AI-T2-01

コンテンツ審査を題材とした生成AI機能実装のベストプラクティス

石見 和也
アマゾン ウェブ サービス ジャパン合同会社
デジタルサービス技術本部
シニア ソリューションアーキテクト

本セッションについて

内容

本セッションでは、コンテンツ審査を題材として、生成 AI を実プロダクトに実装する際に直面する **考慮点** を整理し、具体的な **打ち手** をご紹介します
(L200 ～ L300)

想定される対象者

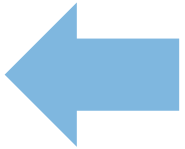
- 生成AI (特にAmazon Bedrock) の利用経験はあるが、いざプロダクトに導入するとなると求められる品質とのギャップを感じている
- 実際に生成AIをプロダクト導入する際に直面する問題や対策を理解しておきたい

※ RAGやOSS LLMの話は含みません。Amazon BedrockでClaudeを扱うケースに着目します

ECサイトへのコメント投稿を審査する場合を考える

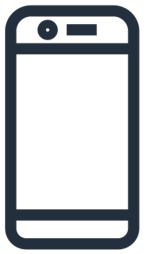
「このパンすごい美味しかった」

「値段の割にボリュームがない」



LLM

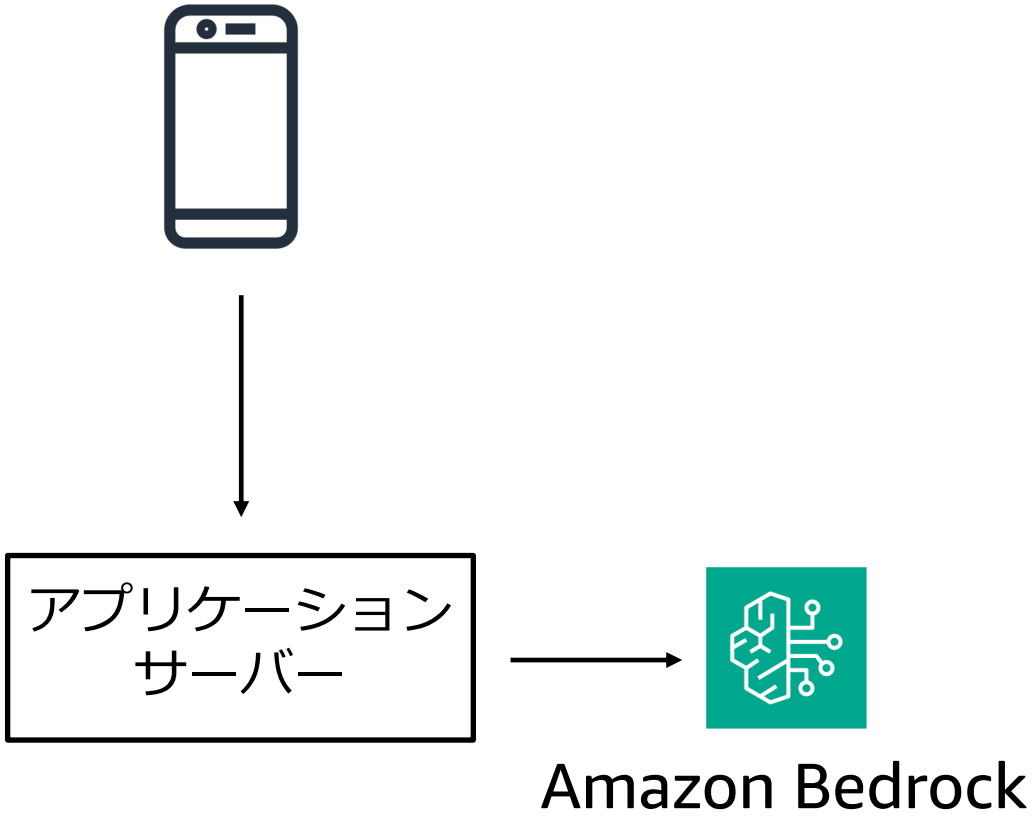
誹謗中傷 ❌



スパム投稿 ❌

解決できないか？

ECサイトへのコメント投稿を審査する場合を考える



Prompt

次のコメントを不適切か判断してください。

「このパンすごい美味しかった」

実際プロダクションに導入するとなると考慮点は多い

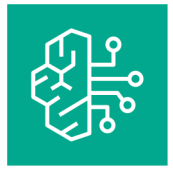
そもそも人力や従来の
MLモデルではだめなの？

想像している基準で精度良く
分類してくれない

レスポンスが案外遅い

リリース後の監視や改善は
どうする？

アプリケーション
サーバー



Amazon Bedrock

Prompt

次のコメントを不適切か判断してください。

「このパンすごい美味しかった」

生成AI APIからエラーが返ってきた時にどうしようか？

全てのコメントをLLMで処理すると案外高くつきそう

セキュリティ面も気をつけてと言われたが何を気にする？

実際プロダクションに導入するとなると考慮点は多い

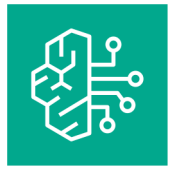
他の手段との棲み分け

精度

レスポンス速度

LLMOps

アプリケーション
サーバー



Amazon Bedrock

Prompt

次のコメントを不適切か判断してください。

「このパンすごい美味しかった」

可用性・スループット

コスト

セキュリティ

LLMの実導入における考慮点と打ち手

他の手段との棲み分け

人、ルールベース、従来のMLモデル

精度

評価 / モデル / Prompt / タスク分解

コスト

モデルサイズ / タスク分解 / 非同期処理

可用性・スループット

クオータ / リトライ / 複数リージョン / 非同期処理

レスポンス速度

モデル選定 / リージョン

LLMOps

APIのメトリクス監視 / LLMOps Tool

セキュリティ (別セッションでカバー)

