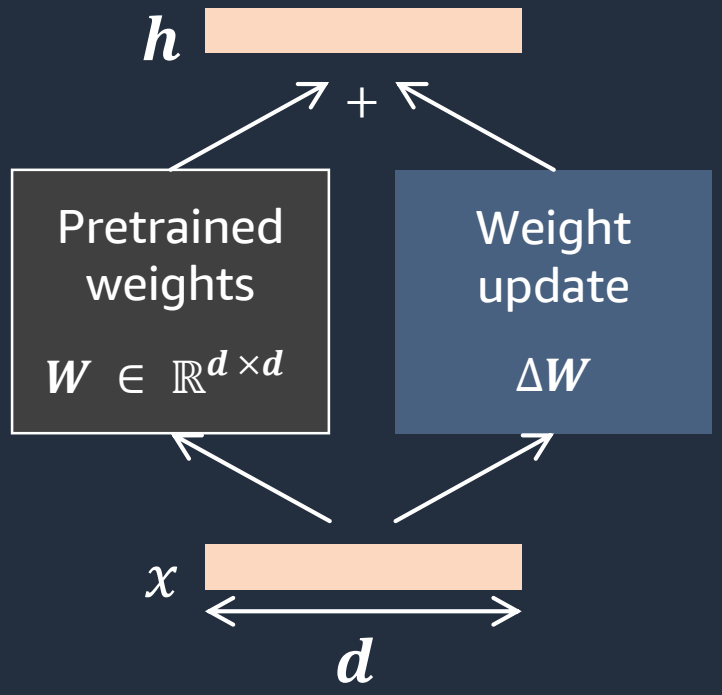
セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
    1. Large Model Inference の難しさ
    2. **効率的なモデル微調整 LoRA の紹介**
    3. Multi-tenant LoRA serving on Amazon SageMaker
    4. サービングのコスト概算
4. まとめ

事例ではテナント単位でモデルをファインチューニングしている。コスト効率よく提供可能？

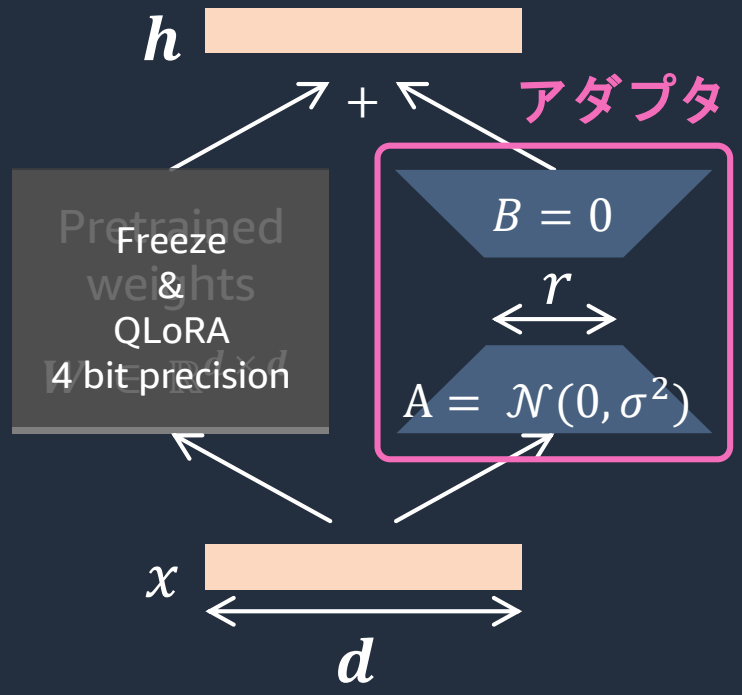
効率的なモデル微調整 LoRA (LOW-RANK ADAPTATION)

通常の Fine-tuning での重み更新



$$\mathbf{h} = W_0 x + \Delta W x$$

LoRA での重み更新



$$h = W_0 x + BAx$$

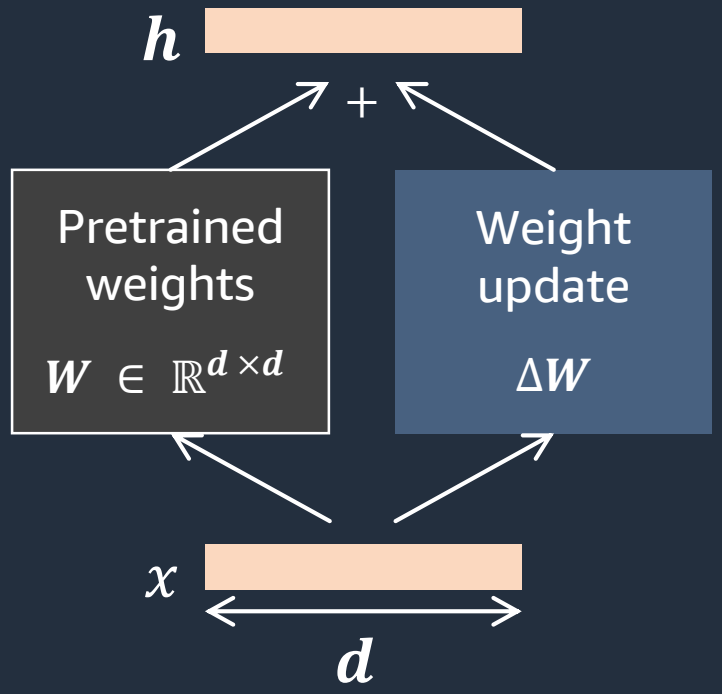
低ランク分解

- PEFT (Parameter-Efficient Fine-Tuning) の一種
- *GPU メモリ要件を 3 分の 1、パラメータ数を最大 10,000 倍削減

