TIPS Framework

**ABSTRACT:** 智慧編程快速演化的浪潮中，雖然大型語言模型(LLM)已能處理程式碼分類與生成任務，但其龐大運算需求與難以調控的黑盒特性，使其無法被廣泛應用於教育環境、邊緣設備與中小型開發場景。現有方法非依賴高資源模型，即依賴大量標註數據，導致程式錯誤分類與語意生成精度始終難以落地。為突破此瓶頸，本研究提出一套專為資源受限環境設計之提示工程架構TIPS Framework，旨在利用小樣本學習與推理鏈結構，於中階語言模型中建構出媲美大型模型之程式理解與生成能力。TIPS Framework 結合輕量級分類模型、語意導引之範例選取機制與逐步推理模板，建立可擴展、可解釋且具推理一致性的多模組提示系統。本研究在多執行緒程式上進行實證，結果指出，TIPS Framework在精確率、召回率與 F1 分數皆顯著超越傳統分類器與主流提示法，特別在不平衡資料集中一樣可以維持一定泛化能力。TIPS Framework不僅是一項高效能提示框架，更進一步顛覆了對模型大小與推論品質間的傳統認知，證明語義設計與推理結構的組合足以改變模型行為與輸出品質。TIPS Framework 不只是一項技術優化，更是一項語言模型民主化與落地化的關鍵策略，為 LLM 在教學、自動除錯與開發現場的普及應用開啟新的可能。

**Keywords:** Prompt Engineering, Multithreaded Programming, In-Context Learning, Chain-of-Thought, Large Language Models, Code Generation

# Introduction

隨著大型語言模型(Large Language Models, LLM)與自然語言處理(Natural Language Processing, NLP)迅速發展，程式碼生成已成為研究與應用領域中的關注焦點(Chen et al., 2024)。GPT-4、Claude、Gemini 等LLM已被證實能透過自然語言提示成功執行程式碼生成任務，為開發流程帶來重大轉變。LLM具備自然語言轉化結構化程式碼的能力，不僅簡化了程式設計流程，更有潛力加速軟體開發週期(Haque, 2025)。

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。LLM具備即時提供支援的特性，能在程式開發過程中隨需協助使用者，特別適用於自我導向學習與獨立開發情境(Cao & Dede, 2023)。同時，LLM亦展現出回應多樣學習風格的潛能，透過自然語言說明、簡明摘要與逐步教學等多元形式，提升學習效率與實作表現(Pesovski et al., 2024)。LLM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)。

在程式碼理解與生成上，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(Rasnayaka et al., 2024; Xia et al., 2022; Zeng et al., 2022)。Xia et al.(2022)提到，預訓練模型在程式理解上表現出色，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)探討了提示工程(Prompt Engineering)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，LLM在程式碼生成上的表現，表現仍難以與自然語言生成任務相提並論(Espejel et al., 2023)。第一，在程式碼生成上，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)。缺漏分號、括號或冒號等細微錯誤，都會導致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。第二，目前大多數表現最佳的LLM屬於封閉源碼(Closed Source)，使用無法存取模型架構與訓練細節，限制了研究再現性與可塑性。第三，當前表現較佳的LLM高度依賴GPU進行推論，對於個人開發者與資源有限的小型機構有造成「可及性」與「運行成本」的雙重障礙。高階GPU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。即便部分LLM提供開放介面提供免費存取，使用過程中仍會牽涉到資料上傳、雲端處理等潛在的隱私與資安風險。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(Wu et al., 2024)。

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PaLM(Chowdhery et al., 2023)、GPT-3(Brown et al., 2020)與 GPT-4(OpenAI, 2023)等代表性語言模型分別擁有 1750 億、5400 億至高達 1.7 兆個參數，展現卓越的語言理解與生成能力，然而模型規模亦伴隨著部署成本急遽上升與計算資源的高強度需求，對實際應用造成不小挑戰。相對地，Mistral(Jiang et al., 2024)與 LLaMA(Dubey et al., 2024)等新一代小型模型僅以約 70 億參數，在多項任務上展現出與LLM相當，甚至在特定應用情境中更具優勢的效能表現。這類小型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(Quantization)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，盡可能保留生成任務中的效能表現。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模型上，並在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，LlamaCPP等工具函式庫(Khosravi, 2024)提供量化模型在CPU上執行的能力。LlamaCPP是一種專為 GGUF(GPT-Generated Unified Format)檔案設計，高效的二進位儲存與推論格式。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。

儘管小型模型與量化技術在運算效率與模型部署上已帶來顯著改善，但模型表現仍難已達到程式碼生成任務所需的結構嚴謹、語意邏輯複雜與低錯誤容忍度等挑戰(Chen et al., 2021)。而提示工程(Prompt Engineering)的發展視為彌補模型能力限制的重要技術。透過精心設計的輸入提示，引導模型展現潛藏的結構化知識與推理能力(Alier et al., 2024; Li et al., 2023)。Prompt Engineering讓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、few-shot與Chain-of-Thought(CoT)。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)。Hiraou (2024)提到，適當的prompt可以顯著提升模型在程式碼補全、錯誤修復與樣板生成任務上的表現，尤其在結合 few-shot 或 CoT 策略後，能有效強化模型對任務模式的理解與邏輯推理能力。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。

因此，本研究提出一套TIPS (Template-Integrated Prompting System) Framework，著重於小型模型在資源受限環境下的程式碼分類與生成能力。TIPS Framework 以 Small Model 為核心運算載體，結合 In-Context Learning (ICL) 與 Chain-of-Thought (CoT) 策略，構築多層次的提示設計流程，導引LLM於Small Model在程式碼分類與生成上產生更具邏輯性與正確性的輸出結果。進一步地，我們聚焦在Pthread 平行程式碼的生成場景，了解TIPS Framework在在臨界區管理、條件變數應用與排程策略選擇等多執行緒設計面向的生成表現。透過融合輕量推理與高品質提示，展示TIPS Framework 在資源受限環境下仍具備實質推論能力與教學應用潛力。The main contributions of this paper are as follows。

(J個我是看別人paper都要寫這個，就跟著寫看看了)

* 本研究提出TIPS Framework專為資源受限環境下的程式碼分類與生成任務設計。該框架結合小型模型與大型語言模型(LLM)，並融入 In-Context Learning(ICL)與 Chain-of-Thought(CoT)提示策略，以強化推理邏輯與分類精度。
* 本研究針對 Pthread 多執行緒程式碼生成任務進行任務導向應用，著重於同步策略的識別與具結構性的並行程式碼產出，提供一套可於教學與實務場景中落地的提示式解決方案。
* 實驗結果顯示，TIPS Framework 在分類正確率與程式碼生成品質上皆優於傳統機器學習模型與基礎提示法，無論在平衡或不平衡資料集上皆展現出穩定且具泛化能力的效能。
* 透過組件層級的消融實驗，本研究進一步驗證 CoT 推理鏈、小模型預測輔助與語義相似示範範例等設計元素，對整體推論品質與穩健性具關鍵影響，證實 TIPS Framework 具備高解釋性與技術擴展潛力。

