

大規模言語モデルを活用したソフトウェア開発の生産性向上に向けて



WillBooster株式会社 代表取締役

東京通信大学 准教授

早稲田大学 研究院客員准教授

坂本一憲 (exkazuu@willbooster.com)

2023年7月29日 @ DX交流フォーラム

自己紹介：坂本 一憲

- 所属・肩書
 - WillBooster株式会社 代表取締役, 東京通信大学 准教授, 早稲田大学 研究院客員准教授, リビングロボット株式会社 アドバイザ
 - スマートエスイー・DXで機械学習とビジネスインテリジェンスを担当
- 略歴
 - 早稲田大学 情報理工 鷺崎研究室にて学士・修士・博士号取得
 - 国立情報学研究所 助教およびベンチャー企業 CTOを経て起業
- プログラミングと最先端IT技術が大好き。最近はLLMに全力投球。

生成AI

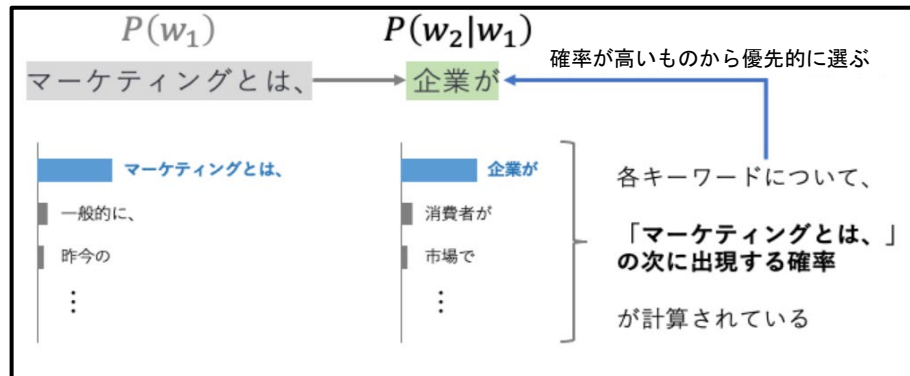
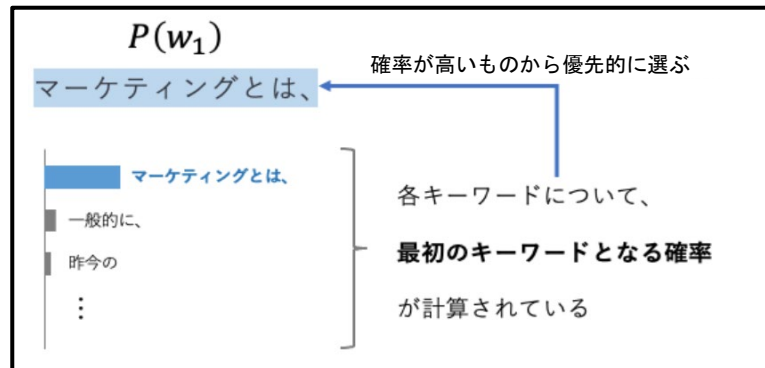
- 生成AI（Generative AI）とは、機械学習技術の1つであり、学習データから構築したモデルを用いて、オリジナルデータを生成するAIを指す。
- 一般に、生成AIは、様々なコンテンツを学習データとするディープラーニング関連の技術（例えば、Transformer）により構築された非常に大規模な機械学習モデルをベースとする。
- 生成AIには、テキスト生成や画像生成、動画生成、音声生成などさまざまな種類がある。代表的な製品として、テキスト生成AIの「ChatGPT」や「Bard」、画像生成AIの「Stable Diffusion」や「Midjourney」がある。

大規模言語モデル（LLM）

- 大規模言語モデル（Large Language Models; LLM）とは、ディープラーニング等の技術を用いて構築された言語モデルである。「大規模」とは、従来の自然言語モデルと比べ、「計算量」「データ量」「パラメータ数」を大幅に増やして構築されていることに由来している。
- 大規模言語モデルをファインチューニングなどすることによって、テキスト分類や感情分析、情報抽出、文章要約、テキスト生成、質問応答といった、さまざまな自然言語処理（Natural Language Processing; NLP）タスクに適応できる。
- 大規模言語モデルの代表例としては、2018年にGoogleが発表した「BERT」や、2020年にOpenAIが発表した「GPT-3」などが挙げられる。2022年12月に発表された「ChatGPT」は、チャット（対話）向けに改善した大規模言語モデルを活用したWebサービスであり、大規模言語モデルの応用例の一つである。

LLMの仕組み

LLMでは、あるトークン（LLMが扱う文字列の単位。人間が認識する単語などの単位とは異なる。）の次に発生する確率の高いトークンを繋げることで、回答を生成している。常に最も発生確率が高い回答を作ることできるが、通常はランダム要素を加えながら、確率が高いものを優先的に選ぶ。



ChatGPT

- 非営利法人OpenAI Inc.とその子会社である営利法人OpenAI LPが開発・提供するLLMをベースとしたWebサービス。
- LINEなどのようなメッセージングアプリのユーザインタフェースを備えており、対話形式でLLMから回答を得られる。
- 2023年7月現在、有償プランでは、GPT-3.5とGPT-4の2種類のLLMを選択して回答を得られる。また、有償プランではプラグインも利用できる。
 - GPT-4の回答速度は遅いが、既存のLLMの中で最も回答の質が高い。
- 本資料ではGPT-3.5とGPT-4の総称として「GPT」という表記を用いる。