「AI-T2-03: 生成 AI アプリケーション開発における セキュリティ・コンプライアンスのポイント」

1 / 6

LLMと従来の手法の棲み分け

- 人力や従来のMLモデルではだめなのか？
- どのような場合にLLMの活用がハマるのか？

コンテンツレビューにおけるLLM活用の棲み分け

	人間	ルールベースの処理	従来のMLモデル	LLM
精度	高	中	高	高
コスト	中	低	低 - 中	低 - 高
準備期間	低	低	高	中
専門人材	不要	エンジニア	データサイエンティスト	エンジニア
導入後の処理速度	遅	早	早	早

どれを選ぶか、どれを組み合わせるのが良いかは (当たり前だが) 状況による

※ 強い利点がある部分を赤でハイライトしている

コンテンツレビューにおけるLLM活用の棲み分け

	人間	ルールベースの処理	従来のMLモデル	LLM
精度	高	中	高	高
コスト	中	低	低 - 中	低 - 高
準備期間	低	低	高	中
専門人材	不要	エンジニア	データサイエンティスト	エンジニア
導入後の処理速度	遅	早	早	早

高い精度が求められるが、今すぐ本件にアサインできる高度な人材がいない場合

⇨
- 有人コンテンツ監視専門企業や、社内のカスタマーポートで対応
- エンジニアがいればNGワードなどルールベースの処理から実装

コンテンツレビューにおけるLLM活用の棲み分け

	人間	ルールベースの処理	従来のMLモデル	LLM
精度	高	中	高	高
コスト	中	低	低 - 中	低 - 高
準備期間	低	低	高	中
専門人材	不要	エンジニア	データサイエンティスト	エンジニア
導入後の処理速度	遅	早	早	早

大規模なプロダクトでレビュー数も多い + データサイエンティスト組織が存在する場合

⇨
- データのラベリングから高度なMLモデルの学習・推論までの流れを実装
- 人の作業が求められる領域を大幅に削減

コンテンツレビューにおけるLLM活用の棲み分け

	人間	ルールベースの処理	従来のMLモデル	LLM
精度	高	中	高	高
コスト	中	低	低 - 中	低 - 高
準備期間	低	低	高	中
専門人材	不要	エンジニア	データサイエンティスト	エンジニア
導入後の処理速度	遅	早	早	早

MLモデルの改善や運用に機械学習人材を割くまでの体制は取りづらいが、
今いるアプリケーションエンジニアで可能な限り自動化を試みたい場合

⇨ • LLMの活用により例えばアプリケーションエンジニアも試行錯誤可能に

コンテンツレビューにおけるLLM活用の棲み分け

	人間	ルールベースの処理	従来のMLモデル	LLM
精度	高	中	高	高
コスト	中	低	低 - 中	低 - 高
準備期間	低	低	高	中
専門人材	不要	エンジニア	データサイエンティスト	エンジニア
導入後の処理速度	遅	早	早	早

既に従来のMLモデルで対処できている組織であっても、以下の場合LLM活用が有用

- 従来のMLモデルで判断しきれなかった領域を更に高度なLLMで自動判別したい
- 学習データは少ないが、整備されている審査ガイドラインを活用したい
- 審査の判断理由や改善点も含めて出力させたい

2 / 6

精度改善に向けた打ち手

- 精度改善はどこから進めていけばよいのか？
- Promptの修正を繰り返すものの目指すべき精度とのギャップが埋まらない

精度改善の近道は簡易的な評価の仕組み作りから

試行錯誤するうちに評価基準が徐々に変化することも多いので、まずは改善のIterationを回すために参考になるような最低限の評価の仕組みを用意

評価データの作り方：

- 人が評価基準を参考にテストしたい入力と出力のペアを作成
- 評価基準を元にLLMで入力と出力のペアを作成
- サービスのログに対して人がアノテーション (カスタマーサポートが人手でチェックしたものを活用する場合もこちら)

実際の評価：

- 審査してOK/NGの2値判定するような場合は正誤が明確で機械的に判断可能
- チャットボットの回答など評価が曖昧なものは人やLLM(LLM-as-a-Judge)で判断

最先端のモデルを活用する

2024/10/22にClaude 3.5 Sonnet (v2)がリリース、Claude 3.5 Haikuがアナウンス

	Claude 3.5 Sonnet (new)	Claude 3.5 Sonnet	GPT-4o	Gemini 1.5 Pro
Graduate level reasoning *GPQA (Diamond)*	65.0% 0-shot CoT	59.4% 0-shot CoT	53.6% 0-shot CoT	59.1% 0-shot CoT
Undergraduate level knowledge *MMLU Pro*	78.0% 0-shot CoT	75.1% 0-shot CoT	—	75.8% 0-shot CoT
Agentic coding *SWE-bench Verified*	49.0%	33.4%	—	—
Code *HumanEval*	93.7% 0-shot	92.0% 0-shot	90.2% 0-shot	—
Math problem-solving *MATH*	78.3% 0-shot CoT	71.1% 0-shot CoT	76.6% 0-shot CoT	86.5% 4-shot CoT

※ Amazon Bedrock上でもClaude 3.5 Sonnet v2 はオレゴンリージョンで利用可能
※ Claude 3.5 Haikuも、text only版から近日公開予定。性能はClaude 3 Opus相当

Promptの原則をおさえる

明確で直接的な指示

例（マルチショットプロンプト）の使用

Claudeに考えさせる（CoT）

XMLタグを使用する

Claudeに役割を与える（システムプロンプト）

Claudeの応答を事前入力

複雑なプロンプトのチェーン化

長文コンテキストのヒント

Anthropic User Guide

あなたはECサイトのコメントをレビューする役割が与えられています。以下に与えられたガイドラインに正確に従って、投稿コメントが不適切な内容かどうかを判断してください。

