ソフトウェア工学タスクにおける LLMカスケードのコスト削減効果

オープンソースソフトウェア工学研究室

学生番号：S2620051
佐久間 一颯

背景：LLMとその課題

LLM（Large Language Model）とその課題

近年は様々なタスクの処理に利用されている

例）文章要約，翻訳，コード生成，バグ修正

◎高性能モデルの利用コストが高額すぎる

例）GPT-5とGPT-5-nanoでは25倍もの差

➤<u>コストが障壁となり，実務での導入が懸念される</u>

モデルルーティング

タスクの特性に応じて最適なモデルを選択し処理する技術

➤計算コストの削減が見込める

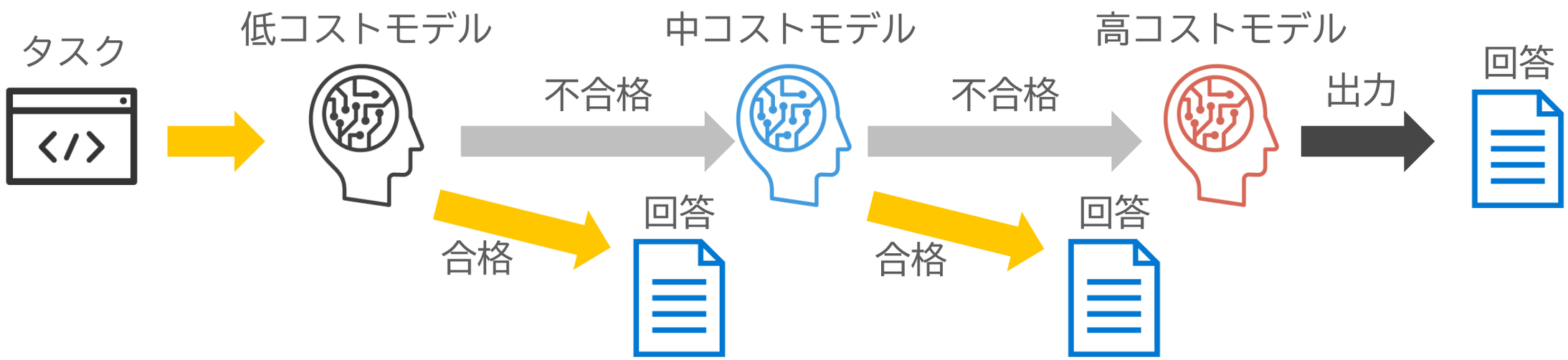
背景：LLMカスケード

LLMカスケード

安価なモデルから順にモデルを使用していく手法

例）FrugalGPT[1]

➤LLMの<u>計算コストの削減</u>が見込める



図：FrugalGPTの概要

・自然言語処理(NLP)タスクにおいて<u>98%のコスト削減</u>に成功

[1] Lingjiao Chen, Matei Zaharia, James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance

先行研究の課題点

NLPタスクに絞って実験を行っている

NLPタスク以外の形式のタスクでは同様の効果があるかわからない

```
mmlu_item = {
    "question": "半径が5の円の面積として最も近い値はどれか？",
    "choices": ["78.5", "31.4", "15.7", "62.8"],
     "answer" : 0  #"78.5"が正解
}
```

出力は選択肢
短いテキスト形式

図：NLPタスクの一例[2]

```
def add_numbers(a: int, b: int) -> int:
    """
    2つの整数aとbを受け取り，その和を返す関数を実装せよ.
    >>> add_numbers(2, 3)
    5
    """
```

出力はコード
長い場合は数十行

図：SEタスクの一例[3]

➤LLMカスケードはSEタスクでも同様のコスト削減効果があるのか

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryl Michalewski, David Dohan, Carrie Cai, Michael Terry, Quoc Le, Charles Sutton, Program Synthesis with Large Language Models
[3] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, Jacob Steinhardt, Measuring Massive Multitask Language Understanding

