AI コーディングアシスタント

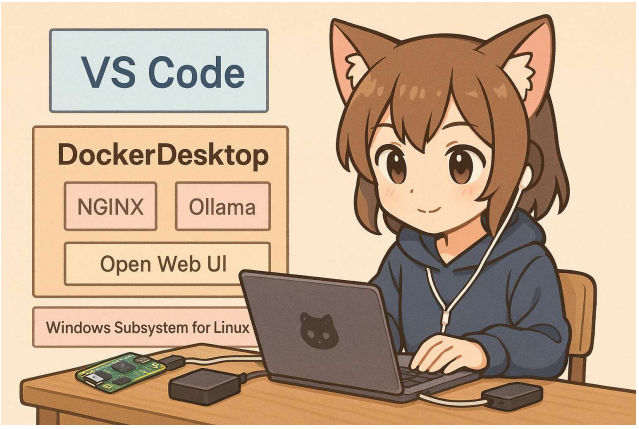
補足資料

氏森充

初めに

本資料は、コンピュータ技術雑誌「Interface（インターフェース）」（CQ 出版社）に掲載した記事の補足資料です。

本文で紹介した内容をさらに一歩掘り下げ、実際の環境で試してみたい方に向けて、誌面の都合により掲載できなかった「LLM をコーディング用途で試用した際の比較検証」について、動作確認に使用したプロンプト例、検証結果の詳細、ソフトウェアのインストール手順などの情報をまとめています。



目次

1. LLM コーディング用途での試用比較

1.1.動作比較用プロンプト

1.1.1. PNG 画像ファイルを表示する GUI プログラムの作成用プロンプト

次の仕様の Python のプログラムを作成してください
・PNG 画像ファイルを表示する GUI
・GUI にはファイル選択メニュー（ボタン）と選択したファイルの画像を表示するボタンがある

1.1.2. サーバ・クライアント型の通信プログラムの作成用プロンプト

server.py と client.py
それぞれ main 関数から実行可能なファイルと定義してください
■サーバクライアント型 API 通信プログラム
●プログラム仕様
・サーバ・クライアント型 socket 通信（複数のクライアントが同時にアクセス可能）システム
・通信プロトコルの内容は API で定義
・API の項目は、次の内容
open：API の接続の開始時の認証を行いセッションを開始します。
save：クライアントからデータを送信し、サーバに保存します。
　　この時データに名前を付けて送信します。
load：サーバに保存されたデータをクライアントにロードします。
　　クライアントからは、データの名前を指定してロードします。
list:サーバに保存されているデータのリストをクライアントに送ります。
close：セッションを開放します。
●API（構造は JSON）
1：コマンド：OPEN
例：{"api": "open", "username": "user1", "password": "pass123"}
＊応答
例：{"status": "ok", "session_id": "<session_id>" }
・session_id は動的に作成されたランダムな値

2：コマンド：SAVE
例：{"api": "save", "session_id": "ABC123", "name": "<FileName>", "data": "{...}" }
・name はクライアントからサーバに送るファイルの名称
・data はファイルの内容
＊応答
例：{"status": "ok", "session_id": "<session_id>}
・session_id は,open で作成された値

3：コマンド：LOAD
例：{"api": "load", "session_id": "<session_id> "name": "<FileName>" }
・name は、受信するファイルの名称
・session_id は,open で作成された値
＊応答
例：{"status": "ok","session_id": "<session_id>",
"name": "config.json" ,"data": "{...}" }
・name はファイルの名称
・data はファイルの内容
・session_id は,open で作成された値

4：コマンド：LIST
例：{"api": "list", "session_id": "<session_id>" }
・session_id は,open で作成された値
＊応答
例：{"status": "ok", "session_id": "<session_id>",
"names": ["<FileName>","<FileName>",...] } }
・names には、server に保存しているデータの名称をリストて返す
・session_id は,open で作成された値

5：コマンド：CLOSE
1：サーバ → クライアント API
{"api": "close", "session_id": "<session_id>" }
・session_id は,open で作成された値
＊応答
{"status": "ok", "session_id": "<session_id>" }
・session_id は,open で作成された値

●認証
接続に使用する PORT は 12345
ユーザは、user1 パスワードは pass123

●プログラムファイル
サーバ用のプログラムファイルと、クライアント用プログラムを分けて作成してください

1.1.3. 迷路を自動生成するプログラムの作成用プロンプト

穴掘り法を利用して2次元迷路自動生成するプログラムを作成して

1.2.検証結果詳細

各 LLM が生成したソースコード（SRC）の検証結果を以下に示します。
なお、本検証では、各 LLM が出力したプログラムコードを一度 ChatGPT-4o にレビューさせたうえで、筆者自身が内容を確認し、必要に応じて編集や追記を行っています。

評価結果は次に示す表現としています。
本誌に記載している「評価基準一覧表」に沿って詳細を判定し、評価項目毎の評価としています。
細かな判断は、ありますが、プログラムコードが概ね次の出来具合だと読んでください

評価結果については、以下の表現に基づいて記載しています。
・本誌に掲載している「評価基準一覧表」に沿って各項目を詳細に判定し、評価項目ごとに「判定判断内容一覧表」判定内容をもとに評価結果を行っています。
・一部に細かな判断の揺れはありますが、プログラムコード全体としては概ね次のような出来栄えであるとご理解ください。

表 1.2-1 判定判断内容一覧表

No	評価項目	評価結果	判定内容
1	機能実装度	良好	仕様（プロンプトの要求内容）に沿った実装が行われており、全体として完成度が高い構成となっている
		概ね良好	仕様（プロンプトの要求内容）に沿った実装が行われているが、動作自体には影響しない仕様抜けがある
		要改善	仕様（プロンプトの要求内容）とは異なった実装や、エラーハンドリングやエッジケース等の処理が抜けている
		不十分	仕様（プロンプトの要求内容）と異なった実装且つ、実行エラーを解消できない。
2	コーディングスタイル	良好	PEP 8 準拠でコードの品質や構造も整理されている。機能毎にモジュール分割もされている
		概ね良好	PEP 8 準拠でコードの品質や構造も整理されているが、モジュール分割がされていない
		要改善	若干の PEP 8 準拠違反があるが、見やすい構造で記述されている
		不十分	まったく PEP 8 に準拠していない。
3	コメントとドキュメント	良好	モジュール、クラス、関数ごとに適切な docstring が記述されており、適切な箇所にもコメントが記述されている。
		概ね良好	コメントや docstring が記載されているが、その記述内容が薄い
		要改善	docstring が無くコメントも少ない。
		不十分	コメントや docstring が一切記述されていない。

4	例外処理	良好	適切な例外の捕捉や、エラーもでリソースが正しく開放され、メッセージも理解しやすい内容となっている・
		概ね良好	適切な例外の捕捉や、エラーもでリソースが正しく開放されているが、メッセージが表示されない。
		要改善	例外の捕捉は無見込まれているが、リソースの解放やメッセージが表示されない。
		不十分	例外処理や異常値対応がされていない。

1.3.検証に使用した LLM のライセンス一覧

No	モデル名	作者	ライセンス
1	codellama	meta	Llama Code Acceptable Use Policy LLAMA 2 COMMUNITY LICENSE AGREEMENT
2	DeepSeek-Coder-v2	DeepSeek	DeepSeek License Agreement MIT License
3	Gemma3	google	gemma license
4	Phi-4	Microsoft	MIT ライセンス
5	qwen2.5-coder	Alibaba Cloud	Apache License Version 2.0
6	codestral	Mistral AI	Mistral AI Non-Production License
7	GPT-4o	OpenAI	https://openai.com/ja-JP/policies/row-terms-of-use/
8	Gemini	google	https://policies.google.com/terms?hl=ja https://ai.google.dev/gemini-api/terms?hl=ja
9	Claude 3.5 Sonnet	Anthropic	https://www.anthropic.com/legal/aup
10	deepseek-reasoner	DeepSeek	MIT License

第2章　表7を詳しくしたもの

1.3.1. codellama:7b

1. PNG 画像表示プログラム（codellama_07_1.py）

評価項目	評価	評価理由
機能実装度	要改善	ファイル選択から画像表示までの処理は一部実装されているが、import 文の誤記により正しく動作していない。 また、イメージ関連の初期化処理が不足しているため、画像を表示することができない、さらに、仕様に記載されている「画像表示」用のボタンが実装されていない
コーディングスタイル	概ね良好	命名やコード構造は概ね良好。ただし、import 文の順序など、細部には改善の余地がある。
コメントとドキュメント	不十分	コメントや docstring が記載されていない
例外処理	概ね良好	IOError 処理はあるが GUI への通知がない

2. API 通信プログラム（ codellama_07_2c.py, codellama_07_2s.py）

評価項目	評価	評価理由
機能実装度	要改善	セッション管理と API 仕様との不整合あり。複数クライアント対応が未実装
コーディングスタイル	要改善	一部関数やデータが未定義で、モジュール分割もなし。 PEP8 違反はないが構造的に不完全。
コメントとドキュメント	不十分	コメントや docstring が記載されていない。
例外処理	不十分	try/except がないため、接続・送受信失敗でクラッシュする恐れあり。

