AAS

**ABSTRACT:**12313

**Keywords:** 45646

#####主體架構參考paper#####

# 參考文獻

我要寫的是，不同語言模型，然後給不同prompt，去比較生成結果

# Introduction(新版)

近年來，隨著大型語言模型（Large Language Models, LLM）與自然語言處理（Natural Language Processing, NLP）技術的迅速發展，自動程式碼生成已成為研究與應用領域中的關注焦點。GPT-4、Claude、Gemini 等模型已被證實能透過自然語言提示成功執行程式碼生成任務（Chen et al., 2024），為開發流程帶來重大轉變。LLM 具備從直觀語言輸入產生結構化程式碼的能力，不僅簡化了程式設計流程，更有潛力顯著加速軟體開發週期（Haque, 2025）。此外，這類模型亦能生成自然語言文本，提供具情境關聯的即時回應，廣泛應用於寫作輔助、教材補充與程式語言學習指引等情境（Steiss et al., 2024）。LLM之所以備受關注，主要來自其在多種應用情境中的高適應性與即時回應能力。第一，LLM能夠提供隨需支援，讓使用者在程式開發過程中隨時獲得協助，特別適用於自我導向學習或獨立開發的場景(Cao & Dede, 2023)。第二，LLM具備因應不同學習風格的潛力，能透過多種方式(如自然語言說明、簡明摘要、逐步教學)呈現內容，進一步提升使用者的學習效率與實作表現(Pesovski et al., 2024)。第三，LLM也展現出在程式碼撰寫上的輔助價值，能協助進行程式碼補全、除錯建議與重構優化，為程式教育與實務開發帶來新的可能性(Khazanchi & Khazanchi, 2024)。

相較於傳統的程式理解與修復技術，LLM 在程式碼生成方面展現出顯著優勢。多項研究指出，LLM 不僅能識別與修正原始碼錯誤，亦能掌握程式語意以產生有效程式碼，整體效能優於以往方法（Rasnayaka et al., 2024；Xia et al., 2022；Zeng et al., 2022）。Xia et al.（2022）進一步強調，預訓練語言模型於程式修復任務中表現出色，能精確執行完整函式生成、程式碼補全與單行修正，並展現跨語言與多開發環境的高度泛化能力。Cao et al.（2025）則探討提示設計對除錯效能的影響，藉由引入高依賴性與複雜性錯誤程式，有效評估 LLM 於真實除錯任務中的應用表現。然而，儘管 LLM 在程式碼生成上已有實質突破，其表現仍難以與自然語言生成任務相提並論(Espejel et al., 2023)。生成可執行且語意正確的程式碼相較於文本生成更具挑戰性，主因在於模型需同時滿足自然語言指令的準確性與程式語法、語意的嚴格要求（Wong et al., 2023）。即便是細微錯誤，如缺漏分號、括號或冒號，皆可能導致編譯失敗或邏輯錯誤，顯示出程式碼生成對語言理解與邏輯推理能力提出更高門檻。

在實際開發用途中，儘管主流LLM語言理解任務與程式碼生成任務中展現出卓越表現。然而在應用推廣仍面臨顯著挑戰。第一，目前大多數表現最佳的LLM屬於封閉源碼（Closed Source），使用者無法存取模型架構與訓練細節，限制了研究再現性與應用可塑性。第二，這些模型高度依賴高效能圖形處理器（GPU）進行推論，造成「可及性」與「運行成本」的雙重障礙，特別對個人開發者與資源有限的小型機構影響深遠。高階GPU設備價格昂貴，且通常不隨一般消費級電腦提供，使得高效運行這些模型的能力不具普遍性。此外，建置與維護這類運算基礎設施所需經費龐大，可能迫使機構調整預算配置，進而排擠其他關鍵研究與開發活動，削弱其在AI創新生態系中的參與度與競爭力。此外，即便部分LLM以開放介面提供免費存取，使用過程中仍可能牽涉資料上傳與雲端處理，導致潛在的隱私與資安風險（Wu et al., 2023）。特別在開發含有敏感資訊的應用場景中，缺乏對模型內部運作與數據處理流程的透明控制，進一步提高資料外洩的憂慮。

