

CoLadder: Manipulating Code Generation via Multi-Level Blocks

Ryan Yen, Jiawen Zhu, Sangho Suh, Haijun Xia, Jian Zhao
2024/10

UIST 2024

研究の概要

- | コーディングにおいてLLMにフラットなテキストで指示を出す現状からの脱却
- | 階層的なブロックで意図を組み立てるGUI（CoLadder）を提供する
- | 生成コードの制御性とプログラマーの理解度を向上させることを目的としている

背景：LLMによる開発の弊害

従来のLLMアシスタントでは、プログラマーは「どんな手順で実現するか」という意図を形成するプロセスを飛ばしがちになる

その結果、生成された巨大なコードを検証する段階で認知負荷が増大し、制御の喪失に繋がる

課題

言語化の限界

フラットなテキストプロンプトでは、人間の頭の中にある「タスクの構造」を正しく表現できない

プロンプトとコードの対応

大量のコードが一気に出力されるため、「プロンプトのどの部分が、コードのどこに対応しているか」を追跡できない

COLADDER: デザインプロセス

理解とアイデア

熟練プログラマー6人によるインタビューや実際のLLMを利用した開発における戦略を確認

プロトタイプ

デザイン目標の確立とフィードバック、認知ウォークスルーに基づく反復開発

展開と評価

有用性のユーザー評価

インタビューによる知見と目標

課題	戦略	設計目標
C1: 意図を外部化するための非構造化プロンプト	S1: タスク分解	DG1: 階層的プロンプト構造
	S2: 階層構造	
C2: 意図からコードへの制御損失	S3: 増分生成	DG2: コード修正のためのプロンプトの直接操作
	S4: インシチュ生成	
	S5: コードセグメントの再配置	
	S6: コードセグメントの置換	
C3: 破壊的コンテキスト切り替え	S7: プロンプト作成中の結果評価	DG3: プロンプト作成中のコード評価の有効化
C4: プロンプトからコードへの不明確な対応	S8: プロンプトにコードを追加	DG4: プロンプトコードの強化対応
	S9: コメントによる評価	