国内外のLLM開発の動向

- 海外（モデル非公開）
 - GPT-4 (OpenAI社), Claude 2 (Anthropic社), PaLM 2 (Google社)
- 海外（モデル公開）
 - FreeWilly2 (Stability AI社), Llama 2 (Meta社), Falcon (TII)
 - Llama 2はGPT3.5-Turboに匹敵、FreeWilly2は一部で凌ぐ！
- 国内・日本語向け
 - NEC LLM (NEC社), OpenCALM (サイバーエージェント社), Japanese-GPT-NeoX (rinna社), Japanese-MPT-7B (Lightblue社)

プロンプト

- プロンプトは、LLMに与える入力文のこと。
- プロンプトの主要な構成要素
 - 文脈: タスクに関連する背景情報
 - タスク: 実行させたい具体的な作業
 - データ: タスクの実行に有益なデータ
 - フォーマット: 入力形式や出力形式
- 基本的に、回答に有益な情報を与えれば与えるほど、回答の質が上がる。
- 何が効くか分からないため、プロンプト改善のために試行錯誤が必須。



小学生にChatGPTの紹介をしたいです。そこで、10歳の子供でも理解できるように、200文字以内で以下の文章を要約してください。

ChatGPTは、OpenAIによって開発された最先端の自然言語処理（NLP）技術であるジェネレイティブ・プレトレーニング・トランスフォーマー（GPT）に基づくAIチャットボットです。自然言語の理解力と生成能力を備えたこのモデルは、ユーザーとの自然な対話を実現し、適切で関連性のある回答を提供するように設計されています。GPTは、より大規模で進化したモデルが開発されるにつれ、繰り返し改善されています。

まず、ChatGPTは、一般的なテキストデータと知識を大量に学習していることが特徴です。これにより、豊富な語彙と広範囲のトピックに対する理解力が向上しています。大規模なコーパスから学習した言語モデルは、豊富な知識と情報が含まれ、情報提供や会話の助けとなる多様な回答を生成することができます。

トークン

多くのLLMにはトークン数の単位で、入出力のサイズに上限が設けられている。

例えば、GPT-4では最大トークン数が32Kと8Kのモデルがある。Anthropic社のClaudeは100Kなど、トークン数は差別化に繋がる。

GPT-3 Codex

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens. <- spaces

親譲の無鉄砲で小供の時から損ばかりしている。小学校に居る時分学校の二階から飛び降りて一週間ほど腰こしを抜ぬかした事がある。なぜそんな無闇をしたと聞く人があるかも知れぬ。別段深い理由でもない。新築の二階から首を出していたら、同級生の一人が冗談に、いくら威張っても、そこから飛び降りる事は出来まい。

Clear

Show example

Tokens	Characters
281	409

概ね、英語は1単語で1トークンだが、日本語は1文字1トークン。なお、空白は1文字で1トークンなので要注意！

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens. <- spaces

????の????????で????の????から????かりしている。????に????る????????の????から????????りて一????????????こしを????かした????がある。な????んな????をしたと????く人があるかも知れぬ。????????い????でもない。????の????から????を????していたら、????生の一人が????に、いくら????っても、????から????????りる????は????まい。

TEXT

TOKEN IDS

ソフトウェア開発における プロンプトパターン (回答はGPT-3.5-Turboによる)

ソフトウェア開発のプロンプトパターン

- アメリカのVanderbilt大学のWhiteらが2つのパターン集を提案
 - A prompt pattern catalog to enhance prompt engineering with chatgpt [1]
 - ソフトウェア開発で活用できる汎用的なプロンプト集
 - Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design [2]
 - （主にWebアプリの）ソフトウェア開発に特化したプロンプト集
- 後者の論文のパターンを紹介してから、我々のノウハウを紹介する。
 - 例示するプロンプトは論文の内容を参考に、我々が改良している。

[1] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., ... & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. arXiv preprint arXiv:2302.11382.

[2] White, J., Hays, S., Fu, Q., Spencer-Smith, J., & Schmidt, D. C. (2023). Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv preprint arXiv:2303.07839.

パターンの一覧 (1/2)

カテゴリ	パターン名	本資料での呼び名	有用性
Requirements Elicitation	Requirements Simulator	要求仕様に基づくシミュレーション	◎
	Specification Disambiguation	要求仕様の明確化	△
	Change Request Simulation	変更のシミュレーション	○
System Design and Simulation	API Generator	API仕様書の生成	◎
	API Simulator	API仕様のシミュレーション	◎
	Few-shot Example Generator	利用例の生成	△
	Domain-Specific Language (DSL) Creation	ドメイン特化言語 (DSL) の作成	△
	Architectural Possibilities	アーキテクチャ候補の提案	◎

パターンの一覧 (2/2)

カテゴリ	パターン名	本資料での呼び名	有用性
Code Quality	Code Clustering	モジュール化を意識したコード生成	◎
	Intermediate Abstraction	抽象化を意識したコード生成	◎
	Principled Code	原則に則ったコード生成	○
	Hidden Assumptions	コードに隠された仮定の推論	◎
Refactoring	Pseudo-code Refactoring	擬似コードによるリファクタリング	○
	Data-guided Refactoring	データフォーマットの変更対応	◎

要求仕様に基づくシミュレーション

- パターン名 : Requirements Simulator Pattern
- 背景とモチベーション
 - 欠陥を早期に発見できればできるほど、修正工数を削減できる。
 - 要求に関する欠陥は、要件定義のフェーズにおいて検出したい。
- LLMによる解決
 - 要求の一覧を入力した上で、LLMに要求を満たすシステムになりきってもらい、特定のタスクが実現可能か否かを判定してもらう。
 - 実現できない場合、実現に必要な要求を提案してもらう。

要求仕様に基づくシミュレーションの例 (1/2)

入力

Requirements: ""