ガイドライン：
<guideline> 詳細なガイドライン </guideline>

投稿コメント：
<comment> メッセージ </comment>

レビュー結果を以下のようなJSON形式で出力してください:
{
“violation”: <メッセージが不適切なら”true”、適切なら”false”のbool値>,
“explanation”: <ガイドライン違反がある場合のみ含めてください>
}

https://docs.anthropic.com/ja/docs/build-with-claude/prompt-engineering/overview

Promptの原則をおさえる

冗長で膨大なガイドラインを全て
含めると精度低下の原因に

人手やLLMで事前に整理するのも重要

審査項目ごとのNGなコメント例を
多数加えるのも一つのアプローチ

あなたはECサイトのコメントをレビューする役割が与えられています。以下に与えられたガイドラインに正確に従って、投稿コメントが不適切な内容かどうかを判断してください。

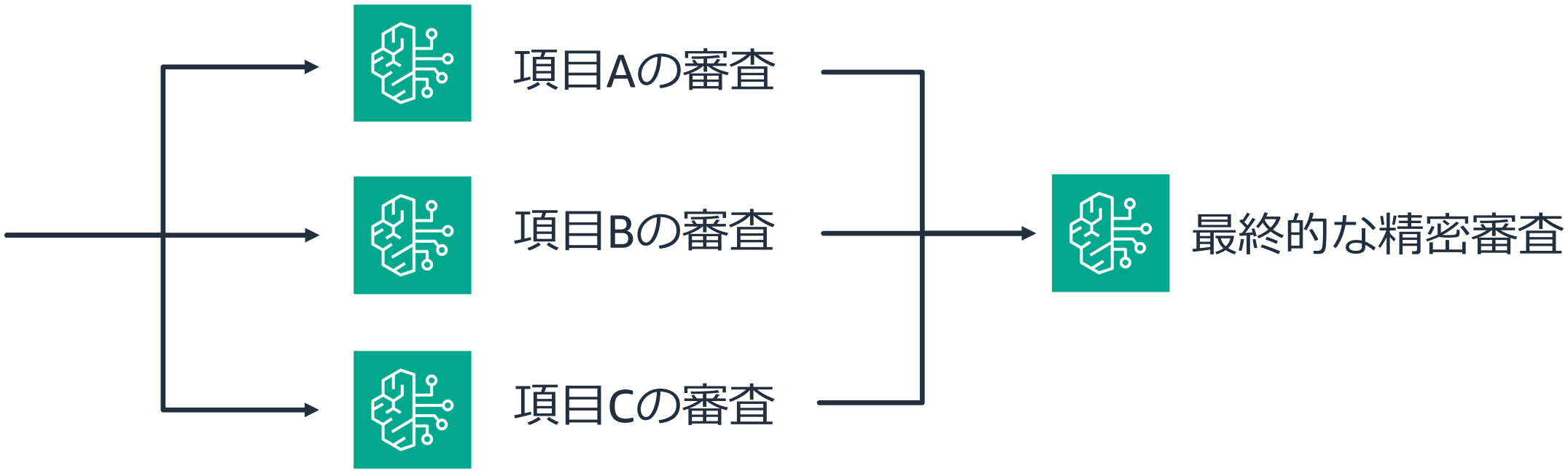
ガイドライン：
<guideline> 詳細なガイドライン </guideline>

投稿コメント：
<comment> メッセージ </comment>

レビュー結果を以下のようなJSON形式で出力してください:
{
  “violation”: <メッセージが不適切なら”true”、適切なら”false”のbool値>,
“explanation”: <ガイドライン違反がある場合のみ含めてください>
}

タスク分解 と Prompt Chaining

複雑な作業を一つのLLMで処理するのではなく、 いくつかのサブタスクに分割し
それぞれ個別のLLMで処理するアプローチ



Pros: 各タスクの精度が向上、サブタスクごとに個別で改善しやすい (Traceability)
Cons: コストや処理時間が増加する可能性がある。工夫次第では削減の場合も

https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/chain-prompts#example-analyzing-a-legal-contract-with-chaining