3. 迷路作成プログラム（codellama_07_3.py）

評価項目	評価	評価理由
機能実装度	要改善	穴掘り法とは異なり、ランダムに壁を埋めているのみで、通路が形成されていない
コーディングスタイル	良好	分かりやすい命名、整ったインデントだが、アルゴリズムとして中途半端な構造となっている
コメントとドキュメント	概ね良好	最低限のコメントあり。ただし迷路作成の処理を説明するコメントはなく誤解を招きやすい。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.2. codellama:13b

1. PNG 画像表示 GUI（codellama13_1.py）

評価項目	評価	評価理由
機能実装度	不十分	ファイル選択から画像表示までの処理は一部実装されているが、import 文に誤記がある。 また、イメージ関連の初期化処理が不足しているため、画像を表示することができない。さらに、仕様に記載されている「画像表示」用のボタンが実装されていない。
コーディングスタイル	概ね良好	PEP8 に準拠しており、クラスや関数の構造、および命名が明確、全体として視認性が高く、読みやすいコードとなっている。
コメントとドキュメント	不十分	コメントや docstring が記載されていない。
例外処理	不十分	例外処理が一切されていない。

2. API 通信プログラム（サーバ:codellama13_2s.py,クライアント:codellama13_2c.py）

評価項目	評価	評価理由
機能実装度	要改善	セッション・複数クライアント対応も未対応、クライアント処理が未完成。
コーディングスタイル	概ね良好	命名や構造は良いけど import 順に乱れがある
コメントとドキュメント	不十分	コメントや docstring が記載されていません。
例外処理	不十分	socket および JSON の処理に try-except がなく、例外発生時にクラッシュする恐れがある。エラーハンドリングの追加が必要。

3. 迷路生成プログラム(codellama13_3.py)

評価項目	評価	評価理由
機能実装度	要改善	ランダム生成はされているが、厳密な穴掘り法ではない
コーディングスタイル	概ね良好	コード全体はシンプルで可読性があるが、独自のロジックの部分の処理内容が読み取りにくくなっている。
コメントとドキュメント	要改善	最低限のコメントは記述されているが、処理の内容や意図を説明するものではなく、コードの理解を助けるには不十分。さらに、関数やクラスに対する docstring も記載されておらず、全体として可読性と保守性に課題がある。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.3. Deepseek-coder

1. PNG 画像表示 GUI（deepseek-coder_1.py）

評価項目	評価	評価理由
機能実装度	概ね良好	ファイル選択から画像表示までの流れは正確に実装されており、GUI もフレームを用いて整理されてる。 画像の表示自体は行われているが、表示領域に応じた縮小処理がなく、大きな画像がはみ出してしまう。
コーディングスタイル	良好	PEP8 に沿っており、クラス構造も適切
コメントとドキュメント	概ね良好	コメントは少ないものの、メソッド名が明確で読みやすい
例外処理	良好	ファイル選択・表示時に明確なエラーメッセージを出しており、適切なハンドリングがあり。

2. API 通信プログラム（deepseek-coder_2s.py,deepseek-coder_2c.py）

評価項目	評価	評価理由
機能実装度	良好	全 API を網羅しており、ファイル保存先やセッション管理も仕様に沿って適切に実装されている。 複数クライアント対応もスレッドによって実現されており、全体として堅実な構成となっている。
コーディングスタイル	要改善	ソケット処理・API 形式共に明確。 一方で、API の処理が if／elif によって1関数だけで定義されているが、API 毎のモジュール化が必要、スレッド処理や接続終了処理で PEP8 に非準拠箇所あり。
コメントとドキュメント	要改善	コメントが少なく、関数や API ごとの役割が明記されていない。
例外処理	要改善	一部エラーメッセージはあるが、通信エラー・構文エラー等への備えは不十分。

3. 迷路生成プログラム：deepseek-coder_3.py

評価項目	評価	評価理由
機能実装度	概ね良好	穴掘り法により迷路生成ができているが、マップサイズの制限（奇数を指定）が無いため、壁の間引きや枝分かれ制御に課題がある。
コーディングスタイル	良好	命名・スペース・関数構成すべて PEP8 に準拠している numpy の使用により配列管理も簡潔
コメントとドキュメント	要改善	関数の目的や処理の流れに関する説明が少ない。 docstring が未記述。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.4. Gemma3 12 b

1. PNG 画像表示 GUI （ Gemma3_1.py ）

評価項目	評価	評価理由
機能実装度	良好	ファイル選択と画像表示が正常に動作し、GUI 構成も仕様通り。表示失敗時のフィードバックもあり。
コーディングスタイル	良好	インデント、命名、関数分割ともに明瞭。PEP8 準拠している。
コメントとドキュメント	良好	クラス・関数に docstring が付与され、処理の流れを理解しやすい。
例外処理	良好	ファイル未選択・読み込み失敗などの例外に明確に対応し、ユーザーへメッセージ表示もされる。

2. API 通信プログラム（ gemma3_2s.py,gemma3_2c.py）

評価項目	評価	評価理由
機能実装度	概ね良好	全 API をカバーし、ファイル保存やセッション管理も仕様準拠複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	概ね良好	ソケット処理・API 形式共に明確。 一方で、API の処理が if／elif によって1関数だけで定義されているが、API 毎のモジュール化が必要
コメントとドキュメント	要改善	各関数に docstring はあるが、API を説明するコメントが無い
例外処理	良好	クライアント・サーバ両側にエラー時の例外処理が実装されており、ユーザビリティも考慮されている。

3. 迷路作成プログラム （gemma3_3.py）

評価項目	評価	評価理由
機能実装度	良好	穴掘り法を用いた迷路生成処理が正確に動作。開始点・通路の構築もランダム化されている。
コーディングスタイル	良好	PEP8 準拠、再帰関数を中心に、シンプルかつ明快な設計。変数名や関数名も意味が通っており、視認性が高い
コメントとドキュメント	良好	各関数・処理に docstring やコメントがあり、関数の役割・引数・戻り値が明確。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.5. Gemma3_27b

1. PNG 画像表示 GUI (Gemma3_27b_1.py)

評価項目	評価	評価理由
機能実装度	良好	ファイル選択と PNG 画像表示が機能しており、画像リサイズ処理も加えられている。
コーディングスタイル	良好	PEP8 準拠で明瞭。命名や構造も一貫性あり。
コメントとドキュメント	概ね良好	docstring は無いが、要所にはコメントがあり、最低限の可読性はある。
例外処理	良好	ファイル未選択、読込エラー時に適切な処理とフィードバックあり。

2. API 通信プログラム(gemma3_27b_2s.py ,gemma3_27b_2c.py)

評価項目	評価	詳細
機能実装度	概ね良好	全 API をカバーし、ファイル保存先やセッション管理も仕様準拠複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	概ね良好	ソケット処理・API 形式共に明確。 一方で、API の処理が if／elif によって1関数だけで定義されているが、API 毎のモジュール化が必要
コメントとドキュメント	要改善	クライアントは、docstring ありだが、Serverには docstring もコメントもない
例外処理	良好	無効なセッション、認証失敗、不正な JSON 等への応答が適切に処理されている。

3. 迷路作成プログラム(gemma3_27b_3.py)

評価項目	評価	詳細
機能実装度	良好	穴掘り法に基づいた迷路生成が正しく行われており、表示も整っている。
コーディングスタイル	良好	PEP8 準拠で、変数や関数名も簡潔明瞭。
コメントとドキュメント	良好	docstring が記述されており、処理概要も簡潔に記載。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.6. phi4

1. PNG 画像表示 GUI （ phi4_1.py）

評価項目	評価	詳細
機能実装度	概ね良好	ファイル選択→画像表示の流れが正確で、GUI もフレームによって整理されている。 画像も表示しているが、画面に合わせた縮小が欲しい
コーディングスタイル	良好	クラス設計、関数分割、命名規則ともに明瞭で一貫性がある。
コメントとドキュメント	良好	各関数に簡潔な docstring あり。全体構造もわかりやすい。
例外処理	良好	ファイル未選択時や読込エラーに対して適切な警告を表示しており、ユーザー対応も良好。

2. API 通信プログラム（ phi4_2s.py,phi4_2c.py）

評価項目	評価	評価理由
機能実装度	概ね良好	全 API をカバーし、ファイル保存先やセッション管理も仕様準拠。複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	良好	構造は整っており、関数分離と命名も一貫性あり
コメントとドキュメント	概ね良好	サーバ側には docstring がなく、補足コメントも控えめですが、構造が簡潔で理解しやすいです。
例外処理	概ね良好	try-except の使用あり。ただし詳細なハンドリングは弱め

3. 迷路作成プログラム（phi4_3.py）

評価項目	評価	詳細
機能実装度	良好	穴掘り法による迷路生成が正しく行われ、偶数指定時も内部で調整される。
コーディングスタイル	良好	関数名・変数名が明確で、再帰構造も整っている。
コメントとドキュメント	概ね良好	関数に docstring が無いため、外部向け説明はやや不足。しかし処理は理解しやすく書かれている。
例外処理	要改善	偶数サイズに対する調整がある、しかし異常なサイズ入力（負数など）には未対応。

1.3.7. qwen2.5-code_07b

1. PNG 画像表示 GUI (qwen2.5-coder_1.py)

評価項目	評価	詳細
機能実装度	要改善	PNG ファイルを選択し、画面の指定サイズにリサイズして表示しているが、表示ボタンが無く、指定サイズが固定のため画像に歪がある。
コーディングスタイル	良好	命名、インデントともに PEP8 準拠で整っている。 コメントが全く無い
コメントとドキュメント	概ね良好	明確な関数構造で理解は可能だが、docstring は未記載。
例外処理	良好	画像読込エラーに対してメッセージボックスで警告表示している。