研究目的

研究目的

SEタスクにおけるLLMカスケードの有効性を検証する

アプローチ

- ベースライン（GPT-5）とLLMカスケードを比較

  コスト削減効果と性能の差があるかを検証

- SEタスク（**コード生成**）とNLPタスク（**四択問題**）を使用

  タスクの違いが効果に与える影響を検証

リサーチクエスチョン

本研究のリサーチクエスチョン

RQ1：**コスト削減効果**はどの程度か

SEタスクでもNLPタスクと同等のコスト削減率を出せるのか

RQ2：**性能はどの程度変化**するのか

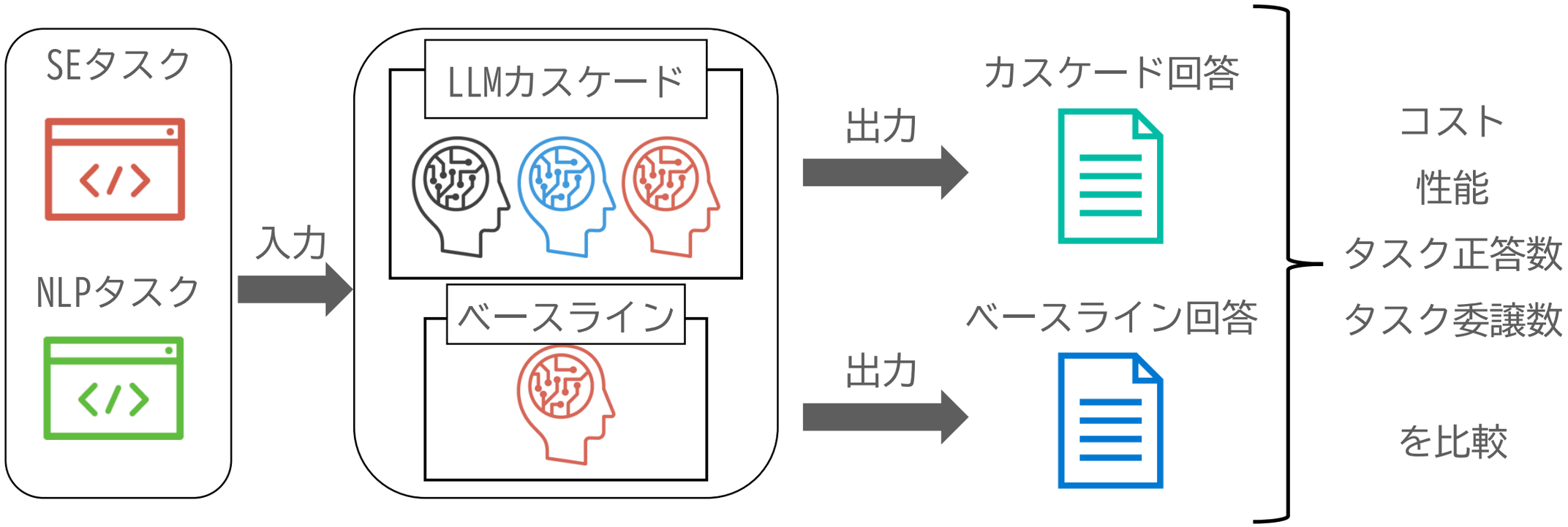
低コスト，中コストモデルのみが解けるタスクがあるのか

RQ3：各モデルの**正答数と委譲数**はどの程度か

LLMカスケードの運用効率を可視化する

例）低コストモデルの使用率が多い ➤ カスケードの効果有

実験設定：実験概要



図：実験概要

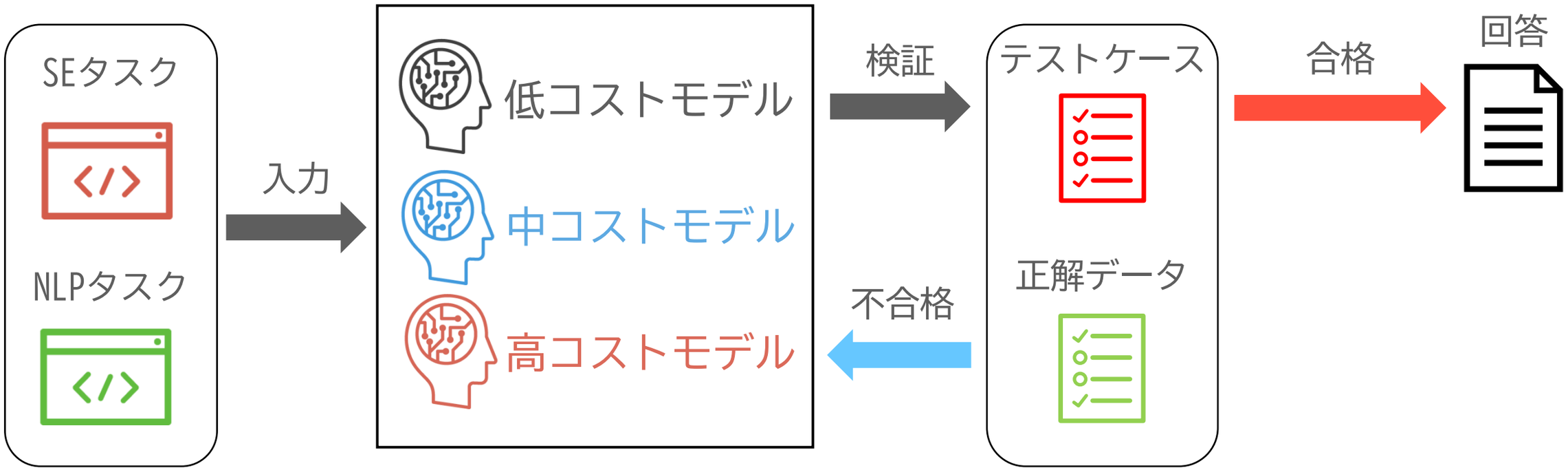
出力の上限

出力トークン数は2048トークンを最大に設定

・今回のタスクにおいて十分なトークン量としている[4]

[4] https://github.com/bigcode-project/bigcode-evaluation-harness/blob/main/docs/README.md

実験設定：LLMカスケードの概要



図：LLMカスケードの概要

LLMカスケードの判定基準

Pass@1（SEタスク）

テストケースに<u>生成された回答</u>を通すことで合否判定

Exact Match（NLPタスク）

正解データと<u>生成された回答</u>を比較して合否判定

データセット

●使用するデータセット

表：使用するデータセット

種類	問題数	タスク内容
HumanEval	164問	SEタスク（コード生成）
MBPP+	378問	SEタスク（コード生成）
MMLU	500問	NLPタスク（一般知識問題）
HEADLINES	500問	NLPタスク（金融知識問題）

●使用するモデル

表：使用するモデル

位置	モデル名	入力コスト(100万トークン)	出力コスト(100万トークン)