# Related Work

## LLM Applications in Parallel Programming

當前LLM在軟體工程與平行設計(Parallel Design)領域中的應用，不再侷限於自然語言處理，而是深入滲透至真實開發場景與高效能計算(High-Performance Computing, HPC)任務中。其能力涵蓋從程式修復、API 理解到平行化程式碼生成與錯誤偵測等複雜任務，成為未來智慧軟體開發的重要支柱(Fan et al., 2023)。LLM的應用涵蓋理解、修復與生成三大面向。Yang et al.(2023)提出 APICKnow 模型，透過對微調LLM，有效辨識 Stack Overflow 討論中之 API 實體與語意關係。即使缺乏大量標註資料，該方法仍展現卓越表現，顯示出強大的語意理解能力。Huang et al.(2022)則以 code-masked 語言模型處理程式碼中的 fully qualified name(FQN)模糊問題，顯著提升模糊片段中 API 名稱的解析效果。這些研究揭示了 LLM 對於程式碼語意的深層理解能力，尤其是在與自然語言融合的程式任務中表現突出。

在生成程式碼上，Cao et al.(2025)透過系統性實驗，評估 ChatGPT 在多語言程式修復任務中的泛化能力。結果顯示，ChatGPT在多樣錯誤類型與語法結構下皆具高度適應性，展現其跨語言與語法的修復潛能。Nashid et al., (2023)針對少樣本問題提出了一種基於檢索的提示選擇策略，有效提升LLM在程式修復與測試斷言生成上的表現。Ahmed & Devanbu (2022)在LLM效能受限於專案脈絡缺失下，透過加入專案內的範例提示進行微調，有助於提升生成的相關性與內容深度。這些研究不僅證明 LLM 可被視為靜態分析工具的補充，更顯示其可在軟體工程流程中扮演具備語境感知能力的輔助開發者。

在平行設計(Parallel Design)領域中，過往平行設計(Parallel Design)需仰賴開發人員針對程式架構進行手動優化與平行化處理。平行設計(Parallel Design)常伴隨資料競爭(data race)。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data race。進而導致不可預期的系統當機。而為了避免data race。平行設計(Parallel Design)中通常會實作同步機制。例如條件變數(condition variables)與互斥鎖(mutex locks)，以確保執行緒安全。而文獻則提出靜態分析(static analysis)、動態分析(dynamic analysis)與執行階段驗證(runtime verification)等多種策略來因應。例如Serebryany et al., (2011)於LLVM編譯器中整合ThreadSanitizer，實現動態競爭偵測。Liew et al., (2024) 針對GPU程式引入結合靜態分析與領域特定抽象的技術，強化錯誤捕捉準確性。Choi et al., (2002)提出一套混合靜態與動態分析架構，增強對物件導向程式中競爭情境的辨識能力。Malakar et al., (2024)提出的RaceFixer能在偵測錯誤後提供對應的同步策略建議，如自動插入互斥鎖或條件變數以修正問題。儘管這些方法在錯誤偵測與修復上已有顯著成效，然而技術本身高度依賴流程與語法分析，導致在跨語言或異構架構上的遷移應用仍具挑戰性。而LLM能提供語意對齊、語境推理與多範式語言適應性，能補足傳統工具不足之處。尤其是面對結構模糊、語法不一致或不完整的輸入時，LLM所展現的彈性與修補能力已成為研究者關注焦點。不僅止於支援程式碼生成與理解的語言模型，而是逐步進化為能處理高結構性與領域特定問題的智慧引擎。

近期研究則嘗試將LLM導入以簡化開發流程與擴展應用效能。例如Chen et al., (2023)開發LM4HPC模型，運用LLM進行程式碼生成與效能優化。該模型專注於高效能運算(HPC)情境下的語意理解與結構調整，展現LLM在自動化設計與性能上的韌性。Kadosh et al., (2023)提出以源碼圖表示與 Transformer 模型相結合的架構，能預測 OpenMP 平行迴圈中的 pragma 語句，從而自動識別平行化潛力。Ding et al., (2023) 則以HPC領域問答資料集微調LLaMA推出HPC-GPT，使其協助與生成 HPC 程式碼。Chen et al., (2024)提出得OMPGPT，專門針對OpenMP程式碼優化進行訓練，可主動產生pragma建議，展示領域特定語言模型在高效能程式生成上的強大能力。這些發展都在凸顯，LLM正由單純生成工具轉化為具備編譯導向知識與結構理解的專業系統。

不僅是新模型的開發，ChatGPT 與 GitHub Copilot等現有通用工具亦被證實在平行設計(Parallel Design) 上具有實際價值。Mišić & Dodović (2024)分析Copilot 在平行程式碼生成任務中的協助潛能，尤其是在語法提示與模板填寫任務中能大幅降低開發門檻。Alsofyani & Wang (2024)則利用LLM偵測 OpenMP 並行錯誤。上述研究共同彰顯出，即便非為專業 HPC 訓練的 LLM，只要經過適當提示設計與任務引導，LLM便可達成跨語言、跨專案、跨任務的泛化學習，為平行設計(Parallel Design)中的關鍵決策提供支持，成為促進程式設計自動化、強化並行任務可控性與推動開發者培育的關鍵核心。

## Assessing the Practicality of LLM Code Generation Based on Correctness and Efficiency

儘管LLM在平行設計與高效能運算應用中展現強大潛力，其程式碼生成能力亦逐漸被導入實際開發流程中，但這也引發對其生成品質與執行效率的廣泛關注。但與傳統開發工具相比，LLM生成之程式碼常帶有不確定性，這種預測式生成機制已在自然語言處理(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)引發類似的可信度疑慮。

在程式碼生成的脈絡中，我們主要關注於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。常見的衡量指標包括Memory Usage、CPU Utilization、Run Time與、Code Complexity(Coignion et al., 2024; Qiu et al., 2024)。Huang, et al., (2024)提出一套專門評估LLM 生成程式碼正確性與效率的測試集EFFIBENCH。研究指出，即使 LLM 生成的程式碼在功能上是正確的，但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)。此結果歸咎於目前的主流訓練範式為「下一個程式碼 token 預測」，傾向局部模式的補全，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)。Dou et al., (2024)與Hou and Ji (2025) 比較了人類程式設計師與LLM的表現，發現LLM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，而人類設計師所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，也凸顯出當前LLM不足之處。Niu et al., (2024)的研究納入準確率與執行時間作為效率指標，並在HumanEval、MBPP 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。Paul et al.,(2024)分析了HUMANEVAL(Chen et al., 2021)、MBPP(Austin et al., 2021)等現有資料集，並採用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)研究的重點為 LLM 所產生程式碼的正確性，對於效率並未關注。Du et al., (2024)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。

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)。Han et al.(2024)提到，完整的參數微調雖能提升表現，但代價是大量運算資源與潛在的泛化能力下降。雖然 Parameter-Efficient Fine-Tuning(PEFT)策略在一定程度上能降低微調成本，但也容易導致災難性遺忘(catastrophic forgetting)，亦即模型可能在新任務表現良好，卻遺失先前已學習的關鍵能力。這也凸顯出採用針對任務語境設計的提示工程(prompt engineering)成為更具可行性與可控性的選項，在不犧牲效率與穩定性的前提下，平衡結果的準確性與可解釋性。

