AMPO：自動多分支提示優化

Sheng Yang [1*†] Yurong Wu [1*†] Yan Gao [2‡] Zineng Zhou [3†] Bin Benjamin Zhu [2] Xiaodi Sun [2] Jian-Guang Lou [2‡]
Zhiming Ding [1] Anbang Hu [2] Yuan Fang [2] Yunsong Li [2]
Junyan Chen [2] Linjun Yang [2]
[1] 中國科學院軟體研究所 & 中國科學院大學 [2] Microsoft
[3] 中國科學院計算技術研究所 & 中國科學院大學
{yangsheng22, wuyurong20, zhouzineng22}@mails.ucas.ac.cn; {yan.gao, binzhu, sunstifler, jlou, anbhu, juliefang, yunsongli, Junyan.Chen, Yang.Linjun}@microsoft.com; zhiming@iscas.ac.cn

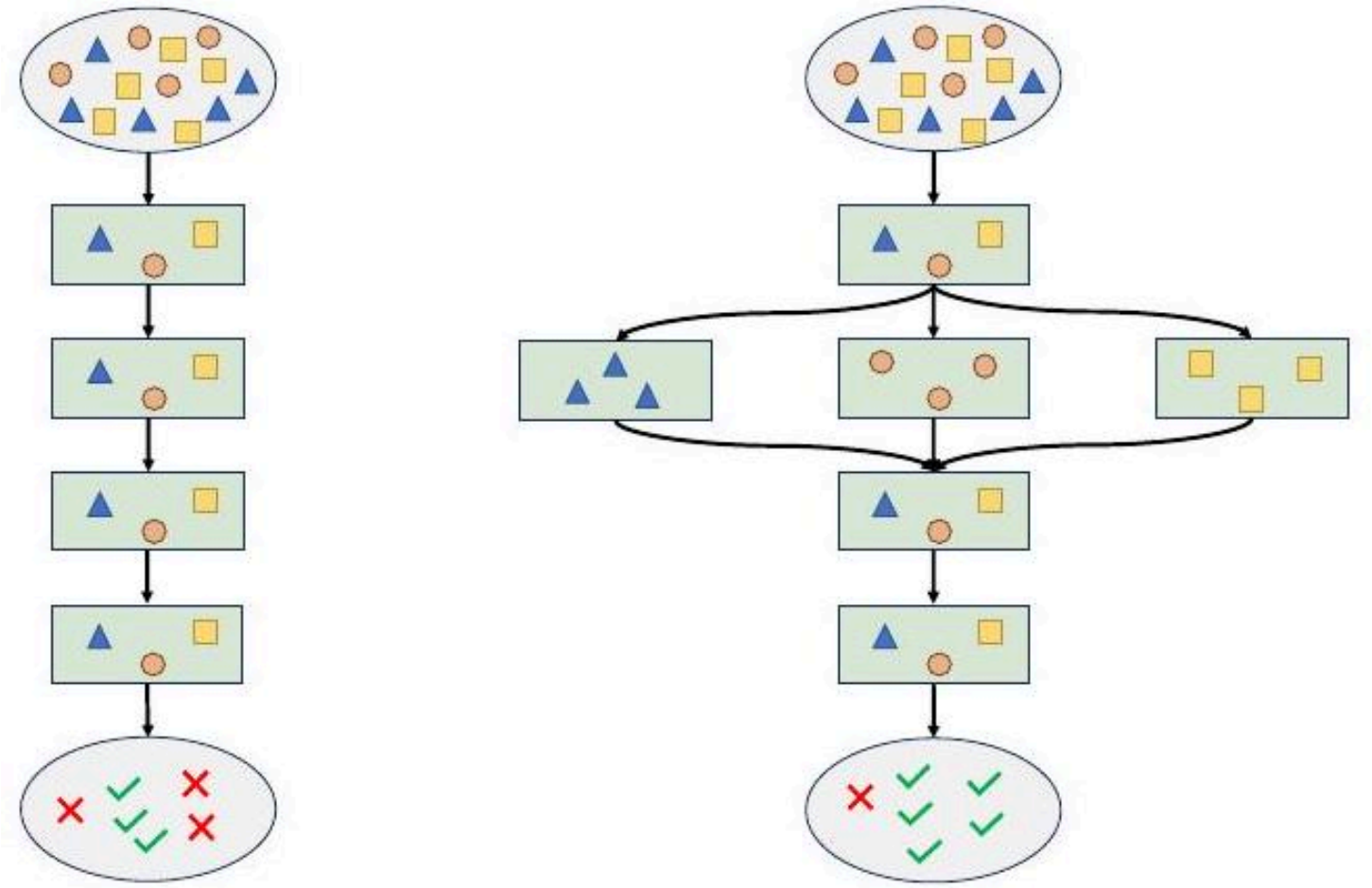
摘要

提示工程（prompt engineering）對提升大型語言模型（LLMs）的效能非常重要。處理複雜問題時，提示工程師往往會從範例中萃取多種模式，並注入相關解法來優化提示，以達到令人滿意的結果。然而，現有的自動提示優化技術僅限於產生單一流程的指示，難以處理多樣化的模式。在本文中，我們提出 AMPO，一種能夠利用失敗案例作為回饋，反覆發展為多分支提示的自動提示優化方法。我們的目標是探索以多分支結構化提示來更好地處理複雜任務中多重模式的新途徑，為此我們引入三個模組：Pattern Recognition、Branch Adjustment 與 Branch Pruning。在五項任務的實驗中，AMPO 持續達到最佳成績。此外，由於我們採用最小搜尋策略，我們的方法也展現出顯著的優化效率。

1 引言

提示工程涉及為大型語言模型 (LLMs) 創建最佳提示以提升其表現。它在 LLM 應用的開發中變得越來越重要（Brown et al., 2020；Welleck et al., 2022）。要創建合適的提示通常需要大量的人力、專業知識以及多次試錯（ZamfirescuPereira et al., 2023）。因此，研究高效的自動提示工程技術是至關重要的（Zhang et al., 2022；Chen et al., 2023）。

近年來，基於 LLMs 的自動提示優化方法已被廣泛應用——



\captionsetup{labelformat=empty}
Figure 1: Figure 1: 我們的提示優化方法旨在將單一流程的指令迭代優化為多分支格式，以處理多種模式。

ration (Zhou et al., 2022; Yang et al., 2023; Schnabel and Neville, 2024)。這些研究通常將 LLMs 作為提示優化器，用以逐步增強目標模型的提示。特別是，一種具前景的範式是基於回饋的優化方法，其中以 LLM 為基礎的提示優化器像人類專家一樣分析並修正失敗案例 (Pryzant et al., 2023; Wang et al., 2023b; Ye et al., 2023)。這些方法明確利用 LLMs 的自我反思能力來精煉提示，並模擬「梯度」朝向錯誤下降方向的步驟，然後「反向傳播」到提示。目前，這一提示優化範式已取得可觀的進展並引起廣泛關注。

儘管基於回饋的優化方法取得了成功，我們發現優化後的 prompts 主要是透過仔細重寫某些步驟、試圖提供更多細節來達成。本質上，它們主要是一種單一流程的指示。如圖 1 所示，處理複雜問題時，事先對問題進行分類可以幫助 LLM 更有系統地分析問題，也更容易找到合適的解決方案（Thomas et al., 2023；Ren et al., 2024；Mao et al., 2023）。顯而易見，產生單一流程指示的 LLM 優化器難以應對各種模式。例如，現有的基於回饋的方法要麼無法提供有效的修正，要麼在失敗案例未被單一流程指示處理時導致大量回歸（Ma et al., 2024）。

