

AIコーディングでレイヤーの低いコードの実装はできるか

hiro0920_六合

自己紹介

- バックエンドエンジニアと名乗りたいものの、ミドルウェアを書いているエンジニア
- 主にCとC++、WebAssemblyを出力するのにRust、機能拡張しやすさのためのモジュールに向けてGolangなどを書いています。
 - FluentBitとかFluentdとか
 - RubyのC拡張とか
 - WebAssemblyとか
 - 文字コードとか色々

前提

- 大部分はChatGPTの無料枠(ChatGPT-4o)で試したものです。
- ChatGPT-4oでは実験しきれない箇所のみGemini 2.5 Proも使用しています。

諸注意

- このスライドはhiro0920_六合個人の実験結果であり、所属会社の意見を代表するものではありません。
- LLMなどのAIを使ってバグ報告はこれまでにないほど手軽にすることができるようになりました。
しかし、AIのhallucinations(ハルシネーション)を見破れない領域で気軽にバグ報告を生成するのを推奨するものではありません。
- 生成AIは用法と良識を守って使ってください。生成AIを足掛かりにするのは大丈夫ですが、AIの出力をそのままコピーしてしまうとコードの品質を保つのが難しくなります。AIはあくまで補助として利用し、最終的には人間が内容を十分に理解・検証し、自身の責任でコードを作成・提供しましょう。

CHALLENGE #1 SIMD

- SIMDとは一回のCPUの計算サイクルに対して複数の値の一連の組を演算する命令
 - X86_64のSSE, AVX命令
 - AArch64(ARM 64bit)のNEON命令
 - RISC-VのRISC-V Vector(RVV)命令