## Semantic-Guided Prompting and Compact Model Development

隨著開源模型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Fried et al., (2022)提出的InCoder與Li et al., (2023)提出的StarCoder是專為程式碼訓練的模型。InCoder在訓練階段採用了「因果遮蔽」(causal masking)，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)。而StarCoder則是在訓練中引入了 FlashAttention 機制，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，該模型基於 LLaMA-2 進行微調，利用「因果遮蔽」(causal masking) 訓練補全模型，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)。雖然到後期，LLaMA-2已經被LLaMA-3(Curto et al., 2024)與 LLaMA-3.1(Dubey et al., 2024)所更新，雖說LLaMA-3與LLaMA-3.1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LLaMA系列持續更新外，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)。而改進的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)。

此外，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。Jiang et al., (2024)提出了具有突破性的語言模型 Mistral，這是一個擁有 70 億參數的高效 Transformer 架構。Mistral 採用了 Grouped-Query Attention(GQA)提升推理效率(Ainslie et al., 2023)，並引入 Sliding Window Attention(SWA) 策略，使其能夠處理任意長度的輸入序列(Beltagy et al., 2020)。在多項標準基準測試中，Mistral 的表現全面超越 LLaMA-2 13B，包 括程式碼生成等任務，甚至在部分情境中也優於更大型的 LLaMA-2 34B 模型(Touvron et al., 2023)。基於 Mistral 的技術成功，後續開發的 Mixtral 8x7B 模型更進一步採用了 稀疏專家混合架構(Sparse Mixture of Experts, SMoE)，在提升推理效率與回應能力的同時，顯著擴展了模型的處理能力。實驗結果顯示，Mixtral 8x7B 在各項基準測試中不僅超越 LLaMA-2 70B，亦在多數任務中與 GPT-3.5 媲美(Jiang et al., 2024)。Zephyr(Tunstall et al., 2023) 與 MiniCPM(Hu et al., 2024) 等模型進一步探索微調與偏好對齊(如 UltraChat、UltraFeedback、DPO)策略，即使參數量相對較小，也能展現強大性能。而這也突顯了一個核心觀點：模型能力的釋放，不僅來自架構創新，更高度依賴於提示技術的進化。

提示工程(Prompt Engineering)的發展始於零樣本提示(Zero-Shot Prompting)，Zero-Shot Prompting 是指在沒有任何任務範例的情況下，僅透過指令語句，引導語言模型完成特定任務的推論方式。隨後的少樣本提示(Few-Shot Prompting)則是在提示語句中加入幾個任務範例(input-output pairs)，以引導模型學習任務格式與邏輯，進而執行後續輸入的推理任務(Brown et al., 2020)。Chain-of-Thought(CoT)則由 Wei et al., (2022) 提出，強調模擬逐步推理過程，引導語言模型在作答前先「思考」、再「回答」，以強化其邏輯推理能力，特別適用於數學與複雜問答等多步任務。Niu et al., (2024)針對不同任務類型設計相應提示策略，例如對於簡單問題採用簡化提示(simple prompt)，而在面對複雜任務時則使用 CoT 提示。研究結果顯示，CoT 不僅可提升模型準確性，亦具備計算效率優勢：能使平均執行時間降低 15%，記憶體使用量下降約 12%，證實其為一項實用且高效的提示優化工具。Zhang et al., (2022) 在自動化流程上提出自動化 Chain-of-Thought提示(Auto-CoT)，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)基於CoT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)則提出「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。

提示工程(Prompt Engineering) 的推陳出新，不僅顯著提升語言模型在複雜任務中的表現精度與穩定性，也為模型的可控性、可解釋性與資源效率帶來實質進展。在提示工程(Prompt Engineering) 快速演進的同時，如何將其系統性地整合為可應用於特定任務的提示框架，仍是一項尚待解決的重要挑戰。本研究提出的TIPS Framework，基於上述提示工程(Prompt Engineering)成果，設計一套針對小型模型輔助、語意相似檢索、推理鏈構建與逐步範例生成的綜合性提示系統。TIPS Framework不僅強化CoT推理流程的一致性，也讓提示工程(Prompt Engineering) 能更有效地對應真實程式任務中的語意與邏輯結構。不僅回應了目前提示技術應用上的模組化缺口，也實證其能顯著提升生成準確性、解釋能力與計算效率，展現出提示工程從策略方法邁向實務框架的下一階段潛力。

# The Proposed Framework: TIPS Framework

TIPS Framework如圖OO所示。共涵蓋Training the small model、Building CoT prompts、Identifying similar demonstrations與Small model-augmented prompting四個階段。核心在於利用(In-Context Learning, ICL)與(Chain-of-Thought, CoT)技術來設計高品質提示語，藉此提升LLM在程式生成任務中的表現。

Figure 1：An overview of the proposed TIPS Framework

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

## Training the small model

我們首先使用建構一個小型模型作為TIPS Framework的前端模組，用來強化ICL構建。並激發 LLM 隱含的微調能力。這種策略能夠減輕LLM在偵測任務中常見的幻覺(hallucination)、資料分佈差異(data distribution discrepancies)等效能退化問題。我們首先從資料集當中準備程式碼片段，並定義為Equation(1)。其中為樣本筆數，表示一個樣本的原始程式碼(輸入特徵)。 ∈ {0,1,2,…,n}表示對應的真實標籤。接著，我們定義小型模型為Equation(2)。其中是模型對預測出的類別標籤， ∈ [0, 1]為模型對預測結果的信心值(confidence)。而訓練的目標是最小化損失函數為Equation(3)。其中為預測標籤與真實標籤之間的損失。透過這樣的小型模型訓練流程，我們能夠建立一個對ICL具備辨識能力的模型，並將其結果作為後續建構 CoT 提示語與支援 LLM 推論的依據。

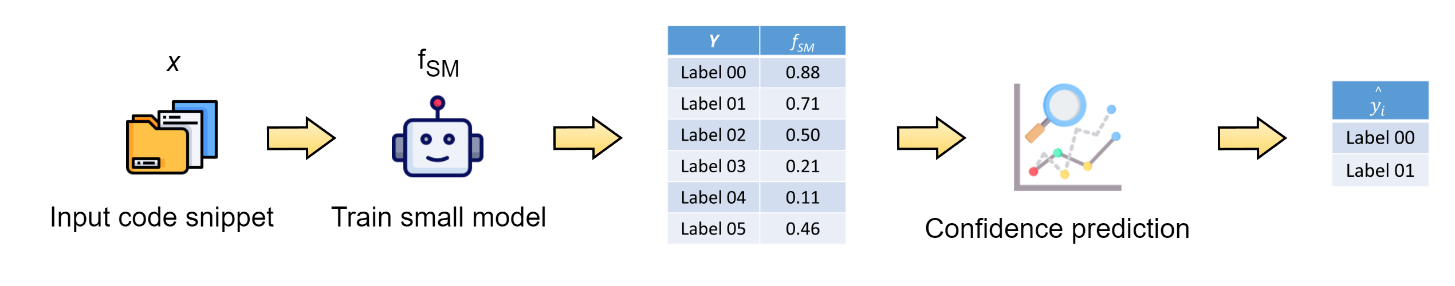
## Buidling CoT prompts

TIPS Framework 的第二階段旨在為每個輸入的程式碼片段構建一條CoT，以輔助後續LLM進行更精確的程式碼生成。不同於傳統僅包含輸入、輸出的prompt，CoT 會在提示中加入一系列中間推理步驟，將複雜問題拆解為多個較簡單的子問題，模擬人類逐步推理的過程，從而提升模型的邏輯判斷能力與最終輸出品質。