提案システム

CoLadder

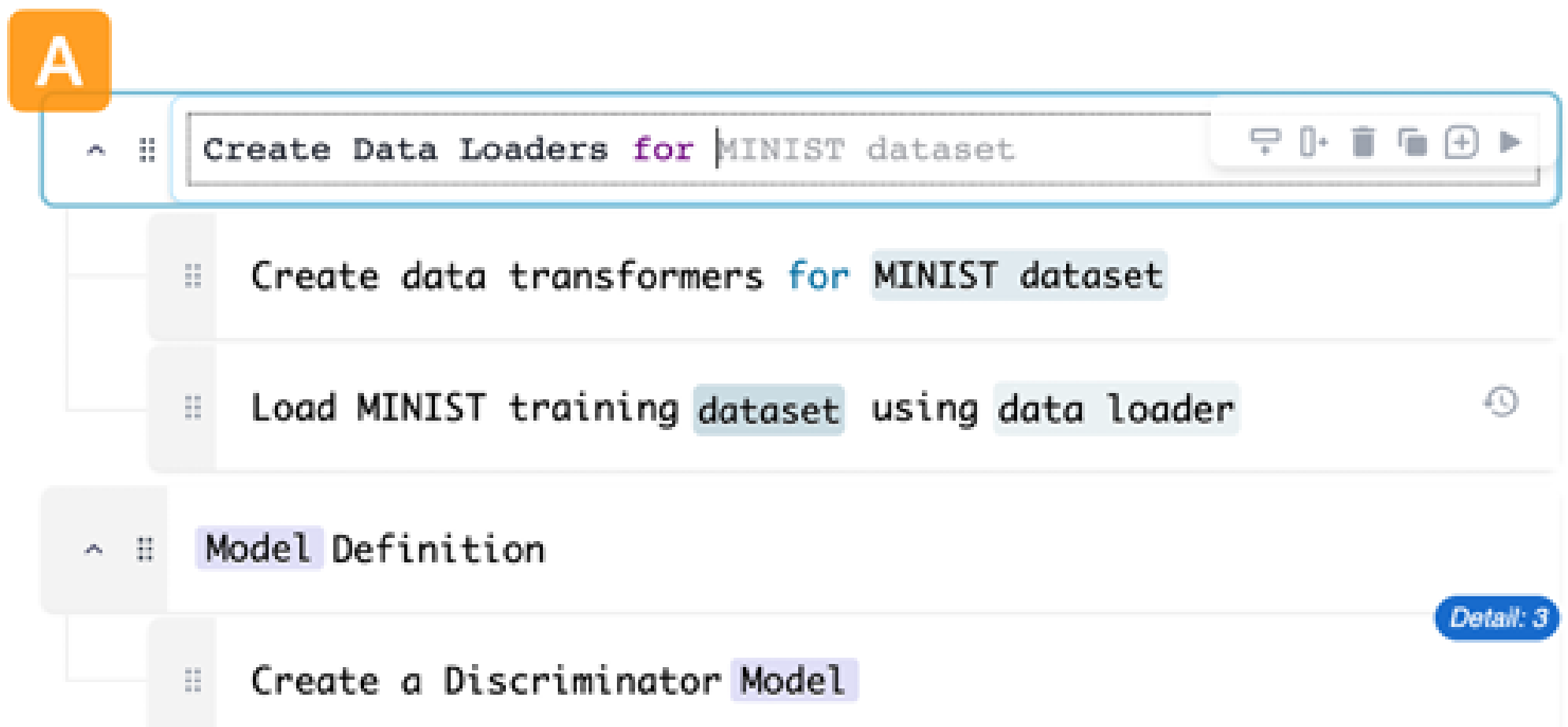
階層的なタスク分解、直接的なコードセグメント操作、プロンプト作成中の結果評価を促進することでプログラマーを支援するシステム

構成コンポーネント

- A. プロンプトツリーエディタ
- B. コードエディタ
- C. プロンプトブロック
- D. 実行結果パネル

A: プロンプトツリーエディタ

プログラマーがプログラミングタスクを小さなプロンプトブロックに分解することで、自分の意図を外部化する



C:プロンプトブロック

プログラマーが一つのプロンプトブロックに混合モードでプロンプトを作成できるようにし、コードと自然言語の両方を組み込める

C

```
nn.Sequential
  for _ in range(3):
    ConvTranspose2d
    BatchNorm2d
    ReLu
```

segmenting code...