為解決上述問題，研究逐漸將焦點轉向參數較少、運算效率更高的輕量級模型。儘管如 PaLM（Chowdhery et al., 2023）、GPT-3（Brown et al., 2020）與 GPT-4（OpenAI, 2023）等代表性模型分別擁有 1750 億、5400 億至高達 1.7 兆個參數，顯示其龐大的語言建模能力，但其龐大模型規模亦導致部署成本與計算資源需求相對提高。相對地，新一代小型模型如 Mistral（Jiang et al., 2024）與 LLaMA（Dubey et al., 2024）僅以約 70 億參數，在多項任務上展現出與大型模型相當，甚至在特定應用情境中更具優勢的效能表現。這類模型在推論效率與硬體資源需求間取得更佳平衡，為低資源場景下的應用提供可行解方。此外，量化技術的興起也有助於加速模型推論。透過將模型權重轉換為低精度格式，大幅降低記憶體與運算需求的同時，盡可能保留下游任務中的效能表現。例如 Activation-aware Weight Quantization（AWQ）（Lin et al., 2024）與 Generative Pre-trained Transformers Quantization（GPTQ）（Frantar et al., 2022）等方法，已被廣泛應用於壓縮 LLM，並在多項任務中維持接近原始模型的準確度。而為了支援量化模型於低資源環境中部署，工具函式庫如 LlamaCPP（Khosravi, 2024）提供了在 CPU 上執行已量化模型的能力。LlamaCPP 為一種高效的二進位儲存與推論格式，專為 GGUF（GPT-Generated Unified Format）檔案設計，優化了模型載入速度與記憶體使用。使得 LLM 得以在無需高階 GPU 的情況下運行，進一步降低技術進入門檻，擴展其在各類邊緣設備與本地環境中的應用潛能。

除了模型架構與推論效率的優化，提示工程（Prompt Engineering）的發展亦成為提升 LLM 性能的關鍵策略之一。透過精確設計的提示語句，可引導模型生成更具語境關聯與任務對應性的回應（Alier et al., 2024）。提示工程使使用者無需對模型進行再訓練，便能以低成本激發其潛在知識與推理能力，進而擴展模型的應用範疇。具代表性的策略包括 zero-shot 與 few-shot。zero-shot 允許模型僅根據任務描述即執行新任務，few-shot 則藉由提供極少數範例，強化模型對任務模式的理解與應答準確性（Brown et al., 2020）。Chain-of-Thought（CoT）則是將複雜任務拆解為一系列邏輯步驟，逐步引導模型進行中介推理，有效提升其在數學推論、邏輯判斷等高階認知任務中的表現（Zhang et al., 2022）。透過這類提示設計策略，模型不僅能展現出更強的泛化能力，也進一步降低應用門檻，拓寬 LLM 於教育、程式輔助與語意理解等多元場景中的應用潛力。

寫引言節尾跟研究問題

本研究以作業系統課程中的 Pthread 教學為場域，探討 ChatGPT 所生成之 Pthread 程式碼在教學上的應用潛力與侷限。我們提出兩個核心研究問題：

RQ1：不同類型的提示（zero-shot、few-shot、CoT）在 Python 程式碼生成任務中的效能差異為何？

RQ2：Prompt phrasing（語序、語義、語法結構）如何影響 LLM 的生成正確率？

RQ3：在不同提示下，LLM 在語法正確性、邏輯正確性、執行成功率三方面表現有何差異？

RQ4：CoT Prompting 是否顯著提升模型對於邏輯與函式設計任務的表現？

# Related Work：

## LLM 在軟體工程與平行程式設計中的應用

強調LLM於軟體工程與平行程式的**任務導向應用性**，聚焦其在真實開發與並行環境中的效能與挑戰

* LLM於軟體工程任務的應用（程式修復、摘要、API理解）
* LLM於高效能運算（HPC）與平行程式碼的研究應用（如OpenMP pragma生成、平行化潛力分析）
* 專案脈絡下的少樣本提示選擇、錯誤修復任務
* LLM輔助平行化、錯誤偵測、程式碼建議（含ChatGPT、GitHub Copilot）

大型語言模型(LLMs)已在多個領域展現出變革性的潛力，包括軟體工程、數學(Guo, et al., 2023)、高效能運算(HPC)( Chen, et al., 2023) 以及自然語言處理(Cahyawijaya, et al., 2021)。在軟體工程領域中，LLMs 被廣泛應用於程式修復、程式碼摘要與程式碼生成(Ahmed & Devanbu, 2022)。