鑑於現實場景的複雜性，我們主張單一的指令流程不足以應對。LLM 優化器應從失敗案例中識別多種模式，進而探索並開發更具適應性的提示結構。基於此動機，我們提出 AMPO，一種將提示適度轉換為多分支格式，並以失敗案例為指引來優化提示的方法。此過程的關鍵在於創建與任務複雜度相匹配的自適應提示結構。為達成此目標，我們引入三個模組： (1) Pattern Recognition 負責分析失敗案例的模式。 (2) Branch Adjustment 可自適應地在新增分支以應對新模式或提供更多細節以強化現有分支之間進行選擇。 (3) Branch Pruning 採用預剪枝與後剪枝技術以防止提示過擬合。 我們的多分支提示結構的優點在於，它讓目標 LLM 能在推理階段自主決定每個樣本的最適路徑，從而增強處理多樣型態與複雜度的能力。

我們在五個任務上進行實驗，涵蓋不同層次的複雜度。實驗結果顯示，AMPO 在來自一般自然語言理解（General NLU）與領域知識（Domain Knowledge）的五個任務中持續取得最佳成果。此外，與其他基於回饋的優化方法相比，我們的方法由於採用了最小搜尋策略，展現出令人印象深刻的優化效率。

我們的貢獻如下：
(1) 我們提出 AMPO，一種自動提示優化方法，可利用失敗案例作為回饋，迭代構建多分支提示。值得注意的是，AMPO 目前是已知第一個被設計用於生成多分支提示的提示優化方法。
(2) 據我們所知，我們是首個探索以多分支方式構建提示，以更好處理複雜任務中多重模式的研究。我們的結果顯示，AMPO 能夠從資料分佈自動構建有效的多分支提示，從而容納複---
複雜任務。
(3) 我們在五個任務上進行實驗。AMPO 持續優於其他基於回饋的提示優化方法。同時，實驗結果顯示，由於我們採用了最小化搜尋策略，AMPO 顯現出令人印象深刻的優化效率。

2 激勵示例

為了說明在處理複雜問題時識別並處理不同模式的重要性，我們以搜尋查詢理解任務為例。在此任務中，我們使用 LLM 以當前查詢和使用者的搜尋歷史作為輸入，來預測個人化的查詢意圖。俗話說「十個人眼中有十個哈姆雷特」，預測個人化意圖相當困難。根據專家經驗，最佳方法是先將使用者的搜尋行為分類為不同類別，然後為每個類別制定預測個人化意圖的策略。例如，若當前查詢為重新尋找（refinding）查詢——與先前查詢相同——最可能的意圖是重訪使用者最近點選的相同網頁。若當前查詢是由任何先前查詢衍生的重新改寫（reformulation）查詢，則應根據改寫的性質來個人化意圖，例如將原查詢擴展或篩選。 當查詢為領域偏好查詢，且使用者的歷史顯示對與當前查詢相關的特定領域有偏好時，我們應利用此領域偏好來精煉查詢意圖。受此啟發，AMPO 引入多個分支以處理各類別，並自適應地擴展或修剪這些分支以達到最適當的覆蓋範圍。

3 相關工作

3.1 提示指令的結構

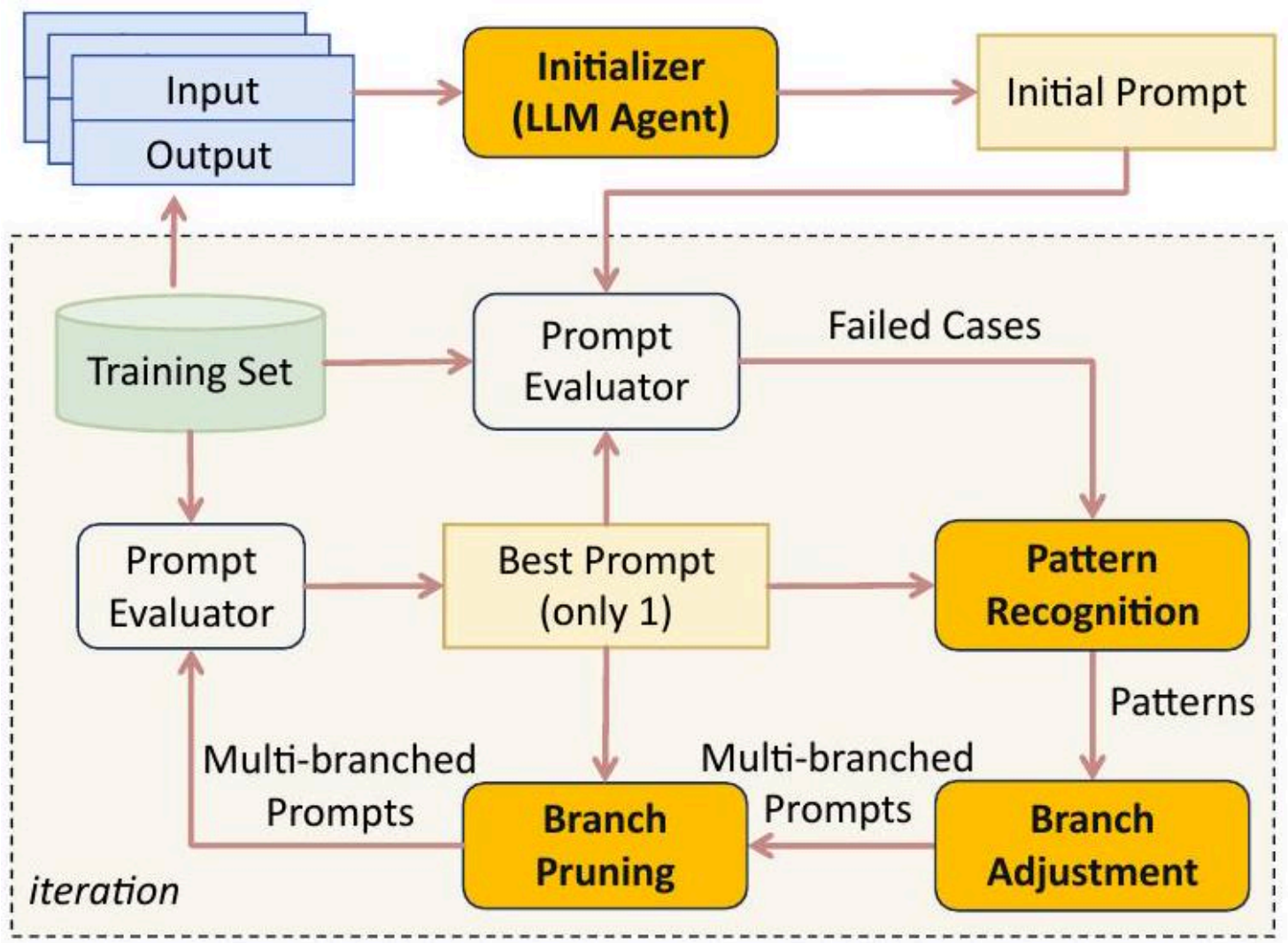
提示顯著影響 LLMs 的效能（White et al., 2023；Liu et al., 2023）。大量研究證實，將問題拆解為多個中間步驟可明顯提升輸出品質（Wei et al., 2022），特別是在需要推理的任務（Wang and Zhou, 2024；Yasunaga et al., 2023）與規劃的任務（Wang et al., 2023a；Zhang et al., 2023）。為了進一步改善這些中間步驟，許多研究者利用模型的上下文學習能力（Kojima et al., 2022；Shum et al., 2023）。然而這些
雖然提示設計方法能增強 LLMs 的推理與生成能力，但它們仍依賴序列化的單流程結構，對於更複雜的情況仍不夠完善。在這項工作中，我們希望探索以多分支結構來構建提示的新方法，以更好地處理複雜任務中的多重模式。我們假設多分支提示結構可以讓目標 LLM 在推理過程中自主判斷每個樣本最合適的路徑。