1. ユーザーは新規登録でき、自分のユーザーネームとパスワードを設定できる。
2. ユーザーは自分のユーザーネームとパスワードを使用してログインできる。
3. ログインしたユーザーは新しいツイートを作成できる。
4. ユーザーは自分のプロフィールページで自分のツイートを見ることができる。
5. ユーザーは他のユーザーのプロフィールページを訪れて、そのユーザーのツイートを見ることができる。
6. ユーザーは他のユーザーをフォローすることができる。
7. ユーザーは自分がフォローしているユーザーのツイートを自分のフィードで見ることができる。
8. ユーザーは他のユーザーのツイートを「いいね」することができる。

シミュレーション対象
の要求仕様

シミュレーション
の依頼文

Now, I want you to act as this system. Use the requirements to guide your behavior. I am going to say, I want to do X, and you will tell me if X is possible given the current requirements, or any additional requirements are needed. Then, choose one from the below:

- If X is possible based on the current requirements and no additional requirements are needed, provide a step-by-step set of instructions on how I would accomplish it and provide additional details that would help implement the requirement.
- If additional requirements are needed to do X, or there is possibility that any additional requirements are needed, write the missing requirements to make it possible as user stories.

Your output should be written in Japanese.

自分のツイートを削除したい。

実現可能かどうかを判定
するタスクの内容

要求仕様に基づくシミュレーションの例2 (2/2)

出力

追加の要件:

9. ユーザーは自分のツイートを削除できる。

手順:

1. 自分のユーザーネームとパスワードを使用してログインします。
2. プロフィールページに移動します。
3. 自分のツイート一覧を表示します。
4. 削除したいツイートを選択します。
5. ツイートの詳細ページに移動します。
6. ツイートの下にある「削除」ボタンをクリックします。
7. 確認メッセージが表示された場合は、削除を確定します。

注意事項:

- ツイートを削除すると、元に戻すことはできません。
- 削除したツイートは他のユーザーからは見えなくなりますが、システム上では完全に削除されるわけではありません。

API仕様書の生成

- パターン名 : API Generator Pattern
- 背景とモチベーション
 - API仕様書の作成は面倒である。コードから生成するツールは存在するが、実装してから仕様書を生成すると、仕様の検討が遅くなる。
- GPTによる解決
 - 要求の一覧を入力した上で、何らかのフォーマットを指定して、API仕様書の生成を依頼する。なお、本セミナーでは論文の例に倣い、OpenAPI SpecificationというREST APIの仕様記述言語を採用。

API仕様書の生成の例 (1/7)

入力

API仕様書生成の依頼文

Generate an OpenAPI specification for a web application that would implement the listed requirements.

Note that your answer should be written in Japanese.

API仕様書の生成対象
の要求仕様

Requirements: ""

1. ユーザーは140字以内の新しいツイートを作成できる。
2. ユーザーは自分のプロフィールページを開いて、自分の最新ツイート40件を見ることができる。
3. ユーザーは他のプロフィールページを開いて、そのユーザの最新ツイート20件を見ることができる。
4. ユーザーは他のユーザーのツイートを「いいね」することができる。
5. ユーザーは他のユーザーのツイートにおいて、自分が付与した「いいね」を解除できる。
6. ユーザーは自分のツイートを削除できる。

""

API仕様書の生成の例 (2/7)

出力 (1/5)

以下は、上記の要件を満たすWebアプリケーションのためのOpenAPI仕様です。

openapi: 3.0.0

info:

title: ツイートアプリケーションAPI

description: ユーザーがツイートを作成、表示、操作するためのAPI

version: 1.0.0

paths:

/tweets:

post:

summary: 新しいツイートを作成する

requestBody:

required: true

content:

application/json:

schema:

type: object

properties:

content:

type: string

maxLength: 140

required:

- content

responses:

'200':

description: ツイート作成成功

入力パラメータとして、contentという文字列型のプロパティを持つJSONを受け取る。

ツイート作成時に、作成したツイートのIDも返してほしいが、返していない。後ほど修正を依頼する。

API仕様書の生成の例 (3/7)

出力 (2/5)

```
/users/{userId}/profile:
  get:
    summary: ユーザーのプロフィールページを表示する
    parameters:
      - name: userId
        in: path
        description: プロフィールを表示するユーザーのID
        required: true
        schema:
          type: string
    responses:
      '200':
        description: プロフィール表示成功
        content:
          application/json:
            schema:
              type: object
              properties:
                username:
                  type: string
                latestTweets:
                  type: array
                  items:
                    type: object
                    properties:
                      content:
                        type: string
                      timestamp:
                        type: string
                        format: date-time
```

API仕様書の生成の例 (4/7)

出力 (3/5)

```
/users/{userId}/tweets:
  get:
    summary: ユーザーの最新ツイートを取得する
    parameters:
      - name: userId
        in: path
        description: 最新ツイートを取得するユーザーのID
        required: true
        schema:
          type: string
    responses:
      '200':
        description: 最新ツイート取得成功
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  content:
                    type: string
                  timestamp:
                    type: string
                    format: date-time
```

API仕様書の生成の例 (5/7)

出力 (4/5)

```
/tweets/{tweetId}/like:
  post:
    summary: ツイートを「いいね」する
    parameters:
      - name: tweetId
        in: path
        description: 「いいね」する対象のツイートのID
        required: true
        schema:
          type: string
    responses:
      '200':
        description: 「いいね」成功

/tweets/{tweetId}/like:
  delete:
    summary: ツイートの「いいね」を解除する
    parameters:
      - name: tweetId
        in: path
        description: 「いいね」を解除する対象のツイートのID
        required: true
        schema:
          type: string
    responses:
      '200':
        description: 「いいね」解除成功
```

API仕様書の生成の例 (6/7)

出力 (5/5)

URLのパスの「{tweetId}」は入力パラメータになっており、ツイートIDを埋め込む必要がある。

```
/tweets/{tweetId}:  
delete:
```

summary: ツイートを削除する

parameters:

- name: tweetId

in: path

description: 削除するツイートのID

required: true

schema:

type: string

responses:

'200':

description: ツイート削除成功

入力パラメータの仕様

このOpenAPI仕様は、ユーザーが新しいツイートを作成し、プロフィールページを表示し、他のユーザーのツイートを表示し、ツイートを「いいね」し、ツイートを削除するためのエンドポイントを提供します。

API仕様書の生成の例 (7/7)

先程の出力で得られたAPI仕様書では、ツイート作成時にツイートIDを返していなかった。IDを返してくれるように、対話の続きとして入力を与える。

入力

ツイートを作成した際に、ツイートのIDを返すように修正してください。

出力

以下は、修正を加えたOpenAPI仕様です。

```
openapi: 3.0.0
info:
  title: ツイートアプリケーションAPI
  description: ユーザーがツイートを作成、表示、操作するためのAPI
  version: 1.0.0
```

... 以下略 ...

API仕様に基づくシミュレーション

- パターン名 : API Simulator Pattern
- 背景とモチベーション
 - API仕様書に欠陥が含まれている場合、できる限り早期に検出して修正したい。しかし、実際に動かさずに欠陥を見つけることは困難。
- GPTによる解決
 - 先程のパターンで生成したAPI仕様書を入力して、GPTにAPIになりきってもらい、APIの動作のシミュレーションを依頼する。シミュレーションを通して、仕様書の欠陥や抜け漏れを見つける。

API仕様に基づくシミュレーションの例 (1/3)

新しく対話を始めても良いが、長いAPI仕様書を入力する手間を省くため、API仕様書の生成の例で示した対話の続きとして入力を与える。

入力

API仕様に基づいたシミュレーションの依頼文

Act as this web application based on the OpenAPI specification. I will type in HTTP requests in plain text and you will respond with the appropriate HTTP response based on the OpenAPI specification.

```
POST /tweets  
{ "content": "hello" }
```

シミュレートしたい
リクエストの1個目。

API仕様に基づくシミュレーションの例 (2/3)

出力

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "tweetId": "1234567890"
}
```

API仕様に基づくシミュレーションの例 (3/3)

入力

```
DELETE /tweets/1234567890
```

出力

```
HTTP/1.1 200 OK
```

入力

```
DELETE /tweets/1234567890
```

出力

```
HTTP/1.1 404 Not Found
```

同じツイートを二回削除しようとする、存在しないことを示すレスポンスを返してくれる。

実は、404のレスポンスは仕様書に載っていないが、404を返した方がより適切なAPIと言える。

パターン以外の活用事例 (断りがないものはGPT-3.5-Turbo)

パターン以外の有用な活用事例

- 要件定義・設計関連
 - ペルソナ作成・典型的な要求の洗い出し・ドメインモデリングなど
- 実装・リファクタリング
 - 基本的には素直に依頼すれば良いが、社内での経験上ノウハウがある
- テスト
 - テストは特に工夫しなくとも、GPTにテスト対象のコードを与えて、テストコードの生成を依頼すれば良い。テスト生成はツール化しやすいため、今後のツール化の期待が読み取れる研究を 1 件紹介する。

ドメインモデリング

- 背景とモチベーション

- ソフトウェア開発の手法の一つであるICONIXでは、要求仕様書から名詞および名詞句を抽出して、抽出したものからドメインモデルの構成要素を選び出し、モデルを設計するが、作業に時間がかかる。

- GPTによる解決

- 名詞および名詞句の抽出や、抽出されたモデルの要素の候補を選定する作業、ドメインモデルを設計してクラス図を作図する作業などを、GPTに依頼する。得られた成果物を叩き台として活用する。

ドメインモデリングの例 (1/8)

入力

以下の要求仕様から、名詞および名詞句を抽出してください。

Requirements: ""

1. ユーザーは、新しいタスクを作成する。
ユーザーは、タイトル、説明、期日、担当者を指定してタスクを作成できる。
タスクの作成時は、タスクの状態は「未着手」である。
タスクはデータベースに保存され、一意の識別子が割り当てられる。
2. ユーザーは、作成したタスクの一覧を表示する。
ユーザーは、全タスクの一覧を表示できる。
一覧にはタスクのタイトルと状態（未着手、進行中、完了）が表示される。
3. ユーザーは、タスクの詳細を編集する。
ユーザーは、タスクのタイトル、説明、期日、担当者を編集できる。
4. ユーザーは、不要なタスクを削除する。
ユーザーは、タスクを削除できる。
削除されたタスクはデータベースから完全に削除される。
5. ユーザーは、タスクの状態を変更する。
ユーザーは、タスクの状態（未着手、進行中、完了）を変更できる。
6. ユーザーは、タスクにコメントを追加する。
ユーザーは、タスクにコメントをテキストで追加できる。
コメントには作成者、作成日時、本文の情報が含まれる。
コメントはタスクの詳細ビューに表示され、古いものから順に表示される。

""

ドメインモデリングの例 (2/8)

出力

名詞および名詞句:

- ユーザー
- タスク
- タイトル
- 説明
- 期日
- 担当者
- 状態
- 未着手
- データベース
- 一意の識別子
- 一覧
- 進行中

- 完了
- 詳細
- 編集
- 不要なタスク
- 削除
- 状態変更
- コメント
- テキスト
- 作成者
- 作成日時
- 本文
- 情報
- 詳細ビュー
- 古いもの

ドメインモデリングの例 (3/8)

先程の対話の続きとして入力する。

入力

与えられた要求仕様と、抽出した名詞および名詞句をもとにして、ドメインモデルを設計したいです。概念の類似する項目を統合したり、ドメインモデルで扱う必要のない項目を削除したりしてください。

出力 (1/2)

以下は、与えられた要求仕様と抽出した名詞および名詞句をもとに作成されたドメインモデルの一例です。概念の類似する項目を統合し、ドメインモデルで扱う必要のない項目を削除しました。

Task (タスク)

- タスクID (一意の識別子)
- タイトル
- 説明
- 期日
- 担当者
- 状態 (未着手、進行中、完了)

ドメインモデリングの例 (4/8)

出力 (2/2)

Database (データベース)

- タスクの保存と取得を担当

User (ユーザー)

- タスクの作成、一覧表示、編集、削除、状態変更、コメント追加を行う

Comment (コメント)

- タスクへのコメント
- 作成者
- 作成日時
- 本文

以上のドメインモデルは、与えられた要求仕様を基にしてタスク管理システムの基本的な機能を表現しています。必要に応じて、モデルを追加、変更、拡張することができます。

ドメインモデリングの例 (5/8)

先程の対話の続きとして入力する。

入力

整理した項目を使って、ドメインモデルのクラス図を作成してください。クラス図は、PlantUMLの記法で表現してください。

出力 (1/3)

以下は、与えられた項目を使って作成されたドメインモデルのクラス図です。クラス図はPlantUMLの記法で表現しています。

@startuml