API 通信プログラム(qwen2.5-coder_2s.py , qwen2.5-coder_2c.py)

評価項目	評価	詳細
機能実装度	概ね良好	仕様 API 全種に対応、セッション管理も正確、複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	良好	命名、関数分離が明確で、構成も整理されている。
コメントとドキュメント	要改善	クライアントは、コメントも docstring があるが、Serverにはどちらも記述されていない。
例外処理	良好	セッション ID やファイル存在チェックなど、基本的な例外処理が網羅されている。

迷路作成プログラム (qwen2.5-coder_3.py)

評価項目	評価	詳細
機能実装度	要改善	穴掘り法ではなく、壁を掘り進む独自方式。仕様(穴掘り法)とは異なるが動作自体は問題ない。
コーディングスタイル	概ね良好	可読性は良好だが、関数内処理がやや詰め込み気味。
コメントとドキュメント	要改善	docstring もコメントも一切なく、独自方式の事もあり処理内容の理解に時間がかる。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.8. qwen2.5-code_14b

1. PNG 画像表示 GUI (qwen25_14b_1.py)

評価項目	評価	評価理由
機能実装度	概ね良好	ファイル選択と表示ボタンが仕様に沿って動作 画像も表示しているが、画面に合わせた縮小が欲しい
コーディングスタイル	良好	PEP8 準拠、tk.Tk を継承した構成で、GUI 部品も明確に分離されている
コメントとドキュメント	良好	各関数に docstring あり、処理も整理されている
例外処理	良好	ファイル読み込みや表示に対して適切な例外処理あり

2. API 通信プログラム(qwen25_14b_2s.py, qwen25_14b_2c.py)

評価項目	評価	評価理由
機能実装度	概ね良好	仕様 API 全種に対応、セッション管理も正確、複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	良好	クラス構造で分離され、役割が明確に定義されている
コメントとドキュメント	良好	クラス・関数構成と処理が直感的で、可読性が高い
例外処理	概ね良好	不正セッション・ファイル未発見などは検出しているが、サーバ・ネットワークエラーへの try-except が限定的。

3. 迷路生成プログラム(qwen25_14b_3.py)

評価項目	評価	評価理由
機能実装度	不十分	穴掘り法ではなく独自アルゴリズムとなっている。 迷路が作成できる時もあるが、境界判定が不十分。
コーディングスタイル	要改善	インデントと命名は概ね良好だが、構造の整理に余地あり
コメントとドキュメント	要改善	最低限のコメントはあるが、アルゴリズムの概要説明が不足し、理解に時間がかかる
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.9. codestral

1. PNG 画像表示 GUI（codestral_1.py）

評価項目	評価	評価理由
機能実装度	要改善	イメージ表示は正しくできているが、表示ボタンがなく、選択後即表示で仕様に未適合
コーディングスタイル	概ね良好	インデントや命名は明確だがグローバル変数使用がやや不適切
コメントとドキュメント	不十分	関数や構造に docstring や説明コメントがない
例外処理	不十分	例外処理が存在せず、ファイル選択失敗に非対応

2. 通信プログラム（ codestral_2s.py / codestral_2c.py）

評価項目	評価	詳細
機能実装度	不十分	仕様の内容に対応。ただし、session_id がサーバ内部でのみ保持されており、複数クライアントへの同時対応に未対応、close のレスポンス形式が仕様と不一致
コーディングスタイル	要改善	命名は適切だが、複数ステートメントが一部 1 行に詰め込まれている。
コメントとドキュメント	要改善	コメントは最小限で、処理内容の説明や docstring が不足。
例外処理	要改善	認証失敗やファイル未発見には対応しているが、ネットワーク障害時や JSON デコードエラーに対する処理がない。

3. 穴掘り法による 2 次元迷路生成（codestral_3.py）

評価項目	評価	詳細
機能実装度	良好	穴掘り法による迷路生成が適切に実装されている。
コーディングスタイル	良好	命名、インデントともに一貫しており読みやすい。
コメントとドキュメント	概ね良好	簡潔なコメントがあるが、関数に docstring がない。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.10. gpt-4o

1. PNG 画像表示 GUI gpt-4o_1.py

評価項目	評価	詳細
機能実装度	良好	ファイル選択、画像表示、エラーメッセージ表示など GUI 機能を広範にカバーしている。
コーディングスタイル	良好	命名、インデント、クラス構成すべてが明瞭で可読性が高い。
コメントとドキュメント	概ね良好	docstring はないが、関数名と構造で十分に意図が読み取れる。
例外処理	良好	ファイル未選択時、読み込みエラー時など、ユーザへのフィードバックが整っている。

2. API 通信プログラム（gpt-4o_2s.py ， gpt-4o_2c.py）

評価項目	評価	詳細
機能実装度	概ね良好	仕様 API 全種に対応、セッション管理も正確、複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	概ね良好	両者ともに構成・命名が明瞭で、PEP8 に準拠。 API がモジュール化されていない。
コメントとドキュメント	概ね良好	関数ごとに処理が整理されているが、docstring は不足気味。
例外処理	概ね良好	セッションの妥当性、認証、データ存在チェックなど各ポイントで例外を適切に処理している。 ただし、recv 時の JSON デコード失敗などに対する例外処理は未実装。

3. 迷路生成プログラム(gpt-4o_3.py)

評価項目	評価	詳細
機能実装度	概ね良好	完全な穴掘り法とは言えないが迷路は作成できている。入力値に関しては、コメントで奇数を入力するように促している。
コーディングスタイル	良好	簡潔な構成で関数分離が明確。変数名も適切。
コメントとドキュメント	概ね良好	関数内のコメントは適切だが、関数定義に docstring が無い
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

1.3.11.gemini

1. PNG 画像表示 GUI （ gemini_1.py ）

評価項目	評価	詳細
機能実装度	概ね良好	ファイル選択・画像表示が正しく動作。状態切替やエラー時の対応も含まれている。
コーディングスタイル	良好	PEP8 に従い、命名やインデントも統一され読みやすい構造となっている。
コメントとドキュメント	概ね良好	必要最低限の docstring はあるが、全体の処理意図や要点に対するコメントが少ない。
例外処理	良好	ファイル未選択・ファイル読み込み失敗などに対応している

2. API 通信プログラム（ gemini_2s.py, gemini_2c.py）

評価項目	評価	詳細
機能実装度	概ね良好	仕様 API 全種に対応、セッション管理も正確、複数クライアント対応もスレッドで実現。 クライアントが、UI では無く単なる試験プログラムになっている。
コーディングスタイル	良好	両プログラムとも PEP8 に準拠し、インデント・命名が統一されている。関数分割と責務の明確化もできており、可読性が高い。
コメントとドキュメント	良好	サーバ側は全関数に docstring あり。クライアント側は関数に簡潔な docstring がある。
例外処理	良好	サーバ側は通信エラー、JSON 解析エラー、セッション不正、ファイル IO 失敗など幅広く対処。クライアント側も通信失敗時に処理継続不可とする明確な分岐あり。

3. 迷路生成プログラム(gemini_3.py)

評価項目	評価	詳細
機能実装度	良好	穴掘り法による正確な迷路生成と境界チェックが行われている。
コーディングスタイル	良好	命名・構造・インデントすべて明確。PEP8 も遵守。 内部ロジックも整っており理解しやすい
コメントとドキュメント	良好	docstring が明確で、処理意図も適切に説明されている。
例外処理	良好	不正な引数に対する ValueError での対応が適切に実装されている。

1.3.12. anthropic_claude-3.7-sonnet

1. PNG 画像表示 GUI (claud_1.py)

評価項目	評価	詳細
機能実装度	良好	要件に沿ってファイル選択・表示機能を正しく実装。エラーハンドリングやアスペクト比保持も適切。
コーディングスタイル	良好	PEP8 に準拠し、関数名・変数名も適切で可読性良好。フレームやレイアウトの使い方も適切です。
コメントとドキュメント	良好	関数ごとの docstring はないが、処理の意図はコメントで明示。
例外処理	良好	GUI 操作時のエラーに対し、適切にメッセージ表示処理を記述。

2. API 通信プログラム(claud_2s.py / claud_2c.py)

評価項目	評価	詳細
機能実装度	良好	要件通りの API 構造・認証・データ保存・読み込み等を網羅。セッション管理も実装。複数クライアント対応もスレッドで実現。
コーディングスタイル	良好	明確な関数分離・命名、一貫性のある実装となっている。
コメントとドキュメント	良好	すべての関数に docstring あり。処理の意図が明確。
例外処理	良好	通信エラー、JSON フォーマットエラー、ファイル I/O など多様な例外に対応。

3. 迷路生成プログラム (claud_3.py)

評価項目	評価	詳細
機能実装度	良好	穴掘り法に基づき、2D 迷路を正しく自動生成。保存・表示機能、入口出口設定も含む。
コーディングスタイル	良好	クラス設計、インデント、命名規則など良好。
コメントとドキュメント	良好	全体にわたり適切なコメントあり。docstring もあり。
例外処理	概ね良好	入力値のバリデーションはあるが、ファイル保存時の例外処理は省略されている。