3 / 6

コスト最適化に向けた打ち手

- 全てのコメントをLLMで処理すると案外高くのではないか？

コスト感の確認

200文字相当のコメントが **1日に100個** 投稿されたとして、これを処理する料金はどれぐらいを想像しますか？

Claude 3.5 Haiku (軽量モデル): 約 〇〇円 / 月
Claude 3.5 Sonnet (高性能モデル): 約 〇〇円 / 月

200文字相当のコメントが **1秒に1個** 投稿された場合は？

Claude 3.5 Haiku: 約 〇〇円 / 月
Claude 3.5 Sonnet: 約 〇〇円 / 月

コスト感の確認

200文字相当のコメントが **1日に100個** 投稿されたとして、これを処理する料金はどれぐらいを想像しますか？

Claude 3.5 Haiku (軽量モデル): 約 150円 / 月
Claude 3.5 Sonnet (高性能モデル): 約 2千円 / 月

200文字相当のコメントが **1秒に1個** 投稿された場合は？

Claude 3.5 Haiku: 約 15万円 / 月
Claude 3.5 Sonnet: 約 200万円 / 月

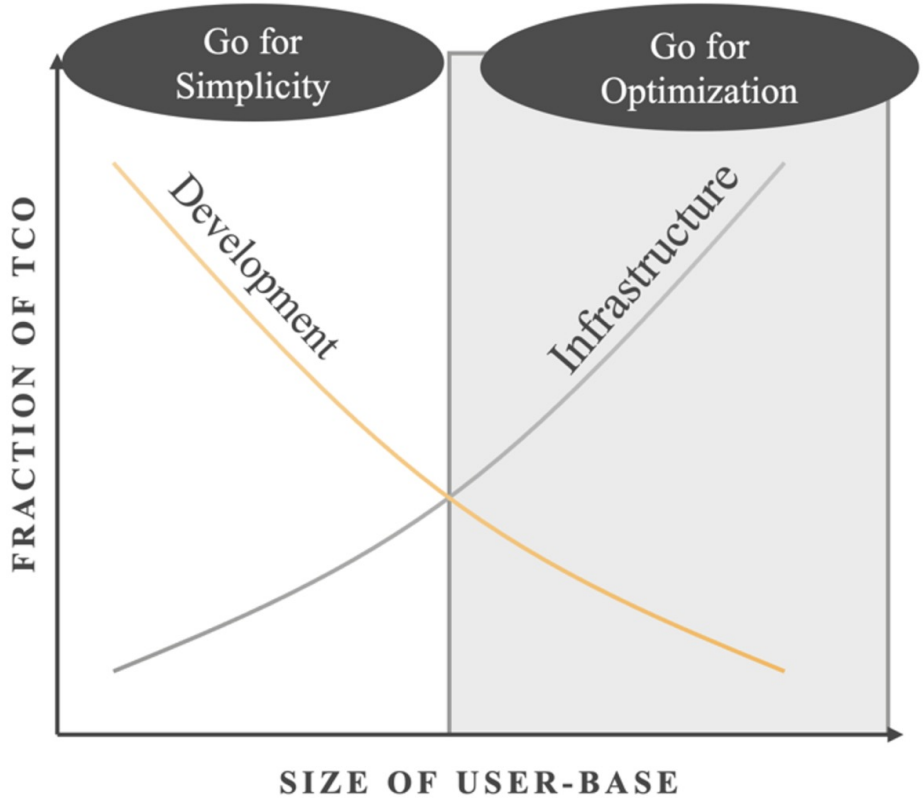
コストと複雑さに関するメンタルモデル

利用規模が小さい場合

過度に最適化せず、シンプルなアプローチを推奨
例：高精度なLLM一つで完結させる

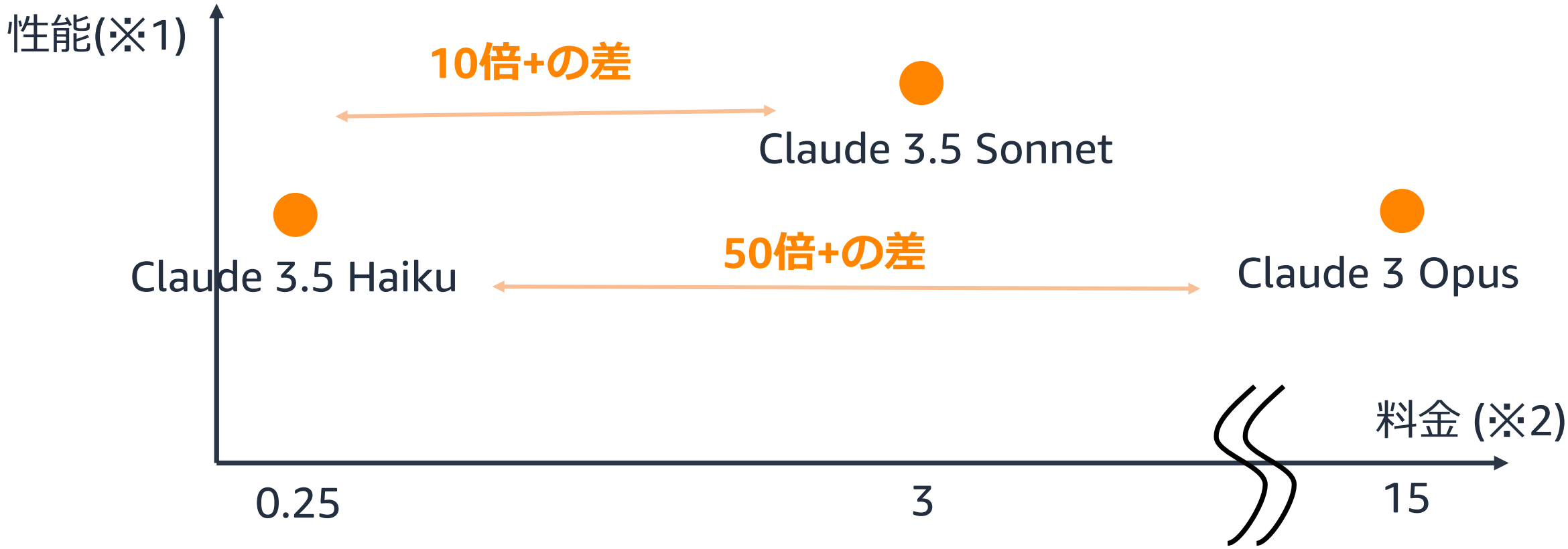
利用規模が大きい場合

エンジニアリングや科学的な取り組みに注力して
長期的なインフラコスト最適化に注力する価値あり
例：タスクを分解してそれぞれにLLMを適用



Amazon science: How task decomposition and smaller LLMs can make AI more affordable

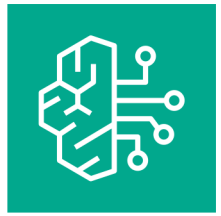
モデルによるコストの違い



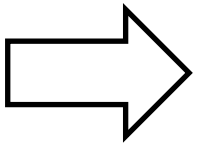
※1 ここでは特定の指標の値ではなく定性的な表記をしている
※2 100万トークンあたりのInput token料金 (USD)。output tokenも比率は同じ