*https://arxiv.org/pdf/2106.09685

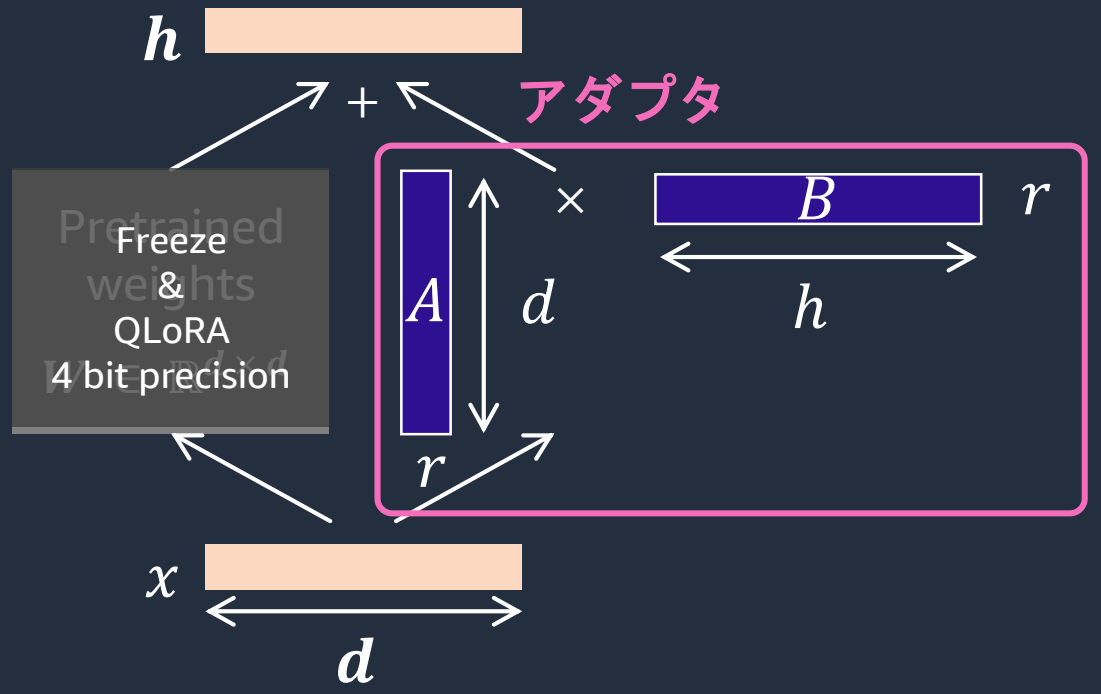
効率的なモデル微調整 LoRA (LOW-RANK ADAPTATION)

通常の Fine-tuning での重み更新



$$\mathbf{h} = W_0 x + \Delta W x$$

LoRA での重み更新



$$h = W_0 x + BAx$$

低ランク分解

- r はハイパーパラメータ

https://arxiv.org/pdf/2106.09685

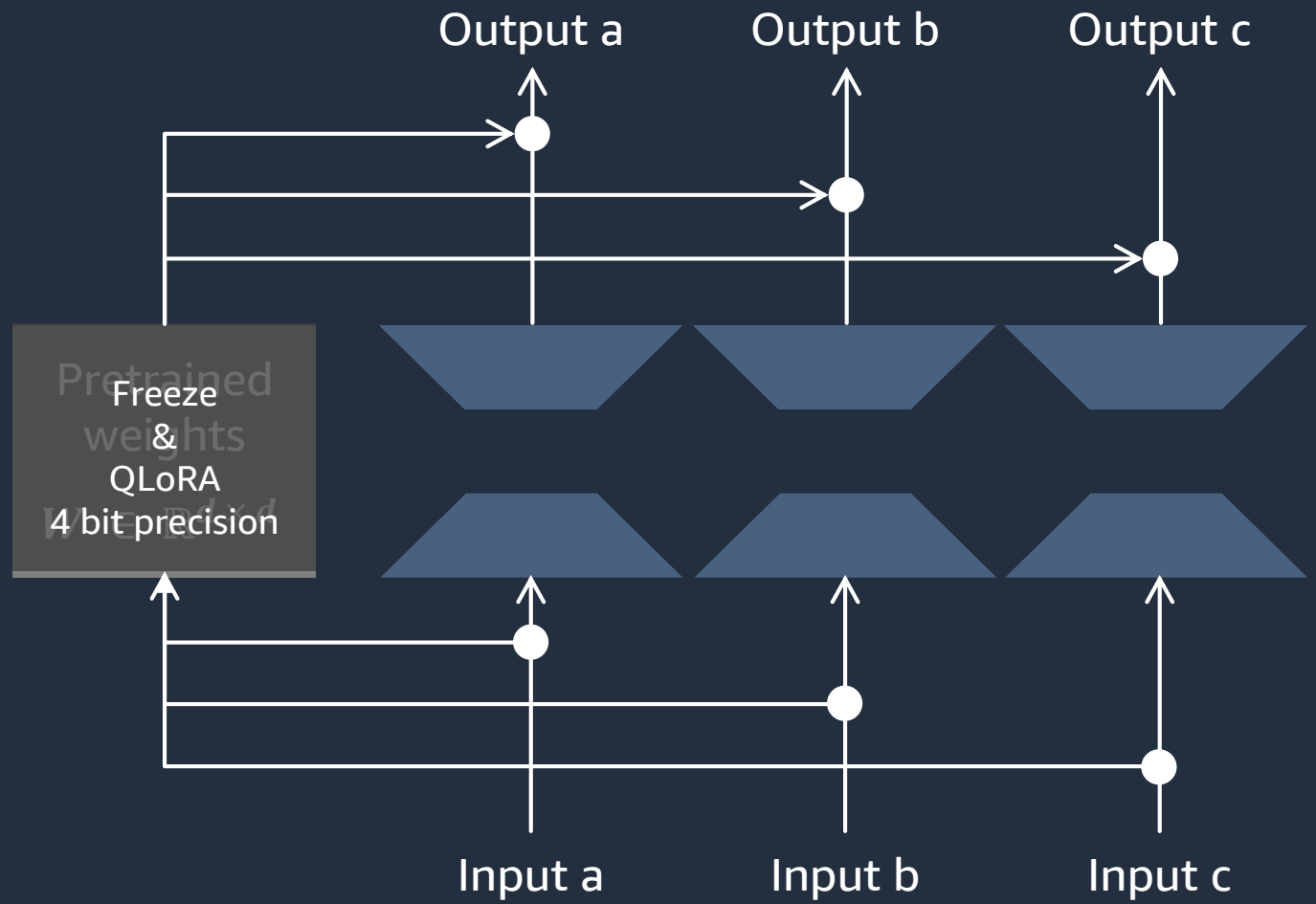
LoRA Serving の効率化アイデア

Merged Option:

Pretrained Weight × テナント分の GPU メモリ要求

Unmerged Option:

単一 Pretrained Weight + アダプタでメモリ効率化



https://arxiv.org/pdf/2311.03285

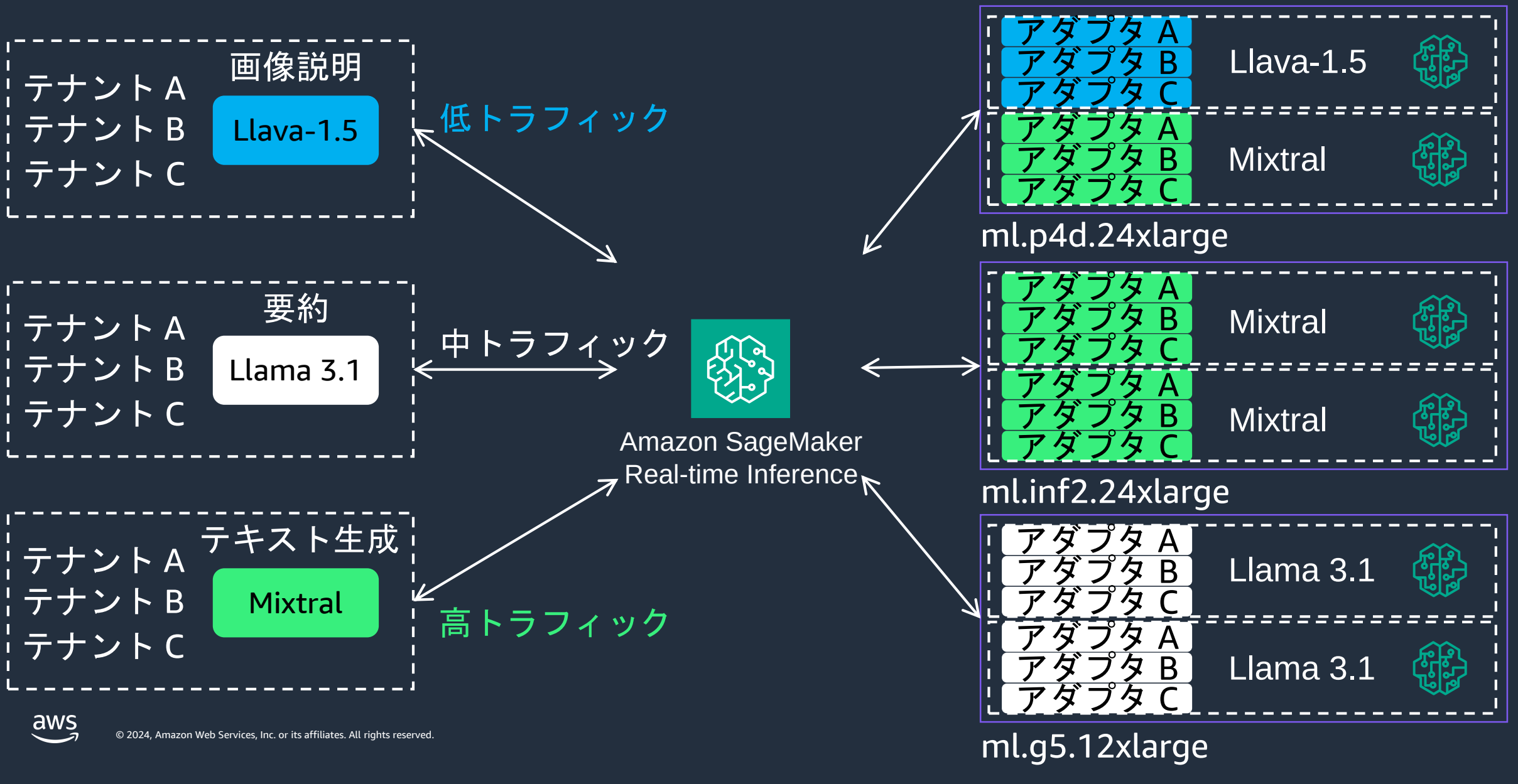
アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