B:コードエディタ

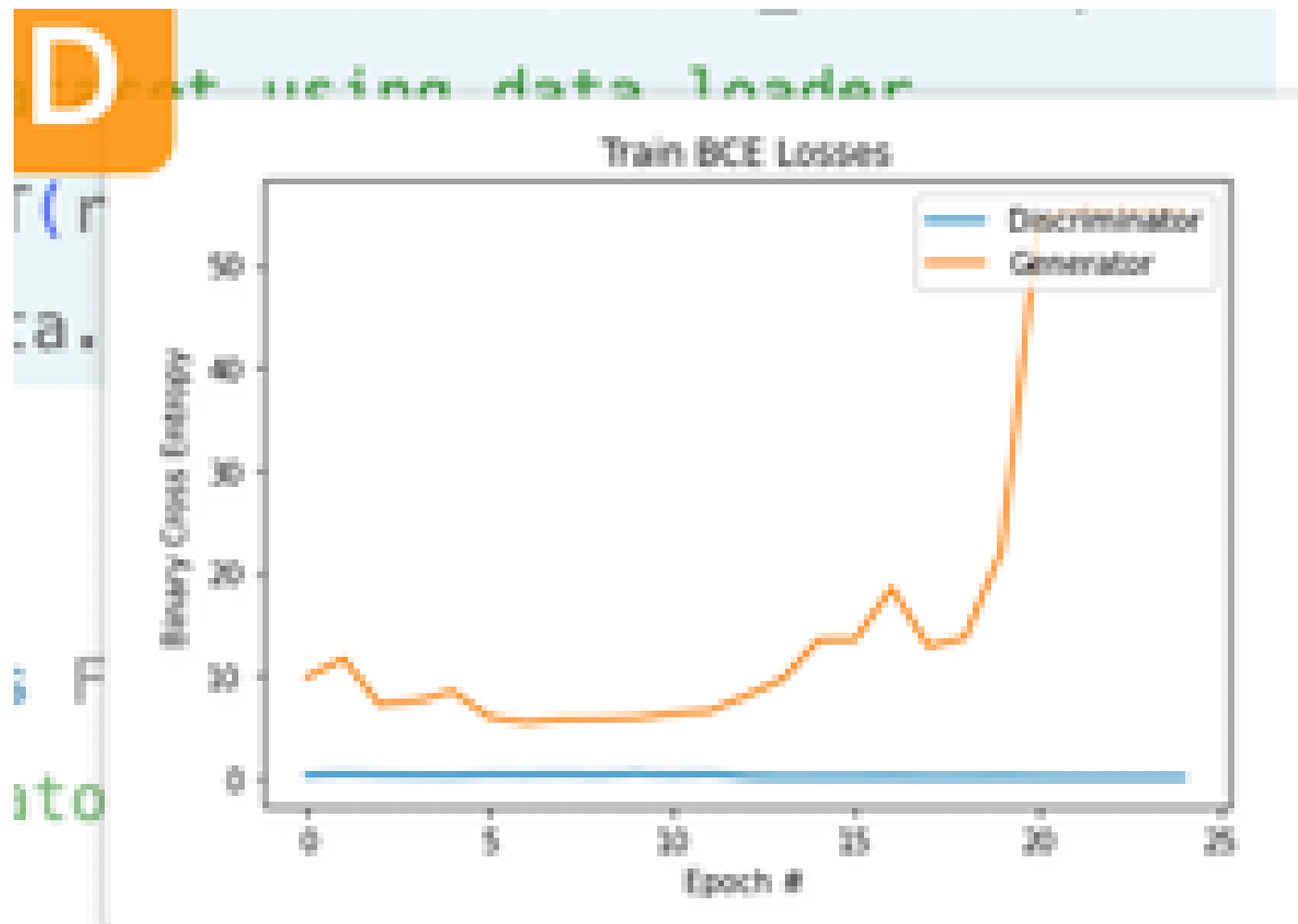
プログラマーが生成されたコードを評価し、プログラムを直接編集できるようにします



```
1 # [ID: 1] Create Data Loaders for MNIST dataset
2 import torch
3 from torchvision import datasets, transforms
4 # [ID: 2] Create data transformers for MNIST dataset
5 transform = transforms.Compose([
6     transforms.ToTensor(),
7     transforms.Normalize((0.5,), (0.5,))
8 ])
```

D:実行結果パネル

プログラマーは現在のコードをコンパイルして実行し、コードエディタの下に結果やエラーを表示することもできます



システム画面の全体像

A

Create Data Loaders for MINIST dataset

Create data transformers for MINIST dataset

Load MINIST training dataset using data loader

Model Definition

Create a Discriminator Model

C

```
nn.Sequential
  for _ in range(3):
    ConvTranspose2d
    BatchNorm2d
    ReLu
```

segmenting code...