```
class Task {  
  - taskId: int  
  - title: string  
  - description: string  
  - dueDate: date  
  - assignee: string  
  - status: string  
  + createTask(): void  
  + getTaskList(): List<Task>  
  + editTask(): void  
  + deleteTask(): void  
  + changeStatus(): void  
  + addComment(): void  
}
```

ドメインモデリングの例 (6/8)

出力 (2/3)

```
class Database {
+ save(task: Task): void
+ get(taskId: int): Task
+ delete(taskId: int): void
}

class User {
+ createTask(): void
+ getTaskList(): List<Task>
+ editTask(): void
+ deleteTask(): void
+ changeStatus(): void
+ addComment(): void
}

class Comment {
- taskId: int
- creator: string
- createdAt: date
- content: string
}

Task "1" *-- "1" Database
Task "1" *-- "*" Comment
User "1" o-- "1" Task

@enduml
```

ドメインモデリングの例 (7/8)

本来、ドメインモデルではメソッドまでは扱わないが、削除すれば良い。基本的に情報が多い分には困らない。

出力 (3/3)

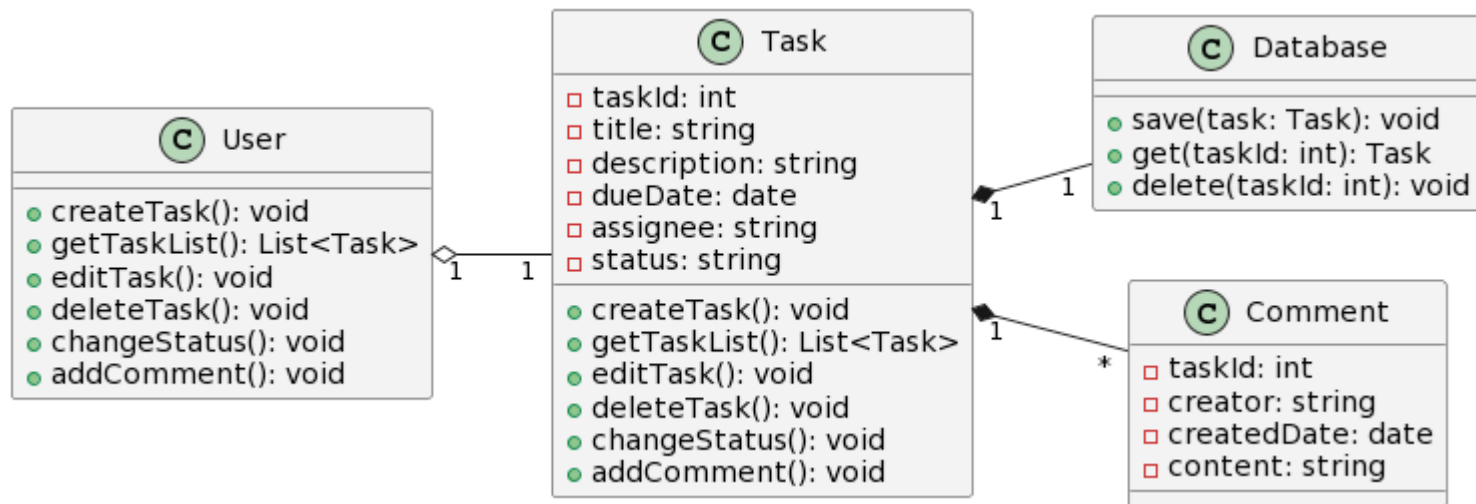
このクラス図では、Taskクラスがタスクの属性や操作を表しています。Databaseクラスはタスクの保存と取得を担当しており、Userクラスはタスクの作成、一覧表示、編集、削除、状態変更、コメント追加を行います。Commentクラスはタスクへのコメントを表しています。

各クラスの関係は、関連線で表現されています。TaskクラスとDatabaseクラスの関係は、1対1の関連です。TaskクラスとCommentクラスの関係は、1対多の関連です。また、UserクラスとTaskクラスの関係は、1対多の関連です。

このクラス図は、タスク管理システムの基本的な機能を表現していますが、必要に応じて追加のクラスや関連を追加することができます。

ドメインモデリングの例 (8/8)

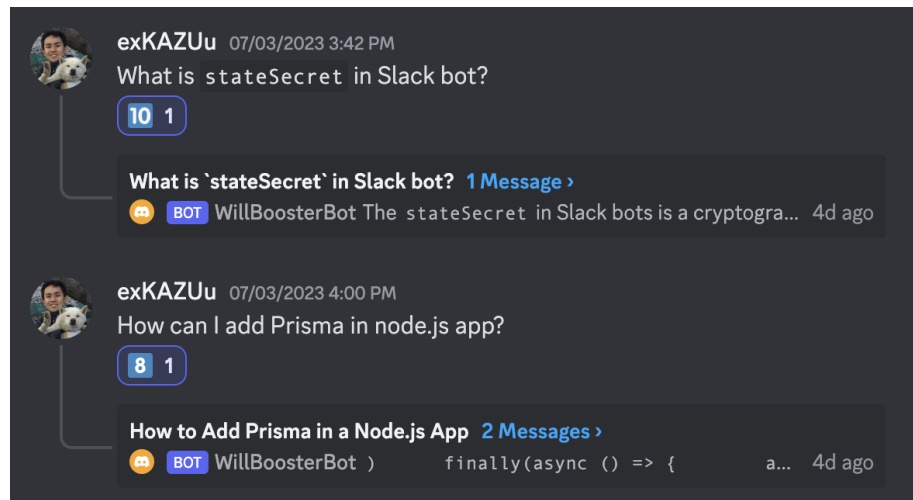
- 生成されたPlantUML記述を <https://www.plantuml.com/> で入力すると、以下のクラス図を作図できる。なお、レイアウトのみ微調整した。



実装・テストにおけるノウハウ (1/2)

以降は、坂本およびWillBooster社のエンジニアの体験を踏まえて、有用なノウハウを説明する。我々は日常的にGPT-4を使っているため、以降で掲載する例は全てGPT-4の回答である。

なお、右図のように、GPT-4の回答に対して絵文字で満足度を記録している。2-3日に一度、3名のエンジニア間で自身の経験を共有する時間を設けている。



実装・テストにおけるノウハウ (2/2)

- できる限り豊富な文脈を与える。
- コードを介して文脈を与える。
- 設定ファイルを介して文脈を与える。
- キーワードで文脈を与える。
- 断片的でも知識を与える。
- 正誤確認を前提に知識を検索する。
- 成果物に対して質問する。

コードを介して文脈を与える

- GPTは便利であるが、GPTにコードなどの生成を依頼するプロンプトを記述することに、一定のコストを支払う必要がある。
 - プロンプトを記述するコストが、コードを記述するコストよりも小さくなければ、GPTを使う意味合いは弱まる！
- 既にコードがあり、それに関連するタスクをGPTに依頼したいとき、わざわざ仕様を自然言語で説明せずとも、コードを与えて説明に代えることができる。
 - プロンプト入力の手間を減らすことで、より生産性を高められる！

コードを介して文脈を与える例

実装済みのWebアプリをテストするため、PageObjectModelパターンを採用したテストコードを実装したい。LoginPageのテストコードは実装済みなので、他のページのテストコードの実装をGPTに依頼する。

PageObjectModelとは

ページ（またはページの一部）を一つのクラスとして表現する。ページクラスのオブジェクトを介して、ページ内の各要素（ボタン、テキストボックスなど）を取得したり、各要素に対する操作（クリック、テキストを入力など）を実行できるようにするデザインパターン。



LoginPageのクラス