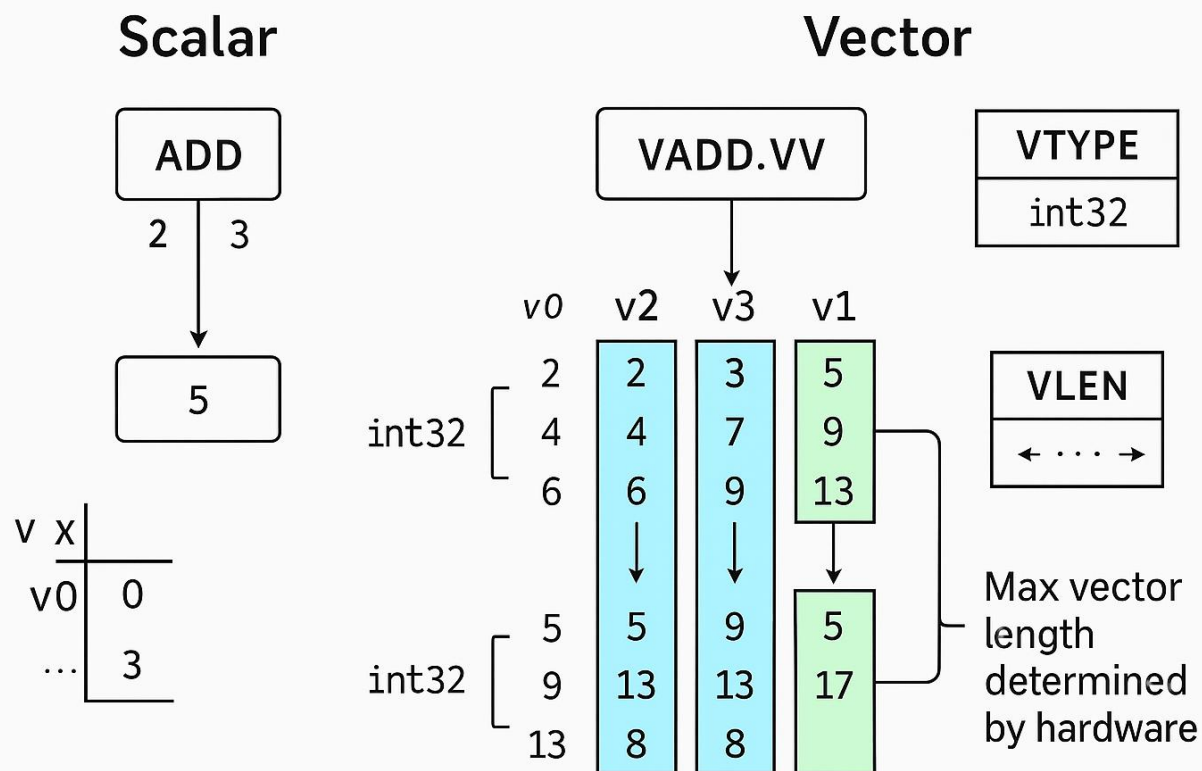
などがあります。

例) 音声処理や画像処理で一つ一つのデータを操作していくよりも、依存関係のない塊ごとに演算を行ったほうが効率的になることがある。

RVV – RISC-VのSIMD

RVVも他のSIMD命令と同じく、
複数の数値の組を
一度に演算でできる。
ただし、命令長は固定ではない。
VLENが命令長にあたり、
バイト境界さえ守れば長くできる。

将来的に異なるVLENを持つ
プロセッサが出てきても
対応可能にしたい。



SIMD命令を使っているコードの潜在的な問題を指摘させる

- 使ったのはFluentBit(ログ収集・転送ツール)のflb_simd.h(文字列演算をSIMD演算を使って速くするヘッダーファイル)
 - ChatGPTの入力フォームで入力長が溢れないうえ、程よく複雑に入り組んでいるコードだったため
- ChatGPTにまずはコンテキストが溢れない程度にコードを貼って指摘させてみる
 - 指摘された全文
 - <https://chatgpt.com/share/681d4eea-2284-800e-a775-da81f7c16a95>
 - ChatGPTのRVVにかかわる箇所の指摘ではVLENというRVVの命令長を暗黙に仮定している危険性を指摘された

```
#define RVV_VEC8_INST_LEN (128 / 8) /* 16 */  
#define RVV_VEC32_INST_LEN (128 / 8 / 4) /* 4 */
```

CHALLENGE #1 -- 問題点の指摘 #2

- 試行錯誤するうち、
 - `__riscv_vsetvl_e8m1(16);`
 - `__riscv_vsetvl_e32m1(4);`
 - を使うことによりハードウェア依存のRVVのVLEN長を取得できるinstructionがあることが分かった。

詳細な説明) 例えば `__riscv_vsetvl_e8m1(n)` は、要素の型 (e8: 8ビット整数)、要素のグルーピング (m1: グループ化なし) を指定し、処理したい要素数 `n` を渡すことで、実際にハードウェアが一度に処理できる要素数 (`vl`) を設定し、その値を返すものです。

RVVのコードを書いた当初はRVVの命令長のVLENを暗黙的に固定したコードを書いていた。

また、ChatGPTに指摘させる前には気づいていなかった。

CHALLENGE #1 -- 問題点の指摘 #3

```
#include <riscv_vector.h>

static size_t vec8_vl_cached = 0;
static size_t vec32_vl_cached = 0;

static inline size_t flb_get_vec8_vl(size_t n)
{
    if (n != vec8_vl_cached) {
        vec8_vl_cached = __riscv_vsetvl_e8m1(n);
    }
    return vec8_vl_cached;
}

static inline size_t flb_get_vec32_vl(size_t n)
{
    if (n != vec32_vl_cached) {
        vec32_vl_cached = __riscv_vsetvl_e32m1(n);
    }
    return vec32_vl_cached;
}
```

ChatGPTからこのようなコードが提案される
→ただしこのままでは使えない。
なので、これを参考にしたものを実機※で
実験すると動かすことができた。

※DC-ROMA RISC-V LAPTOP II

CHALLENGE #1 -- ここまでのまとめ

- ChatGPT-4oモデルでは無料枠の範囲内でも仕様の文脈から大きく文脈を伴わないものに関しては潜在的なバグを指摘できる。
 -  RISC-VのRVVは扱っている人が少ないので、裏取りをしつつ検証用の相手として適していました
 -  ChatGPT-4oに対してのプロンプトではすべて英語でやり取りしたので、日本語ではうまく動かない可能性があります
 - 一般的に大規模言語モデルは英語の学習データ量が最も豊富であり、複雑な技術的指示やコード生成において、英語の方がより精度の高い応答が期待できる場合があります。

CHALLENGE #2 -- WINDOWS API の使用ミス探し

- Windows API のWinsock2内に以下のAPIがある

Syntax

C++

Copy

```
int WINAPI send(  
    [in] SOCKET      s,  
    [in] const char *buf,  
    [in] int         len,  
    [in] int         flags  
);
```