以高效能運算(HPC)來說，LLM 在 HPC 研究中也受到越來越多關注。例如 Chen, et al., (2023) 開發了模型 LM4HPC，應用於程式碼生成與效能優化。Ding et al., (2023)開發出了基於 LLaMA 模型的 HPC-GPT，透過對 HPC 領域的問答資料進行微調，以擴展 LLM 在高效能運算領域的應用能力。

同時，LLMs 也被用於解決 API 相關的挑戰，例如偵測 API 誤用與回答 API 相關問題。例如 Yang et al., (2023) 開發了基於 LLM 得模型 APICKnow，用於從文本中擷取 API 實體及其語意關係。該研究使用 Stack Overflow 資料集進行實驗，證實 APICKnow 無需大量標註資料即可達到高準確率。Huang et al., (2022)則提出透過微調 code-masked 語言模型(MLM)，來解決程式碼片段中 fully qualified name(FQN)解析的問題。這些研究不僅凸顯了 LLM 的能力，也指出其可改進之處，並提供實際應用上的洞見。

Cao et al., (2025) 評估了 ChatGPT 在深度學習程式修復上的能力，透過不同提示設計檢測與修復錯誤。亦強調預訓練語言模型在程式修復方面的優勢，尤其是在跨程式語言與程式碼模式上的泛化能力。

在與程式任務相關的少樣本學習程序中，Nashid et al., (2023) 提出了一種基於檢索的提示選擇方法，引導語言模型進行程式碼修復與測試斷言生成。該方法顯著提升了表現，凸顯了檢索式提示策略在強化少樣本學習中的價值。Ahmed & Devanbu (2023) 聚焦於專案特定的程式碼摘要生成，利用具有專案脈絡的範例進行少樣本訓練，以微調 LLM，進而提升摘要的相關性與品質。

透過多核心處理器的應用，平行程式設計(Pthreads)允許多個執行緒同時運行，進而提升程式效能。儘管執行緒管理能帶來諸多效益，但同時也可能引發如並行錯誤等挑戰，尤其是資料競爭(data races)。當多個執行緒在未正確同步的情況下同時存取同一變數時，就可能發生資料競爭，進而導致不可預期的行為甚至系統當機。為避免此類問題，程式中會實作同步機制，例如條件變數(condition variables)與互斥鎖(mutex locks)，以確保執行緒安全。

要解決資料競爭問題，需要有效的偵測與修復方法。過去與近期的研究提出了多種偵測與修復策略，包括靜態分析(static analysis)、動態分析(dynamic analysis)與執行階段驗證(runtime verification)等方法。(Shi et al., 2024)。這些技術可用來偵測，甚至在某些情況下修復資料競爭。Serebryany et al.,(2011) 提出一種整合於 LLVM 編譯器的動態競爭偵測方法，透過編譯時期插裝技術強化 ThreadSanitizer 的功能。Liew et al., (2024) 則針對 GPU 程式提出一種健全但部分完備的靜態分析技術，結合靜態分析與領域特定的抽象來應對 GPU 並行性的挑戰。Choi et al., (2022) 結合靜態與動態分析，發展出一種能精確偵測多執行緒物件導向程式中資料競爭的方法。Malakar et al., (2024)提出自動化工具 RaceFixer，能分析並行問題並推薦適當的同步機制(如互斥鎖)來修復資料競爭。例如

雖然已有多種技術用於偵測與修復資料競爭，但 LLM 具備獨特優勢，使其在現代平行程式設計中極具價值。近年來，研究顯著推進了人們對 LLM 在平行程式設計中應用潛力的理解。Kadosh et al., (2023) 使用源碼的圖形表示結合 Transformer 模型，預測 OpenMP 的 pragma 並分析迴圈的平行化潛力。OpenMP(Open Multi-Processing) 是一個 API，可讓開發者在共享記憶體多平台上撰寫平行程式碼。Chen et al., (2024)提出 OMPGPT 一款專為此領域設計的 LLM，能協助產生 OpenMP 的 pragma，展現出領域特定 LLM 在自動平行化方面的潛力。包括 ChatGPT 與 GitHub Copilot 在內的工具，也已被應用於平行程式碼生成(Mišić & Dodović, 2024)及 OpenMP 並行錯誤偵測的研究(Alsofyani & Wang, 2024)。這類研究凸顯 LLM 在平行程式設計中的重要性，特別是在提供智慧程式建議與自動化複雜任務(如迴圈平行化與語法結構生成)方面的優勢。LLMs 能適應多種程式設計範式，並從龐大的程式碼資料庫中持續學習，使其能夠不斷演化，並有效因應快速變化的技術環境。結合其高可用性與能整合錯誤偵測、修復與解說等任務的能力，使得 LLMs 成為推進平行程式設計與培育未來開發者的關鍵工具。