低コスト	**GPT-5-nano**	$0.05	$0.4
中コスト	**GPT-5-mini**	$0.25	$2.0
高コスト	**GPT-5**	$1.25	$10.0

評価指標：リサーチクエスチョン

コスト削減率

$$\text{コスト削減率} = 1 - \frac{LLM\text{カスケード総コスト}}{\text{ベースライン総コスト}}$$

ドルベースのAPI利用コストの総額を「コスト」とする

性能評価の指標

$$\text{ベースライン性能} = \frac{\text{GPT-5正答数}}{\text{全問題数}}$$

GPT–5の正答数から求める

$$\text{LLMカスケード性能} = \frac{\text{正答数の合計}}{\text{全問題数}}$$

LLMカスケードによる正答数の合計から求める

結果：RQ1（コスト削減効果）

全タスクで45～78%のコスト削減を達成

表：コスト削減率

種類	ベースライン	カスケード	削減率
HumanEval	$1.29	$0.37	71.26%
MBPP+	$2.59	$1.09	57.96%
MMLU	$1.88	$0.41	78.33%
HEADLINES	$0.54	$0.30	45.39%

HumanEval，MBPP+双方でコストが削減された

◎LLMカスケードはSEタスクにもコスト削減効果がある

結果：RQ2（性能変化）

全タスクで性能が向上した

表：各タスクにおける精度

種類	ベースライン	カスケード	精度の変化
HumanEval	87.20%	90.85%	+3.65%
MBPP+	74.34%	79.89%	+5.56%
MMLU	91.00%	94.00%	+3.00%
HEADLINES	76.00%	76.60%	+0.60%

数%程度だが性能が向上している

➤LLMカスケードを行っても．**性能維持**はできていた

結果：RQ3（各モデルの正答数と委譲数）

低コストモデルで大半の問題を処理

表：各コストモデルの正答数

タスク名	GPT-5-nano	GPT-5-mini	GPT-5
HumanEval	111	38	0
MBPP+	233	68	1
MMLU	410	55	5
HEADLINES	374	5	4