同時，為了在生成推理鏈時提供可信度保障，我們引入了符合性預測(Conformal Prediction, CP)機制(如Equation(4)所表示)。

其中是測試樣本的預測標籤集合，是一個可能的預測標籤，為非符合性分數，衡量與之間不一致的程度。CP的核心是：量化模型預測的不確定性，並為每個預測結果提供信心水準的保證。具體而言，CP會定義一個非符合性函數(non-conformity function)，用來衡量模型的預測結果與實際觀察值之間的差異程度。這個非符合性分數(non-conformity score)，可以視為「預測不一致程度」的量化指標。之後再透過校準集(calibration set)對此函數進行校準，再將其應用到測試樣本上。大致流程如Figure 2所表示。

Figure 2：An example of CP process



在TIPS Framework中，CoT的構建流程可分為兩個步驟。第一，先利用已訓練的小型模型對測試樣本進行初步預測，並生成候選標籤集合。第二，將該集合與小型模型的結果一併整合至 CoT template中。Template由以下五個邏輯推理階段組成：

* Structural Analysis：針對輸入程式碼片段進行語法與控制流程的結構性檢視，辨識關鍵語句、模組邊界與潛在功能區段。
* Semantic Interpretation：分析程式碼上下文及語意特徵，推斷各結構元素的功能與邏輯作用。
* Classification Judgment(分類判斷)：基於結構及語意分析，推斷程式碼所對應的任務類別或同步策略。
* Label Mapping(標籤映射)：於多類別選項中選擇最適合的預測標籤，並處理不確定性情境下的排除、延伸或補充判斷。
* CoT Construction(推理整合)：將前述推理步驟串接為具邏輯一致性的自然語言推理鏈，形成最終提示語。

舉例而言，假設某段輸入程式碼被小型模型初步分類為” Mutex-based strategy for critical section protection.”。 而符合性預測(CP)回傳的候選標籤集合為{critical\_section, resource\_pooling}，這個集合以 95% 的信心水準涵蓋了正確標籤。此時系統就會依據：

* 小型模型的預測結果: ” Mutex-based strategy for critical section protection.”
* CP 給出的候選集合: {critical\_section, resource\_pooling}
* CoT 推理模板(Semantic Understanding → Structural Analysis → Classification Judgment → Label Identification → CoT Construction)

生成一條專屬於該樣本的 CoT 提示語，協助 LLM 在後續階段產生具備正確同步策略(如 pthread\_mutex\_lock() / unlock())的並行程式碼，提升最終輸出之正確率與可靠性。

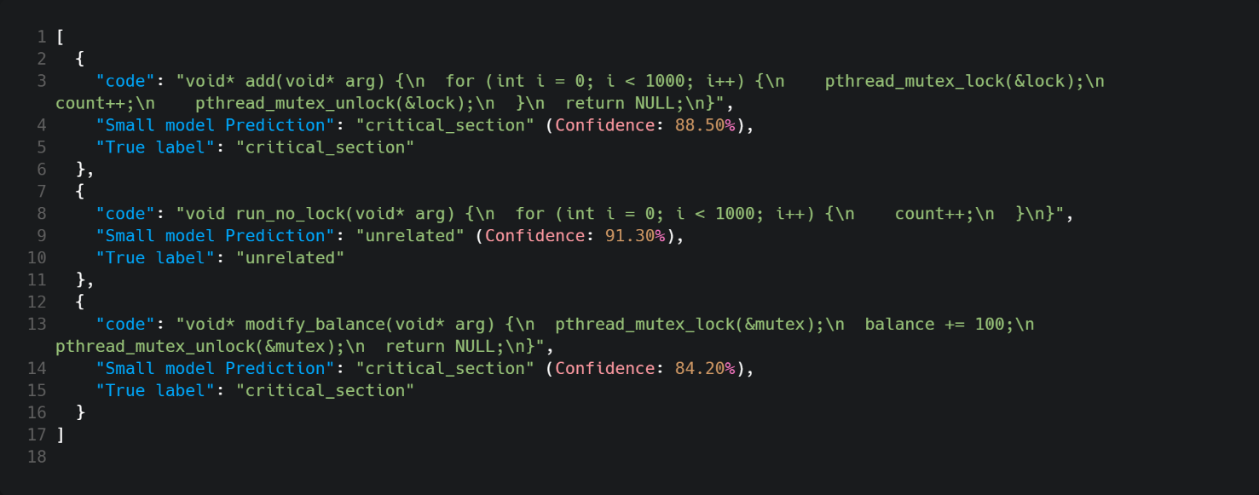
## Identifying Similar Demonstrations

在Prompt Engineering中，In-Context Learning(ICL)是一種常見且有效的使用策略。如何設計高品質的示範樣本(demonstrations)一直是關鍵研究問題之一(Zhao et al., 2023)。透過提供與任務相關的示範範例，LLM 能夠從中隱式學習梯度方向，並透過注意力機制執行類似於微調(fine-tuning)的適應過程(Von Oswald et al., 2023)，從而改善對下游任務的推理能力。我們定義示範集合(demonstration set)為Equation(5)。其中為將訓練資料轉換為自然語言提示的函數。而ICL的的推理流程可表示為Equation(6)。

其中為任務描述(task instruction)，為示範集合，為待預測樣本，為LLM 輸出的預測結果。為了程式碼生生任務中ICL的效果，本系統會同時考慮語意相似度(Semantic Similarity)、字面相似度(Lexical Similarity)與語法相似度(Syntactic Similarity)來找出與當前輸入樣本最相似的範例。語意相似度(Semantic Similarity)如Equation(7)所表示。我們先將程式碼轉換為向量嵌入表示，並使用餘弦相似度衡量兩段程式碼在語意層面的接近程度。字面相似度(Lexical Similarity)如Equation(8)所表示。我們將程式碼分詞為 token 集合，並利用 Jaccard 相似度度量兩者的交集與聯集比例。語法相似度(Syntactic Similarity)則如Equation(9) 所表示。我們將抽象語法樹(AST)轉換為節點序列，並利用 Levenshtein 距離衡量兩者的編輯差異。

最終，我們將上述三項相似度加權整合為總體相似度分數(如Equation(10)所表示)。其中。並在本研究中設定為等權重。系統將從示範資料庫中選擇相似度最高的程式碼片段，作為 ICL 提示語的一部分(如所Figure 3表示)。以幫助 LLM 更準確地捕捉任務特徵並生成正確程式碼。

Figure 3：An example of demonstrations in ICL prompt



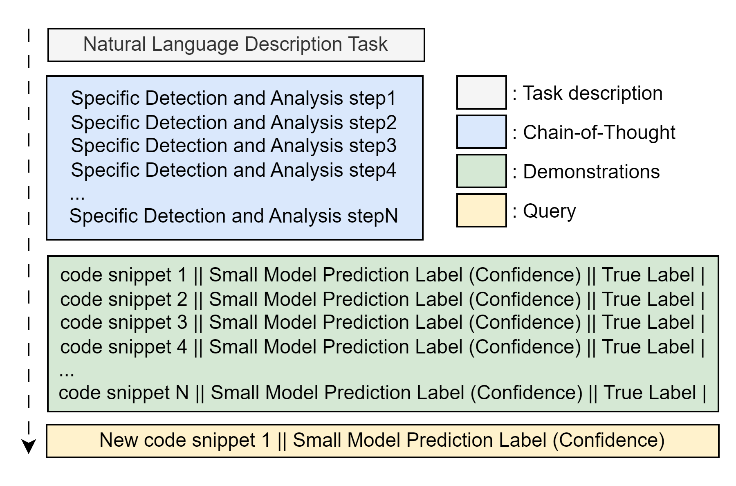
## Model-augmented Prompting

在完成客製化的 Chain-of-Thought(CoT)提示語構建，以及最相關 In-Context Learning(ICL)示範樣本的選取後，TIPS Framework 進入最後一階段：將小型模型(Small Model)的預測結果、信心分數與 CoT 推理鏈整合為一個統一的提示語結構，用於引導大型語言模型(LLM)進行程式碼生成任務。該模型增強提示語設計包含兩個核心組件：

