軽量言語モデルによるコード生成システム

Agentic code workbench

野尻美保子(KEK)

“Articulating Assumptions in AI-Generated Scientific Analysis through Task Decomposition 2607.05762

CoLLM: AI engineering toolbox for end to end deep learning in collider analysis 2602.06496

言語モデルと研究

ChatGPT, Claude

研究スタイルの変化

- コード作成の手間はほぼ0
  - デバッグ: 一定のチェック(pytest, 実行テスト）ここがほぼ律速
  - 仕様策定：対話的サポート

一方で

- モデル自体が高コスト、エネルギー消費、持続可能性
- ネットのアクセスがない。
- たとえば、大規模研究施設で、どのようにAI を活用していくか(レガシーコード、レガシーシステム)
- 言語モデルを観察：　「生成」「検証」「修正」

今日の話のまとめ

できるだけ小さい言語モデルにコードを書かせたい。

小規模LLMでも、タスクを分解し、ドメイン知識・実装・監査・人間の介入点を明示すれば、科学計算コード生成はかなり実用に近づく。

重要なのは「大きなモデルに全部任せる」ことではなく、「人間の作業構造をworkbench として外在化する」こと。

僕らはコードを書く時に何をしているか

- やりたいことを決める
- 大体のフローを考える

    どこにif をいれるか/どの作業を補助関数にするか/どこでループを回すか/

    どこにカウンターをいれるか
- コードを走らせる。
- グラフにみてまともな結果になっているか確認する。
- アウトプットから入力を遡って、なぜ思った通りになっていないか考える。
- 間違いを修正する。

“Agentic code workbench” for small LLM

- 14B=1.4x 10^{10} のモデルで動くコード生成システム

  (Chat さんは多分 2x10^{12})

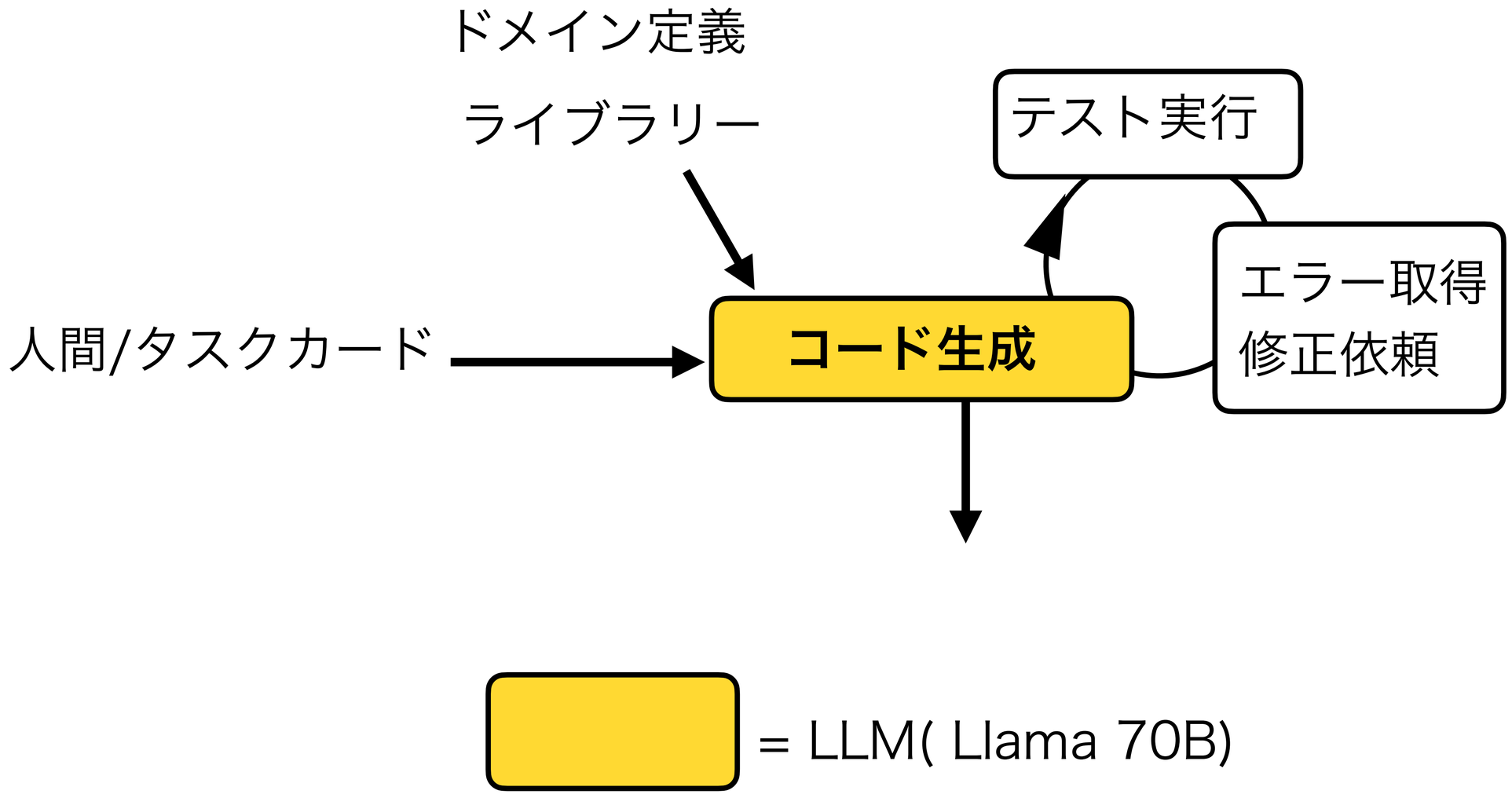
  - コンテクスト長の壁を乗り越えるタスク分解で、オンプレモデルで動作
  - 物理ドメインは自由に変更可能
  - 人の指示の明確化（タスクカード）
  - コード資産の利用や蓄積 全体設計→ **helper selector**
  - コード生成指示の曖昧度監査 （ユーザー補助）