このソケット通信を行う
APIの使用方法のミスを見つけること
はできるか

CHALLENGE #2 -- WINDOWS API の使用ミス探し

```
struct mk_iovec
{
    void    *iov_base;    /* Base address of a memory region for input or output */
    size_t  iov_len;      /* The size of the memory pointed to by iov_base */
};

31 static inline ssize_t writev(int fildes, const struct mk_iovec *iov, int iovcnt)
32 {
33     int i;
34     uint32_t bytes_written = 0;
35
36     for (i = 0; i < iovcnt; i++) {
37         int len;
38
39         len = send((SOCKET)fildes, iov[i].iov_base, (int)iov[i].iov_len, 0);
40         if (len == SOCKET_ERROR) {
41             uint32_t err = GetLastError();
42             // errno = win_to_posix_error(err);
43             bytes_written = -1;
44             break;
45         }
46         bytes_written += len;
47     }
48
49     return bytes_written;
50 }
51
```

このコード片からAPIの使用ミスを見つけることは可能なのか？

ここでのmk_iovec構造体とwritev関数は、UNIX系のOSで標準的なiovec構造体とwritevシステムコール(複数のメモリ領域のデータを一度のシステムコールで書き出す機能)をWindows上で模倣しようとしたものです。iov_baseがvoid*型になっているのはそのためです。

CHALLENGE #2 -- WINDOWS API の使用ミス探し

3. Implicit Conversion of `void *` to `const char *` in `send()`

- **Problem:** MSVC may warn or error on type mismatch when calling `send()`.
- **Fix:** Cast properly:

```
c                                     コピーする   編集する

len = send((SOCKET)fildes, (const char *)iov[i].iov_base, (int)iov[i].iov
```

このコード片からAPIの使用ミスを見つけることは可能なのか？

AIの回答

CHALLENGE #2 -- WINDOWS API の使用ミス探し

```
struct mk_iovec
{
    void    *iov_base;    /* Base address of a memory region for input or output */
    size_t  iov_len;      /* The size of the memory pointed to by iov_base */
};

31 static inline ssize_t writev(int fildes, const struct mk_iovec *iov, int iovcnt)
32 {
33     int i;
34     uint32_t bytes_written = 0;
35
36     for (i = 0; i < iovcnt; i++) {
37         int len;
38
39         len = send((SOCKET)fildes, iov[i].iov_base, (int)iov[i].iov_len, 0);
40         if (len == SOCKET_ERROR) {
41             uint32_t err = GetLastError();
42             // errno = win_to_posix_error(err);
43             bytes_written = -1;
44             break;
45         }
46         bytes_written += len;
47     }
48
49     return bytes_written;
50 }
51
```

このコード片からAPIの使用ミスを見つけることは可能なのか？

第二引数が(char *) または (const char *)を要求しているが、渡ってくるのはvoid*ポインターと見分けていた。

C言語ではvoid*から他のポインタ型への暗黙的な変換が許容されることが多いですが、最近のバージョンのMSVCではより厳格で、このような変換は明示的なキャストが必要です。

CHALLENGE #2 -- WINDOWS API の使用ミス探しのまとめ

- Windows APIに関してもMSDNのドキュメントから学習しているようす。
- コンテキストを絞って指摘させると確かに人間が判別してミスをしていると判定できる箇所と同じ個所を指摘していた。
 - この実験は既にAIを使わない修正をした後にAIにも検証させたものです。
 - 指摘された全文
 - <https://chatgpt.com/share/68246a02-814c-800e-a396-1697cb06da04>
- ⚠ ChatGPT-4oに対してのプロンプトではすべて英語でやり取りしたので、日本語ではうまく動かない可能性があります(再掲)
 - 一般的に大規模言語モデルは英語の学習データ量が最も豊富であり、複雑な技術的指示やコード生成において、英語の方がより精度の高い応答が期待できる場合があります。(再掲)

CHALLENGE #3 RUBYの言語バインディング

- Ruby言語にはFFIとしてCの関数を呼ぶ仕組みがある
 - この仕組みはRubyの開発者はとても書きやすく、読みやすくかけると主張している。しかし、ドキュメントは詳しくは書かれておらず、その辺に転がっているソースコードを読むしかない。
 - →これをChatGPTのLLMモデルで書くことはできないだろうか？