* Chain-of-Thought Prompts：引導後續LLM逐步推理，從而提升其判斷過程的透明度與可解釋性。避免 LLM 直接進行「端到端」輸出時可能產生的跳躍性推論錯誤，增強模型的邏輯一致性。
* 小模型預測結果與信心分數(Prediction with Confidence Score)：透過小型模型的預測標籤以及其對應信心分數，後續LLM能夠獲取額外的先驗資訊，理解輸入樣本與候選標籤之間的潛在關聯性。當小型模型的信心水準高於某一閾值時，LLM 可以優先採納該預測結果; 反之，LLM 會傾向依賴其自身的推理過程進行決策。

最終完整的prompt template如Figure 4所示，TIPS Framework允許後續LLM在處理含糊、具多義性或存在潛在歧異的程式碼時，根據情境自主選擇是否信任小型模型輸出，或是依靠自身推理進行修正決策。TIPS Framework不僅提高了程式碼生成的準確率與魯棒性，也增強了最終結果的可解釋性，為自動化策略識別與平行程式設計輔助提供了更具彈性與可靠性的解決方案。

Figure 4：The final prompts template



# Experimental Design

## Datasets

本研究聚焦於多執行緒程式中常見的同步錯誤類型： data race。data race通常源於開發人員未能正確使用mutex或其他同步機制保護共享資源，導致多個執行緒同時訪問並修改該資源，進而引發競爭條件與非預期的程式執行結果。我們從GitHub上的開源專案中擷取Pthreads原始碼檔案[[1]](#footnote-1)。每份程式碼長度約介於 50 至 250 行之間。這些程式原始版本皆包含正確實作之 mutex 機制，以確保臨界區的同步安全性。為了進行系統性對比實驗，我們依據資料分佈特性構建了兩種資料集：平衡資料集(Balanced Dataset)與不平衡資料集(Imbalanced Dataset)。

平衡資料集(Balanced Dataset)包含數量相同的「正確版本」與「錯誤版本」程式碼樣本。錯誤版本是由原始正確程式碼經人工修改而產生。每個錯誤樣本僅包含單一類型的資料競爭缺陷，例如缺少必要的 pthread\_mutex\_lock() 呼叫。此資料集旨在評估TIPS Framework與其他baseline models在標準、資訊對稱情境下的分類準確率及程式碼生成品質。該設計有助於排除資料分佈偏差的影響，專注於測試不同提示語策略(prompting strategies)的純粹效果。

不平衡資料集(Imbalanced Dataset)是為了模擬真實軟體開發環境中常見的資料分佈不均現象。不平衡資料集(Imbalanced Dataset)的樣本可能同時包含 2 - 4 種不同型態的資料競爭缺陷，例如缺少鎖、錯誤解鎖、鎖覆蓋範圍不足等。旨在評估TIPS Framework面對**多重錯誤類型及資料不平衡情境**下的泛化表現與魯棒性(Robustness)。

## Baseline model

為評估本研究所提出的 TIPS Framework 於多執行緒程式碼生成任務的表現。我們選定Support Vector Machine (SVM)(Joachims, 1999)、Adaboost(Freund & Schapire, 1996)、Bagging(Breiman, 1996)作為Baseline Model。這些Baseline Model具備可重現性，且過往多應用於類似任務或資料設定，能有效作為本研究所提出方法之對照組。由於本研究的任務同時涵蓋分類判斷與程式碼生成兩個面向，分類任務主要聚焦於檢測程式碼中錯誤類型的辨識率，可形式化為一個典型的監督式分類問題。因此，選擇傳統分類器作為基準模型能夠提供與語言生成模型(LLM-based approach)在分類層級上的公平對照，進一步檢驗 TIPS Framework 的增益效應。

Support Vector Machine(SVM) 是一種常用的監督式分類演算法，透過尋找能最大化不同類別間距(margin)的超平面進行分類。SVM 特別適用於高維度特徵空間，並可搭配不同核函數(如線性核、多項式核、徑向基核 RBF)來處理非線性分類問題。SVM 以其良好的泛化能力著稱，在程式碼缺陷檢測與軟體度量分析等任務中均取得穩健表現。

Adaptive Boosting(Adaboost)屬於提升式集成學習方法，通過迭代性地訓練多個弱分類器(通常為決策樹)，並根據分類誤差率動態調整樣本權重，最終以加權投票機制形成一個強分類器。Adaboost 在處理特徵複雜或資料存在雜訊的分類任務中表現良好，並已被證實在多個自然語言處理與軟體工程領域任務(例如程式碼意圖識別、缺陷預測)中具有效能優勢。

Bootstrap Aggregating(Bagging)是一種平行式集成學習策略，透過自助抽樣(sampling with replacement)生成多個訓練子集，並在各子集上分別訓練獨立的分類器，最後以多數決匯總預測結果。此方法能有效降低模型的方差，提升預測穩定性，尤其適用於避免單一分類器過度擬合的情境。

## Evaluation Metrics

為量化本研究所提出之TIPS Framework 的分類效能，本文採用三項常見且標準化的指標：Precision、Recall與F1-score。各項指標的計算皆基於標準混淆矩陣中的四種基本統計量：真陽性(True Positive, TP)、假陽性(False Positive, FP)、真陰性(True Negative, TN)及假陰性(False Negative, FN)。這些指標同時也為基準模型(Baseline Models)共同採用的評估準則，以確保實驗結果的公平性與可比較性。

Precision如Equation(11)所表示，用以衡量模型在預測為正類時的準確程度，即在所有被模型判定為正類的樣本中，實際為正類的比例。高精確率代表模型能有效避免產生大量誤報(false alarms)。Recall如Equation(12)所表示，映模型對實際正類樣本的覆蓋能力，即在所有真實為正類的樣本中，能被模型正確識別的比例。F1-score如Equation(13)所表示，為精確率與召回率的調和平均數，用以同時平衡模型的預測準確性與完整性，特別適用於類別分布不平衡的資料集環境。

## Research Questions

為全面評估本研究所提出之 TIPS Framework 在多執行緒程式碼分類與生成任務中的效能，我們設計了一系列比較實驗，並將實驗流程分為兩個主要面向：分類判斷與程式碼生成。透過這些研究問題驗證TIPS Framework的整體優勢、各核心組件的貢獻度，以及其在不同資料分佈條件下的適用性與穩定性。在分類判斷部分，我們選定Support Vector Machine (SVM)(Joachims, 1999)、Adaboost(Freund & Schapire, 1996)、Bagging(Breiman, 1996)作為Baseline model作為比較。這些模型涵蓋了傳統分類器與深度語言模型方法，構成多樣化比較基準，藉此全面評估TIPS Framework的生成效能。因此我們提出：