3.2 Automatic Prompt Optimization

LLMs 的進展促成了許多探討自動 prompt 優化技術的研究（Ye et al., 2023；Schnabel and Neville, 2024）。這些研究通常可分為兩類：使用搜尋演算法（Guo et al., 2023；Zhou et al., 2022）或利用 LLMs 的自我反思能力（Pryzant et al., 2023；Yang et al., 2023）來找出最佳 prompt。近期也有嘗試將兩者結合，透過像蒙地卡羅樹搜尋（MCTS）等搜尋演算法與自我反思能力整合（Wang et al., 2023b）。在這種結合方法中，每個 prompt 被視為一個狀態，每次優化視為一個動作，藉由可追溯的樹狀搜尋實現更精細的 prompt 優化。我們的方法屬於自我反思類別。AMPO 與既有方法的關鍵差異在於，AMPO 將 prompt 的結構視為一個優化目標。傳統方法通常將 prompt 視為單一段落，整體進行優化。相對地，AMPO 從錯誤案例中識別多種模式，透過新增分支或強化既有分支，將 prompt 精煉為多分支結構。 這種多分支結構使得 LLMs 能更有效地處理資料分布高度多樣的任務。

4 方法論

給定一個任務 τ ，由提供的訓練集 $D_{\text{train}} = \{(q_1, a_1), (q_2, a_2) \ldots, (q_n, a_n)\}$ 指定，其中 q_i/a_i 是每個條目的輸入/輸出配對，我們從初始提示 P_0 開始。模型輸入由 P 和 q_i 組成，基礎 LLM β 根據 $p_\beta(a_i \mid q_i, P)$ 做出預測。提示優化的目標是找到能使衡量函數 R （例如準確率）在 D_{train} 上達到最大化表現的最佳提示 P^* 。



\captionsetup{labelformat=empty}
圖 2：圖 2：AMPO 的整體框架

這可以形式化定義為一個優化問題：$P^* = \text{argmax}_{P \in S} \sum_i R\left(p_\beta\left(a_i \mid q_i, P\right)\right)$，其中 S 表示自然語言提示的樣本空間。

4.1 概述

AMPO 的目標是利用失敗案例作為回饋，迭代地發展出一個多分支提示。此過程的關鍵在於產生一個能依任務複雜度自適應的提示結構。為達成此目標，我們的方法主要包含三個模組：(1) Pattern Recognition 負責分析失敗案例的模式；(2) Branch Adjustment 負責在新增分支以應對新興模式與根據已識別模式強化現有分支之間取得適當的可調適性平衡。適當的平衡對於提示結構的自適應性至關重要；(3) Branch Pruning 採用前剪枝與後剪枝技術以防止提示過擬合，進一步確保提示結構的可調適性。AMPO 的整體框架如圖 2 所示。

一開始，根據 few-shot 範例 (Zhou et al., 2022) [1] 生成初始提示 P_0。接著，我們開始以失敗案例作為回饋進行迭代式提示優化。演算法細節見演算法 1。以下我們對每個模組進行詳細介紹。

4.2 模式識別

給定當前提示 P 以及從訓練集 D_{train} 中抽取的 K 個錯誤範例，本模組的目標是執行錯誤分析以找出

```
Algorithm 1 AMPO
Require: Initial prompt: \(P_{0}\), Training set: \(D_{\text {train }}\), Valid
    tion set: \(D_{\text {val }}\), Iteration limit: \(T\)
    // LLM agents
```

```
Analyzer: Conduct error analysis.
Summarizer: Condense error reasons into patterns and
assign important scores.
Revisor: Edit the multi-branched prompt.
Initiate \(P=P_{0}\)
while \(t \leq T\) do
    Evaluate \(P\) on \(D_{\text {train }}\) and sample \(K\) failed cases
\(E=\left\{\left(x_{i}, y_{i}\right) \mid\left(x_{i}, y_{i}\right) \in D_{\
    // Pattern Recognition
    LLM-Analyzer conducts error analysis and iden-
tifies the error reasons for each failed case: \(R=\)
\(\left\{r_{1}, r_{2}, \ldots r_{K}\right\}\)
    LLM-Summarizer condenses \(R\) reasons into \(M\) pat-
terns and assign important scores from each pattern:
Patt \(=\left\{\left(\right.\right.\) patt \(\left._{1}, s_{1}\right), \ldo
    // Branch Adjustment
    Selects top \(N\) patterns by important scores.
    LLM-Revisor optimizes \(P\) based on top \(N\) Patt and
obtained optimized prompts \(P_{\text {opt }}=\left\{P 1, \ldots P_{N}\righ
    // Branch Pruning
    LLM-Revisor prunes the optimized prompts \(P_{o p t}\) and
obtained \(P_{\text {pruned }}=\left\{P 1, \ldots P_{N}\right\}\)
    Evaluate \(N\) pruned prompts \(P_{\text {pruned }}=\)
\(\left\{P 1, \ldots P_{N}\right\}\) on \(D_{\text {val }}\) and select the
\(P^{*}\).
    Update \(P=P^{*}\)
end while
return The best prompt.
```

每個錯誤範例的根本原因。為了達成此目的，我們為名為 LLM-Analyzer 的 LLM 代理建立了逐步指示。LLM-Analyzer 的元提示見附錄 A.3。通過分析輸出的原因，我們發現 LLM 所生成的原因常常相似，即使失敗案例不同。這導致基於回饋的方法優化效率低。此外，在 AMPO 中，相似的回饋可能導致提示內出現冗餘分支，最終影響其效能（如第 5.5 節所示）。

為了解決上述問題，我們使用另一個 LLM 代理（命名為 LLM-Summarizer），在每次迭代中將所有失敗案例的原因摘要為不同的模式。納入此摘要有幾個好處：(1) 它減少了重複的原因，與其他基於反饋的方法相比，能顯著提高優化效率。(2) 將原因摘要為模式可提升泛化能力，從而最小化多分支提示中的冗餘分支。此外，我們要求 LLM-Summarizer 為每個模式分配重要性分數
。在本文中，我們依重要性分數選擇前 N 個模式。這使得在分支調整過程中能選擇重要的模式，進一步減少需探索的提示數量，從而提升效率。LLM-Summarizer 的元提示載於附錄 A.4。

4.3 分支調整

Branch 調整模組的目標是根據 LLM-Summarizer 彙總出的模式來優化多分支提示。關鍵在於釐清某個模式應該用來強化現有分支還是建立新分支。最佳選擇取決於具體任務。例如，較複雜的任務往往會呈現更多樣的模式，此時多分支提示較為理想，因其具擴展性並能處理各種模式。相反地，對於較簡單的任務，單流程指示更為有效，因為它們較易遵循且更具健壯性。