表：各コストモデルの委譲数

タスク名	GPT-5-nano	GPT-5-mini	GPT-5
HumanEval	164	53	15
MBPP+	378	145	77
MMLU	500	90	35
HEADLINES	500	126	121

LLMカスケードにより高コストモデルの使用頻度を大幅に削減

➤LLMカスケードはコスト削減の直接的な要因となる

考察

SEタスク間でのコスト削減効果の差

○**問題仕様の明確さ**に関係があると考えられる

HumanEval（コスト削減率 71.26%）

```
def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
```

図：HumanEvalの一例

関数シグネチャ，各値の型が事前に定義

MBPP+（コスト削減率 57.96%）

```
%プロンプト
 "Write a function to find squares of individual elements in a list."

%テストケース
 assert square_nums([1,2,3,4,5,6,7,8,9,10])==[1,4,9,16,25,36,49,64,81,100]
 assert square_nums([10,20,30])==([100,400,900])
 assert square_nums([12,15])==([144,225])
```

図：MBPP+の一例

モデルが文から全て推測しなければならない

考察

性能向上の要因分析

◎ GPT-5単体では不正解だが，カスケードで正解のタスクが存在

表：HumanEvalにおけるGPT-5単体で不正解，LLMカスケードで正解となったタスクの分析

問題番号	内容	回答の有無	不正解の原因
47	中央値の計算	×	トークン上限到達
76	べき乗判定	×	トークン上限到達
129	グリッド最短パス	×	トークン上限到達
130	隣接要素入れ替え判定	×	トークン上限到達
132	ネストした括弧の判定	×	トークン上限到達
147	要素数と合計の積	○	ロジックエラー

➤**出力の簡潔さの違い**によって正答数に差が出てくる

- GPT-5は詳細にタスクを理解しようとして，推論でトークンを使い切る
- GPT-5-nano，miniは簡潔に回答するため，同タスクで正解を出せる

考察

性能向上の要因分析

◎ 1問はロジックエラーで失敗している（HumanEval/147）

・LLMカスケードの方ではGPT-5-miniが正解

```
def get_max_triples(n):
    """
    You are given a positive integer n. You have to create an integer array a of length
n.
      For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1.
      Return the number of triples (a[i], a[j], a[k]) of a where i < j < k,
    and a[i] + a[j] + a[k] is a multiple of 3.

    Example :
      Input: n = 5
      Output: 1
      Explanation:
      a = [1, 3, 7, 13, 21]
      The only valid triple is (1, 7, 13).
    """
```

図：HumanEval/147のタスク内容

考察

性能向上の要因分析

```
if n < 3:
        return 0
    q, r = divmod(n, 3)
    c0 = q + (1 if r >= 2 else 0)  # count of i with i % 3 == 2 => a[i] % 3 == 0
    c1 = n - c0                    # count of i with i % 3 in {0,1} => a[i] % 3 == 1

    def comb3(m):
        return (m * (m - 1) * (m - 2)) // 6 if m >= 3 else 0

    return comb3(c0) + comb3(c1)
```

内部関数を定義したりして
内容が複雑化

図：HumanEval/147に対するGPT-5の回答

```
rem = n % 3
    base = n // 3
    # indices with i % 3 == 2 produce a[i] % 3 == 0
    c0 = base + (1 if rem >= 2 else 0)
    # the rest produce a[i] % 3 == 1
    c1 = n - c0
    result = 0
    if c0 >= 3:
        result += c0 * (c0 - 1) * (c0 - 2) // 6
    if c1 >= 3:
        result += c1 * (c1 - 1) * (c1 - 2) // 6
    return result
```

必要最低限の計算量で
内容もシンプル

図：HumanEval/147に対するGPT-5-miniの回答

GPT-5miniの方が簡潔でテストに通りやすいコードを生成している

まとめ

SEタスクにおけるLLMカスケードの有効性を検証

実証した全タスクで45～78%のコスト削減を達成

- NLPタスクと同様のコスト削減率を示した
- モデル性能も維持しながらコスト削減が可能

◎LLMカスケードはSEタスクにも有用であることが分かった

今後の課題

他のLLMやSEタスクでの検証

- より複雑なタスクでも同等の結果が得られるか

テストケース無しでの判定可能な手法の開発

- 本研究は理想的な状況での実証（テストケース有，正答例有）
- 実務においてはテストケースが無いことがある

生成AIのモデル崩壊の
メカニズム解明と抑止技術の構築

オープンソースソフトウェア工学研究室

学生番号：S2620051
佐久間 一颯

背景

生成AIの普及とデータの枯渇

日本企業の**87.6%**が生成AIを業務効率化に導入済[1]