* RQ1：相較於傳統分類模型(如 SVM、Adaboost、Bagging)，本研究提出之 TIPS Framework 是否能透過提示設計與 LLM 整合，有效提升 Pthread 程式碼分類與生成任務的整體表現？

同時，在真實軟體開發場景中，錯誤樣本通常呈現不均衡分佈，並可能同時包含多種類型的資料競爭錯誤。為驗證TIPS Framework在不理想條件下的適應性，我們提出以下問題：

* RQ2：在面對樣本分佈不平衡與程式碼多樣性提升的情境下，TIPS Framework 是否仍能維持穩定的推論品質與生成效率？

此外，為了解各組件對整體框架貢獻度，我們設計消融實驗(ablation experiments)，逐一剔除或替換下列三個核心模組，以觀察其對最終分類效能的影響：

* Small Mode：負責在示範階段產生初步分類結果，提供輔助資訊給後續LLM 參考。
* Chain-of-Thought (CoT)：引導 LLM 進行邏輯化推理流程，幫助其系統化分析程式碼結構。
* Demonstrations：提供與目標樣本最相似的範例，促進後續LLM 在類似情境下的準確推論。

我們提出以下問題：

* RQ3：TIPS Framework 的各核心組件在分類與生成任務中分別帶來何種貢獻與影響？

在程式碼生成上，本研究想探究TIPS Framework 是否能透過提示設計有效提升LLM在 Pthread 程式碼生成任務中的準確性。我們採用 GPT-3.5 作為後續基礎生成的LLM，並設計兩種提示策略進行比較。一種是本研究基於TIPS Framework所生成的prompt template; 另一種是僅提供任務描述與輸入程式碼的**Basic Prompt，該提示謂包含額外推理輔助資訊。因此我們提出：**

* **RQ4：**本研究提出之 TIPS Framework 是否能透過提示設計有效提升LLM在 Pthread 程式碼生成任務中的準確性？

# Result

為了驗證本研究所提出之TIPS Framework在分類判斷上是否優於傳統分類模型，我們以SVM、Adaboost與Bagging作為比較。Table 1呈現了各模型在多個 Pthread 任務類別下的表現(Precision = 0.97, Recall = 0.90, F1-Score = 0.94)。顯著優於SVM(Precision = 0.80, Recall = 0.63, F1-Score = 0.78)。其中，在 Mutex Management 與 Condition Variables 兩個任務類別中，TIPS Framework 分別取得 F1-score 0.95 與 0.95，明顯提升了模型對關鍵同步策略的辨識能力。

Table 1：Performance Comparison of Small Models and TIPS Framework in Pthread Code Generation Tasks (Balanced Dataset)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **SVM** | | | **Adaboost** | | | **Bagging** | | | **TIPS Framework** | | |
|  | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** |
| Thread Lifecycle | 0.91 | 0.40 | 0.57 | 0.88 | 0.70 | 0.78 | 0.70 | 0.78 | 0.88 | 1.00 | 0.89 | 0.89 |
| Mutex Management | 0.71 | 0.80 | 0.89 | 0.89 | 0.73 | 0.80 | 0.80 | 0.80 | 0.80 | 1.00 | 0.91 | **0.95** |
| Condition Variables | 0.63 | 0.60 | 0.75 | 0.88 | 0.80 | 0.78 | 0.88 | 0.80 | 0.80 | 1.00 | 0.91 | **0.95** |
| Thread Pool Creation | 0.83 | 0.70 | 0.82 | 0.91 | 1.00 | 1.00 | 0.95 | 1.00 | 0.95 | 0.95 | 0.91 | 0.93 |
| Scheduling Policy Assignment | 0.91 | 0.66 | 0.89 | 0.89 | 0.88 | 1.00 | 0.78 | 1.00 | 1.00 | 0.92 | 0.88 | 0.96 |
| **Average** | 0.80 | 0.63 | 0.78 | 0.89 | 0.82 | 0.87 | 0.82 | 0.88 | 0.89 | **0.97** | **0.90** | **0.94** |

Note. Abbreviations: Pre = Precision; Re = Recall; F1 = F1 Score

為了進一步探究TIPS Framework在不平衡資料分佈下是否仍能維持穩定效能。研究結果如Table 2所呈現。可以觀察到。所有模型在不平衡資料集(Imbalanced Dataset)中整體效能均顯著下降，但下降幅度因模型而異。TIPS Framework(Precision = 0.82, Recall = 0.42, F1-Score = 0.57)還是略為優於SVM(Precision = 0.42, Recall = 0.22, F1-Score = 0.28)與Bagging(Precision = 0.85, Recall = 0.41, F1-Score = 0.55)。

Table 2：Robustness Evaluation of Small Models and TIPS Framework under Imbalanced Pthread Task Distribution

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **SVM** | | | **Adaboost** | | | **Bagging** | | | **TIPS Framework** | | |
| Task | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** |
| Thread Lifecycle | 0.53 | 0.05 | 0.17 | 0.88 | 0.05 | 0.20 | 0.88 | 0.15 | 0.33 | 0.77 | 0.12 | 0.32 |
| Mutex Management | 0.42 | 0.45 | 0.55 | 0.73 | 0.42 | 0.52 | 0.80 | 0.52 | 0.63 | 0.94 | 0.64 | 0.76 |
| Condition Variables | 0.00 | 0.00 | 0.00 | 0.45 | 0.54 | 0.49 | 0.70 | 0.59 | 0.64 | 0.85 | 0.45 | 0.60 |
| Thread Pool Creation | 0.48 | 0.02 | 0.05 | 0.98 | 0.51 | 0.67 | 1.00 | 0.26 | 0.51 | 0.71 | 0.28 | 0.44 |
| Scheduling Policy Assignment | 0.69 | 0.59 | 0.63 | 0.77 | 0.49 | 0.59 | 0.85 | 0.51 | 0.64 | 0.83 | 0.63 | 0.72 |
| **Average** | **0.42** | **0.22** | **0.28** | 0.96 | 0.40 | **0.49** | **0.85** | **0.41** | **0.55** | **0.82** | **0.42** | **0.57** |

Note. Abbreviations: Pre = Precision; Re = Recall; F1 = F1 Score

為深入分析 TIPS Framework 各核心組件在多執行緒程式碼分類與生成任務中的貢獻度，我們設計了一系列消融實驗(ablation experiments)。逐一剔除Small Model、Chain-of-Thought (CoT)、Demonstrations三個主要模組。Table 3呈現了在不同組件配置下模型的表現。在移除 Small Model 的配置下(w/o Small Model)，TIPS Framework表現(Precision = 0.75, Recall = 0.50 F1-Score = 0.58)顯著低於完整框架的平均表現(Precision = 0.86, Recall = 0.62, F1-Score = 0.61)。

在移除CoT 模組(w/o CoT)後，模型表現(Precision = 0.71, Recall = 0.50 F1-Score = 0.59)與w/o Small Model 結果接近(Precision = 0.75, Recall = 0.50 F1-Score = 0.58)。但在 Thread Lifecycle 和 Thread Pool Creation 任務中的 Recall 僅分別為 0.25 和 0.29，明顯低於完整框架的 0.50 與 0.29。顯示 CoT 能夠協助 LLM 分解複雜程式邏輯，提升少數類別樣本的檢測率。