在新增分支以處理新模式與為現有分支提供更多細節之間的適應性 在此模組中，我們使用一個稱為「LLM-Revisor」的 LLM 代理來修改多分支提示。具體而言，我們為 LLM-Revisor 定義了兩種操作類型：(1) 新增分支以處理新模式，(2) 提供附加細節以強化現有分支。透過分析現有分支與新模式，LLM-Revisor 判斷應該在深度上擴展提示（加入更多細節）或在廣度上擴展提示（加入更多分支）。此方法的優勢在於其彈性，允許根據任務的複雜度與模式分佈來調整提示結構。LLM-Revisor 的逐步操作說明載於附錄 A.5。

4.4 分支修剪

透過徹底檢視 LLM-Revisor 的編輯過程，我們觀察到多分支提示容易發生過擬合。通常在經過數輪迭代後，提示在測試集上的效能開始下降。與此同時，提示在感知上看似分支越來越多。這種情況發生在 LLM 記住訓練資料中的邊緣案例而無法抓住重要模式時。

受到機器學習中剪枝技術的啟發（Esposito et al., 1997; Kwon et al., 2022），我們提出兩種可能的解決方案：(1) 預剪枝透過提早停止避免提示進一步擴張。在每次迭代後，我們檢查交叉驗證誤差。如果誤差沒有顯著下降則停止。透過提早剪枝，我們獲得較精簡的提示，較不易對訓練資料過擬合。(2) 後剪枝則與預剪枝相反，允許提示成長至其完整深度。特別地，我們在 meta prompt 的最後加入一個步驟，讓 LLM-Revisor 再次檢視整套指令並刪除任何分支，以提升指令的泛化能力。

5 Experimental

5.1 Baselines

人類指令：人類提示是由人類根據其對原始資料集的理解與認知所設計的指示。對於每一項任務，我們採用領域專家撰寫、來自學術資料庫或專業網站的指令，並進行相應的修改以使其適配於我們的任務。

Chain-of-Thought (CoT) (Wei et al., 2022) 在問題後附加「讓我們一步一步思考。」以觸發模型的推理過程。

Chain-of-Thought 指令（CoT Instructions）(Wei et al., 2022)：我們隨機選取 5 個案例，並透過少量示範學習生成對應的思考鏈。之後，我們手動優化提示以提升表現。此提示亦被用作其他基於回饋的提示優化方法的初始提示。

APO (Pryzant et al., 2023)：APO 從錯誤範例生成自然語言層級的梯度，然後利用這些梯度對提示進行逆向編輯。

OPRO (Yang et al., 2023)：OPRO 利用歷史提示、分數與錯誤範例來引導 LLM 生成得分更高的提示。與 APO 不同，OPRO 在優化過程中不提供明確的回饋。

PromptAgent (Wang et al., 2023b)：PromptAgent 採用蒙地卡羅樹搜尋（MCTS）演算法來策略性地優化提示流程。

5.2 任務

為了在廣泛的應用上全面評估我們方法的有效性，我們從不同領域精心挑選了 5 個任務進行深入實驗：知名的文本分類任務 TREC (Voorhees and Tice, 2000)、廣為認可的情感分類任務 SST-5 (Socher et al., 2013) 以及大規模閱讀理解任務 RACE (Lai et al., 2017)。此外，我們選擇了兩個來自生物醫學領域的專業任務，這些任務在撰寫專家級提示時明確需要領域見解，分別為醫學問答任務 MedQA (Jin et al., 2021) 與 MedMCQA (Pal et al., 2022)。詳細資料集資訊請見附錄 A.1。

5.3 實作細節

在本研究中，我們使用 GPT-3.5-turbo 與 GPT-4turbo 作為目標模型，並以 GPT-4-turbo 擔任優化器。為了全面捕捉可能的錯誤回饋，我們將 Analyzer 的 temperature 參數設為 1。對於 Revisor，我們將 temperature 設為 0 以確保修改的精確性。我們從壞案例中抽樣 $K = 5$ 個，並為 LLM-Revisor 選取排名第一的 $N = 1$ 種模式進行優化。因此，我們僅保留一個 prompt 進行迭代。在 APO、PromptAgent 與 AMPO 的迭代過程中，我們從訓練資料中抽樣 10% 作為驗證集以評估 prompt 的效能。在實驗中，我們對每個實驗執行三次，並報告在測試集上評估結果的平均值。

5.4 主要結果

在表 1 中，我們展示 AMPO 所產生的 prompts 與 Human Instruction、CoT、CoT Instructions、APO、OPRO 及 PromptAgent 在三個領域五項任務上的比較。我們觀察到本方法在所有任務上顯著提升準確率，驗證了我們方法在優化 prompts 上的有效性。

AMPO 在複雜任務上顯著超越其他方法。以 MedQA 任務為例，該任務包含各種病況與複雜情境。因此，在處理此類問題時，LLMs 必須根據病人的具體情況識別不同模式，並提供最適合的治療方案。如表 1 所示，無論使用 GPT-3.5-turbo 或 GPT-4-turbo，人類-

LLMs	方法	一般自然語言理解 (General NLU)			領域知識	
		SST-5	TREC	RACE	醫學問答	醫學多選題
GPT-3.5-turbo	人類	51.56	69.60	79.25	61.25	45.75
	推理鏈 (CoT)	50.00	63.00	77.75	50.50	47.25
	推理鏈指示 (CoT 指示)	49.56	67.75	78.25	68.25	48.25
	APO	52.00	69.00	78.00	72.25	48.00
	OPRO	52.44	70.50	78.75	70.50	46.75
	PromptAgent	54.22	72.50	80.75	71.75	47.50
	OURS	55.781 .56	**76. 00** ↑ 3.50	$81.75^{1.00}$	**76. 50** ↑ 4.25	$48.75^{\uparrow}0.50$
GPT-4-turbo	人類	52.34	70.50	89.75	64.50	65.75
	思維鏈 (CoT)	53.86	63.50	88.50	63.50	69.75
	思維鏈指示	50.33	71.25	91.00	71.75	70.75
	APO	55.25	75.25	90.75	83.25	71.50
	OPRO	56.44	79.50	90.00	76.50	66.00
	PromptAgent	57.33	81.50	91.00	77.00	70.25
	OURS	$59.78^{2.45}$	**82. 00** ↑ 0.50	**91. 25**0.25	**89. 00** ↑ 5.75	73.00 ↑ 1.50

\captionsetup{labelformat=empty}

表 1：GPT-3.5-turbo 與 GPT-4-turbo 在五項任務上的效能比較，最高準確率以粗體標示。向上箭頭表示 OURS 超過第二高分的幅度。

Model	MedQA	TREC	SST-5
AMPO	**89. 00**	**82. 00**	**59. 78**
- 摘要	86.75	81.50	57.00
Δ	−2.25%	−0.50%	−2.78%
- 強化現有	86.25	78.25	55.33
分支	−2.75%	−3.75%	−4.45%
Δ	82.25	77.75	53.78
- 新增分支	−6.75%	−4.25%	−6.00%
Δ			