## LLM 生成程式碼的品質與效率問題

此節可強調您研究動機中的\*\*「效率瓶頸與挑戰」\*\*，為後續研究目的鋪陳基礎。

* 稀缺資源場景（IoT、邊緣裝置）下的挑戰，LLM常見的效率低落原因（如 token 預測傾向局部補全、prompt 含糊造成幻覺）
* 程式碼正確性、效率、記憶體與運算資源的評估指標
* GPT與人類程式設計者的效能比較（正確性 vs. 效率）
* 基準數據集（HumanEval、MBPP、EFFIBENCH）的評估方式（含 pass@k）

程式碼效率常見的衡量指標包括記憶體使用量、CPU 使用率、執行時間(Run Time)與程式碼複雜度。Dou et al., (2024)與 Hou and Ji (2025)比較了 LLM 與人類程式設計師的表現。發現 GPT-4 雖然在結構化任務中表現優異，但在面對模糊任務時表現不佳，而人類設計師撰寫的程式碼在執行時間與記憶體使用方面通常更有效率，突顯出 LLM 與人類解法在優化方面的落差。Niu et al., (2024)的研究中，除了正確性外，研究也納入執行時間作為效率指標，並在 HumanEval、MBPP 這些基準資料集中(聚焦 Python 問題)，與 LeetCodeEval 資料集(涵蓋 C++ 題目)上進行測試。結果指出，LLM 的執行效率仍落後於人類所撰寫的程式碼。

然而，這些先進神經網路工具的預測性質也引發了跨領域的可信度疑慮，例如自然語言處理(Deanda et al., 2015)、電腦視覺(Zhang et al., 2019)、自動駕駛(He et al., 2024)與醫療保健領域(Deanda et al., 2025)。程式碼生成的脈絡中，主要關注於 LLM 所生成程式碼的實際可用性，尤其在硬體受限下的環境更需重視效能指標，例如記憶體使用量、CPU 使用率、執行時間與資源消耗(Coignion et al., 2024; Qiu et al., 2024)。

為了評估生成程式碼的正確性，即是否達成預期功能。Paul et al.,(2024)分析了現有的基準資料集。例如 HUMANEVAL(Chen et al., 2021)、MBPP(Austin et al., 2021)等方式來評估 LLM 所產出的程式碼。相關研究採用 pass@k 作為效能評估指標。pass@k 意旨在 k 次生成中至少成功一次的機率。然而 Paul et al.,(2024)所研究的重點為 LLM 所產生程式碼的正確性，但對效率問題則未多加關注。Du et al., (2024)則提到，程式碼的效率應該被定義在資源消耗最小的情況下完成任務的能力。這樣的程式碼才能提升使用者體驗、節省能源，並使應用更具成本效益。

儘管近期的基準研究開始重視程式碼的功能正確性，像是 EFFIBENCH 是一套專門用來評估 LLM 生成程式碼正確性與效率的測試集(Huang et al., 2024)。但研究也指出，即使 LLM 生成的程式碼在功能上是正確的，卻經常在記憶體使用與執行時間方面不如人類撰寫的程式碼(Huang et al., 2024)。此結果源自目前主流的訓練範式，即「下一個程式碼 token 預測」傾向於強調局部模式的補全，而非尋找整體上更有效率的演算法結構(Guo et al., 2024)。儘管語法上正確，但局部補全的方法會產生多餘的程式碼區塊，導致時間與空間複雜度上升，進而造成計算資源的浪費。LLM 生成程式碼的固有低效率，會使應用程式的執行速度變慢、運作成本上升，甚至無法符合系統需求。尤其是 IoT、邊緣運算系統、雲端裝置等資源受限的環境下部署時，記憶體使用量與執行速度為關鍵考量(Bolón-Canedo et al., 2024; Solovyeva et al., 2025)。