而在去除示範範例(w/o Demonstration)後F1-score為 0.60，略低於完整框架的 0.61。但在 Mutex Management 任務中，Precision 由 0.70 升至 1.00，顯示在特定情境下相似範例有助於 LLM 理解同步策略的正確實作方式。

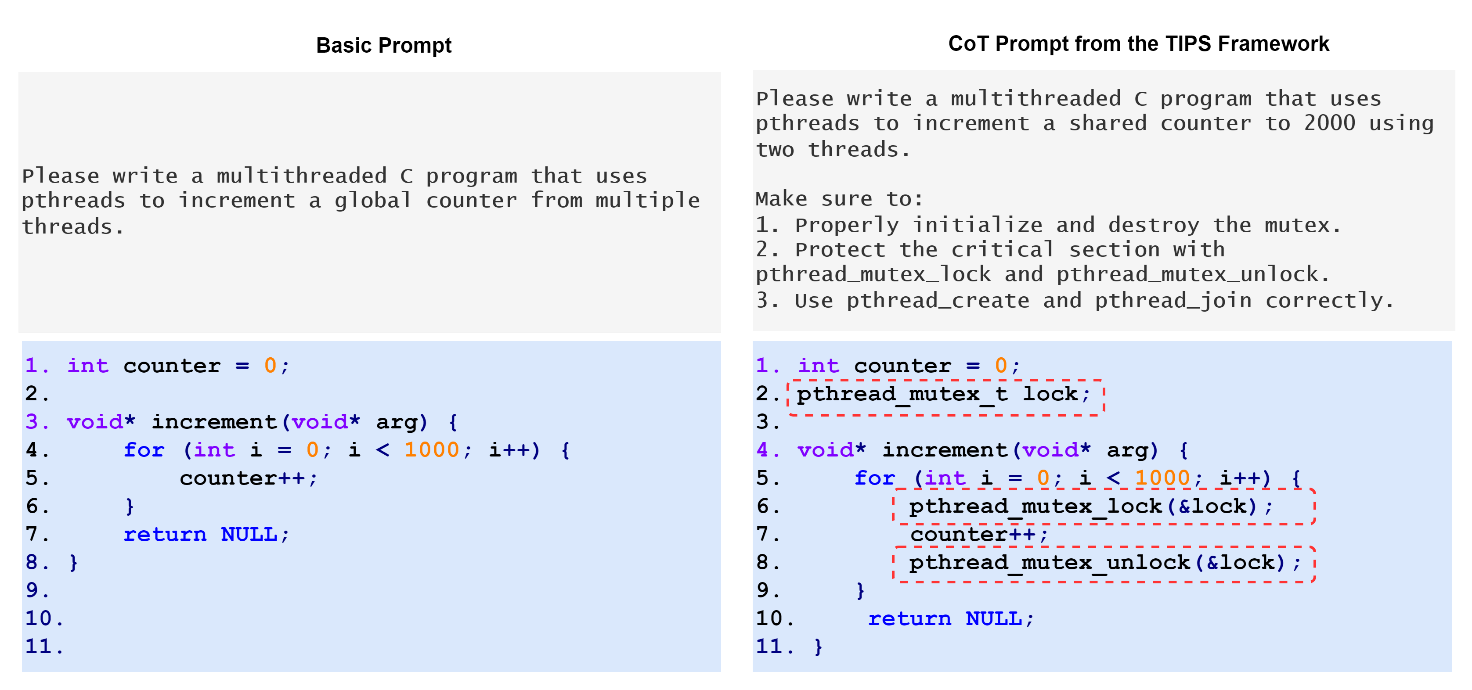
Table 3：Effectiveness of the TIPS Framework Under Different Component Configurations

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **w/o Small model** | | | **w/o CoT** | | | **w/o Demonstration** | | | **TIPS Framework** | | |
| Task | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** | **Pre** | **Pre** | **F1** |
| Thread Lifecycle | 0.40 | 0.50 | 0.44 | 0.25 | 0.25 | 0.25 | 0.29 | 0.50 | 0.36 | 0.20 | 0.50 | 0.29 |
| Mutex Management | 0.84 | 0.64 | 0.73 | 0.82 | 0.72 | 0.77 | 1.00 | 0.72 | 0.84 | 0.70 | 0.76 | 0.73 |
| Condition Variables | 0.74 | 0.58 | 0.65 | 0.70 | 0.58 | 0.64 | 0.68 | 0.62 | 0.65 | 0.82 | 0.75 | 0.78 |
| Thread Pool Creation | 1.00 | 0.22 | 0.37 | 1.00 | 0.29 | 0.45 | 0.92 | 0.28 | 0.43 | 0.93 | 0.29 | 0.45 |
| Scheduling Policy Assignment | 0.79 | 0.56 | 0.71 | 0.80 | 0.66 | 0.73 | 0.84 | 0.60 | 0.70 | 0.76 | 0.81 | 0.79 |
| **Average** | **0.75** | **0.50** | **0.58** | **0.71** | **0.50** | **0.59** | 0.75 | 0.54 | **0.60** | **0.86** | **0.62** | **0.61** |

Note. Abbreviations: Pre = Precision; Re = Recall; F1 = F1 Score

而對於TIPS Framework在協助程式碼生成的表現，結果如Figure 5所呈現。在使用Basic Prompt上，GPT-3.5雖然能生成基本的多執行緒程式碼架構，但並未包含pthread\_mutex\_lock() 與 pthread\_mutex\_unlock()等對於臨界區保護機制的考量。相較之下，基於TIPS Framework 所提供的Prompt Template明確引導模型使用pthread\_mutex\_lock() 與 pthread\_mutex\_unlock() 保護共享資源。

Figure 5：Comparison of Code Generation Results Using Basic Prompt vs. Prompt template from TIPS Framework



# Discussion

本研究所提出之TIPS Framework在多執行緒設計面向的分類任務表現顯著超越傳統分類器。尤其在Mutex Management 與 Condition Variables 任務中，TIPS Framework 透過 CoT 與小模型輔助判斷，有效降低了錯誤分類率。此結果與Ahmed & Devanbu (2022)、Nashid et al. (2023)等研究呼應。在程式碼分類任務中，結合專案上下文的提示策略能提升 LLM 輸出的相關性。Nashid et al. (2023) 的檢索式提示選擇策略雖能改善錯誤修復表現，但其少樣本方案在結構複雜、同步邏輯多變的情境下仍存在召回率不足的問題。而TIPS Framework 則進一步結合小型模型進行先驗篩選，搭配 CoT 推理鏈模擬人類逐步判斷流程，提升了在多類別技術債分類任務中的準確度與穩定性。此外，與傳統機器學習分類器相比，TIPS Framework 不僅提供標籤預測，更透過推理鏈呈現決策過程，使結果具備可解釋性(explainability)。回應了 Fan et al. (2023) 所指出的研究缺口，即傳統分類器在軟體工程應用中雖能達成一定準確率，但缺乏推理可追溯性，限制了實務採用。