1.3.13. Deepseek-reasoner

1. PNG 画像表示 GUI （deepseekR_1.py）

評価項目	評価	詳細
機能実装度	概ね良好	PNG ファイルの表示はできるが、表示ボタンが存在せず、選択と同時に表示される点で仕様逸脱あり、 画像も表示しているが、画面に合わせた縮小が欲しい
コーディングスタイル	概ね良好	PEP8 準拠で統一された命名・インデント、レイアウトも良好。
コメントとドキュメント	概ね良好	関数に docstring ありだが、クラスや全体構成の説明は省略。
例外処理	概ね良好	ファイル読み込みエラーを明示的に捕捉し、エラーダイアログを表示している。

2. API 通信プログラム（ deepseekR_2s.py ，deepseekR_2c.py）

評価項目	評価	詳細
機能実装度	要改善	セッション管理もあり。複数クライアント対応もスレッドで実現。 すべての API を実装しているが、認証にパスワードなしで可能になっている。
コーディングスタイル	良好	クラス設計、関数分離、PEP8 準拠、モジュール化が明確。
コメントとドキュメント	概ね良好	一部 docstring が不足しているが、関数ごとに構造が整理されており読みやすい。
例外処理	概ね良好	サーバ側に try/finally あり。バリデーションも適切。クライアント側も例外に強い設計。

3. 迷路生成プログラム(deepseekR_3. py)

評価項目	評価	詳細
機能実装度	要改善	穴掘り法に近いロジックだが、迷路は作成できない。 入力値が奇数になるように修正されている点は良い
コーディングスタイル	良好	クラス構造、メソッド分離、命名、インデントすべて良好。
コメントとドキュメント	概ね良好	メソッドには docstring がなく説明はやや不足するが、関数構成で理解しやすい。
例外処理	不十分	幅や高さなどのパラメータに対する入力検証が実装されておらず、無効な値が入力された場合に不正な動作やクラッシュが発生する可能性がある。

第3章

2. ソフトのインストール

ステップ2

2.1. Docker Desktop

2.1.1. 概要

Docker Desktop は、コンテナ仮想化を手軽に扱える GUI ツールで、開発環境の構築や管理を効率化します。

URL　　：https://www.docker.com/ja-jp/
License　：Docker Subscription Service Agreement
　　(https://www.docker.com/ja-jp/legal/docker-subscription-service-agreement/)
開発元　　：Docker
バージョン：v0.6.3
サポート　：https://github.com/sponsors/tjbck

2.1.2. インストール環境の事前確認

1. Docker Desktop をインストールする前に、Windows Subsystem for Linux（WSL）の設定を済ませておく必要があります。
2. 以下のコマンドを実行し、WSL のバージョンが WSL2 になっていることを確認してください。

（WSL 以外のバージョン番号に関しては、最新の物になっていればよいです）

```
PS > wsl.exe --update
      要求された操作には管理者特権が必要です。
      ダウンロード中: Linux 用 Windows サブシステム 2.4.13
      インストール中: Linux 用 Windows サブシステム 2.4.13
      Linux 用 Windows サブシステム 2.4.13 はインストールされました。
      この操作を正しく終了しました。
      更新プログラムを確認しています。
      Linux 用 Windows サブシステムの最新バージョンは既にインストールされています。
PS > wsl --set-default-version 2
   WSL 2 との主な違いについては、https://aka.ms/wsl2
   を参照してください
   この操作を正しく終了しました。
```

WSL の設定に問題がなければ、Docker の公式サイトから「Docker Desktop Installer.exe」をダウンロードし、インストールを行ってください。

※ 注意：

商用利用の場合、Docker Desktop の使用には「Docker サブスクリプション（有料ライセンス）」の契約が必要です。企業や組織で利用する場合は、ライセンス条件をご確認のうえ対応してください。

2.1.3. Docker Desktop のダウンロード:

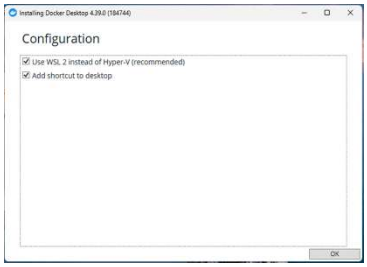
- Docker の公式サイトからインストーラーをダウンロードします。
- Docker Desktop for Windows（画面 2-2-1）

2.1.4. インストーラーの実行:

① ダウンロードしたインストーラーを実行し、画面の指示に従ってインストールを進めます。

② インストール中に「WSL 2 の有効化」に関するオプションが表示されますので、「WSL 2 を使用する」を選択してください（画面 2-2-2）

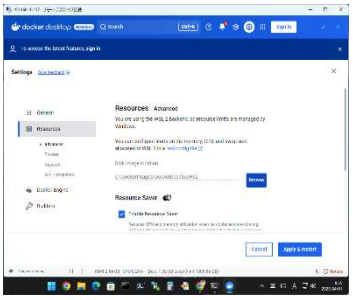
画面 2-1-1



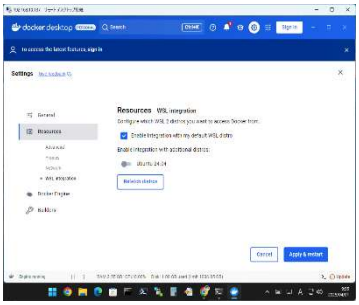
画面 2-1-2

2.1.5. インストール後の設定:

1. インストール完了後（画面 2-3）、Docker Desktop を起動し、必要に応じて設定を行います。
   ① Docker イメージの保存場所の設定（画面 2-2-3）
   ② WSL 関連の設定（画面 2-2-4）



画面 2-1-3



画面 2-1-4

2.1.6. 補足事項

```
PS> docker run --rm -it --gpus=all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
```

① Docker Desktop のインストールには、管理者権限が必要です。
② インストール前に、既存の仮想化ソフトウェアとの競合がないか確認してください。
③ Docker Desktop の利用には、ライセンス取得が必要です。詳しくは、Docker 社のライセンスページを確認してください。
④ Wsl2 で GPU が有効になるか確認してください

2.2. Ollama

2.2.1. 概要

Ollama は、オープンソースコミュニティによって開発されたツールで、ローカル環境で大規模言語モデル（LLM）を手軽に実行できることを特徴としています。

このツールを使用することで、インターネット接続に依存せずに、テキスト生成や会話型 AI の機能を個人のコンピュータ上で利用することが可能になります。

URL	: https://ollama.com/
License	: MIT License
開発元	: Ollama
GITHUB	: https://github.com/ollama
Version	: v0.6.5

2.2.2. 主な特徴

① オフライン環境での利用:: Ollama は、インターネット接続がない環境でも利用可能です。あらかじめ使用するモデルをダウンロードしておけば、以降はオフラインで継続的に利用できます。すべてのデータ処理がローカル環境内で完結するため、機密情報が外部に漏れるリスクを抑えることができます。

② コスト削減: クラウドベースのサービスと異なり、API 利用料などの追加コストを気にすることなく、何度でもモデルを実行できます。

③ 多様なモデルのサポート: Llama 2、Mistral、Phi-3 など、複数のオープンソースモデルをサポートし、用途に応じて選択可能です。

④ llama.cpp の活用: Ollama は内部的に llama.cpp を利用しており、量子化されたモデルを使用することで、高性能な GPU や大量の RAM がなくても LLM を動作させることが可能です。これにより、一般的な PC 環境でも効率的なメモリ管理が実現されています。

2.2.3. インストール用 コマンド(DockerDesktop（Windows）の利用)

Docker Desktop を利用して Ollama をインストールする場合、あらかじめ GPU ドライバーをインストールしておく必要がありますが、CUDA 関連のライブラリを手動で設定する必要はありません。
※補足：Docker Desktop 上の Ollama は Linux 環境で動作しているため、本来であれば CUDA のインストールが必要ですが、現在の Windows 環境では WSL が CUDA に対応しているため、ユーザーが別途 CUDA をインストールする必要はありません。

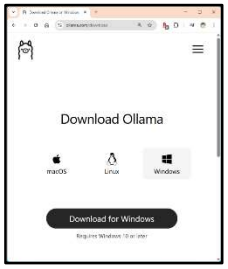
```
PS> docker run -d --gpus=all -v ollama:/root/.ollama -p 11434:11434 --name ollama ollama/ollama
```

① run -d ：バックグラウンドモードでコンテナを動作させる
② --gpus=all ：GPU を利用するモード
③ -v ollama:/root/.ollama ： ホスト側（PC 側）の「ollama」のフォルダを「/root/.ollama」でマウントする。
④ -p 11434:11434 ：ollama コンテナの 11434 ポートを 11434 で外部に公開する
⑤ --name ollama ：コンテナの名を「ollama 」とする
⑥ ollama/ollama ： DockerHub に登録されている ollama 用イメージを利用する