在訓練資料集上，同時也存在有效與無效的程式碼，這會讓模型對於提示語意產生幻覺(hallucination)，進一步導致不必要的能源消耗與在關鍵應用場景下的效能下降(Huang et al., 2025)。Ehsani et al., (2025)與 Khojah et al., (2025)也指出，若 prompt 本身含糊或結構不良，模型可能誤解任務本意，進而產出不必要複雜或效率低落的解決方案。

## 小型與開源 LLM 模型及提示工程的發展

此節不僅回應您研究中關於模型選型與提示策略的設計，也強調小模型+高效率提示的**可行技術解法**。

* 開源與小型LLM（LLaMA、Mistral、Phi、Mixtral、Zephyr、MiniCPM）的表現與架構優勢
* FlashAttention、GQA、SMoE等技術
* Prompt Engineering：zero-shot、few-shot、CoT、Auto-CoT、Self-consistency、LogiCoT
* 提示框架（LangChain、Semantic Kernel）與外部系統整合性
* 提示優化對效率與準確性的貢獻

開源模型也作為私有模型的替代方案逐漸浮現。這些開源模型在程式碼生成任務中同樣展現出卓越的表現。例如，InCoder(Fried et al., 2022) 與 StarCoder(Li et al., 2023) 是專為程式碼訓練的模型。有別於其他編碼模型。，InCoder 在訓練階段採用了「因果遮蔽」(causal masking)目標此方法結合了因果語言模型與遮蔽語言模型的優點，增強了其學習能力(Devlin et al., 2019)。相對地，StarCoder 在訓練中引入了 FlashAttention 機制，此技術能加速注意力運算並降低記憶體使用，進而優化模型效能。

Roziere et al(2023)發布了 Code Llama，使用因果遮蔽技術來訓練補全模型。該模型基於 LLaMA-2 進行微調，並引入了一個「長上下文微調」(Long Context Fine-Tuning，LCFT)階段使模型能處理長達 16,384 個詞元的序列，相較於早期 LLaMA-2 階段所支援的 4,096 詞元是一大進步。儘管 Code Llama 僅有 70 億參數，但其在 HumanEval 與 MBPP 資料集上的表現已超越了參數高達 700 億的 LLaMA 模型，顯示其在處理大規模程式碼挑戰方面的效率與效能(Yu et al., 2024)。然而，LLaMA-2 的效能已被更新的 LLaMA-3(Curto et al., 2024)與 LLaMA-3.1(Dubey et al., 2024)。值得注意的是，LLaMA-3.1 的效能甚至優於 LLaMA-2，這歸因於多項改進：它是一個多語言模型，在預訓練與後訓練階段都受益於更高品質與更大量的數據，且其模型參數規模也更大。

除了持續朝向開源語言模型的努力外，Phi 模型的推出也顯示了在模型規模上競爭的興趣日益增加(Gunasekar et al., 2023)。最初的模型 Phi-1 是一個基於 Transformer 的模型，擁有 13 億參數，專為程式碼任務設計，並強調使用教科書品質的訓練資料(Gunasekar et al., 2023)。改進版本 Phi-1.5(Li et al., 2023) 在 Phi-1 的基礎上加入了下一詞預測目標，並使用數十億詞元的資料集進行訓練。Phi-2 是一個擁有 27 億參數的模型，使用 Phi-1.5 的訓練資料結合新的合成 NLP 文本與篩選過的網站資料進行訓練(Javaheripi et al., 2023)。

受到 LLaMA-1 以及 ChatGPT2、Claude3 與 Microsoft BingChat4 等指令遵循模型成功的啟發，Taori et al., (2023) 推出了 Alpaca。Alpaca 是從 LLaMA-1-7B 微調而來的 70 億參數模型，使用了 52,000 筆指令遵循示例。儘管模型規模遠小於對手，Alpaca 的行為表現卻與 OpenAI 的 text-davinci-003 相似。