```
class LoginPage {  
    constructor(page) {  
        this.page = page;  
    }  
    async navigate() {  
        await this.page.goto("https://example.com/login");  
    }  
    async enterUsername(username) {  
        await this.page.fill("#usernameInput", username);  
    }  
    async enterPassword(password) {  
        await this.page.fill("#passwordInput", password);  
    }  
    async clickLoginButton() {  
        await this.page.click("#loginButton");  
    }  
}
```

PageObjectModelは、テスト対象とテスト間に一対一の関係があるため生成しやすい。

あなたは、TypeScript と playwright を用いて、PageObjectModel のコードを生成するシステムです。
以下であるウェブサイトのページを実装した blitz.js のソースコードをあなたに渡します。あなたはそのソースコードに対応する PageObjectModel のコードを返信してください。

入力

```
````ts
import type { BlitzPage } from '@blitzjs/next';
import { Routes, useParam } from '@blitzjs/next';

.....

const TargetPage: BlitzPage = () => {
 const id = useParam('id', 'number');
 const [value] = useQuery(getValue, { id });

 return (
 <>
 <PageBreadcrumbs
 items={[
 { label: 'ダッシュボード', href: Routes.HomePage() },

]}
 />
 <Card mb={6} py={2}>
 <TableContainer>
 <Table>
 <Thead>
 <Tr>

 </Tr>
 </Thead>
 </Table>
 </TableContainer>
 </Card>
 </>
);
};
````
```

出力

あなたの Blitz.js ソースコードを理解しました。

以下に対応する Page Object Model (POM) の TypeScript ファイルを提供します :

```
````ts
import { Page } from 'playwright'

export class TargetPage {
 private page: Page

 constructor(page: Page) {
 this.page = page
 }

 // 画面要素のセレクト定義
 private selectors = {
 breadcrumbs: '[data-testid="breadcrumbs"]',

 }

 async getBreadcrumbs() {
 return this.page.$$ (this.selectors.breadcrumbs)
 }

}
````
```

社内のコーディング規約を満たさない場合、One-shotで形式を整えることもできる。

[先程のプロンプトに以下を付け加える]

入力

参考として、作成済みの PageObjectModel のソースコードを以下に添付します。

```
````ts
import type { Locator, Page } from '@playwright/test';
import { PageBase } from '../pageBase';
export class SamplePage extends PageBase {
 static async create(page: Page): Promise<SamplePage> {
 const newPage = new SamplePage(page);
 await newPage.breadcrumb.waitFor();
 return newPage;
 }
 readonly breadcrumb: Locator;

 constructor(page: Page) {
 super(page);
 this.breadcrumb = page.getByText('ダッシュボード');
 }

 async openEditPage(): Promise<EditPage> {
 await this.editButton.click();
 return await EditPage.create(this.page);
 }

}
```