FFI(Foreign Function Interface)とは、あるプログラミング言語(ここではRuby)から、他の言語(ここではC言語)で書かれた関数やライブラリを利用するための仕組みのことです。

CHALLENGE #3 RUBYの言語バインディング

- これまでのRubyの言語バインディングの問題
 - SWIGを用いるとほぼ自動的にライブラリの言語バインディングを作成できるものの、使い勝手と命名規則が壊滅的になる
 - 例) ゲッターにget_xxx()と付いてしまったり、セッターにset_xxx()だけが生成される。
 - →本来はxxx()とxxx=()でよいはず

使い方) https://www.swig.org/Doc1.3/Ruby.html#Ruby_nn11

SWIGが生成するコードは例えばget_attribute()やset_attribute(value)といった他言語風の命名規則になりがちで、Rubyプログラマにとっては使い勝手が悪くなる場合があります。

CHALLENGE #3 RUBYの言語バインディング

- 以下の関数シグネチャを持ったRubyの言語バインディングを作りたい

```
int _setmaxstdio( int new_max );  
int _getmaxstdio( void );
```

- 動機としては長く動くアプリケーションで一時的にI/Oのファイル数制限を緩和したいから。

CHALLENGE #3

RUBYの言語バインディング

- 以下の関数シグネチャを持った
Rubyの言語バインディングを作りたい

```
int _setmaxstdio( int new_max );  
int _getmaxstdio( void );
```

おおよそOK

```
#include <windows.h>  
#include <stdio.h>  
#include "ruby.h"  
  
static VALUE mWin32;  
static VALUE cStdIO;  
  
/* Getter: int _getmaxstdio(void); → Ruby: Win32::StdIO.maxstdio */  
static VALUE stdio_get_maxstdio(VALUE self) {  
    return INT2NUM(_getmaxstdio());  
}  
  
/* Setter: int _setmaxstdio(int); → Ruby: Win32::StdIO.set_maxstdio and maxstdio= */  
static VALUE stdio_set_maxstdio(VALUE self, VALUE val) {  
    int new_max = NUM2INT(val);  
    int result = _setmaxstdio(new_max);  
  
    if (result == -1) {  
        rb_sys_fail("_setmaxstdio failed");  
    }  
  
    return INT2NUM(result);  
}  
  
void Init_stdio_ext(void) {  
    mWin32 = rb_define_module("Win32");  
    cStdIO = rb_define_class_under(mWin32, "StdIO", rb_cObject);  
  
    // Getter: maxstdio  
    rb_define_method(cStdIO, "maxstdio", stdio_get_maxstdio, 0);  
  
    // Setter: set_maxstdio  
    rb_define_method(cStdIO, "set_maxstdio", stdio_set_maxstdio, 1);  
  
    // Setter alias: maxstdio=  
    rb_define_method(cStdIO, "maxstdio=", stdio_set_maxstdio, 1);  
}
```

RUBYの言語バインディングを生成する時の問題点

- プロンプト全文

- <https://chatgpt.com/share/6826ce3f-c26c-800e-86cf-45fd312016bf>

さらに、以下の追加ルールに従って実装する必要があります：

- setter（セッター）は、元の関数名の代わりに `set_` プレフィックスを付けて提供する必要があります
- setter は、`=` を伴った名前でも提供する必要があります。その際は元の関数名の `set` プレフィックスを削除して使います
- getter（ゲッター）は、元の関数名に含まれる `get` プレフィックスを取り除いた名前で提供されるべきです（例： `getmaxstdio` は `maxstdio` に）
- 同じ関数でも、`rb_define_method` の第2引数を変更することで、異なる名前で2回定義して使用することが可能です

例：

```
c 📋 コピーする ✎ 編集する  
  
rb_define_method(rb_cIOExt, "set_maxstdio", ioext_set_maxstdio, 1);  
rb_define_method(rb_cIOExt, "maxstdio=", ioext_set_maxstdio, 1);
```

これらは、実際の実装内部では同じ関数を使用しますが、Rubyインタプリタに公開される名前は異なります。

AIに指示する時は明確な機能要件と命名規則の指示が必要。
Rubyの言語バインディングを作る際の命名規則は、少数の方々の共通認識にとどまるようです。

おまけ – RUBYのFIDDLEを用いた場合

- RubyにはfiddleというRubyでのみ完結する言語バインディングの仕組みがあります。
- 以下の関数シグネチャを持ったRubyの言語バインディングを作りたい

```
int _setmaxstdio( int new_max );  
int _getmaxstdio( void );
```