值得一提的是，Chiang et al., (2023)推出的 Vicuna 結合了 LLaMA 與 Alpaca 的優勢，其參數數量略高於 Alpaca，達到 130 億。雖然上述模型的參數量最高為 130 億。Jiang et al., (2024)推出了突破性的模型 Mistral。Mistral 是一個擁有 70 億參數的語言模型，其使用 Grouped-Query Attention(GQA)提升推論速度(Ainslie et al., 2023)。並透過 Sliding Window Attention(SWA)有效處理任意長度的序列，降低成本(Beltagy et al., 2020)。Mistral 的表現超越了 LLaMA-2 13B 在所有測試基準上的成績，包括程式碼生成任務，甚至優於已釋出的 LLaMA-2 34B 變體(Touvron et al., 2023)。基於 Mistral 的成功，改進版 Mixtral 8x7B 採用了稀疏專家混合(Sparse Mixture of Experts，SMoE)架構。這項改進使 Mixtral 在所有基準上超越或匹敵 LLaMA-2 70B 與 GPT-3.5 模型的效能，同時仍採用與 Mistral 7B 相同的基本架構。其主要差異在於結構設計：Mixtral 每層包含八個前饋區塊(稱為「專家」)，提升了模型的處理效率與回應能力(Jiang et al., 2024)。

Zephyr 是 Mistral-7B 的升級版本(Tunstall et al., 2023)，透過 ∼20 萬筆 UltraChat 資料集(Ding et al., 2023) 進行微調，並進一步利用約 6.4 萬筆 UltraFeedback 資料集進行對齊(Cui et al., 2023)。該模型在 MT-Bench 基準上超越了 LLaMA2 70B。Hu et al., (2024)提出的 MiniCPM 模型在 MT-Bench 表現上顯著超越了 Zephyr-7B-alpha。與 Zephyr-7B 類似，MiniCPM 同樣使用監督式微調與 DPO 演算法，但其模型更為精簡，僅有 20 億參數。

提示工程(Prompt Engineering)的發展歷程始於「零樣本提示(Zero-Shot Prompting)」，這種方法讓模型在沒有特定訓練資料的情況下，僅依據一般性的指令執行任務。隨後「少樣本提示(Few-Shot Prompting)」的出現引入少量範例，以協助模型更準確地理解任務內容(Brown et al., 2020)。Chain-of-Thought(CoT)由 Wei et al., (2022) 提出，進一步推進提示技術，透過逐步推理的方式，顯著提升模型在推理任務中的表現。Niu et al., (2024) 採用了不同的提示設計技巧，例如針對簡單問題使用簡易提示(simple prompt)，針對複雜問題則使用 Chain-of-Thought(CoT)提示。CoT 將問題分解為邏輯步驟，藉此減少冗餘、提升效率。結果顯示，CoT 可使執行時間降低 15%，記憶體使用量降低 12%，是一種實用的優化工具(Wei et al., 2022)。

在自動化流程上，Zhang et al., (2022) 提出了「自動化 Chain-of-Thought(Auto-CoT)提示」，能自動產生多樣化的推理鏈，在無需人工設計的情況下提升準確度與穩定性。Wang et al., (2022) 在 CoT 提示的基礎上提出「自我一致性(Self-Consistency)」機制，透過產生多條推理路徑，並選出最一致的答案，藉此進一步提升解題準確性與穩健性。Zhao et al., (2023) 提出了「邏輯推理提示(LogiCoT)」，將符號邏輯原則引入推理過程，用於驗證每個推理步驟的正確性，以降低錯誤與幻覺(hallucination)的發生。