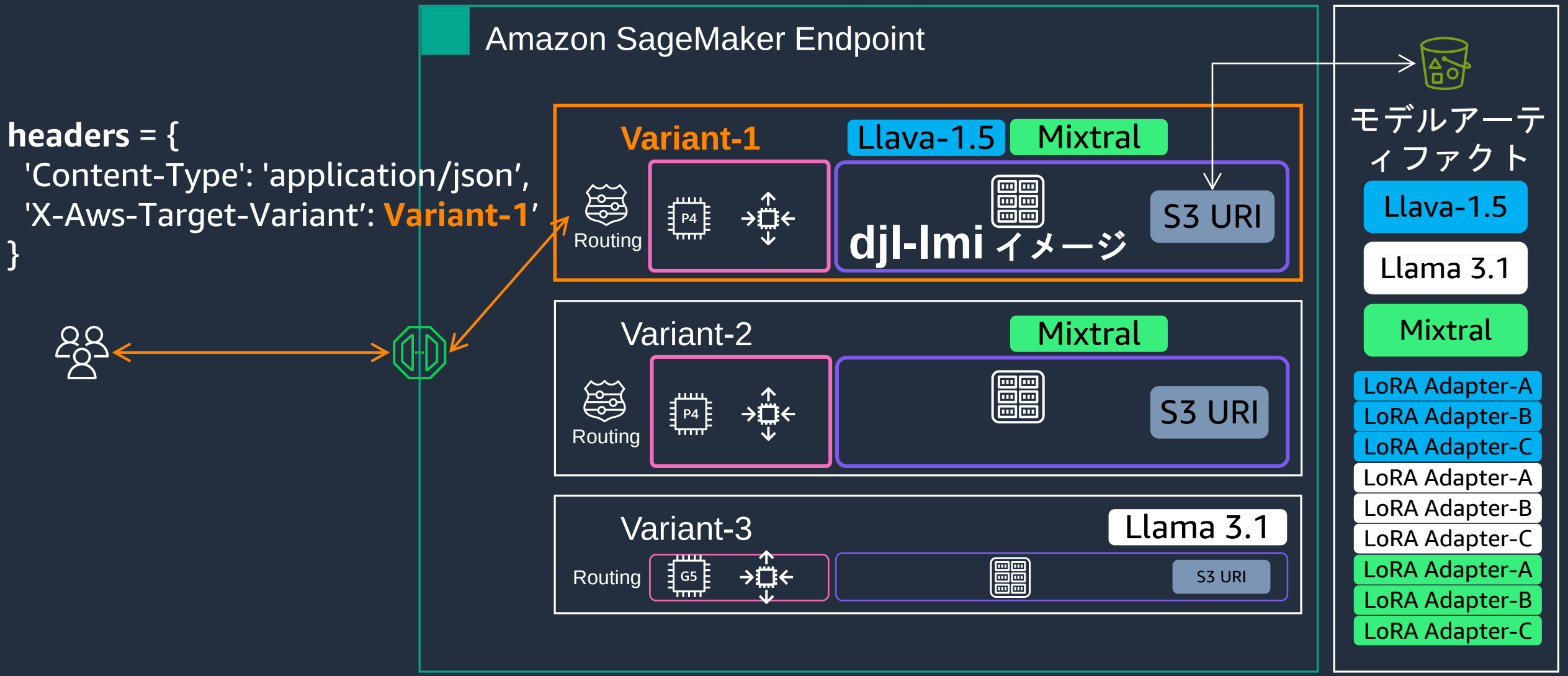
1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
    1. Large Model Inference の難しさ
    2. 効率的なモデル微調整 LoRA の紹介
    3. Multi-tenant LoRA serving on Amazon SageMaker
        1. **Amazon SageMaker Endpoint**
        2. Multi-tenant LoRA Serving のコンテナ技術スタック
        3. Multi-tenant LoRA Serving on Amazon SageMaker の構築方法
    4. サービングのコスト概算
4. まとめ

事例で利用された推論サービング機能を提供する AWS サービス

Amazon SageMaker Real-time Inference



マルチテナンシーのための推論エンドポイント構造



アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
    1. Large Model Inference の難しさ
    2. 効率的なモデル微調整 LoRA の紹介
    3. Multi-tenant LoRA serving on Amazon SageMaker
        1. Amazon SageMaker Endpoint
        2. **Multi-tenant LoRA Serving のコンテナ技術スタック**
        3. Multi-tenant LoRA Serving on Amazon SageMaker の構築方法
    4. サービングのコスト概算
4. まとめ

Amazon SageMaker Endpoint で利用するコンテナイメージの説明

Multi-tenant LoRA Serving のコンテナ技術スタック

djl-lmi, LMI イメージの一つ

DJL Serving

LMI-Dist backend, Adapters

Rolling batching
全リクエスト完了を待たずに次のリクエストを処理 (Dynamic batching の改善)

S-LoRA Library (スケーラブル LoRA サービング)

3. Heterogeneous Batching

2. Unified Paging

vLLM Library

Paged Attention

Dynamic batching

Punica CUDA adapter kernels

S-LoRA Library

1. Dynamic adapter loading

4. Optimized Tensor Parallel

https://aws.amazon.com/jp/blogs/machine-learning/improve-throughput-performance-of-llama-2-models-using-amazon-sagemaker/

Multi-tenant LoRA Serving のコンテナ技術スタック

djl-lmi, LMI イメージの一つ

DJL Serving

LMI-Dist backend, Adapters

Rolling batching
全リクエスト完了を待たずに次のリクエストを処理 (Dynamic batching の改善)

S-LoRA Library (スケーラブル LoRA サービング)