而在不平衡資料集上，雖然所有模型效能均顯著低於平衡資料集結果。但TIPS Framework仍優於大多數傳統分類器，顯示該框架能部分緩解資料稀疏帶來的性能衰退。針對此結果，先前文獻亦有提到LLM 與傳統分類器在不平衡或長尾分佈情境下效能會下降。Mišić & Dodović (2024)提到GitHub Copilot 在低頻率並行錯誤樣本上表現不穩定，需額外依賴靜態分析工具補強。Niu et al. (2024) 亦指出，標準提示策略在稀有錯誤類型偵測的 Recall 下降幅度可高達 40%，導致系統在實務場景中錯漏率偏高。相比之下，TIPS Framework 藉由小型模型進行前置分類過濾，並結合CoT與demonstration提供LLM 額外上下文資訊，降低了錯誤類別樣本不足造成的判斷不確定性。這種多模組提示設計的增益效果，使得 TIPS Framework 在非理想資料分佈下仍能維持較佳的 Precision 與整體穩定性。

接著，我們進一步探討 TIPS Framework內各組件對多執行緒程式碼分類的影響，同時也為了回應當前研究的一樣缺口，多數基於提示工程的 LLM 研究證實提示策略能提升推理表現，但鮮少針對複合式提示框架中各子模組的實質貢獻進行系統性評估(Wei et al., 2022; Brown et al., 2020)。而實驗結果指出，移除Small Model後相較於完整TIPS Framework顯著降低，顯示初步分類訊息能有效協助 LLM 收斂推理方向，減少誤判。移除CoT後，雖與 Small Model 消融結果相近，但在但在「Thread Lifecycle」與「Thread Pool Creation」任務中 Recall顯著降低，凸顯 CoT 能提升 LLM 對少數類別樣本的檢測率，並支撐多步邏輯判斷。而在移除Demonstrations後，F1-score雖然小幅度下降，但在Mutex Management 任務等特定情境中，能明顯改善 Precision，表示相似範例提示有助於 LLM 生成同步策略正確的程式碼。

最後，我們表現了TIPS Framework在多執行緒設計面向程式碼生成的成效，先前文獻多針對一般程式碼生成進行探討，例如 Ahmed & Devanbu (2022) 與 Nashid et al. (2023) 證實加入範例提示能提升 LLM 生成程式碼的語境一致性。但並未針對多執行緒同步控制任務進行深入評估。Niu et al.,(2024)與Mišić & Dodović, (2024)則提到模型在涉及多執行緒同步控制時仍易出現遺漏臨界區保護或誤用同步原語的問題。而TIPS Framework則補足了此缺口，證實當提示設計結合 CoT 與範例檢索時，LLM 能更精確掌握同步原語的正確使用，避免多執行緒程式碼中常見的臨界區保護遺漏問題。呼應 Zhao et al. (2023) 提出的「邏輯推理提示(LogiCoT)」概念，顯示邏輯化推理鏈能引導 LLM 在程式碼生成過程中遵循正確步驟，提升整體安全性與可靠性。

# Conclusion

本研究提出基於資源受限環境下設計的提示工程架構TIPS Framework，以提升LLM在程式碼分類與生成任務中的應用效能與推論可控性。TIPS Framework 採用小型語言模型作為語意預篩器，結合In-Context Learning與Chain-of-Thought構建一個具語境理解、語意對齊與多模組推理功能的提示鏈系統。實證結果顯示，TIPS Framework不僅標準分類上全面優於傳統機器學習模型，且在處理資料不平衡、多錯誤類別混雜與程式邏輯複雜度高的真實開發場景中，仍能維持高泛化能力與穩健性。此外，消融實驗(ablation experiments)進一步驗證 TIPS Framework 內部各核心模組對整體性能具關鍵貢獻。多模組整合設計在資訊提供與推理支撐上相輔相成，非單一模組可取代。尤其是小型模型作為前置分類器，不僅減輕 LLM 的推理負擔，更能有效引導其關注高潛在意義的特徵區段，有助於建構具語意對齊與邏輯連貫的提示語結構。這些發現不僅對提示工程策略的設計提供實證支持，也補足過往僅關注 LLM 架構優化與大規模訓練數據的研究取向，強調「提示流程設計本身」即為一種具備強大推理增益潛力的獨立研究主體。

然而，本研究仍存在些許限制，第一，實驗資料主要來自GitHub上之Pthread原始碼專案，雖經人工標註與錯誤注入處理，仍可能存在語言範疇與任務場景上的侷限性。特別是在同步控制邏輯表達與程式碼風格多樣性的部分，資料分佈未能涵蓋更廣泛的語言特性與異構架構(如 Rust 的 async/await 機制或 OpenCL 的多核心管理)。因此，模型在其他程式語言與同步抽象上的推論能力仍需更進一步驗證其可遷移性與語義一致性。第二，TIPS Framework 雖結合小型模型與提示策略，有效提升推論效率與可控性，但其建構過程仍需大量人工參與與特定任務知識，尤其在 CoT 推理鏈模板撰寫與語意相似範例篩選上，需仰賴研究人員主觀經驗與語意理解能力。此一特性可能限制其在不同任務或語境下的快速部署與通用應用能力，亦增加了提示語設計的門檻與成本。第三，TIPS Framework 所能發揮之整體效益，在一定程度上受到後續所連結LLM推理能力的調節影響。當後端所使用之語言模型具備高度語境理解與推理規劃能力時，即便未結合 TIPS 所提供之提示鏈與先驗資訊，仍可能產生接近甚至優於框架輔助後的輸出品質。相對地，於中階推理能力之語言模型中，本研究所設計之語意導向範例選取、結構化推理鏈模板與小模型預篩機制，則展現出顯著的推理引導與語義強化效果。

在實質貢獻上，TIPS Framework，專為資源受限環境下的程式碼分類與生成任務設計。透過結合輕量級小型模型、小樣本學習策略(In-Context Learning)與邏輯推理提示(Chain-of-Thought, CoT)，實現對中階語言模型的語義補強與推理增益，降低對高運算資源的依賴。此一架構亦具高度模組化，可與現有 LLM API 進行低成本整合，適用於教育場景、開發平台與語義輔助工具等多元應用情境，為提升 LLM 在真實世界任務中之可用性與可控性提供一套可行解法。

而在理論貢獻上，本研究回應並深化了提示工程(Prompt Engineering)領域中關於「結構化提示設計」與「語境適應性」的關鍵議題。不同於傳統僅關注提示策略分類(如 zero-shot、few-shot、CoT)，TIPS Framework 採用多模組整合方式，建構由語意相似檢索、小模型分類預測、推理鏈構造與信心調節所組成之提示流程。TIPS Framework揭示LLM與Prompt Engineering間的互補關係，補足過去研究對於「模型基線能力與提示依賴程度交互影響」的探討空缺，拓展了語言模型應用策略設計的理論邊界。

本研究所提出的 TIPS Framework讓LLM的參數與規模不再是唯一解答，而映現出另一種可能，將結構、語義與推理融入提示設計，提供一種具可持續性與高適應性的語言模型應用策略。TIPS Framework 展現了語言模型應用從高度依賴雲端資源、受限於黑盒架構的模式，逐步轉向可部署於本地、適用於教學與開發現場的實用化發展路徑。我們期待的目標是：讓智慧技術不再集中於少數平台與組織手中，而能普遍地服務於日常開發、教育與學習場域。

Reference

1. The benchmark can be found at <https://github.com/MayAlsofyani/Pthread-Benchmark> [↑](#footnote-ref-1)