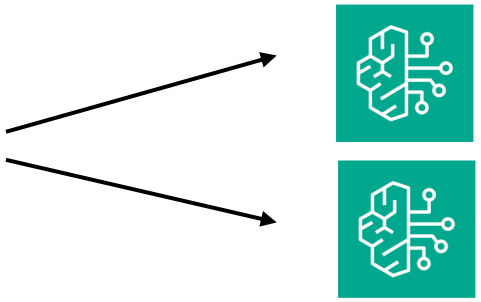
タスク分解もコスト最適化に繋がる

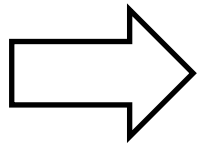
複雑な作業を一つのLLMで処理するのではなく、 いくつかのサブタスクに分割し
それぞれ個別のLLMで処理するアプローチ

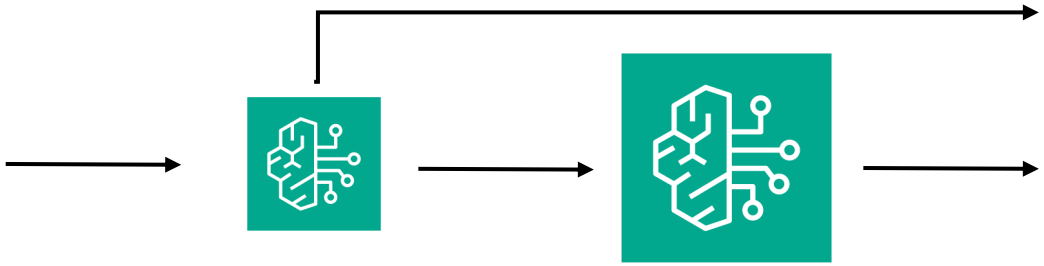


複雑なレビューを
高性能LLMでまとめて処理









簡易な審査
(軽量LLM)

複雑な審査
(高性能LLM)

非同期処理化によるコスト最適化

Amazon Bedrockのバッチ推論機能を利用すると、通常24時間以内にオンデマンドの料金の 50% で処理できる

利用例：
- パーソナライズされたメール文面を定期的に作成
- RAGなどの用途に埋め込みベクトルを一括作成

https://aws.amazon.com/jp/about-aws/whats-new/2024/08/amazon-bedrock-fms-batch-inference-50-price/

4 / 6

可用性・スループット

- 生成AI APIからたまに4xx, 5xxのエラーが返ってくるのはなぜだろうか？
- 今後更に大規模な利用を見込んでいるが何か備えはできるのだろうか？

まずクオータを確認

英語のドキュメントやService Quotasで、一分あたりのリクエスト数やトークン数等の制限（クオータ）を確認。モデルやリージョンによって異なる

クオータに当たっている場合、Bedrockではレスポンスで429,400が返却される
クオータに当たっていないが、過負荷等でリクエストを処理出来ない場合503を返却

クオータを超えて利用したい場合は一度AWSの営業やSAにご相談ください

Name	Default	Adjustable	Description
On-demand InvokeModel tokens per minute for Anthropic Claude 3.5 Sonnet	us-east-1: 400,000 us-east-2: 400,000 us-west-2: 2,000,000 ap-northeast-2: 400,000	No	The maximum number of tokens that you can submit for model inference in one minute for Anthropic Claude 3.5 Sonnet. The quota considers the combined sum of Converse, ConverseStream, InvokeModel and InvokeModelWithResponseStream.

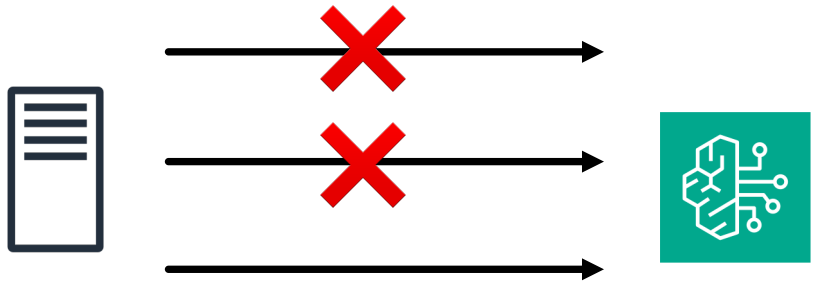
https://docs.aws.amazon.com/general/latest/gr/bedrock.html

リトライの仕組みを必ず意識

APIを呼び出す際にリトライの仕組みがあると扱いやすい

SDKによってもリトライの挙動は少しずつ異なるので一度確認を

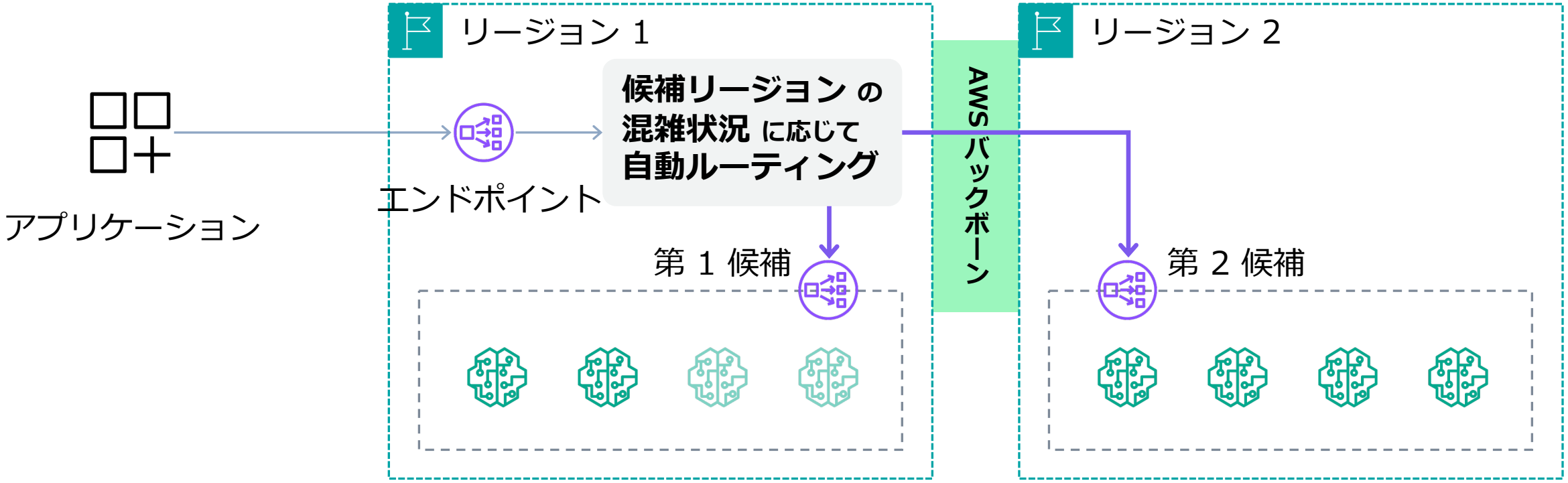
例えば、Pythonで利用されるboto3の場合、特に重要な429, 503に対して
デフォルト設定で最大5回API呼び出しを再試行する