3. Heterogeneous Batching

2. Unified Paging

vLLM Library

Paged Attention

Dynamic batching

Punica CUDA adapter kernels

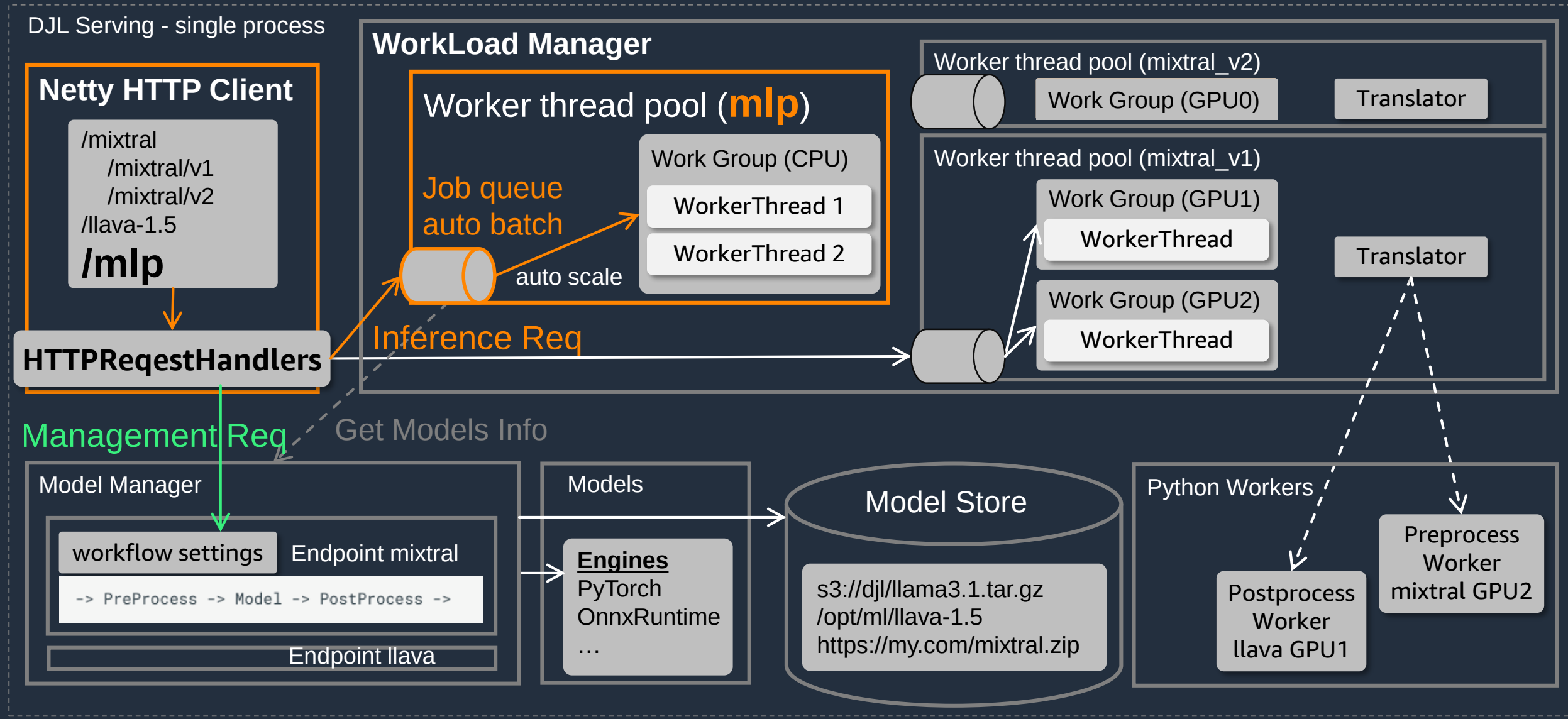
S-LoRA Library

1. Dynamic adapter loading

4. Optimized Tensor Parallel

https://aws.amazon.com/jp/blogs/machine-learning/improve-throughput-performance-of-llama-2-models-using-amazon-sagemaker/

DJL Serving アーキテクチャ



https://docs.djl.ai/master/docs/serving/serving/docs/architecture.html

Multi-tenant LoRA Serving のコンテナ技術スタック

djl-lmi, LMI イメージの一つ

DJL Serving

LMI-Dist backend, Adapters

Rolling batching
全リクエスト完了を待たずに次のリクエストを処理 (Dynamic batching の改善)

S-LoRA Library (スケーラブル LoRA サービング)