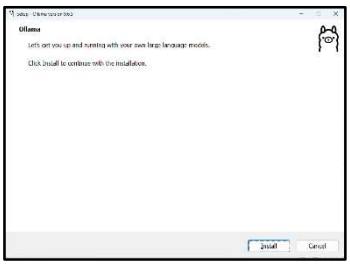
2.2.4. Docker を利用しないインストール（Windows）

① Ollama の HP からインストーラをダウンロードし、実行します。(画面 2 2-1,画面 2 2-2)
② Ollama を GPU 有で動作させる場合は、GPUのドライバーは必要ですが、別途 cuda 関連のライブラリ(CUDA Toolkit)をインストールする必要はありません.
※：Windows の場合、Ollama のインストーラ自体に cuda 関連のツールがインストールされています。
＊：詳細は、「https://github.com/ollama/ollama/blob/main/README.md」を参照
③ ollama をインストールした後に、環境変数をセットしてください。(画面 2 2-3)

#Ollama サーバを外部に公開
OLLAMA_HOST=0.0.0.0
#LLM ファイルの保存先
OLLAMA_MODELS=D:¥0llama¥



画面 2-2-1



画面 2-2-2

画面 2-2-3

ステップ4

2.3.Open WebUI

2.3.1. 概要

OpenWebUI は、ローカルで LLM を実行する環境「Ollama」を直感的に操作できる、Web ベースのユーザーインターフェースです。
モデルの管理やチャットの実行が簡単に行えるほか、複数モデルの切り替えや履歴管理にも対応しており、ローカル環境での LLM 利用をより快適にするツールです。

```
URL      :https://openwebui.com/
License  :BSD 3-Clause License
GitHub   :https://github.com/open-webui
```

2.3.2. 主な特徴:

① 完全オフライン動作: インターネット接続なしで動作するため、データのプライバシーとセキュリティを確保できます。
② 多様な LLM ランナーのサポート: Ollama や OpenAI 互換 API など、複数の LLM ランナーに対応しており、ユーザーはニーズに応じて選択できます。
③ モデルの一元管理: 複数の LLM を一元的に管理でき、必要に応じてモデルの追加や削除、設定変更が簡単に行えます。
④ RAG(Retrieval-Augmented Generation)機能: ユーザーが登録したドキュメントを基に、より適切な応答を生成することが可能です。

2.3.3. Docker を利用したインストール

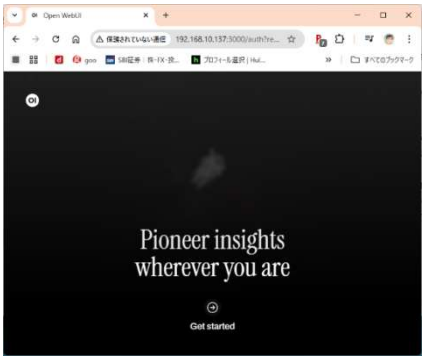
```
PS> docker run -d -p 3000:8080 -e OLLAMA_BASE_URL=http://xxx.xxx.xxx.xxx:11434 -v
open-webui:/app/backend/data --name open-webui --restart always ghcr.io/open-
```

パラメタ説明
① run -d : バックグラウンドモードでコンテナを動作させる
② -p 3000:8080 : コンテナ内の 8080 ポートを外部 3000 で公開する
③ -e OLLAMA_BASE_URL=http://xxx.xxx.xxx.xxx:11434 :環境変数を設定する
環境変数「OLLAMA_BASE_URL」は、接続する ollama のアドレスを設定値ですので、「xxx.xxx.xxx.xxx」には、PC2の IP アドレスを設定してください。
④ -v open-webui:/app/backend/data : PC1の open-webui というパスを、:/app/backend/data というパスでコンテナ側に提供する。これにより「/app/backend/data」配下に記憶されたデータは、docker コンテナには保存されません。これにより、open-webui をアップデートする目的で open-webui の docker コンテナを再構成しても、データを引き継ぐことができます。
⑤ --name open-webui :コンテナの名称を open-webui とする
⑥ --restart always : Docker を開始した時に open-webui も起動する。また、何らかの原因で open-webui がダウンしたても、再起動させる。

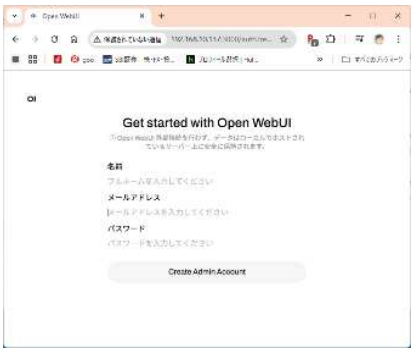
⑦ ghcr.io/open-webui/open-webui:main ： GitHub に登録されている、open-webui のコンテナを利用する

2.3.4. インストール後の設定

1：ブラウザで「http://localhost:3000」を表示（画面 2.3-1）
※サーバーの起動には時間がかかることがあります。接続できない場合は、しばらく待ってから再度アクセスしてください。
2：「Get Started」をクリックしてユーザー登録を行います（画面 2.3-2）。
※ここで入力した情報はローカル環境でのみ使用されるものであり、外部には送信されません。
3：画面右上または左下に表示されているユーザー名のアイコンをクリックし、「管理者パネル」を選択します(画面 2.3-3)。
4：「設定」タブを開き、「接続」を選択します（画面 2.3-4）。
5：接続先一覧に表示されている http://xxx.xxx.xxx.xxx:11434 の右側にある「管理」アイコンをクリックします。
6：「ollama.com からモデルをプル」の下にある入力欄に llama3.2:1b と入力し、右端のボタンをクリックすると、モデルがダウンロードされます（画面 2.3-5）。
7：他のモデルを利用したい場合は、「ダウンロード可能なモデル名にアクセスするには、ここをクリックしてください。」のリンクを開き、表示される「Library」画面から利用可能なモデルを選択してください。

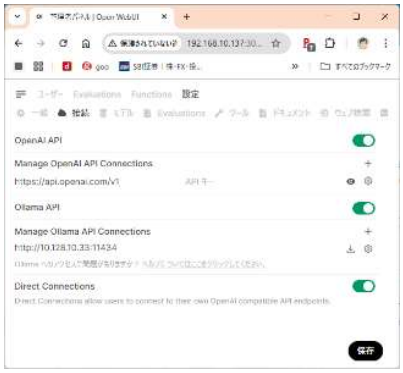


画面 2.3-1



画面 2.3-2

画面 2.3-3



画面 2.3-4

画面 2.3-5

ステップ5

2.4.NGINX

2.4.1. 概要

NGINX（エンジンエックス）は、高速かつ高性能な Web サーバーおよびリバースプロキシサーバーです。
静的コンテンツの配信をはじめ、ロードバランシング、SSL 終端処理、HTTP キャッシュなど、さまざまな機能に対応しています。
軽量でスケーラブルな設計が特徴であり、大規模なトラフィックにも柔軟に対応できる点から、多くのシステムで採用されています。

URL　　:https://nginx.org/
License　:2-clause BSD-like license.（https://nginx.org/LICENSE）
GitHub　:https://github.com/nginx/nginx

2.4.2. 主な特徴

① 静的な HTML、画像、CSS、JavaScript などのファイルを高速に処理・配信できるため、Web サイトの表示速度向上に貢献します。
② クライアントからのリクエストを複数のバックエンドサーバに分散することで、可用性の向上と負荷分散を実現します。
③ イベント駆動型アーキテクチャを採用しており、低いメモリ使用量で数万の同時接続にも高効率で対応できます。
④ セキュリティ強化のための SSL/TLS 通信や、通信効率を向上させる HTTP/2 にも標準で対応しており、現代的な Web の要件を満たします。

2.4.3. インストール（Docker 利用）

Docker を利用して NGINX をインストールする場合は、サンプルの Docker 用設定ファイルをベースに、ご自身の環境に合わせて内容を適宜変更してください。

① Dockerfile（Docker イメージの構築（ダウンロード））

```
FROM nginx:alpine
```

② docker-compose.yml（Docker サービス定義 ）

```
version: '3.8'

services:
  reverse-proxy:
    build: .A
    container_name: reverse-proxy
    ports:
      - "11434:11434"
    volumes:
      - ./default.conf:/etc/nginx/conf.d/default.conf:ro
    restart: always
```

③　default.conf（NGINX 設定ファイル）

```
server {
    listen 11434;
    server_name _;
    location / {
        proxy_pass http://xxx.xxx.xxx.xxx:11434;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

※：xxx.xxx.xxx.xxx” には、ollama をインストールした PC のアドレスを登録してください

この3つの設定ファイルを所定の場所におき、Docker を起動します。