以下に、あなたが受け取っている blitz.js のソースコード  
に対応する pageObjectModel のソースコードを示します。

出力

```
````ts
import type { Locator, Page } from '@playwright/test';
import { PageBase } from '../pageBase';
.....
export class TargetPage extends PageBase {
  static async create(page: Page): Promise<TargetPage> {
    const newPage = new TargetPage(page);
    await newPage.breadcrumb.waitFor();
    return newPage;
  }
  readonly sessionsListButton: Locator;
  .....
  constructor(page: Page) {
    super(page);
    this.sessionsListButton = page.getByRole('link', {
name: 'セッション一覧' });
  }
  .....
  async openSessionsPage(): Promise<SessionsPage> {
    await this.sessionsListButton.click();
    return await SessionsPage.create(this.page);
  }
  .....
}
```

設定ファイルを介して文脈を与える

GPTにコードを生成させる際、導入済みのライブラリを優先的に使いたい。

しかし、導入済みライブラリの確認やプロンプトでの指定に手間がかかる。

入力

以下のような形式のCSVデータが文字列として与えられます。

TypeScriptでCSVデータを配列に変換するコードを書いてください。

必要に応じて使用中のパッケージを使用してください。

<変換対象のCSVファイルと依存ライブラリの設定ファイル（package.json）を添付>

出力（TypeScriptのコード）