\captionsetup{labelformat=empty}

表 2：表 2：AMPO 在未使用摘要、強化現有分支與新增分支之消融實驗結果。報告的是完全匹配分數。

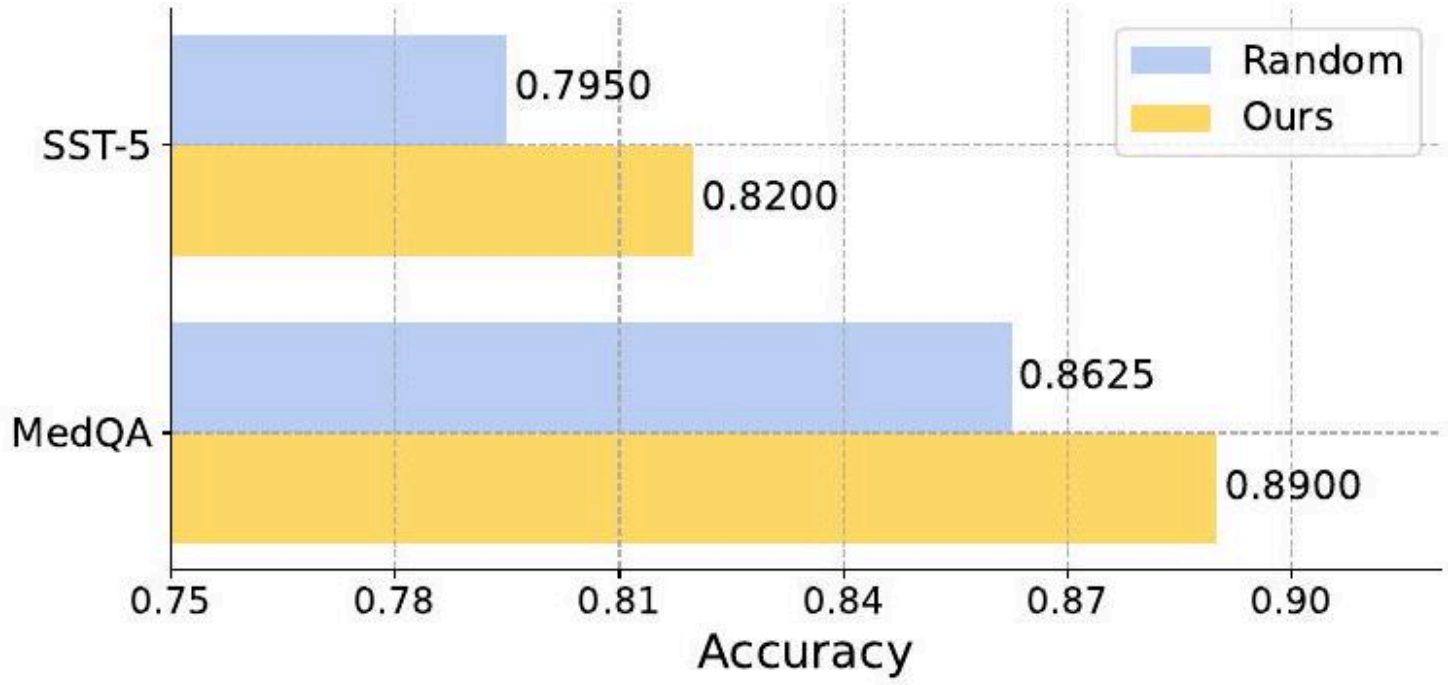
所構建的提示表現最差，可能是因為指示較為概括，與輸入資訊沒有緊密對齊。接著，透過少量示例生成的提示表現較好。再者，當以失敗案例作為回饋補充時，APO、OPRO 及 PromptAgent 的表現能進一步提升。最後，我們的方法相比其他方法（即 Human Instruction、CoT、CoT-Instructions、APO、OPRO 與 PromptAgent）在使用 GPT-4-turbo 時分別達到相對提升 24.50% 、 25.50%, 17.25%, 5.75% 及 12.00% 。原因在於我們的方法以多分支提示來處理複雜情境，先對問題進行分類，而非以單一路徑解決所有問題。實驗結果顯示 AMPO 在複雜任務上顯著超越其他方法。

AMPO 在一般任務上也優於其他方法。以相對一般的任務 RACE 為例。如表 1 所示，由人類和 few-少量示例兩者都表現良好。此時，採用基於回饋的方法如 APO 與 PromptAgent 帶來的改進甚微，甚至可能有害。例如，在 RACE 任務上，經 APO 優化的提示在以 GPT-4-turbo 為目標模型時，表現比初始提示（即 CoT-Instructions）低了 0.25% 。與此同時，AMPO 持續超越其他方法並達到最先進的表現。這表示我們的方法即使在一般情況下也能良好運作。這意味著我們的方法可以靈活地從資料分佈中創建自適應的多分支提示，從而適應複雜或一般的任務。

5.5 消融研究

為了系統性地研究 AMPO 中模式總結（pattern summarization）、多分支的自適應性（adaptivity of multi-branched）與分支剪枝（branch pruning）之影響，我們在三個任務上進行了完整的消融實驗。結果如表 2 所示。

摘要 在 AMPO 的每次迭代中，我們整合了 LLM-Summarizer，將 K 個抽樣失敗案例的錯誤原因總結為 M 種模式。若移除 LLM-Summarizer，由 LLMAnalyzer 所產生的錯誤原因會直接批次地送入 LLM-Revisor。根據表 2 的結果，在 MedQA、TREC 以及 SST-5 任務中，表現分別下降了 $2.25\%, 0.50\%$ 與 2.78% 。這表示在進行修正之前，將錯誤原因總結為共通模式是至關重要的。



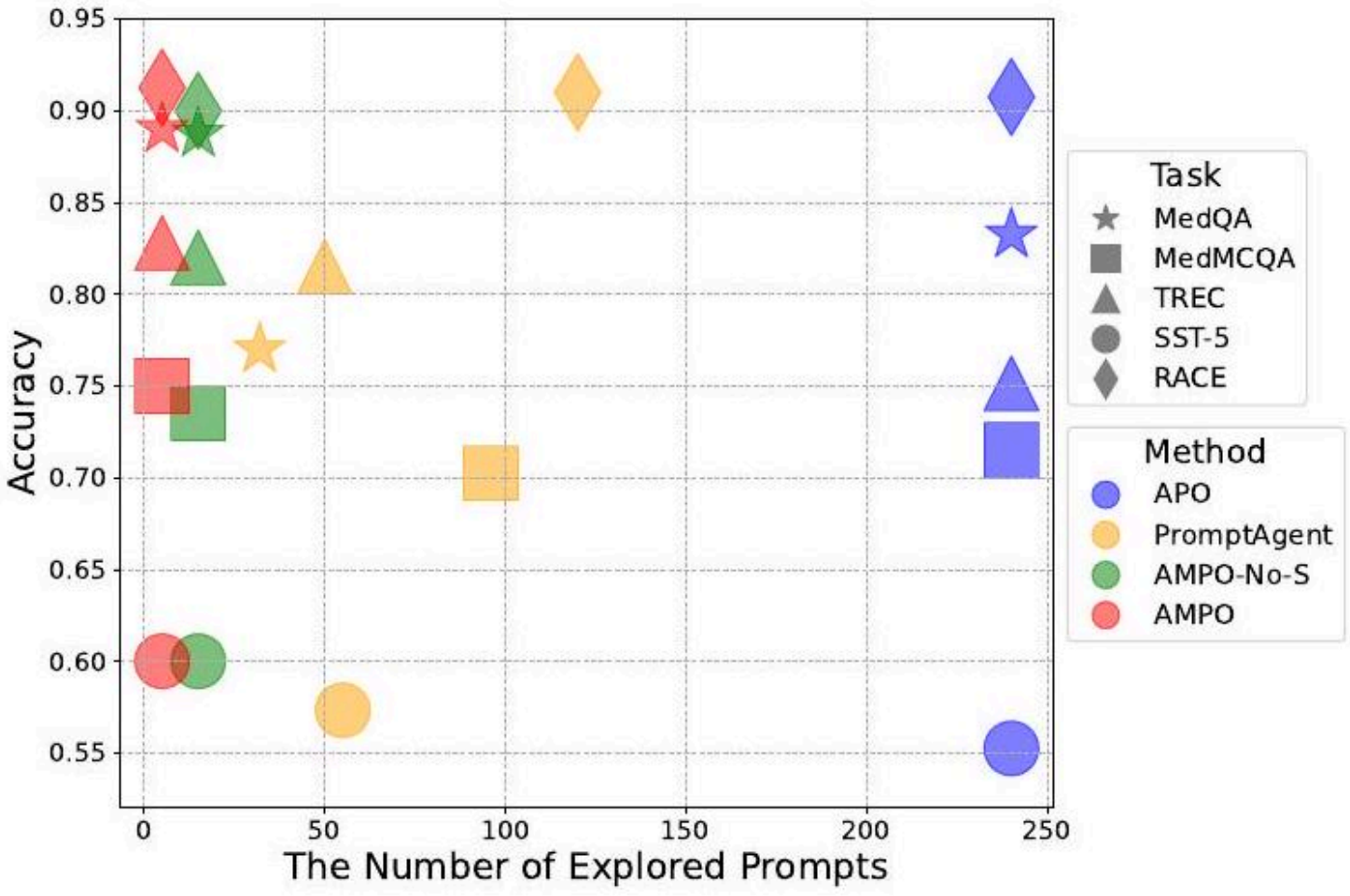
\captionsetup{labelformat=empty}
圖 3：圖 3：我們的模式選擇策略與隨機策略在 MedQA 及 SST5 上的比較。