3. Heterogeneous Batching

2. Unified Paging

vLLM Library

Paged Attention

Dynamic batching

Punica CUDA adapter kernels

S-LoRA Library

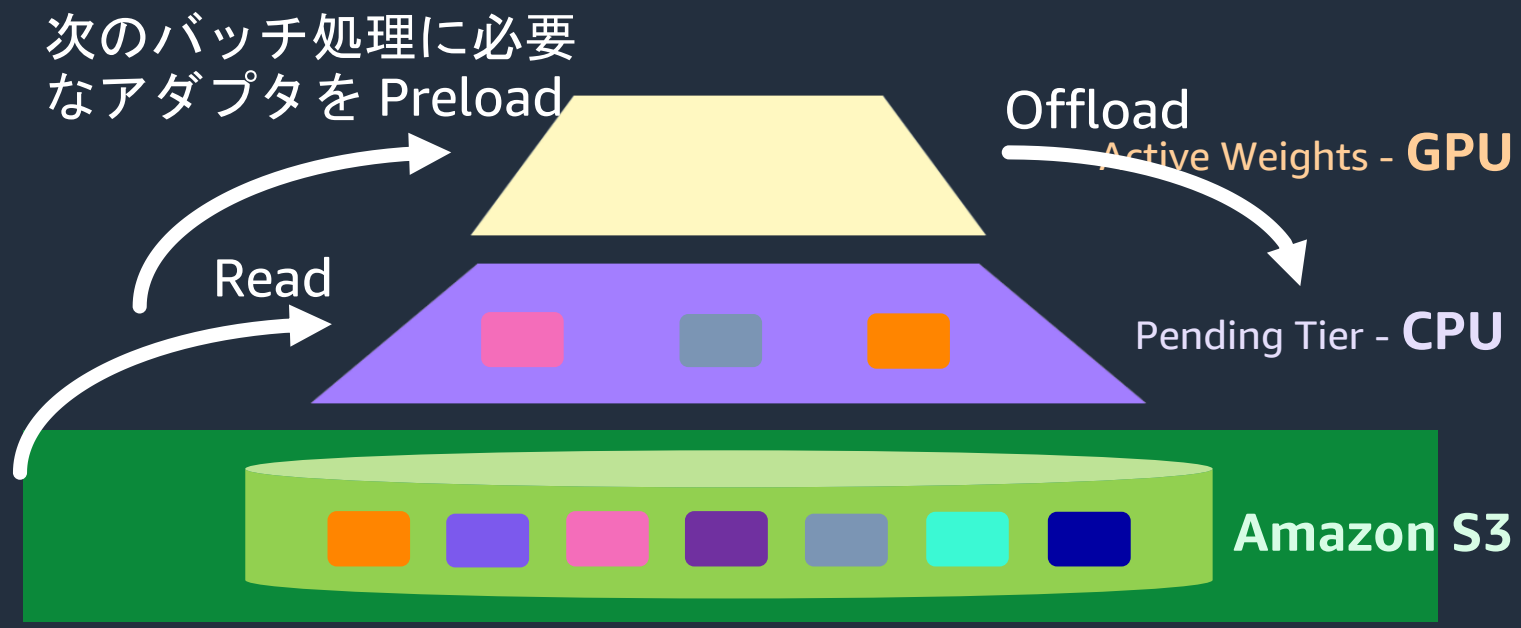
1. Dynamic adapter loading

4. Optimized Tensor Parallel

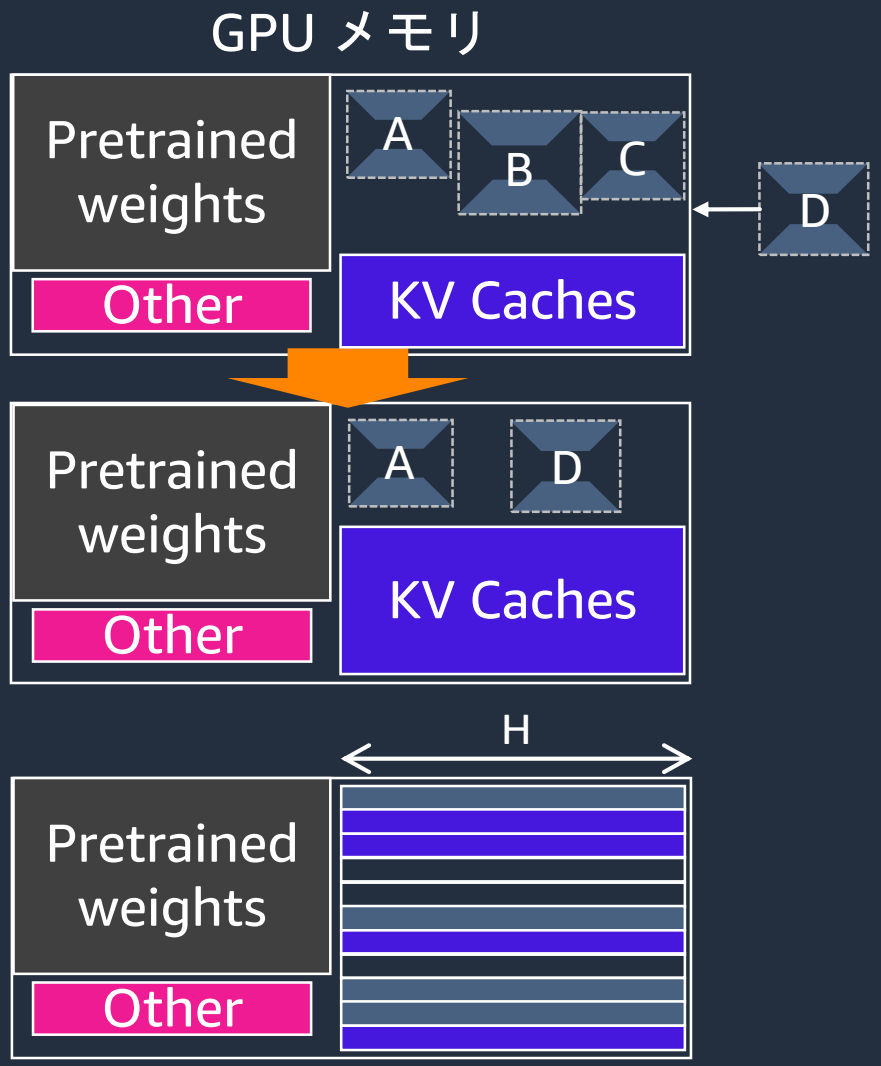
https://aws.amazon.com/jp/blogs/machine-learning/improve-throughput-performance-of-llama-2-models-using-amazon-sagemaker/

S-LoRA 技術: Dynamic Adapter Loading, Unified Paging

Multi-tenant LoRA serving 実現のために解決すべき技術課題
1. GPU メモリに Preload できるアダプタ数が限られる
2. 複数サイズのアダプタの動的ロード・オフロードによるメモリ断片化、IO オーバーヘッド