なかでもシステム開発分野では全工程で使用する支援が進歩している

高度な生成AIを構築するには，高品質な学習データが必要

➤しかし収集され尽くされつつあり，2026年頃にはデータが枯渇することが予想される（2026年問題）

代替案として合成データを使用➤モデル崩壊を起こす危険

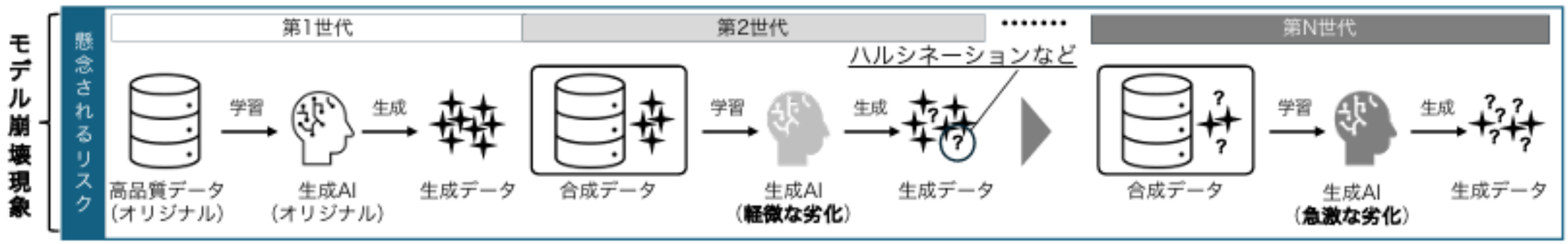
合成データ：既存データに生成AI自身の出力も追加したデータ

[1]デロイト トーマツ：プライム上場企業における生成AI活用の意識調査, 2024年5月.

背景

モデル崩壊

生成AIが合成データで追加学習を繰り返す過程で，
出力の質が劣化し，出力の多様性や正確性が失われていく現象



システム開発分野での危険性

生成AIはシステム開発の全工程に導入済み

例）要件定義，設計，実装，テスト，運用・保守

システム開発は信頼性・性能の持続が社会的に極めて重要

➤モデル崩壊が進めば，品質が静かに劣化する見えないリスクが

先行研究

モデル崩壊の構造的リスクは実証済み[2]

使用モデル：OPT-125m （軽量かつ汎用モデル）

➤現実の開発現場では適用できない可能性大

合成データに加え，元データ有りと無しで比較検証

システム開発分野での適用・検証はまだ無い

ICSE， FSE， ASE等では確認されなかった

[1] E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, and K. Matsumoto, "Studying ReOpened Bugs in Open Source Software," Empirical Software Engineering, Vol.18, No.5, pp.1005-1042, October 2013. (Core A, Q1 Journal)