```
PS＞ docker compose up -d
```

ステップ7

2.5.Visual Studio Code

2.5.1. 概要

Visual Studio Code（VSCode）は、Microsoft が開発した無料のオープンソースコードエディタで、軽量かつ高機能が特徴です。多くのプログラミング言語に対応し、拡張機能により柔軟なカスタマイズが可能です。

License：
MICROSOFT SOFTWARE LICENSE TERMS

2.5.2. インストール手順

「Visual Studio Code」のインストール手順を次に示します。

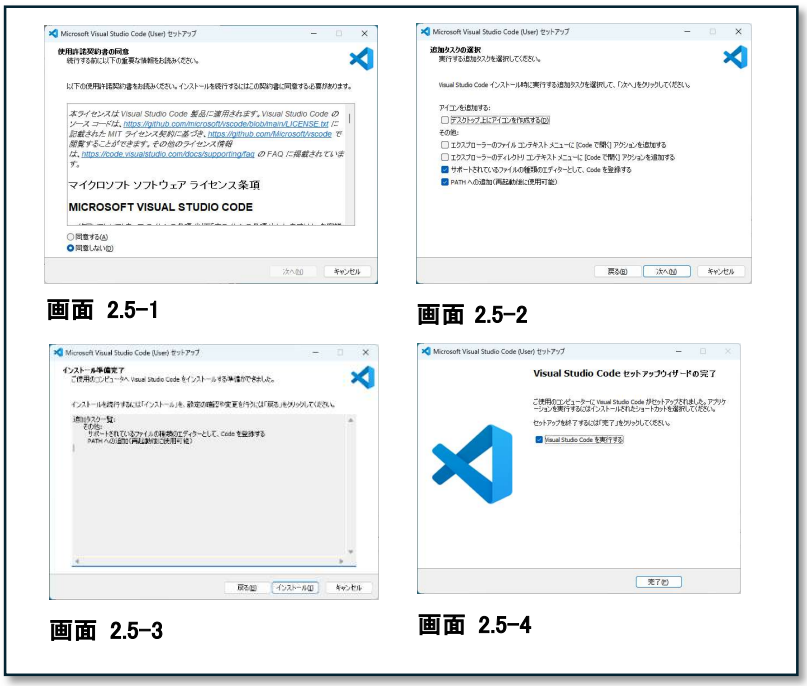
① インストーラのダウンロード
HomePage：https://code.visualstudio.com/
DownloadPage：https://code.visualstudio.com/Download
利用している OS に合ったインストーラをダウンロードする

② インストーラを実行すると「使用許諾契約書の同意」を求められますので、ライセンスの内容をよく読み、同意できるようであれば「同意する」をチャックし「次へ」をクリックする。（画面 2.5-1）

③ 追加タスクの選択（画面 2.5-2）
追加したいタスクを選択して「次へ」を選択し「次へ」をクリック

④ インストール準備関完了（画面 2.5-3）
設定内容が表示されますので、問題無ければ「インストール」をクリック
VisualStudio Code セットアップウィザードの完了

⑤ これで、インストールは完了です。（画面 2.5-4）



画面 2.5-1

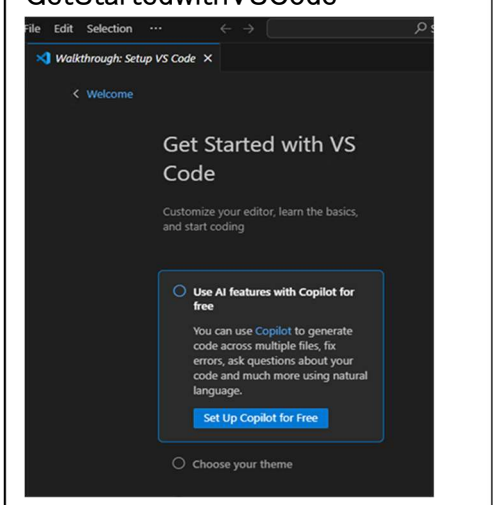
画面 2.5-2

画面 2.5-3

画面 2.5-4

⑥ VsCode の起動
起動後、初めての方は、「GetStartedwithVSCode」の画面が表示されます。(画面 2.5-5)
⑦ 後でも設定できますので、特に必要なければ「エスケープキー」を押すと TOP 画面に移ります。(画面 2.5-6)

GetStartedwithVSCode



画面 2.5-5

TOP 画面

画面 2.5-6

VSCodeの拡張機能を利用してAIコーディング・アシスタントを導入する　　本文の図7～

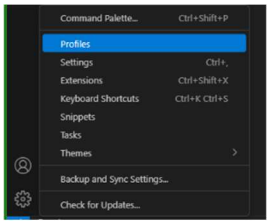
2.5.3. 開発環境設定

開発環境を構築するための準備を次にしめします。

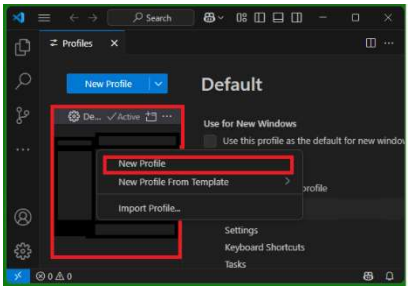
1. プロジェクト用ワークスペースの作成
   開発に利用するコンテンツや拡張機能およびワークスペース用フォルダの設定を管理する「プロファイル」を作成する。
2. AI コーディングアシスタント用
   ローカル用とクラウド用の2種類のプロファイル作成を作成する。
3. ローカル用 LLM用プロファイル
   開発に使用する言語用拡張機能(例:Python)や、 LocalLLM用 AI コードアシスタント(例 Continue)を作成したプラットフォームにインストールする。
4. クラウド用 LLM用プロファイル
   開発に使用する言語用拡張機能(例:Python)や、 クラウドLLM用 AI コードアシスタント(例 GitHub Copilot)を作成したプラットフォームにインストールする。

2.5.4. プロファイル作成

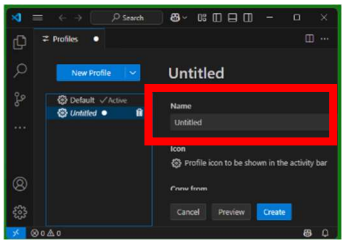
・「アクティビティバー」—>「管理」—>「Profiles」（画面 2.5-7） 本文図7（a）
・「プロファイル画面」
・「プロファイル一覧」右クリック」—>「NewProfile」（画面 2.5-8）図7（b）
・Name 欄に <プロファイル名>を入力（画面 2.5-9）
　例：「AI コーディング」
・ベースにするプロファイルを選択（画面 2.5-10）
　例：Copy from 「Python」を選択
・拡張機能は、無しにして後で必要な物のみを入れる（画面 2.5-11）
　Contents —> Extensions —>「None」
・設定を確定する。「Create」（画面 2.5-11）図7（c）
・プロファイル切り替え(画面 2.5-12)
　例：「アクティビティバー」—>「管理」—>「Profiles」—>[AI コーディング」 図8



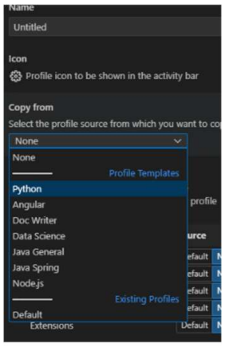
画面 2.5-7



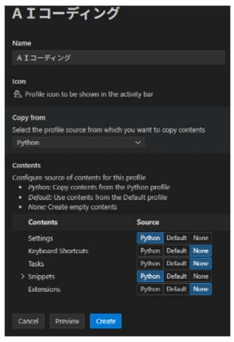
画面 2.5-8



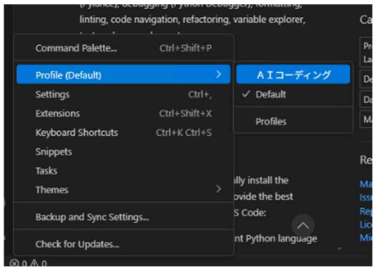
画面 2.5-9



画面 2.5-10



画面 2.5-11



画面 2.5-12

2.5.5. 拡張機能のインストール

1. Python

（ア）Python 用拡張機能です。実行に使用するPythonは、別途インストールする必要があります。

本文
図9（a）

① Pylance、Python Debugger の2つの拡張機能も同時にインストールされます

License
Pylance、Python Debugger
MIT ライセンス
Pylance
MICROSOFT SOFTWARE LICENSE TERMS MICROSOFT PYLANCE EXTENSION FOR VISUAL STUDIO CODE

2. Japanese Language Pack for VS Code