```
// csv-parse パッケージのインポート
import parse from 'csv-parse';
// [具体的な実装]
```

WillBooster社の内製Botでは、ファイルをアップロードすると、自動的にファイルの内容がプロンプトに挿入される仕組みを有している。そのため、ファイルの方がプロンプトの手入力よりも簡単。

LLMベースのテスト生成研究

ChatUniTest: a ChatGPT-based automated unit test generation tool

- ChatGPT (GPT-3.5-Turboモデル) で、テストコードを生成する研究。
- GPTは入力可能なトークン数に制限があり、また、動作しないテストコードを生成してしまう問題がある。
- 提案手法では、トークン数の限界ぎりぎりまで、テストコード生成に有用な文脈情報を与え、また、テストコードが動作しない場合に、ルールベースな手法とGPTの両方を活用して、テストコードを自動修正する。
- 最先端の既存ツールとしてEvoSuitを比較対象に採用している。

提案手法の概要

提案手法はJava言語を対象としており、JUnit 5とMockito 3を使用。

1. テストコード生成の必須情報（テスト対象のメソッド全体、対象メソッドが含まれるクラスのシグネチャ・コンストラクタのシグネチャ）を生成
2. GPTのトークン数の制限に収まる範囲で、できる限り多くの文脈情報（対象メソッドが呼び出す他のメソッドのシグネチャ、対象メソッドが含まれるクラス内のフィールド、対象メソッドが依存するクラスなど）を生成
3. プロンプトに文脈情報を加えて、GPTでテストコードを生成
4. ルールベースとGPTの両方でコンパイルエラー・実行時エラーに対処

実験対象のJavaプロジェクト

- 上部5プロジェクトは2021年12月1日以前から存在するが、下部5プロジェクトは存在しなかったため、GPTの学習対象外であることが保証される。

| Project | Abbr. | Domain | Version | Trained | # Methods |
|---|---------------|--------------------|---------|---------|-----------|
| Commons-Lang [20] | Lang | Utility | 3.12.0 | yes | 3018 |
| Commons-Cli [21] | Cli | Cmd-line interface | 1.5.0 | yes | 250 |
| Commons-Csv [22] | Csv | Data processing | 1.10.0 | yes | 209 |
| Gson [23] | Gson | Serialization | 2.10.1 | yes | 660 |
| Jfreechart [24] | Chart | Visualization | 1.5.4 | yes | 8340 |
| Ecommerce-microservice-backend-app [25] | Ecommerce | Microservices | 695a6d4 | no | 316 |
| Datafaker [26] | Datafaker | Data generation | 1.9.0 | no | 1766 |
| Flink-kubernetes-operator [27] | Flink-k8s-opr | Cloud computing | 1.4.0 | no | 844 |
| Binance-connector-java [28] | Binance-conn | API wrapper | 2.0.0 | no | 474 |
| Event-ruler [29] | Event-ruler | Event Engine | 1.2.1 | no | 485 |
| Total | | | | | 16362 |

EvoSuiteとの比較

- 古くから存在するプロジェクトに関しては、提案手法が優れている。
- 学習対象外の5つについても、行網羅で優位。
- 下5つに限定した平均は
分岐: 90.14% vs. 81.38%
行: 91.77% vs. 93.47%

| Projects | Branch Coverage | | Line Coverage | |
|---------------|-----------------|---------------|---------------|---------------|
| | EvoSuite | Our | EvoSuite | Our |
| Lang | 84.92% | 94.70% | 84.71% | 91.94% |
| Cli | 90.90% | 96.36% | 90.93% | 93.52% |
| Csv | 75.87% | 84.09% | 70.28% | 86.61% |
| Gson | 59.23% | 83.88% | 61.53% | 86.80% |
| Chart | 87.67% | 88.27% | 85.90% | 84.03% |
| Ecommerce | 100% | 100% | 89.58% | 96.35% |
| Datafaker | 91.12% | 89.76% | 58.55% | 86.07% |
| Flink-k8s-opr | 89.72% | 78.76% | 87.14% | 83.02% |
| Binnance-conn | 98.72% | 97.17% | 87.59% | 97.87% |
| Event-ruler | 87.78% | 93.14% | 84.02% | 87.40% |
| Average | 86.59% | 90.61% | 80.02% | 89.36% |

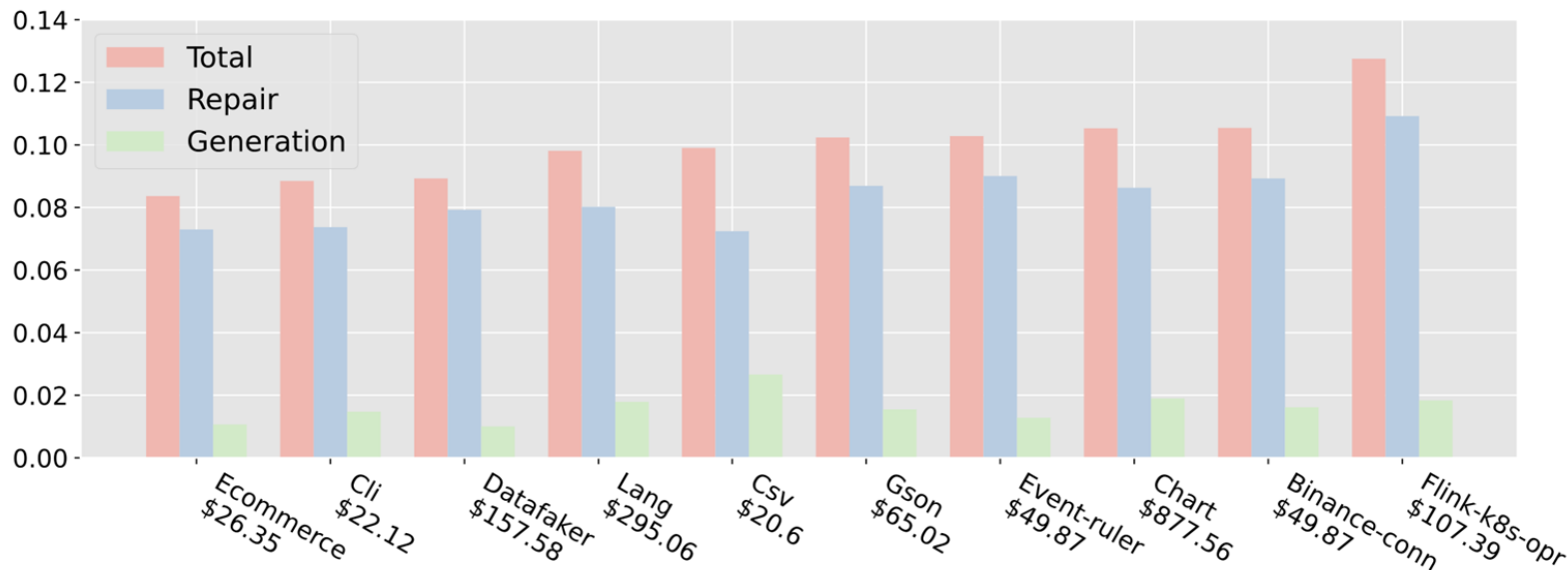
言語モデルベースの従来手法との比較

- 2023年に発表されたA3Testと2020年に発表されたAthenaTestと比較
- 動作するテスト数とテストを生成できたメソッドの割合の両方で優れる。

| Projects | Correct Test Cases (%) | | | Focal Method Coverage (%) | | |
|----------|------------------------|------------|---------------|---------------------------|------------|---------------|
| | A3Test | AthenaTest | ChatUniTest | A3Test | AthenaTest | ChatUniTest |
| Gson | 14.09% | 2.89% | 23.30% | 40.90% | 9.54% | 55.91% |
| Cli | 25.19% | 11.07% | 38.77% | 37.20% | 29.46% | 70.34% |
| Csv | 25.73% | 8.98% | 44.26% | 37.80% | 34.31% | 76.94% |
| Chart | 31.30% | 11.70% | 39.02% | 34.40% | 32.00% | 79.56% |
| Lang | 49.50% | 23.35% | 39.85% | 58.30% | 56.97% | 84.05% |
| Total | 40.05% | 16.21% | 40.14% | 46.80% | 43.75% | 79.67% |

提案手法が費やした金銭的成本

- 当時のgpt-3.5-turboモデルは1000トークンごとに\$0.002（今は入力のみ25%OFF）。最大でも1メソッドあたり\$0.13弱。高い？安い？



- Xie, Z., Chen, Y., Zhi, C., Deng, S., & Yin, J. (2023). ChatUniTest: a ChatGPT-based automated unit test generation tool. arXiv preprint arXiv:2305.04764.

論文から読み取れること

- GPTモデルの学習時から存在するプロジェクトでは、従来技術よりも高いカバレッジのテストを生成できる。学習時に存在しないプロジェクトであっても、行カバレッジにおいてはEvoSuiteよりも優れている。
 - 当然ではあるが、従来技術とGPTモデルで生成したテストを混ぜ合わせれば、確実に従来よりも良いテストを生成できる！
- Chartプロジェクトでは、10万円（\$878）程度の費用がかかった・・・。
- 先程の研究も本研究もGPT 3.5 Turboが利用されているが、最新モデルであるGPT 4を用いれば、より一層の性能向上が期待できる！（費用も。）

まとめ

- ChatGPTのプロンプトパターン
 - Whiteらが提唱するプロンプトパターンを踏まえた活用事例の紹介
- パターン以外の活用事例
 - プロンプトパターンに含まれていない活用事例の紹介
- LLMベースのテスト生成研究
 - LLMを活用した最先端のテスト生成研究の紹介
- WillBooster社では、LLMを活用した研究開発等のパートナーを探索中
 - ご興味のある方は exkazuu@willbooster.com までご連絡ください！