    → **Golden Axe Oracle**

  - 生成コードについての監査→実装解析**(tracer)**の中間出力とそれに基づく評価 **(critique)**

    →人間の介入ポイントの定義の明確化　タスクごとに、言語モデルの能力評価が可能、多様な中間出力とモデル評価

システムの構成(2026.2)



システムの構成 (2026.7)

ドメイン情報は自由に取り替えることができる

全てのステップ中間アウトプット
成功率・時間
などの挙動を確認できる

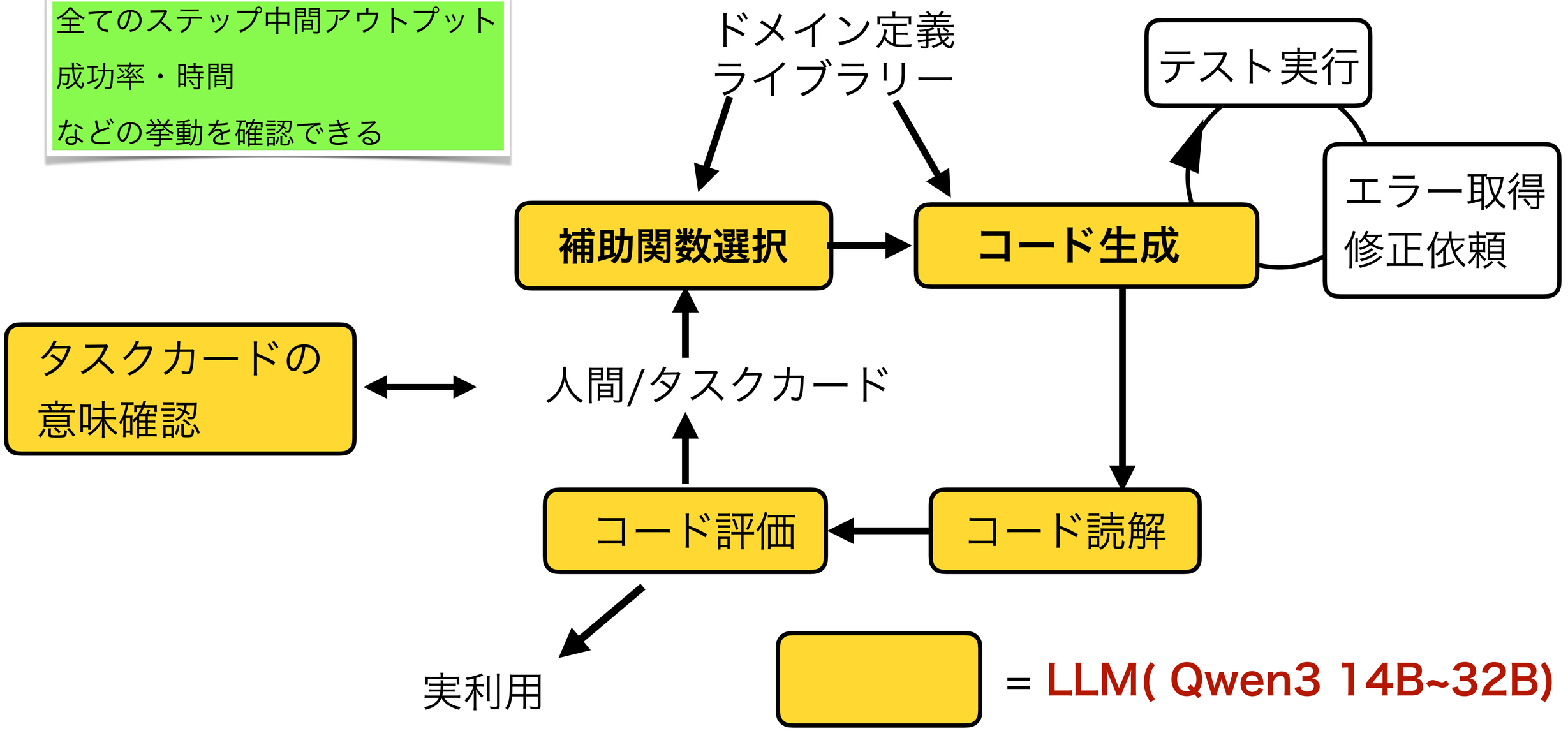


Figure from Graph of Thoughts paperFigure from Graph of Thoughts paper

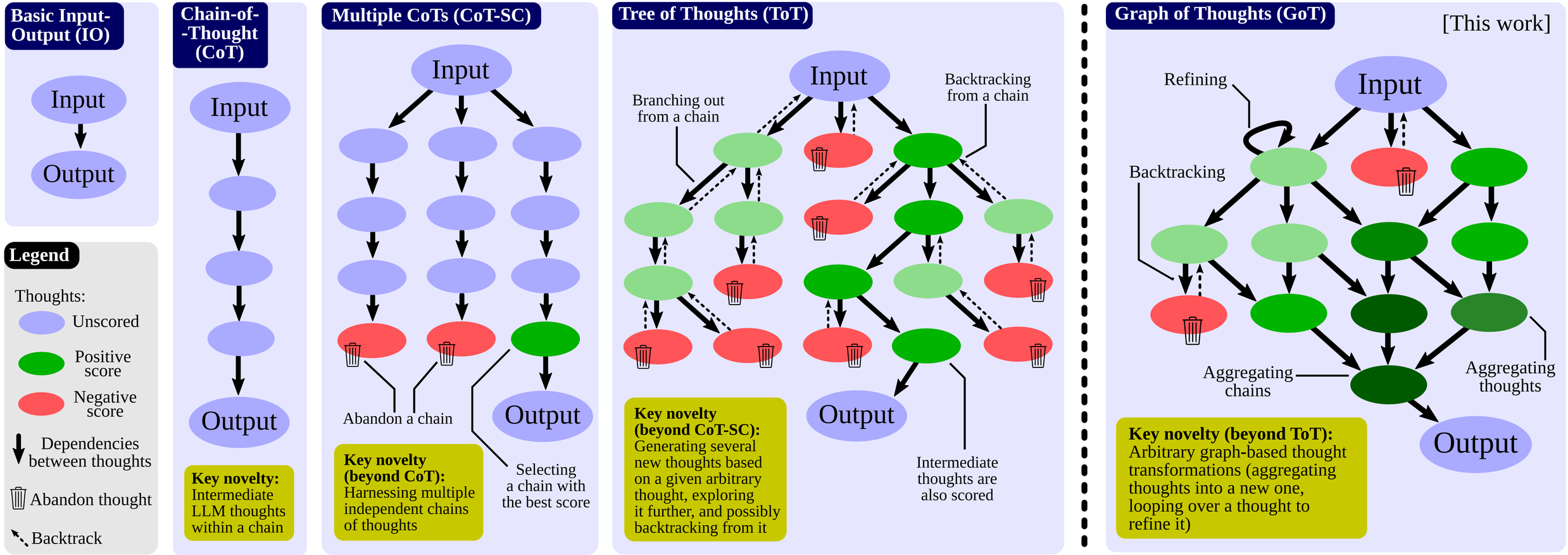


Figure 1: Comparison of Graph of Thoughts (GoT) to other prompting strategies.