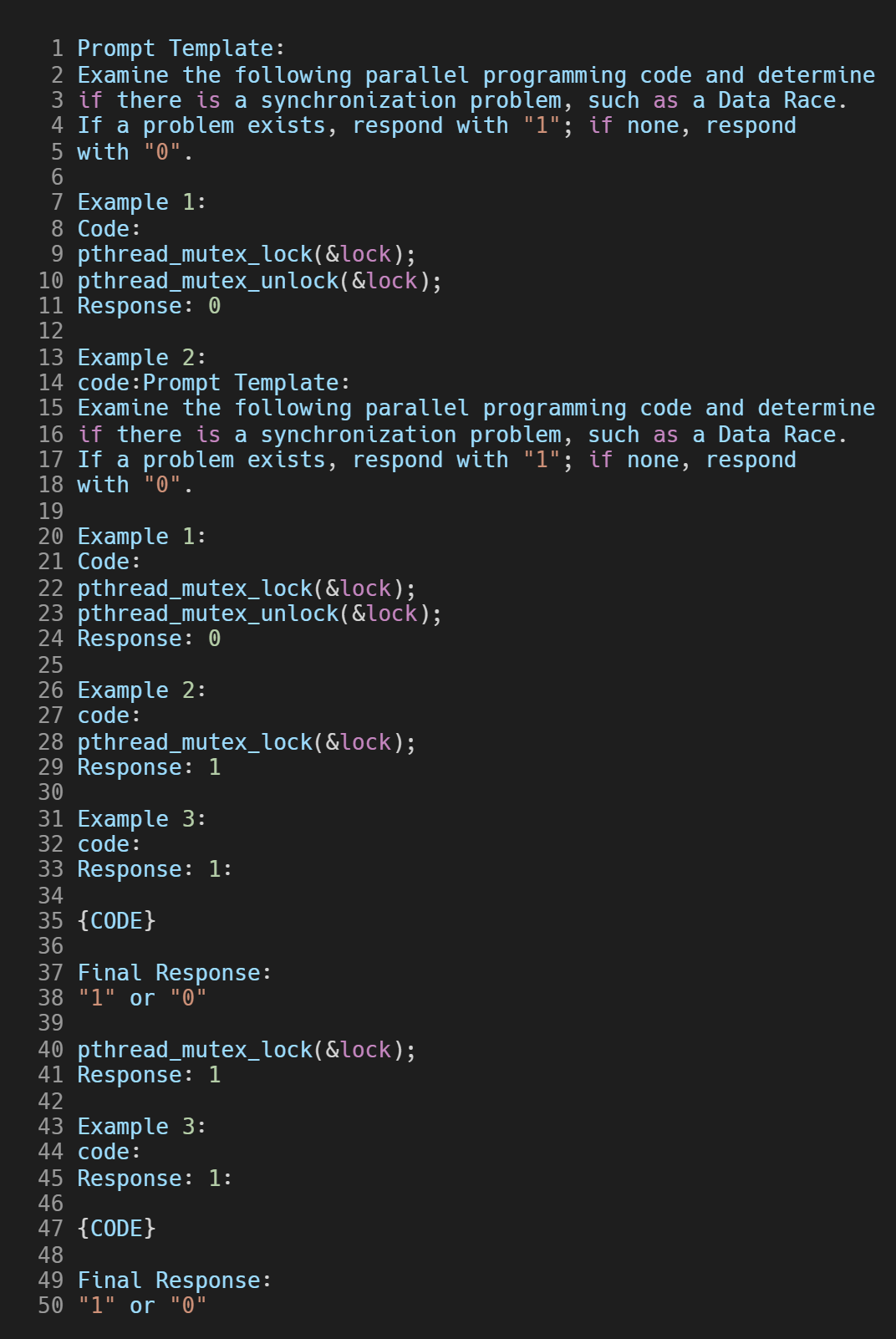
而為了有效實作上述提示技術，研究者也設計了多種提示框架，使這些方法得以實際應用於現實世界中。提示框架在銜接模型能力與使用者實際需求之間，扮演了關鍵橋樑的角色。這些框架提供必要的基礎設施，如整合外部工具、維持歷史資訊、確保輸出結果具備結構性與安全性(Liu et al., 2023)。例如 LangChain 與 Semantic Kernel 等框架可讓 LLM 與資料庫、網頁瀏覽器及其他外部系統互動，進而克服 LLM 的內在限制並擴展其應用範疇(Liu et al., 2023)。透過一個全面而可調適的結構，全面提升人機溝通的品質(Lo, 2023)。

# Study design

資料集

The benchmark can be found at <https://github.com/MayAlsofyani/Pthread-Benchmark>

Figure : Prompt Template for Few-Shot Learning Prompt



# Conclusion

對於LLM是否能有效除錯程式碼(RQ1)，Pthread程式本身涉及複雜的並行性問題，例如data race、synchronization 與deadlock。這些都需要對Thread互動有深入理解。儘管LLM在預訓練上有接受過一般除錯概念的訓練，但其知識可能未涵蓋如 Pthread 這類特殊領域，特別是在同步原語(synchronization primitives)方面。當提示中未明確指出與平行程式相關的細節時，我們開發出的LLM系統傾向提供通用型敘述，因缺乏上下文而難以準確識別具體錯誤。若期望提升LLM在Thread分析方面的效能，可採用更具針對性的提示設計。