1. Dynamic Adapter loading

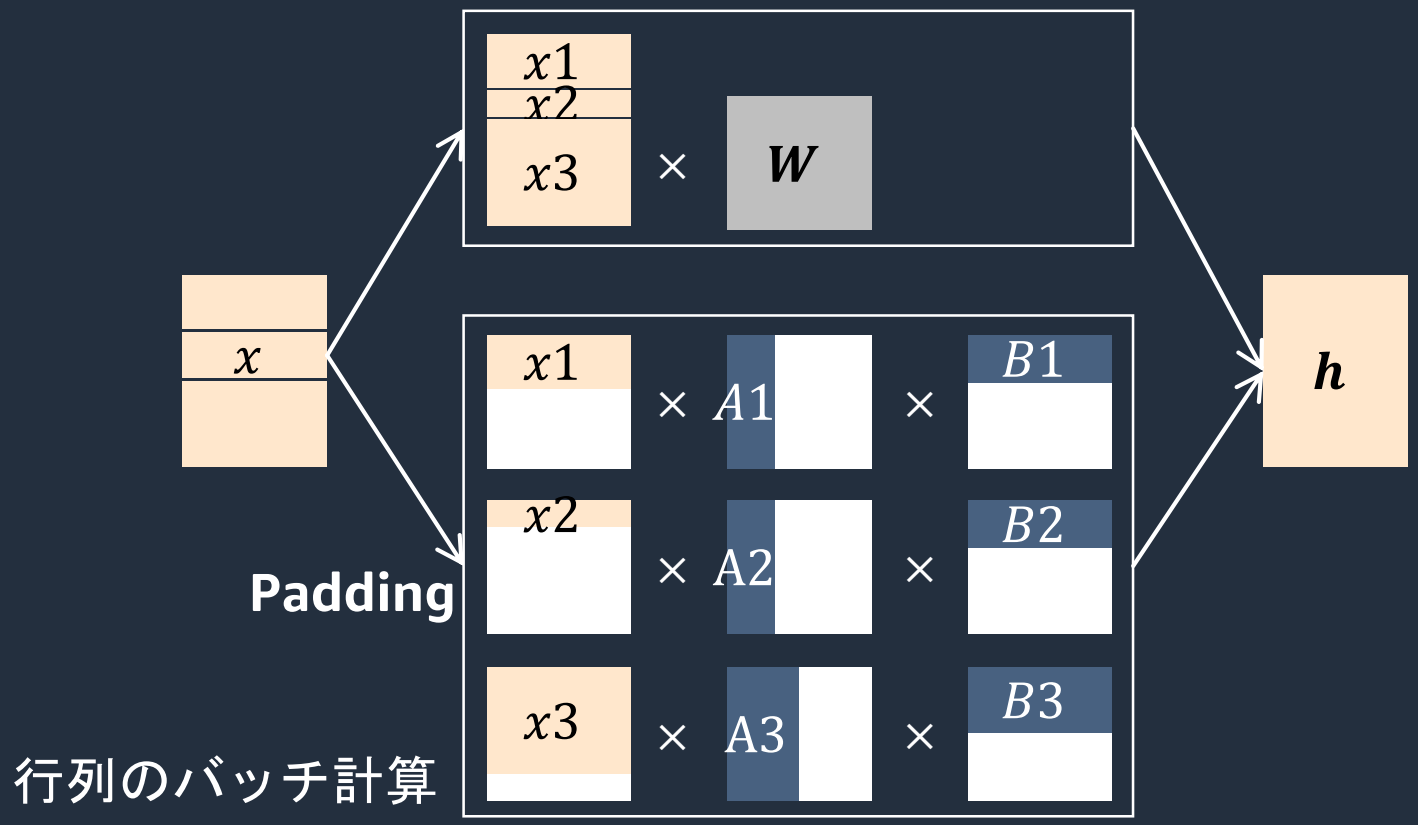


2. Unified Paging

https://arxiv.org/pdf/2311.03285

S-LoRA 技術: Heterogenios Batching

Multi-tenant LoRA serving 実現のために解決すべき技術課題
異なるランクの多数のアダプタのバッチ処理にパディングが必要で計算効率が良くない



Punica CUDA adapter kernels

要求
行列計算で余分なパディングで計算効率を下げたくない。

手法
Unified Paging のメモリレイアウトで様々なランクとシーケンス長を持つ計算を効率的に処理するカーネルを実装

Heterogenious Batching をマルチ GPU でテンソル並列計算できるような工夫
→ 4. Optimized Tensor Parallel

https://arxiv.org/pdf/2311.03285

アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
    1. Large Model Inference の難しさ
    2. 効率的なモデル微調整 LoRA の紹介
    3. Multi-tenant LoRA serving on Amazon SageMaker
        1. Amazon SageMaker Endpoint
        2. Multi-tenant LoRA Serving のコンテナ技術スタック
        3. **Multi-tenant LoRA Serving on Amazon SageMaker の構築方法**
    4. サービングのコスト概算
4. まとめ

サービング環境構築の作業量はどのくらい？

Multi-tenant LoRA Serving on Amazon SageMaker の構築

Step 1: モデル、アダプタ、設定の配置

```
|- model_dir
    |- adapters/
        |--- <adapter_1>/
        |--- <adapter_2>/
        |--- ...
        |--- <adapter_n>/
    |- serving.properties
    |- model.py (optional)
```

```
%%writefile lora-multi-adapter/serving.properties
option.model_id=huggyllama/llama-7b
option.engine=MPI
option.rolling_batch=lmi-dist
option.tensor_parallel_degree=1
option.enable_lora=true
option.gpu_memory_utilization=0.8
```

LoRA serving 設定

Step 2: 数十行のコードでサービング

```
inference_image_uri = image_uris.retrieve(
        framework="djl-lmi",
        region=region,
        version="0.29.0"
    )
model_name_acc = name_from_base(f"lora-multi-adapter")

create_model_response = sm_client.create_model(
    ModelName=model_name_acc,
    ExecutionRoleArn=role,
    PrimaryContainer={"Image": inference_image_uri,
                      "ModelDataUrl": s3_code_artifact_accelerate,
                     })
model_arn = create_model_response["ModelArn"]
```

イメージ取得 & Model 作成

```
endpoint_config_response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[
        {
            "VariantName": "variant1",
            "ModelName": model_name_acc,
            "InstanceType": "ml.g5.12xlarge",
            "InitialInstanceCount": 1,
            "ModelDataDownloadTimeoutInSeconds": 1800,
            "ContainerStartupHealthCheckTimeoutInSeconds": 1800,
        },
```

エンドポイント設定

アジェンダ

セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. **Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り**
    1. Large Model Inference の難しさ
    2. 効率的なモデル微調整 LoRA の紹介
    3. Multi-tenant LoRA serving on Amazon SageMaker
    4. **サービングのコスト概算**
4. まとめ

生成 AI 利用のコスト概算の一例

前提	数値
リージョン	オレゴン
テナント数	300
年間対応件数	3000 万
トークン数 / 対応	In: 307, Out: 533
スループット要求 (RPS)	0.95
平均処理時間 (sec)	4.4
モデル並列数 (0.95 × 4.4)	4.18

サービング試算の共通条件 (10/22 時点、数値は目安)

- Savings Plans for Amazon SageMaker
  - 1 year, No Upfront
- Llama 3.1 Instruct (70B), Max 128K tokens