display the sam... load the MINIST ... load the MINIST ... create a Dis

B

Run Reset

```
1 # [ID: 1] Create Data Loaders for MINIST dataset
2 import torch
3 from torchvision import datasets, transforms
4 # [ID: 2] Create data transformers for MINIST dataset
5 transform = transforms.Compose([
6     transforms.ToTensor(),
7     transforms.Normalize((0.5,), (0.5,))
8 ])
9 # [ID: 3] Load MINIST training dataset using data loader
10 train_dataset = datasets.MNIST(root='./data', train=True, downl
11 train_loader = torch.utils.data.DataLoader(train_dataset, batch
12 # [ID: 4] Load MINIST test dataset using data loader
13 test_dataset = datasets.MNIST(r
14 test_loader = torch.utils.data.
15 # [ID: 5] Model Definition
16 import torch.nn as nn
17 import torch.nn.functional as F
18 > # [ID: 6] Create a Discriminator
```

D

Train BCE Losses

Epoch #	Discriminator Loss	Generator Loss
0	~10	~10
5	~10	~10
10	~10	~10
15	~10	~10
20	~10	~10
25	~10	~45

各ブロックの機能



A. 補足ビュー

会話型UIを通じてプロンプトに追加の詳細を加える

B. 履歴の差分ビュー

変更の種類、プロンプトの反復、およびそれに対応する生成されたコードを観察できる

C. 依存関係の表示

プロンプト内のフレーズを選択すると、システムはツリー構造全体とコードセグメント内の相関するフレーズをハイライトする

システムアーキテクチャ

ベースフレームワーク: Next.js

UIの描画と、APIでのLLM呼び出しを統合して管理

AIエンジン: OpenAI GPT-4 API

階層的なプロンプトツリーを解析し、コードの生成と構造化

UI / エディタ層: Monaco Editor

右側の「メインコード画面」だけでなく、左側の「各プロンプトブロック」の中身にも採用

コード実行環境: Pyodide (Python環境)

ブラウザ上でPythonを直接動かすためのサンドボックス環境

評価設定

対象ユーザ

経験豊富な12人のプログラマー

ベースライン

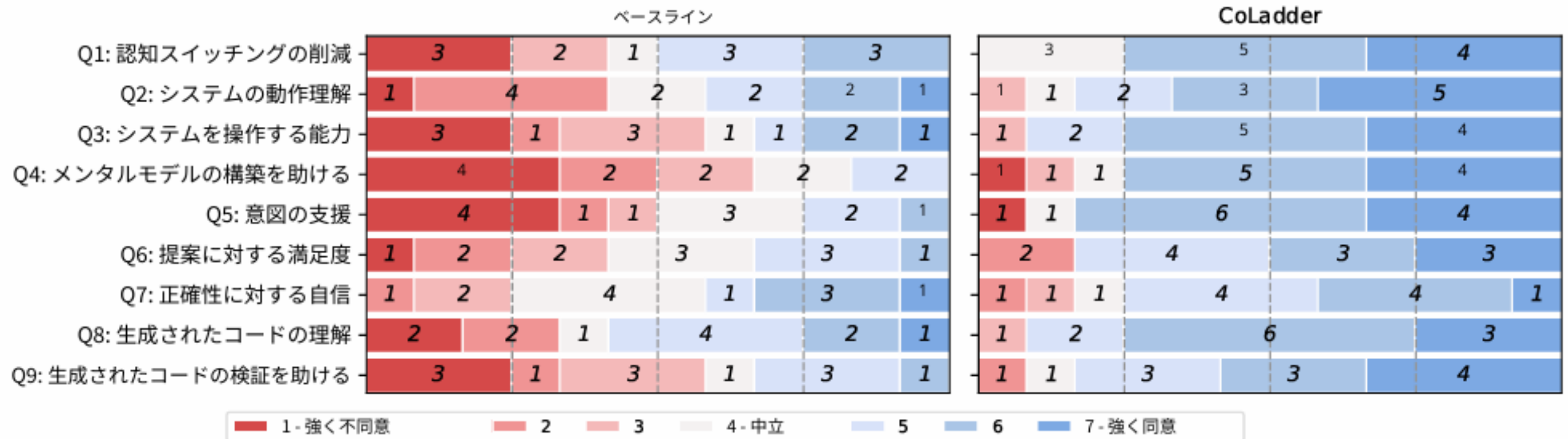
GitHub Copilotに似たインラインコメントに基づいてコードを生成するウェブベースのコードエディタ