コンセプトはGraph of Thoughts（LLM 内部の thought 構造の拡張）と似ている。
本研究では、人間が読める中間成果物としてタスクを系統的に外に出す。

Hugging Face LLM を利用

Hugging Face / transformers を使い、ローカルまたはオンプレ環境で Qwen 系モデルを動かすことを想定。

本研究では結局すべてHugging Face API でテスト

Qwen3 はモデルサイズに対して長めのコンテキストを扱いやすい。同じ条件でモデルサイズの影響を試せるのが魅力。

それ以外ではLlama-70B も試した。

タスクカード

箇条書きでやってほしいことを書く

演算もできる。

[SELECTION_CUTS]

- Select taus with pT > 20 GeV and |eta| < 2.3(カット)

- Require at least 2 taus (カット)

- **Define the leading tau and subleading tau from the selected taus.**

- **Define ditau system from the leading tau and subleading tau.**

[PLOTS_FOR_VALIDATION]

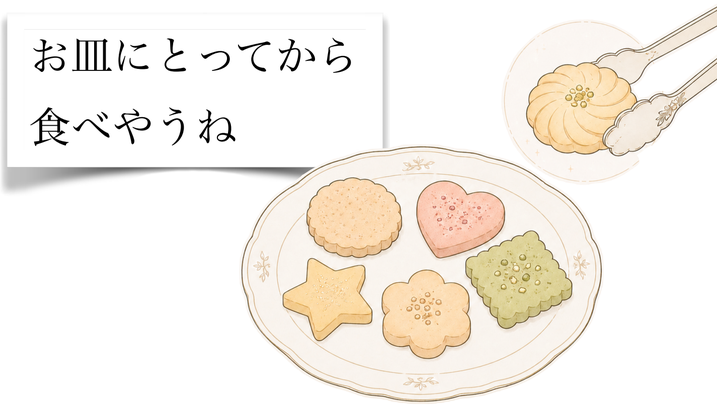
…

[OUTPUT_STRUCTURE]

….

Helper selector

小規模言語モデルにおけるコード生成



1. 補助関数リストをドメイン側から提供
2. システムプロンプトに補助関数リストを渡す
3. 目的にあった補助関数を選んでもらう。
4. helper_selector の選んだ補助関数名を[ADDITIONAL_HELPERS] としてspec に追加

```
You select helper functions from a fixed registry.

Do not write implementation code. Select only exact names present in the registry and do
not invent, rename, shorten, or alias helper names.

Selection rules:

- Cover explicit requirements in the task specification.

- Select a helper only when it supports an explicit requirement.

- Reject superficially related helpers that are not justified by the task.

- Prefer helpers matching the stated object and processing workflow.
```

Helper Registry

```
HelperSpec(
        name="four_momentum",
        section="parser_core",
        summary="Convert an object dict into (E, px, py, pz).",
        keywords=("four momentum", "px", "py", "pz", "energy", "invariant
mass"),
        aliases=("4-momentum", "lorentz components"),
        dependencies=("get_particle_mass",),
    ),
```

Helper Registry の初期抽出は chatGPT などの大規模LLM →人間の監査

Helper selector でありがちなこと:

14B, 32B 存在しないヘルパーを確信を持って選んでくる　←既存コードスタイルの暗唱

8B ともかく単語がマッチすれば、選ぶ

Tracer and Critique コード監査の自動化

“Tracer” module： 指示書(SPEC)に基づく実装を監査

どんな場面でも使えるように「**ドメイン知識と無関係**」にコードを表出（**ファイルに書く**）

→デバックに対する構造化された知識を表現　(とても大変　プロンプトが全体で266行)

```
Produce an "AS-IMPLEMENTED SPEC" using the SAME bullet structure and
order as the original SPEC.
```
（スペックに対応する実装を書き出しなさい）

```
Your analysis must be derived ONLY from the provided CODE.
Do not infer missing steps or assume domain knowledge not visible in code.

For each SPEC bullet:
```
（説明でなく実際のコードを抜書きするように指示）

```
- Identify the concrete code fragment(s) implementing the SPEC item.
- Describe the implementation using the concrete code-level definition.
- Ground every claim in explicit code evidence (variables, expressions,
  or function calls) with line references.
```

“Critique” module = domain critique prompt+ critique prompt (175行)

```
The goal is to identify semantic inconsistencies between the SPEC and the code.
Use the TRACER OUTPUT as the primary source of code-level definitions. Critique prompt is
domain knowledge free.
```
(tracer の出力で判断)

Tracer output

SECTION 1: SPEC BULLET ANALYSIS (faithful extraction of task implementation)

✅ * Select taus with pT > 20 GeV and |eta| < 2.3; computed as: [t for t in get_taus(event) if t['pt'] > 20 and abs(t['eta']) < 2.3]; upstream chain: **get_taus**(event) -> [t for t in ...]; **(L263-L264)**