おまけ – RUBYのFIDDLEを用いた場合

- プロンプト全文

- <https://chatgpt.com/share/6826e50e-8624-800e-b8a8-6c9addfcf4f6>

```
require 'fiddle'
require 'fiddle/import'

module CRT
  extend Fiddle::Importer

  # Load the Microsoft C runtime library
  # Use 'ucrtbase' for modern Windows (Visual Studio 2015+),
  # fallback to 'msvcrt' for older versions.
  begin
    dlload 'ucrtbase.dll'
  rescue Fiddle::DLError
    dlload 'msvcrt.dll'
  end

  extern 'int _setmaxstdio(int)'
  extern 'int _getmaxstdio(void)'
end
```

こちらの方式のほうが簡単に作成できて、
C言語のソースをビルドする必要もない！

FiddleはRubyに標準で添付されているライブラリで、
libffi (Portable Foreign Function Interface
Library) のラッパーです。

ここまでのまとめ

- ChatGPT-4oでもプロンプトに与えるコンテキストを工夫して指示を出すとAIにもある程度低いレイヤーのコードの解釈、修正が可能な様子
- AIモデルの学習していないコードベースではまた違った結果となりそうです
- 機能要件を省いて出力すると、言語化していない部分でコーディング規則を破るコードが出力されて、修正が必要になってしまいました
- 非機能要件としてファイルやモジュールを分ける機能はかなり未熟に感じました

ここまでのまとめ

- ChatGPT-4oでもプロンプトに与えるコンテキストを工夫して指示を出すとAIにもある程度低いレイヤーのコードの解釈、修正が可能な様子
- AIモデルの学習していないコードベースではまた違った結果となりそうです
- 機能要件を省いて出力すると、言語化していない部分でコーディング規則を破るコードが出力されて、修正が必要になってしまいました
- 非機能要件としてファイルやモジュールを分ける機能はかなり未熟に感じました
- ~~Rubyの言語バインディング生成を試そうと始めたら、その週だけWindowsでのRubyのC拡張のツールチェーンが壊れていました(現在は修正済み)~~
- 適切にコンテキストを切って課題設定したものをAIに投げるのが一番大事

WEBASSEMBLYの問題探しー 失敗例

- ChatGPT-4oでも込み入ったコンテキストの機能追加修正はうまくいかなかった
- 例えば、[WebAssembly](#)の[Reference Types](#)の有効無効でWasmのロードが失敗する問題があった
 - しかしこれは修正だけ見ると、↓の一行で済んでしまう

```
if (NOT MSVC)
    set (WAMR_BUILD_LIB_PTHREAD 1)
endif ()
```

```
- set (WAMR_BUILD_REF_TYPES 0)
+ set (WAMR_BUILD_REF_TYPES 1)
```

```
if (NOT MSVC)
    # linker flags
```

WebAssemblyのReference Typesは、数値型だけでなく、関数への参照 (funcref) やホスト環境のオブジェクトへの不透明な参照 (externref) をWebAssemblyモジュールが直接扱えるようにする拡張機能です。

WAMR (WebAssembly Micro Runtime) は、主にIoTデバイスや組み込みシステム向けに設計された、軽量で高性能なスタンドアロンのWebAssemblyランタイムの一つです。

WEBASSEMBLYの問題探しー #2

- ChatGPT-4oでも込み入ったコンテキストの機能追加修正はうまくいかなかった
- ChatGPT-4oがしてきた提案はどれも的外れなものであった
 - やり取りの全文
 - <https://chatgpt.com/share/682c4a15-d060-800e-9680-407da649ee53>

WEBASSEMBLYの問題探しー #3

- こちらからRustc 1.81以降ではWebAssemblyの出力の挙動が変わったんだ、と情報を与えることで、Rustc 1.81での変更点を挙げてくれることができた

✓ TL;DR

Yes, enabling `WAMR_BUILD_REF_TYPES=ON` is necessary and effective for handling modern Wasm binaries generated by Rust ≥ 1.81 , which may implicitly use reference types (like `funcref`, `externref`) even without obvious use of them in your code.

🔍 Why This Happens

🦋 What are Reference Types?