先行研究

モデル崩壊の出力の具体例

Q：「1300年代の～(中世ヨーロッパに関する内容の記述)」
この文章を入力として，続きの文を生成してください

0世代：入力テーマとの関連性が保たれている

➤「ロンドンの～は一，19世紀後半には＝が～．」

1世代：説明のない人物や，少し関係のない建築が登場する

➤「ローマの～や，ブエノスアイレスの～は，教皇○○が～の時に～．」

5世代：関係のないことも話し出す

➤「これは英語，フランス語，ドイツ語～などで翻訳されていて～．」

9世代：全く関係ないことについて述べだす

➤「建築．ウサギの色は黒，白，青，赤，黄色があり，生息地で違う～」

研究目的

本研究の目的

１，汎用型・ドメイン特化型AIのモデル崩壊を実証的に**評価**

２，AI出力の判別技術の**確立**

３，汚染データの影響の定量化および性能との関係の**解明**

４，汚染データの除去技術の**構築**

リサーチクエスチョン

リサーチクエスチョン

①生成AIのモデル崩壊はどの経路で起きるのか

- 劣化するパターンはタスクの種類で変わるのか
- 汎用型とドメイン特化型で劣化速度は変わるのか
- どこから劣化が顕著になるか

②どんな種類のデータ汚染がモデル崩壊の進行に寄与し，それを除去することでどの程度影響を抑制できるのか

- 汚染データの種類（コード，自然言語，仕様書）によって影響度は異なるか
- アンラーニング後，どこまで性能が回復するのか

実験内容①

システム開発タスクに与える影響とメカニズムの解明

①汎用型生成AIを対象とした評価

評価対象タスク

：要件定義，設計情報回復，コード生成，テストケース生成，コード自動修正

使用モデル：Llama4等

追加学習に使用する合成データの比率：5段階以上を考えている

例）0%，25%，50%，75%，100%

追加学習の最大回数：？回

②ドメイン特化型生成AIを対象とした評価

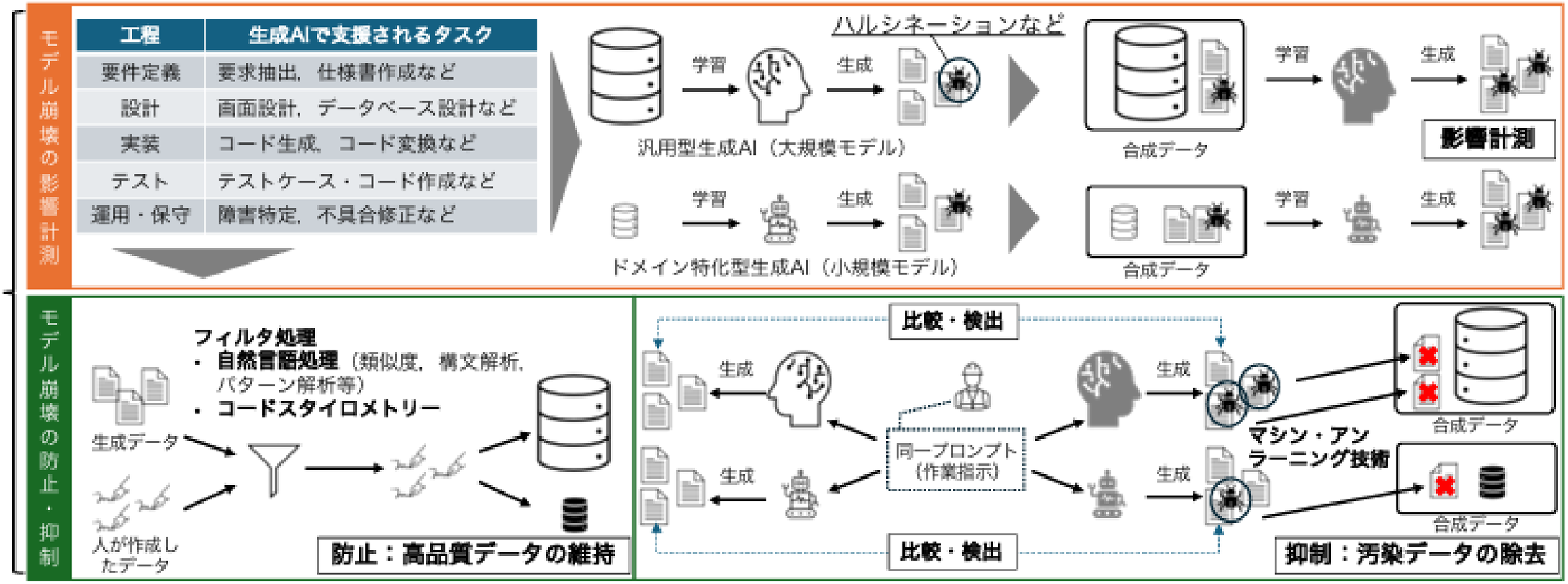
使用モデル：Hugging Faceから公開透明性・適合性・研究実績を基に選んだSLLM

その他①と同様

実験内容②

モデル崩壊の進行を抑制する技術の開発

①生成データのフィルタリング技術の確立と評価

②データ汚染の除去技術の確立と評価

オレンジ＝実験①　緑＝実験②