自適應調整 AMPO 的一項重大創新是其在擴展新分支與強化多分支提示現有分支之間的自適應性，因此有必要透過消融實驗來驗證此功能的重要性。具體來說，我們修改了 LLM-Revisor 的元提示，移除可強化現有分支或新增分支的選項。實驗結果顯示，在未新增分支的情況下，三項任務的平均表現明顯下降 5.67% 。值得注意的是，若移除新增分支的操作，AMPO 會實際退化為 APO。當 LLM-Revisor 只能新增分支時，表現亦平均下降 3.65% ，但仍比僅強化現有分支高出 3.03% 。透過此研究，我們得到兩項結論：(1) 新增分支與強化現有分支皆為發展自適應多分支提示的關鍵；(2) 在處理複雜任務時，新增分支較強化現有分支更為重要。

6 分析

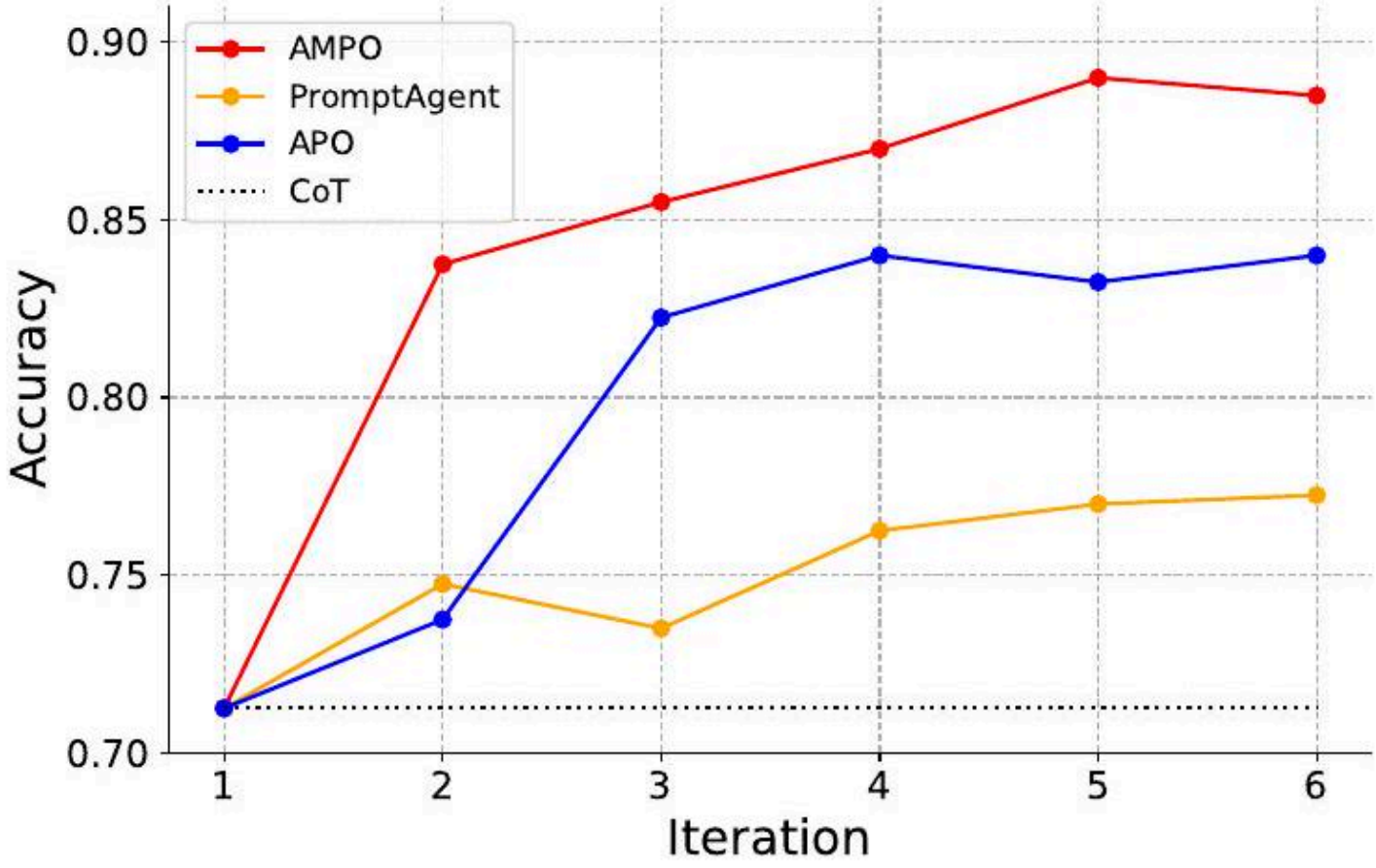
6.1 模式選擇策略

在我們的實驗設計中，我們一開始隨機抽樣 $K = 5$ 個失敗案例，並使用 LLM-Analyzer 進行分析，而不指定固定的失敗原因數量。之後，所有識別出的原因都會交由 LLM-Summarizer，將它們整合成數個主要類別。與此同時，我們要求 LLM-Summarizer 為每個主要原因指派一個重要性分數。接著，我們依重要性分數選出前 $N = 1$ 個最關鍵的模式。圖 3 顯示我們的搜尋策略平均比隨機抽樣提升了 2.75% ，這證明了我們模式選擇策略的有效性。



\captionsetup{labelformat=empty}

圖 4：圖 4：探索效率分析。我們的方法在最少的探索提示下達成最佳結果。水平軸代表中間探索提示的數量，垂直軸代表準確度。此處，AMPO-No-S 指的是沒有 Summarizer 的 AMPO。