（ア）VS Code に日本語用にローカライズされた UI を提供する拡張機能です。

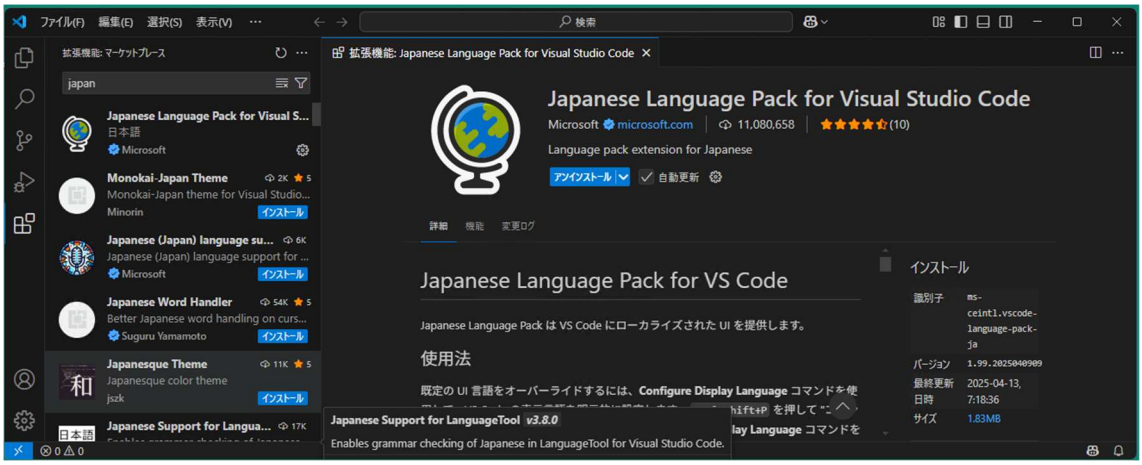


図9（b）

License
MIT ライセンス

2.5.6. ローカル LLM 用拡張機能「Continue」

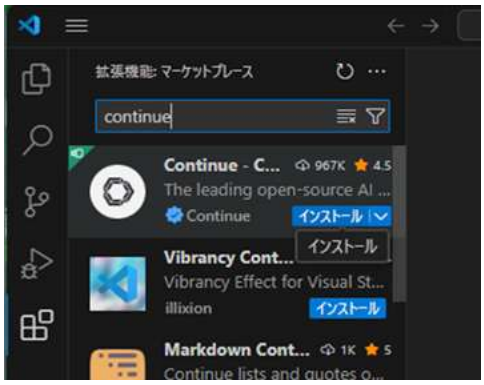
ローカル環境でも利用できる AI コーディングアシスタント拡張機能です。

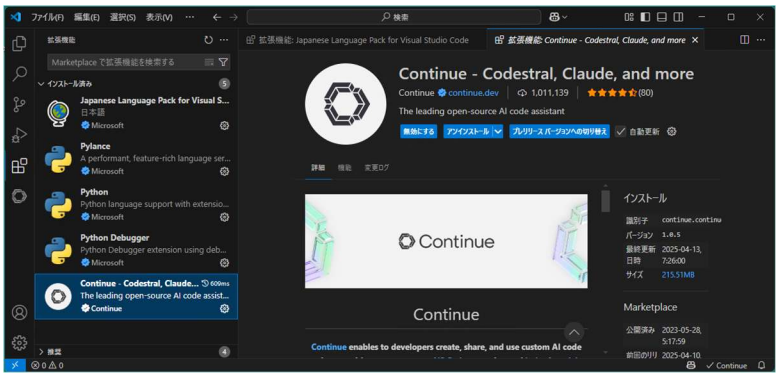
URL　　:https://hub.continue.dev/pricing
License :Apache License 2.0
開発元　:Continue Dev, Inc.
GITHUB :https://github.com/continuedev/continue

拡張機能マーケットプレースから「Continue」をインストールします

図10

マーケットプレース





Continue の設定

① Continue インストール後、VSCode を再起動すると「アクティビティバー」に Continue のアイコンが表示されます。

Continue 初期画面

Continue
アイコン

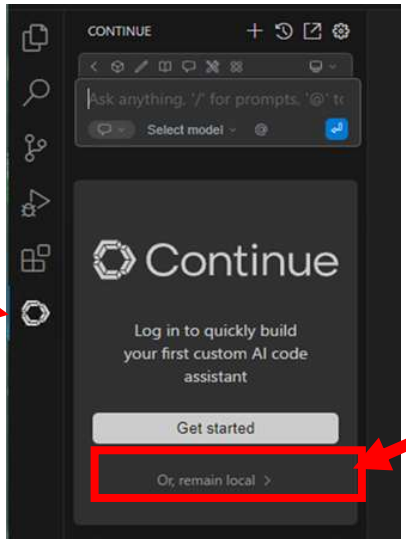


図11

Or.remain local

② インストール直後は、このアイコンをクリックすると「Continue」の初期画面が表示されます。
③ ローカル環境の llm を利用するだけであれば Continue はログインせずに利用できますので、「Or.remain local」を選択します。

④ LLM の設定画面が表示されます。Ollama は、先に Docker にインストールしている物が NGINX（Proxy）を通して認識されています。

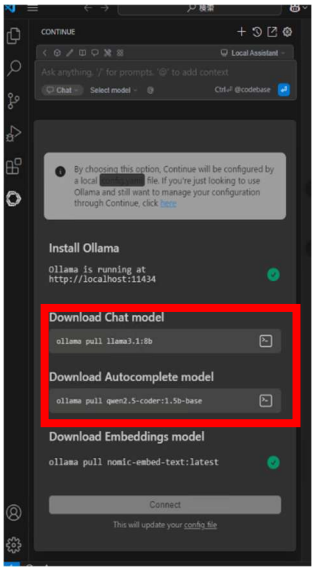
⑤ 次に、Chat Model、Autocomplete model の設定を行います。（画面 2.5-13）

⑥ ollama は、Docker 内にインストールしていますので、Windows 環境からは直接実行はできません。そこで、先にインストールした OpenWebUI を利用して、「llama3.1:8b」と「qwen2.5-coder:1.5b-base」の2つの LLM を Ollama にインストールます。（画面 2.5-14）

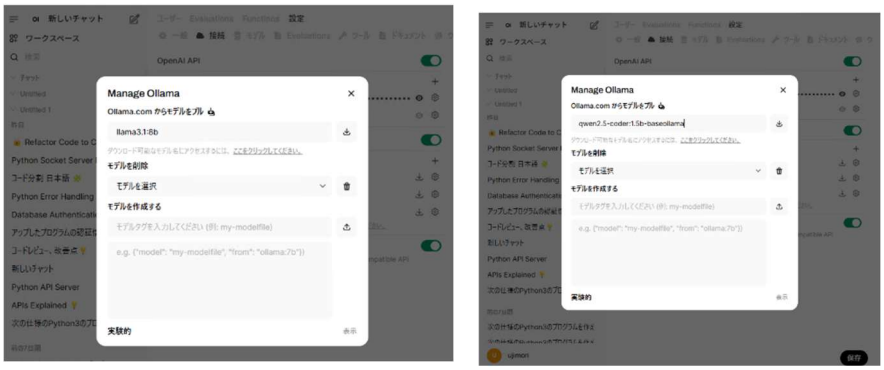
⑦ モデルのインストールが完了すると、全てにチェックが入ります。（画面 2.5-15）

⑧ この後「Connect」をクリックすると、ローカル環境の AI コーディングアシスタント が利用できるようになります。（画面 2.5-16）

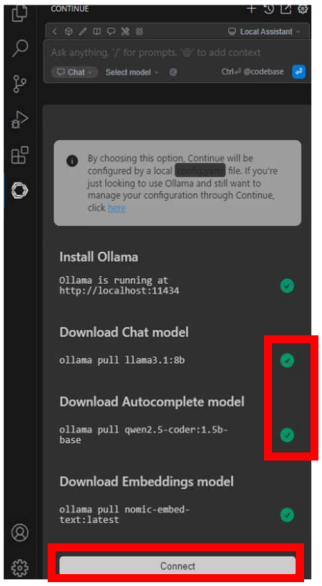
⑨ このプロファイルは、Local 用なので、クラウドに接続する「GitHubCopilot」を「Hide」にしておきます（画面 2.5-17）



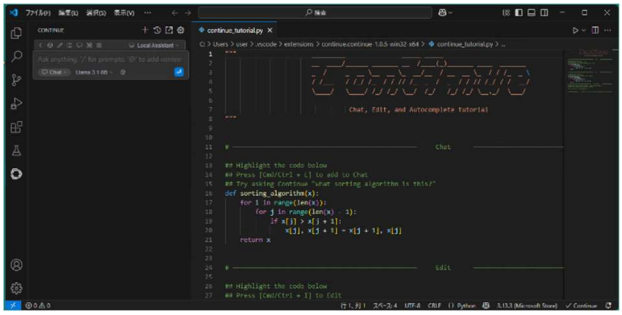
画面 2.5-13 図12



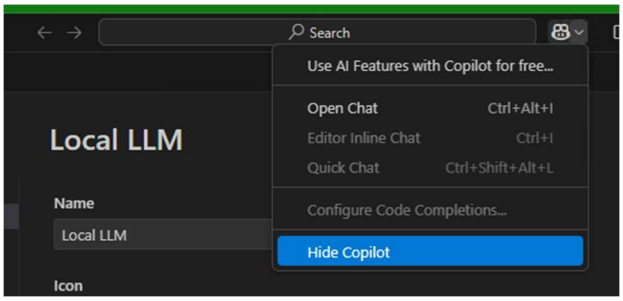
画面 2.5-14 図13



画面 2.5-15 図14



画面 2.5-16



画面 2.5-17

Connect と OpenWebUI との接続設定

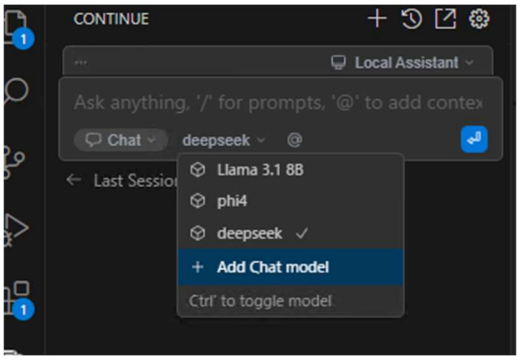
OpenWebUI が提供している「OpenAI」形式の API を利用することで、Continue と OpenWebUI とを接続できます。

個人で使用する場合は、特に必要ありませんが、複数のメンバーでLLMを共有する場合には、開発する内容に応じて利用できるLLMを一元管理することができますので、この方法が有効です。

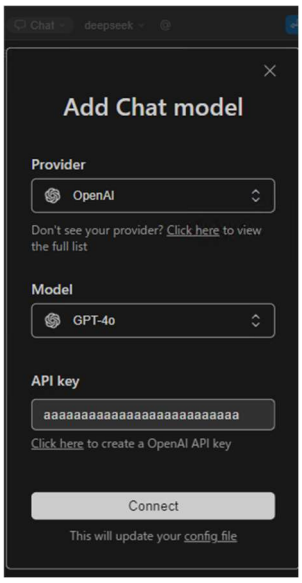
① OpenWebUI の API キーを取得
まず設定メニューから API キーを取得します
設定—＞アカウント—＞API キー

画面 2.5-18

② Continue への登録 Continue の「Chat」画面—＞「AddChatModel」（画面 2.5-19）
③ AddChatModel 設定画面に「Provider：OpenAI」を指定し、モデルは、選択できる物を追加（後で変更します）、APIkey には。OpenWebUI から取得した APIkey を設定し「Connect」をクリック

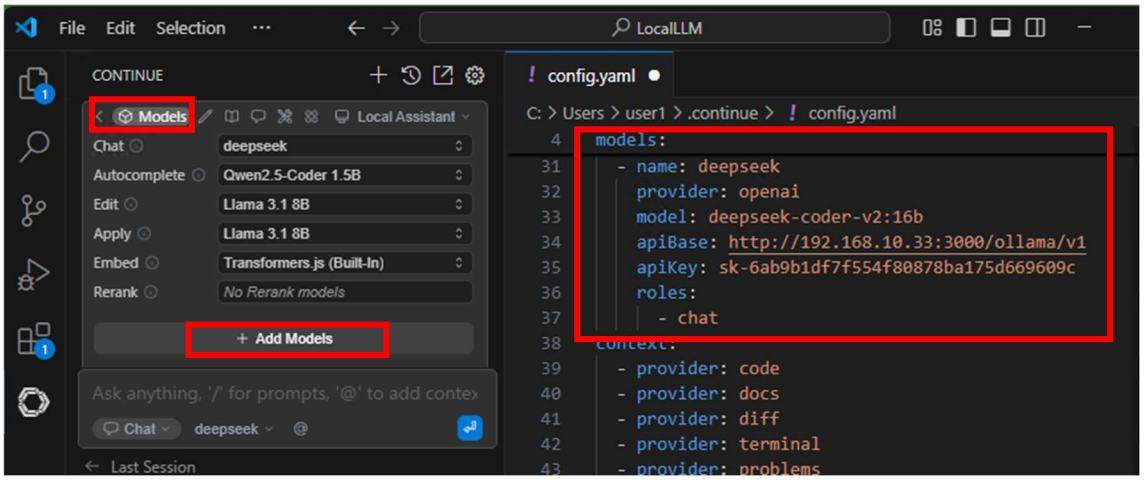


画面 2.5-19 図16



画面 2.5-20 図17

④　config.yaml を編集して「OpenWebUI」に接続する設定行います。
Continue の「Chat」画面—＞「Models」—＞「AddModels」—＞「config.yaml」
表示された config.yaml の内容を直接編集します。



設定項目

name：にモデル名　（任意の値）

model：にフルモデル名

apiBase：に openwebui のアドレスと API パス

設定例

設定ファイルは、現行では「yaml 形式」ですが、古い Ver だと「json 形式」になっています。

（　参考：https://docs.continue.dev/yaml-migration）

■yaml 形式