https://boto3.amazonaws.com/v1/documentation/api/latest/guide/retries.html

複数のリージョンを合わせて利用するケースもある

Amazon Bedrockには、Cross-region inferenceという機能があり、
自動的に複数リージョンを利用することで、より可用性高く安定した推論を実現



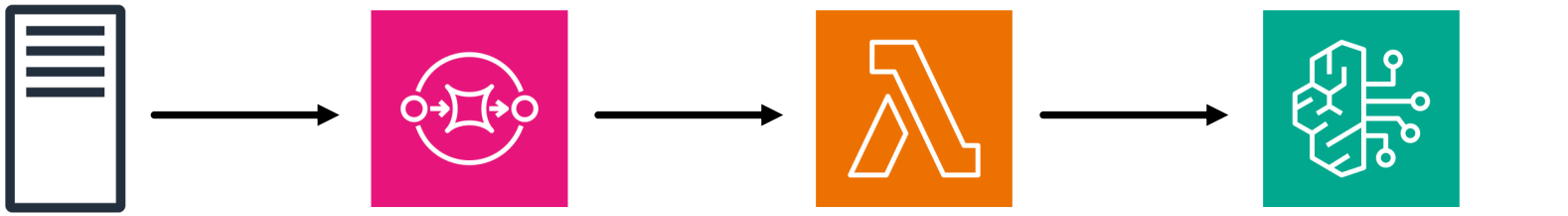
※ 現在米国とヨーロッパが対象

https://docs.aws.amazon.com/bedrock/latest/userguide/cross-region-inference.html

非同期処理化

UX的に可能であれば、非同期処理となるような実装も有用

過負荷で一時的に後段のAPIが利用不能な場合も、エンドユーザーへの影響を小さくできる

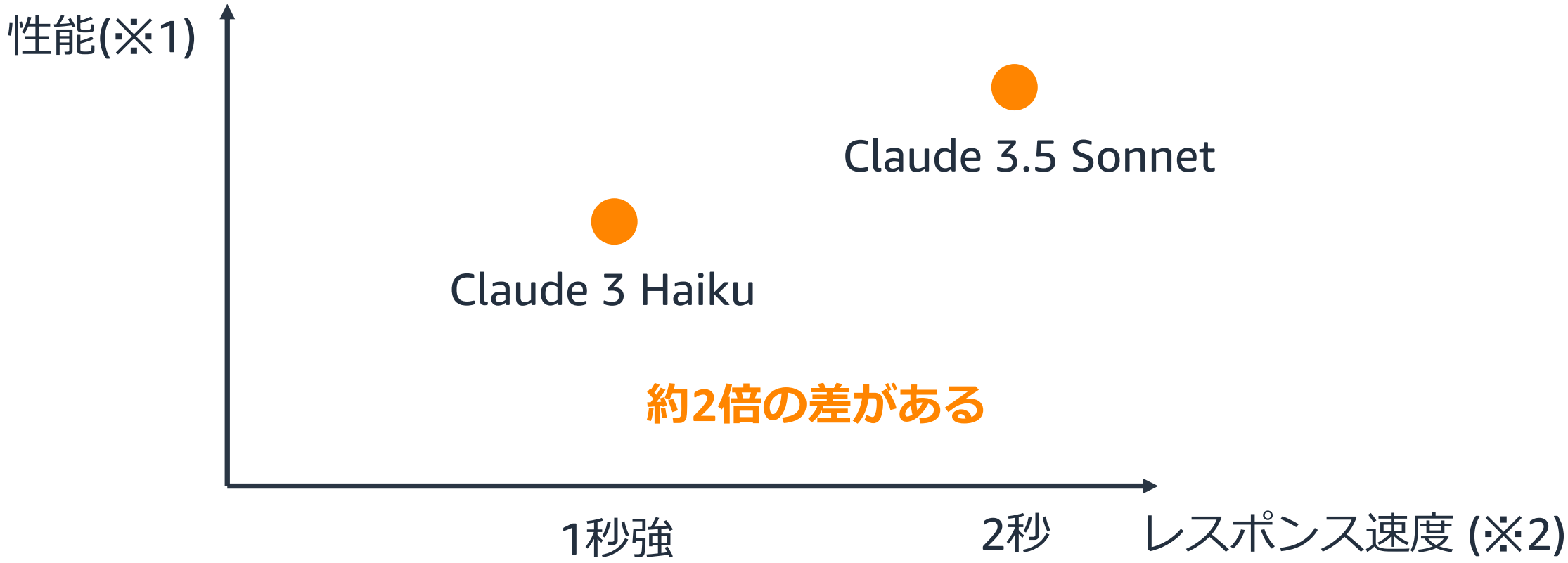


5 / 6

レスポンス速度

- レスポンス速度をもう少し早くできないか？

モデルによるレスポンス速度の違い



※1 ここでは特定の指標の値ではなく定性的な表記をしている
※2 同リージョン、入力・出力100トークンの場合の値。タイミングにより変動するので目安

レスポンス速度観点でのリージョン選定

よく選択肢に挙がるリージョン

- 北部バージニア (us-east-1)　約144 ms
- オレゴン (us-west-2)　約96 ms
- 東京 (ap-northeast-1)　N/A