タスク

12-15分間の機械学習とデータ可視化

評価結果

アンケート



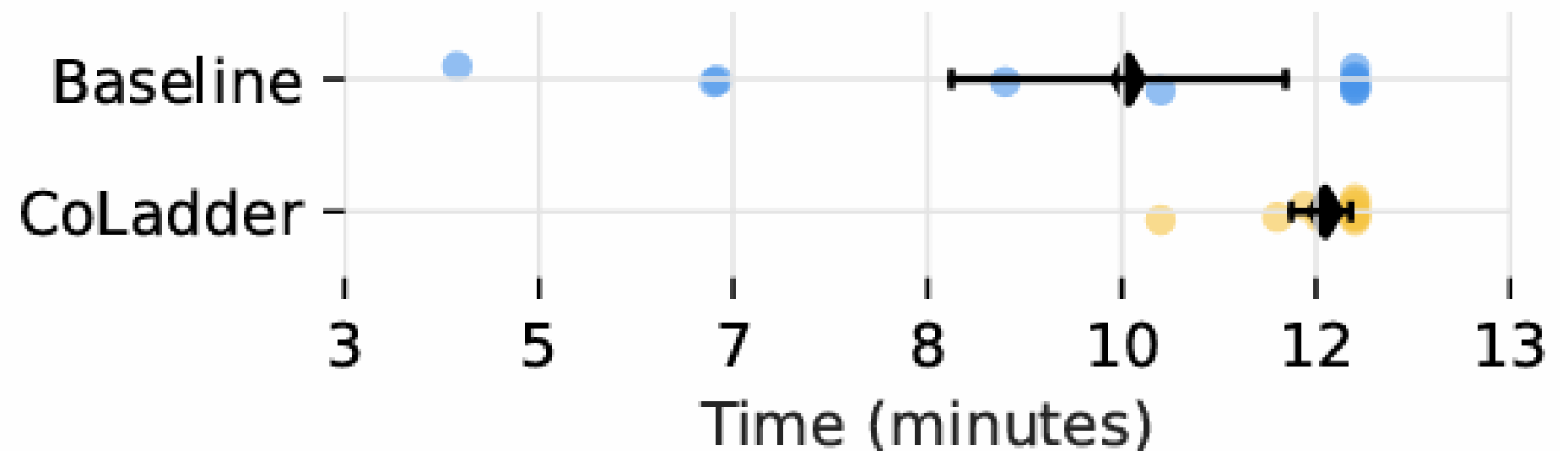
評価結果

タスクの正確性 (Base : CoLadder)