```
- name: deepseek
  provider: openai
  model: deepseek-coder-v2:16b
  apiBase: http://192.168.10.33:3000/ollama/v1
  apiKey: sk-xxxyyyzzz
  roles:
    - chat
```

■json 形式

```
{
  "title": "deepseek",
   "provider": "openai",
   "model": "deepseek-coder-v2:16b",
   "apiBase": "http://192.168.10.33:3000/ollama/v1",
   "apiKey": "sk-xxxyyyzzz"
}
```

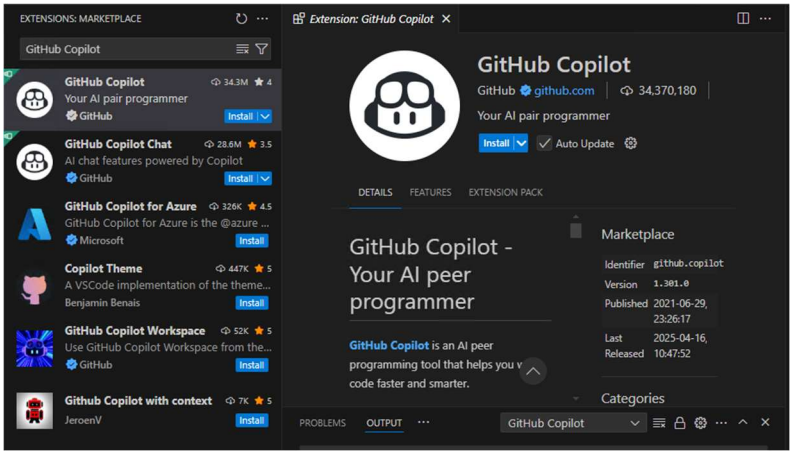
2.5.7. クラウド LLM 用拡張機能「GitHubCopilot」

クラウド環境で利用できる AI コーディングアシスタント拡張機能です。

URL　　:https://docs.github.com/ja
License :GitHub Copilot Product Specific Terms
GitHub Customer Agreemen
GitHub Terms for Additional Products and Features
Subscription plans for GitHub Copilot
開発元　:GitHub
GITHUB :https://github.com/continuedev/continue

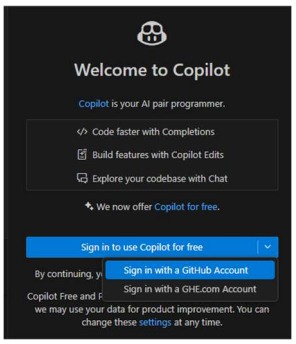
拡張機能マーケットプレースから「Continue」をインストールします

マーケットプレース



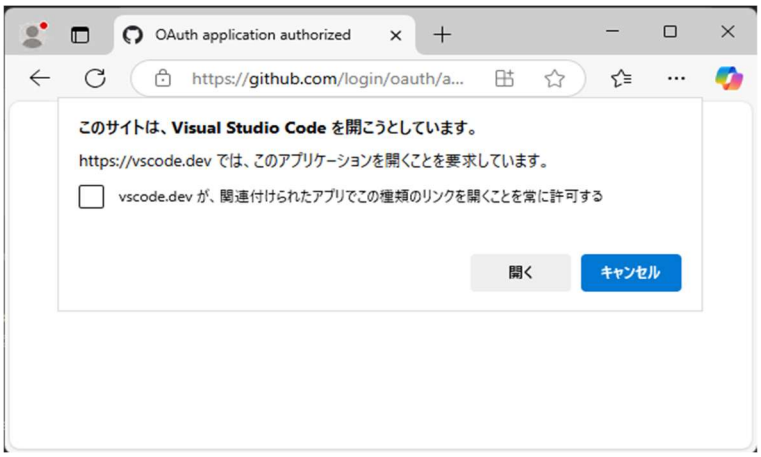
GitHub Copilot の設定

インストールが完了すると「sgine in 」を求めされます。GitHubCopilot の利用には、GitHub Copilot サブスクリプションにアクセスする必要がありますので、ご自身の GitHub Copilot サブスクリプションのアカウントでログインしてくださ。
Copilot のサブスクリプションをまだお持ちでない場合は、Copilot Free プランにサインアップします。
Create your free account のサイトで、無料アカウントを作成できます。

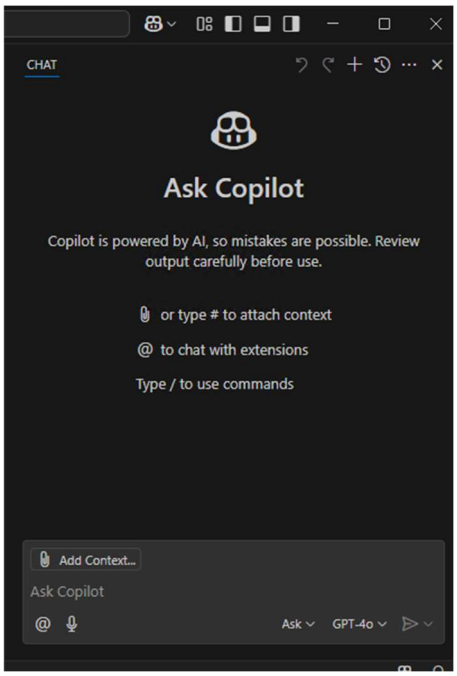


https://github.com/signup?return_to=https%3A%2F%2Fgithub.com%2Fcopilot&source=login

認証用の画面をブラウザが起動しますので、ログインすると、Web の画面から VsCode の起動が要求されますので、認証すると、ブラウザから VSCode を開く（起動）するように指定されますので、「開く」をクリックしてください。



その後、VSCode が再起動すると GitHub Copilot が認証した状態になり、Chat が出来る状態になります。



3. 最後に

本補足資料では、誌面に掲載しきれなかった複数の LLM モデルに対する評価結果や、検証時に使用した具体的なプロンプト例、さらに各種ツールのインストール手順について詳しく解説しました。特に、コーディング支援における LLM の実用性や性能差については、具体的なソースコードの検証を通じて明確に比較しています。また、Docker や Ollama、OpenWebUI などを活用したローカル環境の構築手順も紹介しており、読者が自身の開発環境に導入しやすいよう配慮しています。是非、ご自身の環境で LLM を活用したコーディング体験を進めてみてください

※：本資料に掲載されている各ツールの画面キャプチャは、それぞれの開発元のガイドラインに基づき掲載しています。