} 東京からの参考round trip latency

東京リージョンにアプリケーションがある場合、確かに東京がレスポンスは
早く返るのだが、クオータの上限から考えるとまずはオレゴンを推奨

※ レイテンシは AWS Network Manager Infrastructure Performance でのある断面での参考値

レスポンス速度の改善には何が寄与するのか

レスポンス速度は モデル、 トークン数、 ネットワーク遅延 に主に影響される

より軽量なモデルを利用して性能が許容されるか

入出力のトークンを削減できるか

- Promptに含めるコンテキスト量の削減
- 出力を最小限になるよう指定
- キャシュ
- Fine Tuning

よりネットワーク距離が近いリージョンを利用できるか

レスポンスの遅さを感じさせないUXにできるか

- ストリーミング出力
- 処理が進んでいるようなメッセージ

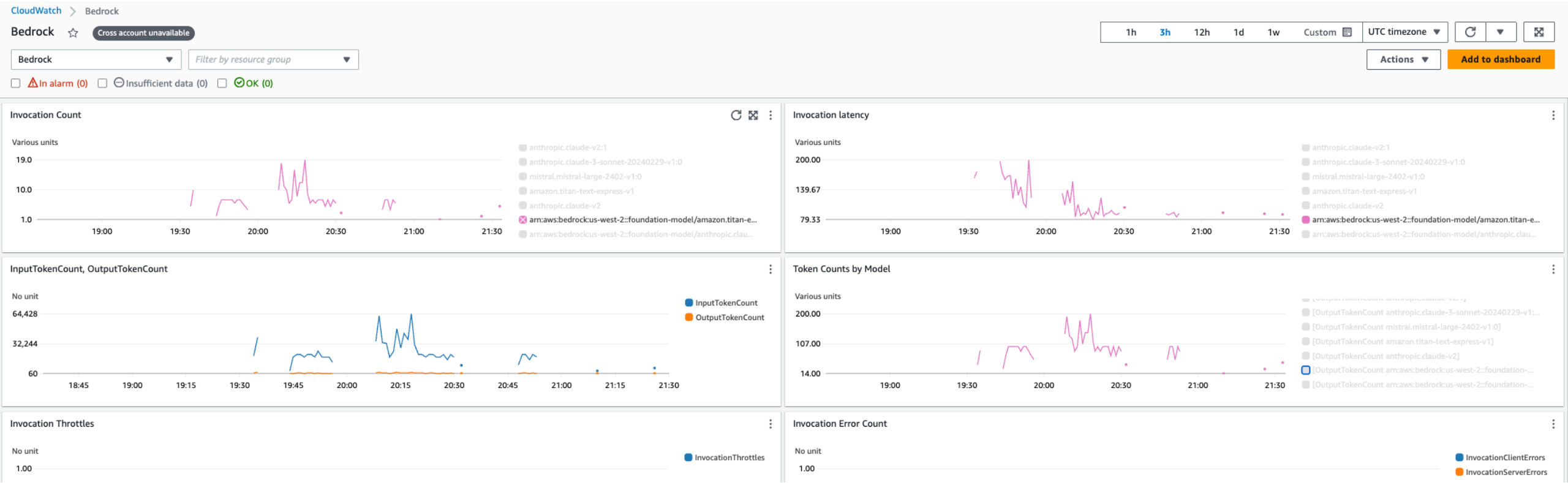
6 / 6

LLMOps

- 生成AI APIのメトリクスはどうモニタリングはどうするか？
- LLMの実行ログを確認してPromptの改善に繋げたい

Bedrockのメトリクスを確認するためのダッシュボードが存在

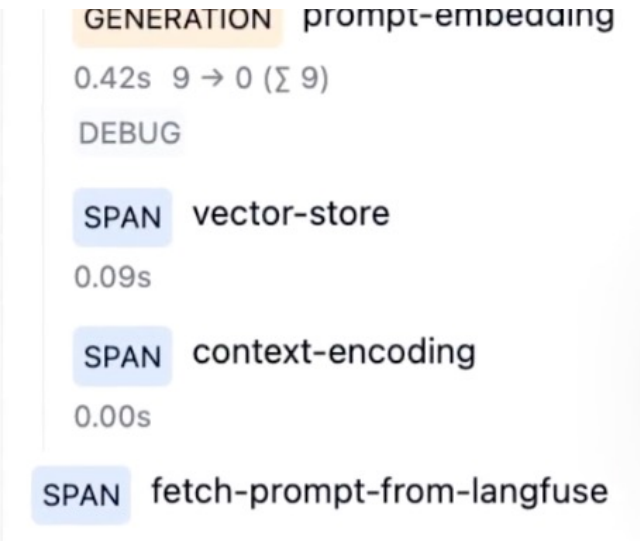
CloudWatchの自動ダッシュボードにBedrock専用ダッシュボードの用意あり
どのモデルがどれぐらい使われているか、クオータに当たりスロットリングしているか、そもそもエラーになっているか等が一目で分かる



https://us-west-2.console.aws.amazon.com/cloudwatch/home?region=us-west-2#home:dashboards/Bedrock