\captionsetup{labelformat=empty}

圖5：圖5：收斂性分析

6.2 探索效率分析

探索效率對於降低計算成本至關重要。因此，我們將本方法與三個強基線比較所探索的提示數量。在圖 4 中，我們在各種任務中以最少的探索提示達成最佳結果。以 MedQA 任務為例，我們對 4 種不同方法執行 5 次迭代，計算它們所產生的提示數量。每次迭代中，APO 從上一迭代的 4 個提示各生成 3 個提示，並從中選出在驗證集上表現最好的 4 個提示作為下一迭代，導致總共 240 個探索提示。PromptAgent 使用深度為 3、寬度為 3 的 MCTS 搜尋策略（Winands et al., 2008），在 5 次迭代後產生總共 52 個提示。與 PromptAgent 與 APO 相比，AMPO 所探索的提示數量分別少 6.4 倍與 48 倍，卻分別達成 12% 與 5.75% 的性能提升。我們方法高效率的主要原因有三點：(1) 我們採用一個

一名 67 歲男性因過去 45 分鐘出現劇烈壓迫性胸骨後疼痛被送至急診。疼痛放射至左肩，伴有 S4 及雙側...已開始給予多巴胺、嗎啡、硝酸甘油與鏈激酶。以下哪一項會是在接下來 24 小時內對此病人最令人擔憂的情況？

AMPO-Optimized Prompt

1. Read the input carefully to understand the context and specifics of the scenario:
– **If** the scenario is clinical, focus on the patient's condition and any special circumstances (like pregnancy).
– **Else** if the scenario is non-clinical, such as a biochemical experiment, ensure comprehension of the scientific principles involved.
2. Identify key information.
...
3. Consider the context:
– **If** it's a clinical scenario, ... **If** the scenario is diagnostic in nature and suggests an embryologic error, ...
– **Else if** the scenario is treatment-oriented, proceed to the next steps with a focus on treatment selection.
– **Else if** it's a non-clinical scenario, ...
4. Review options based on the context:. ...– **Else if** treating, review treatment options and prioritize based on safety and effectiveness. ...– **If** the case involves enzyme kinetics, provide specific guidance on interpreting data such as Km and Vmax.
5. If multiple options seem appropriate:– **If** in a clinical case, ...
6. Make a decision:
– **If** in a clinical scenario, ...
– **If** diagnosing, ...
– **Else if** in a non-clinical scenario, choose the conclusion or next step that best fits the experimental data and scientific knowledge.

Analysis

Firstly, this scenario is clinical, dealing with a patient who presents with severe, retrosternal chest pain, radiating to the left shoulder and associated with sweating-clear indications of an acute myocardial infarction. ... .treatment-oriented, since immediate medical interventions have been applied and their outcomes must be monitored closely. ... . So the answer is A.

Options

(A) Hemoptysis
(B) Easy fatigability
(C) Persistent ST-segment elevation
(D) Shortness of breath

圖 6：這是一個來自 MedQA 的範例，AMPO 優化後的 prompt 使 LLM 做出正確選擇，而其他方法皆未成功。直觀上，AMPO 產生的多分支 prompt 在不同步驟設有分支，使用 if-else 陳述考量各種條件。與單一流程指示相比，它能處理更廣泛的模式，從而達到較佳的表現。我們用不同顏色突顯出 prompt 的各種判斷條件。從此範例可見，多分支 prompt 先引導 LLMs 將問題區分為臨床與非臨床，進一步判斷是否為診斷或治療導向，同時評估患者是否適合接受治療。針對涉及酵素動力學的案例，則會引導 LLMs 提供該領域的專門見解。
貪婪搜尋策略，意即在每一輪只保留一個最好的 prompt。(2) 我們的 LLM-Summarizer 將失敗案例中的所有錯誤原因濃縮為數個模式。(3) 此外，LLM-Summarizer 為每個模式分配一個重要性分數，讓 LLMRevisor 能夠過濾這些模式，進一步減少需要探索的 prompt 數量。

6.3 收斂性分析

為了進一步研究 AMPO 的學習過程，我們監測並視覺化了每一輪過程中 prompt 的效能變化。具體來說，我們記錄並繪製了 MedQA 任務中所有基線在六輪中的效能軌跡，說明了各種 prompt 優化方法效能的演化（見圖 5）。我們觀察到三種優化方法皆呈現整體上升趨勢，但 AMPO 的增幅顯著較大，僅用一次迭代便直接從 71.25% 跳到 83.75% 。多分支 prompt 具備更佳的可擴展性與處理不同模式的更大容量。與其他方法不同，我們的方法能夠擴展
除了修改現有分支外還能新增分支，使其能處理更多情境並降低修改原始提示的難度。

6.4 個案研究

為說明 AMPO 如何利用多分支提示解決問題，我們進行了定性分析。以 MedQA 的一個範例展示，我們的方法能從複雜資料中準確分類各種情境，並為每種情況設計詳細的解法，最終導向正確答案。從此範例的分析結果可見，LLM 先判斷病人狀況為臨床性並發現有急性心肌梗塞的明確指徵，接著判定情境為以治療為導向，最終得出正確答案。

7 結論

在本文中，我們提出了 Automatic MultiBranched Prompt Optimization (AMPO)，它會將提示顯式地轉換為多分支格式，然後利用失敗案例作為反饋進行迭代精煉。
具體而言，我們採用三個 LLM 代理協同工作，並提出引導原則以平衡多分支結構的可適應性。實驗結果顯示，AMPO 相較於現有最先進的基於反饋的優化方法具有更佳的表現，同時顯著提升了優化效率。

8 限制

多分支提示詞優化要求 LLMs 具備強大的邏輯推理能力。為了降低複雜度，我們採用了分而治之的方法，設計了三個角色：LLMAnalyzer、LLM-Summarizer 以及 LLM-Revisor。此外，我們設計了逐步的元指令來指導如何生成一個能適應不同難度任務的自適應提示詞結構。儘管如此，我們的方法仍然依賴於 LLMs 本身的能力。由於當前它們能力的限制，有時模型可能無法嚴格遵循元指令，導致結果不理想。然而，透過未來使用更優秀的 LLMs，我們可以進一步提升此方法的效能。

參考文獻

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. Advances in neural information processing systems, 33:1877-1901.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2023. Instructzero: Efficient instruction optimization for black-box large language models.

Floriana Esposito、Donato Malerba、Giovanni Semeraro 與 J Kay。1997。A comparative analysis of methods for pruning decision trees。IEEE transactions on pattern analysis and machine intelligence, 19(5):476–491。

Qingyan Guo、Rui Wang、Junliang Guo、Bei Li、Kaitao Song、Xu Tan、Guoqing Liu、Jiang Bian 與 Yujiu Yang。2023。Connecting large language models with evolutionary algorithms yields powerful prompt optimizers。arXiv preprint arXiv:2309.08532。

Di Jin、Eileen Pan、Nassim Oufattole、Wei-Hung Weng、Hanyi Fang 與 Peter Szolovits。2021。What disease does this patient have? a large-scale open domain question answering dataset from medical exams。Applied Sciences, 11(14):6421。

Takeshi Kojima、Shixiang Shane Gu、Machel Reid、Yutaka Matsuo 與 Yusuke Iwasawa。2022。Large language models are zero-shot reasoners。Advances in neural information processing systems, 35:22199–22213。