Amazon Bedrock (オンデマンド) *2

利用モデル	$/1K トークン	年額 ($)
Claude 3.5 Sonnet	In: 0.003 Out: 0.015	**267K**
Claude 3 Haiku	In: 0.00025 Out: 0.00125	**22K**
Llama 3.1 (70B)	In: 0.00099 Out: 0.00099	**25K**

Llama 3.1 (70B) サービング	INT4 ml.g5.12xlarge	**49K**
Llama 3.1 (70B) サービング	INT8 ml.g5.48xlarge	**140K**
Llama 3.1 (70B) Fine-tuned	INT8, **S-LoRA** ml.g5.48xlarge	**140K**

*1: https://www.businesswire.com/news/home/20211214006409/en/Forethought-Raises-65-Million-Series-C-to-Accelerate-Digital-Transformation-in-Customer-Service

*2: https://aws.amazon.com/jp/bedrock/pricing/

アジェンダ

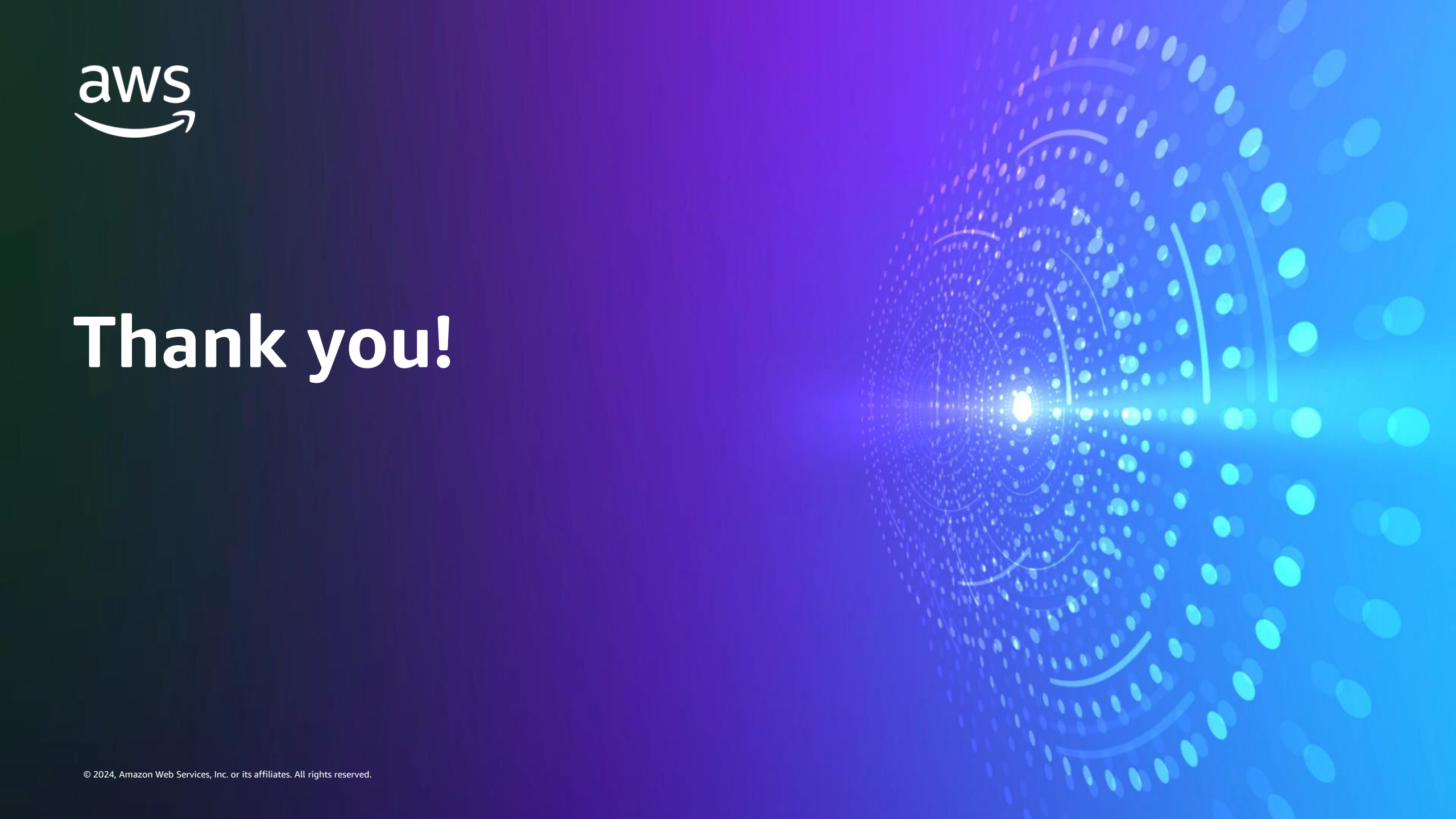
セッションテーマ = SaaS への生成 AI 適用時の考慮点やパターンを整理すること

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. **Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り**
4. **まとめ**

まとめ

1. SaaS と生成 AI の知識整理
2. SaaS サービスへの生成 AI 適用事例紹介
3. Multi-tenant 生成 AI 機能提供時のコスト・性能について深掘り
   多様な日本語オープンモデル活用、スループット要求への対応、Fine-tuned モデル対応、ノイジーネイバー、ティアごとのコスト設計、の**要求に対応するパターンの一つとして**、サービング実装手法について深掘り
   → トークン数課金と比較しても現実的なコストで大規模モデル、数百の Fine-tuning されたモデル、を簡単にサービングするパターンを紹介した

参考
Multi-tenant LoRA Serving on Amazon SageMaker サンプルノートブックの実施
https://github.com/aws-samples/sagemaker-genai-hosting-examples/blob/main/Llama2/Llama2-7b/LMI/llama2-7b-multi-lora-adapters-sagemaker.ipynb



Thank you!