LLMの実行結果をデバッグ、評価、改善するようなツールも広がりを見せている

- Bedrockの「Model invocation logging」の設定で、LLMの入出力ログをCloudWatch LogsやS3に格納可能
- LangSmithやLangFuseをはじめ、トレース、評価、改善も含めたLLMOpsツールも拡張が続いている



https://langfuse.com/

まとめ

再掲：実際プロダクションに導入するとなると考慮点は多い

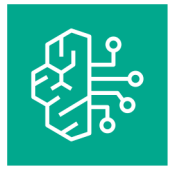
そもそも人力や従来の
MLモデルではだめなの？

想像している基準で精度良く
分類してくれない

レスポンスが案外遅い

リリース後の監視や改善は
どうする？

アプリケーション
サーバー



Amazon Bedrock

Prompt

次のコメントを不適切か判断して
ください。

「このパンすごい美味しかった」

生成AI APIからエラーが返って
きた時にどうしようか？

全てのコメントをLLMで処理する
と案外高くつきそう

セキュリティ面も気をつけて
と言われたが何を気にする？

まとめ：LLMの実導入における考慮点と打ち手

本セッションでは、コンテンツ審査を題材として、生成 AI を実プロダクトに実装する際に直面する 考慮点 を整理し、具体的な 打ち手 をご紹介します

他の手段との棲み分け

人、ルールベース、従来のMLモデル

精度

評価 / モデル / Prompt / タスク分解

コスト

モデルサイズ / タスク分解 / 非同期処理

可用性・スループット

クオータ / リトライ / 複数リージョン / 非同期処理

レスポンス速度

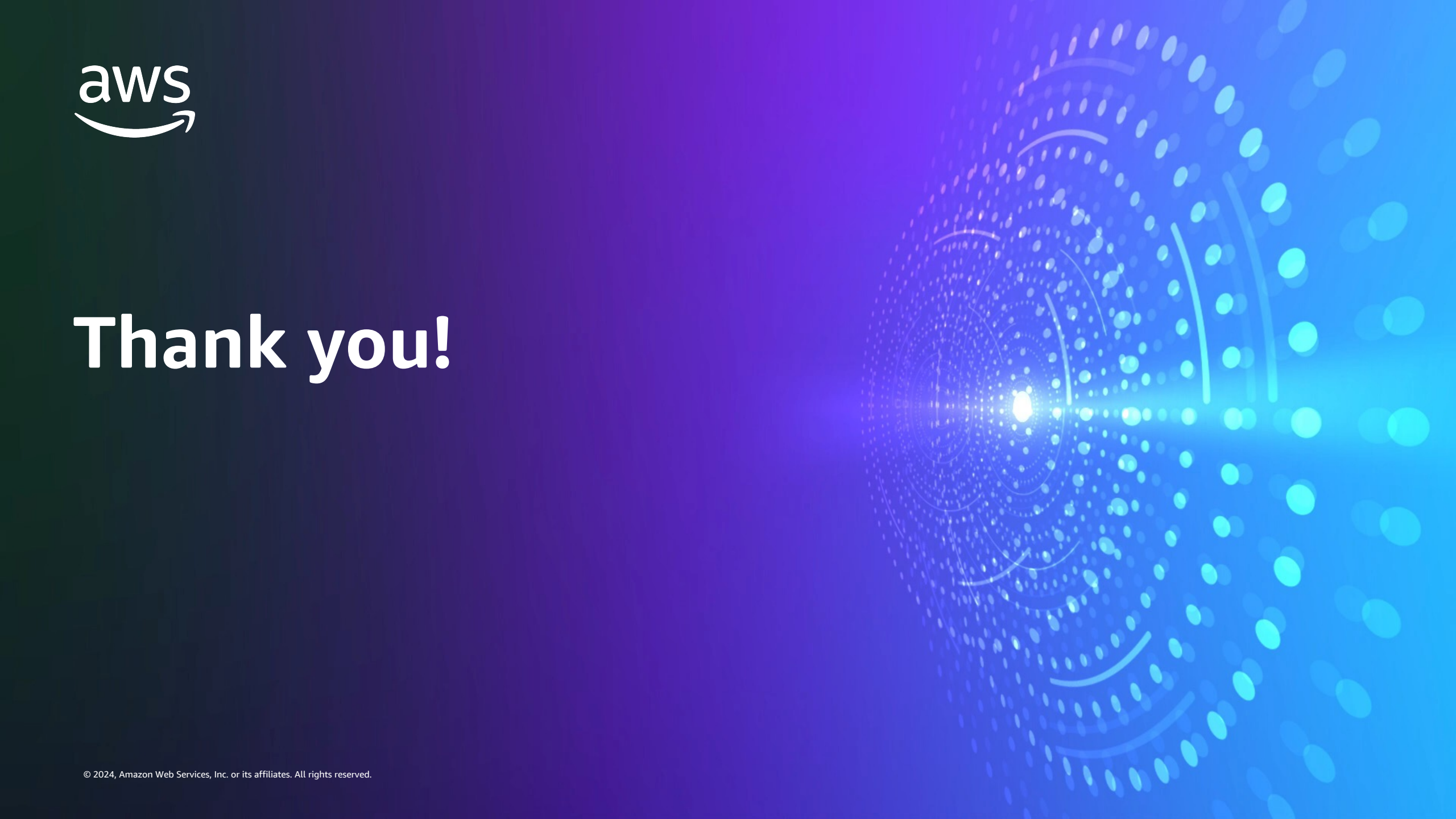
モデル選定 / リージョン

LLMOps

APIのメトリクス監視 / LLMOps Tool

セキュリティ (別セッションでカバー)

「AI-T2-03: 生成 AI アプリケーション開発における セキュリティ・コンプライアンスのポイント」



Thank you!