Reference types (from the [reference types proposal](#)) add:

- `funcref` — references to Wasm functions
- `externref` — references to host values (like strings or objects)

These are not part of the original Wasm MVP spec. **Older builds of WAMR** without `WAMR_BUILD_REF_TYPES` support **reject binaries** that use these types in signatures or tables.

🦋 What's Up with Rust ≥ 1.81 ?

Starting from Rust 1.81:

- The LLVM backend **may emit reference types automatically**, especially if any `externref`-like behavior is inferred.
- Even if you're not using `externref` explicitly, `wasm-bindgen` or subtle use of `fn()` or function pointers can result in `funcref` usage.

So even a simple `wasm32-unknown-unknown` binary may now contain reference types.

GEMINI 2.5 PROの場合 — #4

- Gemini 2.5 Proで問題のログを質問すると、WebAssemblyのコンパイラの問題では？という回答が返ってきた。
 - 実に惜しい。
実はランタイムで無効にしていた機能が悪さをしていたため。
- 回答 <https://g.co/gemini/share/a86ffd895125>

2. Troubleshooting and Resolving:

- **Check Rust Compilation Target Features:** When you compile your Rust code to WASM (e.g., for the `wasm32-unknown-unknown` or `wasm32-wasi` target), the Rust compiler (via LLVM) can enable or disable specific WebAssembly features. You might need to explicitly disable features that are not supported by your WAMR instance.
- You can try to control these features using the `RUSTFLAGS` environment variable or by specifying target features in your `.cargo/config.toml` file. For example, to disable SIMD and other features, you might try something like:

Bash



```
RUSTFLAGS='-C target-feature=--simd,-bulk-memory,-reference-types' cargo
```

The exact features to disable can be a bit of trial and error. Common potentially problematic "newer" features include `simd128`, `bulk-memory`, `reference-types`, `multi-value`, `sign-ext`. Start by disabling a few and see if it resolves the issue.

- Refer to the [Rust documentation on target features](#) and the [Rust blog post on WebAssembly target feature changes](#) for more details.

GEMINI 2.5 PROとCHATGPTの比較まとめ

- ChatGPT-4oの推論単体の提案ではカバーしきれない範囲までGemini 2.5 Proのモデルはあたりをつけていた
- どこまでコーディングを任せられるかはモデルの挙動と学習したリソースのカバー範囲に依存する
- 実際の人間が行うコーディングはWeb上の資料やRFCを参照することも多いため、コストが許せばGemini 2.5 Proを使ってコーディングの補助や発想のヒントを得ることができそう

最後に

- このスライドは各所に生成AIを使って作ったものが含まれています。
- 皆さんお気づきになりましたか？

最後に

- このスライドは各所に生成AIを使って90%以上作ったものが含まれています。
- 皆さんお気づきになりましたか？
 - RVVの説明のポンチ絵
 - 各所に散りばめられているグレーの技術用語の説明

が生成AIが作成したものをベースに作られたものでした。



ご清聴ありがとうございました

時間が余ったら話す用

- Fluent Bitのstrptimeという時刻フォーマットのパーサーはタイムゾーンを入れたパースができていなかった
 - これをGemini 2.5 Proでコーディング補助を回すことは可能か？

```
struct flb_tm my_tm;
```

```
flb_strptime("10:30:00EDT", "%H:%M:%S%z", &my_tm);
```

とか

```
flb_strptime("2023-01-10 10:00:00 PST", "%Y-%m-%d %H:%M:%S %Z", &my_tm);
```

とか

```
flb_strptime("14:00:00 cest", "%H:%M:%S %Z", &my_tm);
```

とか呼んでタイムゾーンを文字列から取得したい。

時間が余ったら話す用

- Fluent Bitのstrptimeという時刻フォーマットのパーサーはタイムゾーンを入れたパースができていなかった
 - これをGemini 2.5 Proでコーディング補助を回すことは可能か？
 - **結果→可能**

時間が余ったら話す用

- Fluent Bitのstrptimeという時刻フォーマットのパーサーはタイムゾーンを入れたパースができていなかった
 - これをGemini 2.5 Proでコーディング補助を回すことは可能か？
 - ChatGPT-4oでは入力に溢れる長さの実験としてはちょうどよい長さ(～1000行程度)のため。
 - **結果→可能**
 - プロンプト全文
 - <https://g.co/gemini/share/f398ac1ae320>
- 既存のコードに合った追加のコードを生成して組み込むことが可能だった