未完了のタスクの割合 50% : 41.67%

完了かつ正確 25% : 50%

タスクに費やした時間

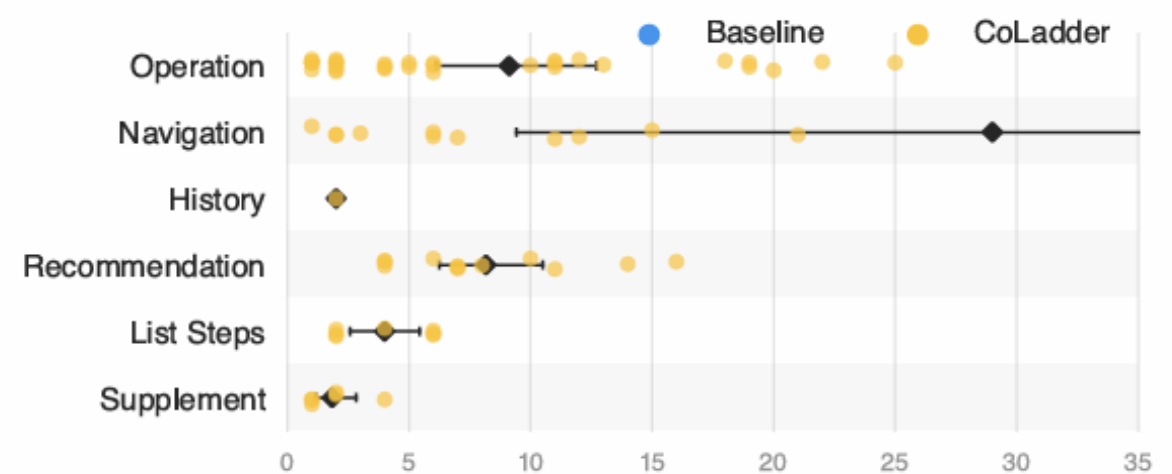
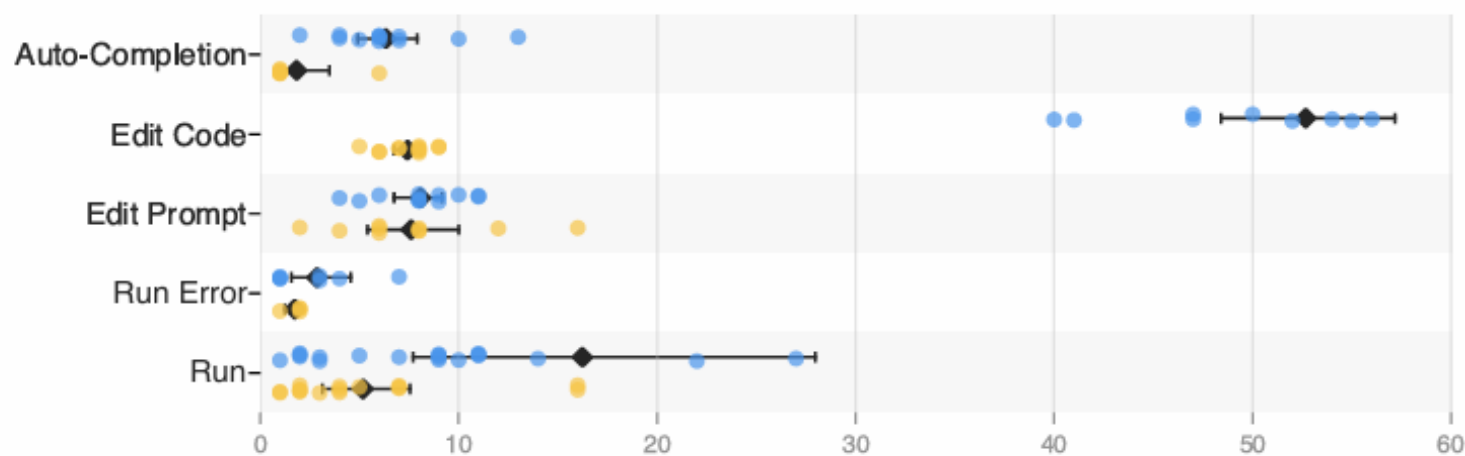


評価結果

行動の変化（操作ログ）

Baselineのユーザーは大量の時間を「生のコードの直接編集」に費やしていた

一方、CoLadderのユーザーはコードを直接いじらず、プロンプトブロックをドラッグ&ドロップしたり、階層を整理したりするブロック操作で問題を解決した



結論

AIアシスタント環境下での場当たりの開発から計画的な開発へ

AIに一気にコードを書かせるのではなく、意図を段階的に実装することでプログラマーの主体性を維持する

「タスクの階層的分解」「モジュール単位の生成」「プロンプト作成時の評価」を提供することで、プログラマーが目標設定から、最終的なコード実装まで、コードを操作・編集する能力を向上

所感

AIを賢くする・プロンプトを最適化するという視点ではなく、人間が正しく意図を組み立てられるようにUIを工夫したという斬新さ

CoLadderは親タスクと子タスクという1次元の包含関係しか表現できないことに発展可能性を感じた