……

✅ * Select invariant mass of the leading and subleading jets (m_jj) > 400 GeV; computed as: m_jj > 400; upstream chain: jet1, jet2 -> **helper_invariant_mass**(jet1, jet2) -> m_jj; **(L284-L285)**

* jets

  - computed/defined as: [j for j in get_jets(event) if **j['pt'] > 30 and abs(j['eta']) < 4.7]**

  - upstream chain: get_jets(event) -> [j for j in …]

  - collection definition: j['pt'] > 30 and abs(j['eta']) < 4.7

  - structure: single-constituent

SPEC (仕様）の曖昧さ

* 数値計算コードの結果は一意
* LLMのよるコード生成: 言語モデルの初期化や“code-fixing stage”でててくる不定性でブレる
* さらに、曖昧なプロンプトが違う結果を呼び込む

—Require at least 2 jets with pT > 30 GeV and |eta| < 2.5

—-Select invariant mass of the two leading jets between 60 and 100 GeV
(W hadronic candidate)

→ A calculate mass for the jet **with η cut**

→ B calculate the mass for selected events **without η cut**

・どう修正するか

・小さい言語モデルを扱っているので、長いプロンプトを書くと、全体が崩れる可能性がある。

・注意深くかかないといけないなら、コードを自分で書いたほうがよくないか？

“Golden Axe Oracle”

曖昧なプロンプトの意味を2択で聞きにくるモジュール

[AMBIGUITY 2] Jet veto scope

Original phrase:

"Veto events with b-tagged jets"

Why this is dangerous:

The phrase does not clarify whether the veto applies to all jets in the event or only to the selected jets used in the dijet system.

Question to user:

Does the b-tagged jet veto apply to all jets in the event or only to the selected jets used in the dijet system?

Golden axe:

The veto applies to all jets in the event.

Silver axe:

The veto applies only to the selected jets used in the dijet system.

tips

小規模言語モデルは自由記述でなく、2択が良い

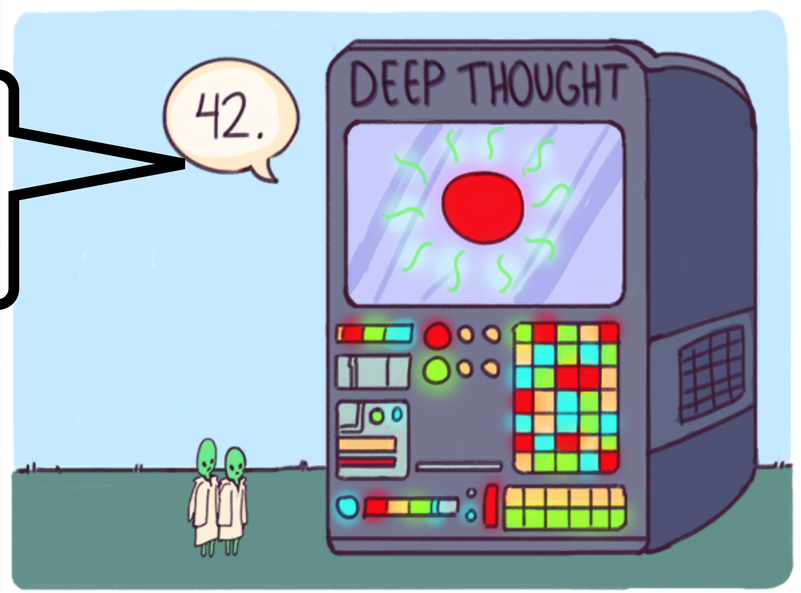
点検内容を指示する”oraclet” option はさらに安定

古典SF から学ぼう

Douglas Adams “ The Hitchhiker’s Guide to the Galaxy. “

Answer of the Ultimate question of life, the Universe, and Everything = 42

after 7.5 million years of calculation



Venkatesh Rao, Breaking Smart: How Software is Eating the World,
“A Tale of Two Computers,” illustration by/with Grace Witherell, 2015

1．中間アウトプットの必要性

2．人間とのコミュニケーションポイントの定義

‘I think the problem, to be quite honest with you, is that you’ve never actually known what the question is.’

‘So once you do know what the question actually is, you’ll know what the answer means.’

小さな言語モデルの声が聞きたい

手順を教えてあげれば、小さいモデルでも複雑なことができる。

重要なのは、「人間の知的作業」を手順として定義することと、人間の意図とのすり合わせする場所を定義すること。

AIが知的だな、って思ったら、知性の定義について、少し考えた方がいい。
小さいLLM の当てはめ問題＋作業手順
→より大きなLLM

次のステップは、状況に応じた作業手順の最適化