Woosuk Kwon、Sehoon Kim、Michael W Mahoney、Joseph Hassoun、Kurt Keutzer、Amir Gholami。2022。A fast post-training pruning framework for transformers。Advances in Neural Information Processing Systems，35：24101-24116。

Guokun Lai、Qizhe Xie、Hanxiao Liu、Yiming Yang、Eduard Hovy。2017。RACE: Large-scale ReAding comprehension dataset from examinations。In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing，pages 785–794，Copenhagen，Denmark。Association for Computational Linguistics。

Pengfei Liu、Weizhe Yuan、Jinlan Fu、Zhengbao Jiang、Hiroaki Hayashi、Graham Neubig。2023。Pretrain, prompt, and predict: A systematic survey of prompting methods in natural language processing。ACM Computing Surveys，55(9)：1-35。

Ruotian Ma、Xiaolei Wang、Xin Zhou、Jian Li、Nan Du、Tao Gui、Qi Zhang、Xuanjing Huang。2024。Are large language models good prompt optimizers? arXiv preprint arXiv:2402.02101。

Kelong Mao、Zhicheng Dou、Fengran Mo、Jiewen Hou、Haonan Chen 與 Hongjin Qian。2023。《Large language models know your contextual search intent: A prompting framework for conversational search》。arXiv pre 印本 arXiv:2303.06573。

Ankit Pal、Logesh Kumar Umapathi 與 Malaikannan Sankarasubbu。2022。《Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering》。收錄於 Conference on health, inference, and

learning，頁 248–260。PMLR。

Reid Pryzant、Dan Iter、Jerry Li、Yin Tat Lee、Chenguang Zhu 與 Michael Zeng。2023。《Automatic prompt optimization with" gradient descent" and beam search》。arXiv pre 印本 arXiv:2305.03495。

Xubin Ren、Wei Wei、Lianghao Xia、Lixin Su、Suqi Cheng、Junfeng Wang、Dawei Yin 與 Chao Huang。2024。《Representation learning with large language models for recommendation》。收錄於 Proceedings of the ACM on Web Conference 2024，頁 3464–3475。

Tobias Schnabel 和 Jennifer Neville。2024。Prompts as programs: A structure-aware approach to efficient compile-time prompt optimization。arXiv preprint arXiv:2404.02319。

KaShun Shum、Shizhe Diao 和 Tong Zhang。2023。Automatic prompt augmentation and selection with chain-of-thought from labeled data。arXiv preprint arXiv:2302.12822。

Richard Socher、Alex Perelygin、Jean Wu、Jason Chuang、Christopher D Manning、Andrew Y Ng 和 Christopher Potts。2013。Recursive deep models for semantic compositionality over a sentiment treebank。收錄於 Proceedings of the 2013 conference on empirical methods in natural language processing，第 1631–1642 頁。

Paul Thomas、Seth Spielman、Nick Craswell 和 Bhaskar Mitra。2023。Large language models can accurately predict searcher preferences。arXiv preprint arXiv:2309.10621。

Ellen M Voorhees 和 Dawn M Tice。2000。建立問答測試集。收錄於第 23 屆國際 ACM SIGIR 年會論文集：資訊檢索研究與發展，頁 200–207。

Lei Wang、Wanyu Xu、Yihuai Lan、Zhiqiang Hu、Yunshi Lan、Roy Ka-Wei Lee 和 Ee-Peng Lim。2023a。Plan-and-solve prompting：透過大型語言模型改善零次鏈式思考推理。arXiv preprint arXiv:2305.04091。

Xinyuan Wang、Chenxi Li、Zhen Wang、Fan Bai、Haotian Luo、Jiayou Zhang、Nebojsa Jojic、Eric P Xing 和 Zhiting Hu。2023b。Promptagent：以語言模型進行策略規劃以實現專家級提示優化。arXiv preprint arXiv:2310.16427。

Xuezhi Wang 和 Denny Zhou。2024。無需提示的鏈式思考推理。arXiv preprint arXiv:2402.10200。

Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Fei Xia、Ed Chi、Quoc V Le、Denny Zhou 等人。2022。Chain-of-thought prompting elicits reasoning in large language models。Advances in Neural Information Processing Systems，35：24824–24837。

Sean Welleck、Ximing Lu、Peter West、Faeze Brahman、Tianxiao Shen、Daniel Khashabi、Yejin Choi。2022。Generating sequences by learning to self-correct。arXiv preprint arXiv:2211.00053。

Jules White、Quchen Fu、Sam Hays、Michael Sandborn、Carlos Olea、Henry Gilbert、Ashraf Elnashar、Jesse Spencer-Smith、Douglas C Schmidt。2023。A prompt pattern catalog to enhance prompt engineering with chatgpt。arXiv preprint arXiv:2302.11382。

Mark H.M. Winands、Yngvi Björnsson、Jahn-Takeshi Saito。2008。Monte-carlo tree search solver。收錄於 Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29–October 1, 2008. Proceedings，卷 6，頁 25–36。Springer。

Chengrun Yang、Xuezhi Wang、Yifeng Lu、Hanxiao Liu、Quoc V Le、Denny Zhou 與 Xinyun Chen。2023。Large language models as optimizers。arXiv preprint arXiv:2309.03409。

Michihiro Yasunaga、Xinyun Chen、Yujia Li、Panupong Pasupat、Jure Leskovec、Percy Liang、Ed H Chi 與 Denny Zhou。2023。Large language models as analogical reasoners。arXiv preprint arXiv:2310.01714。

Qinyuan Ye、Maxamed Axmed、Reid Pryzant 與 Fereshte Khani。2023。Prompt engineering a prompt engineer。arXiv preprint arXiv:2311.05661。

JD Zamfirescu-Pereira、Richmond Y Wong、Bjoern Hartmann 與 Qian Yang。2023。Why johnny can’t prompt: how non-ai experts try (and fail) to design 1 lm prompts。收錄於 Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems，頁 1-21。

Shun Zhang、Zhenfang Chen、Yikang Shen、Mingyu Ding、Joshua B Tenenbaum 與 Chuang Gan。2023 年。使用大型語言模型進行程式碼生成之規劃。arXiv preprint arXiv:2303.05510。

Tianjun Zhang、Xuezhi Wang、Denny Zhou、Dale Schuurmans 與 Joseph E. Gonzalez。2022 年。Tempera：透過強化學習進行測試時提示（test-time prompting）。

Yongchao Zhou、Andrei Ioan Muresanu、Ziwen Han、Keiran Paster、Silviu Pitis、Harris Chan 與 Jimmy Ba。2022 年。大型語言模型是具人類水準的提示工程師。arXiv preprint arXiv:2211.01910。

附錄 A

A. 1 資料切分

在我們的實驗設置中，任務分為兩大類別：General NLU（自然語言理解）與 Domain Knowledge（領域知識）。

在 General NLU 類別中，我們有三個任務：SST-5、TREC 與 RACE。每個任務分配到 100 筆訓練樣本與 50 筆驗證樣本。於測試集方面，SST-5 包含 450 筆樣本，而 TREC 與 RACE 各有 400 筆樣本。

在 Domain Knowledge 類別中，有兩個任務：MEDQA 與 MEDMCQA。與 General NLU 任務相同，這些任務各被分配 100 筆訓練樣本。然而，表格中指出這些任務有 50 筆驗證樣本及 400 筆測試樣本。值得注意的是，MEDQA 與 MEDMCQA 在 Eval 與 Test 欄位的格式存在一些不一致，需要釐清以增進可讀性。詳細資訊請參見表 3。

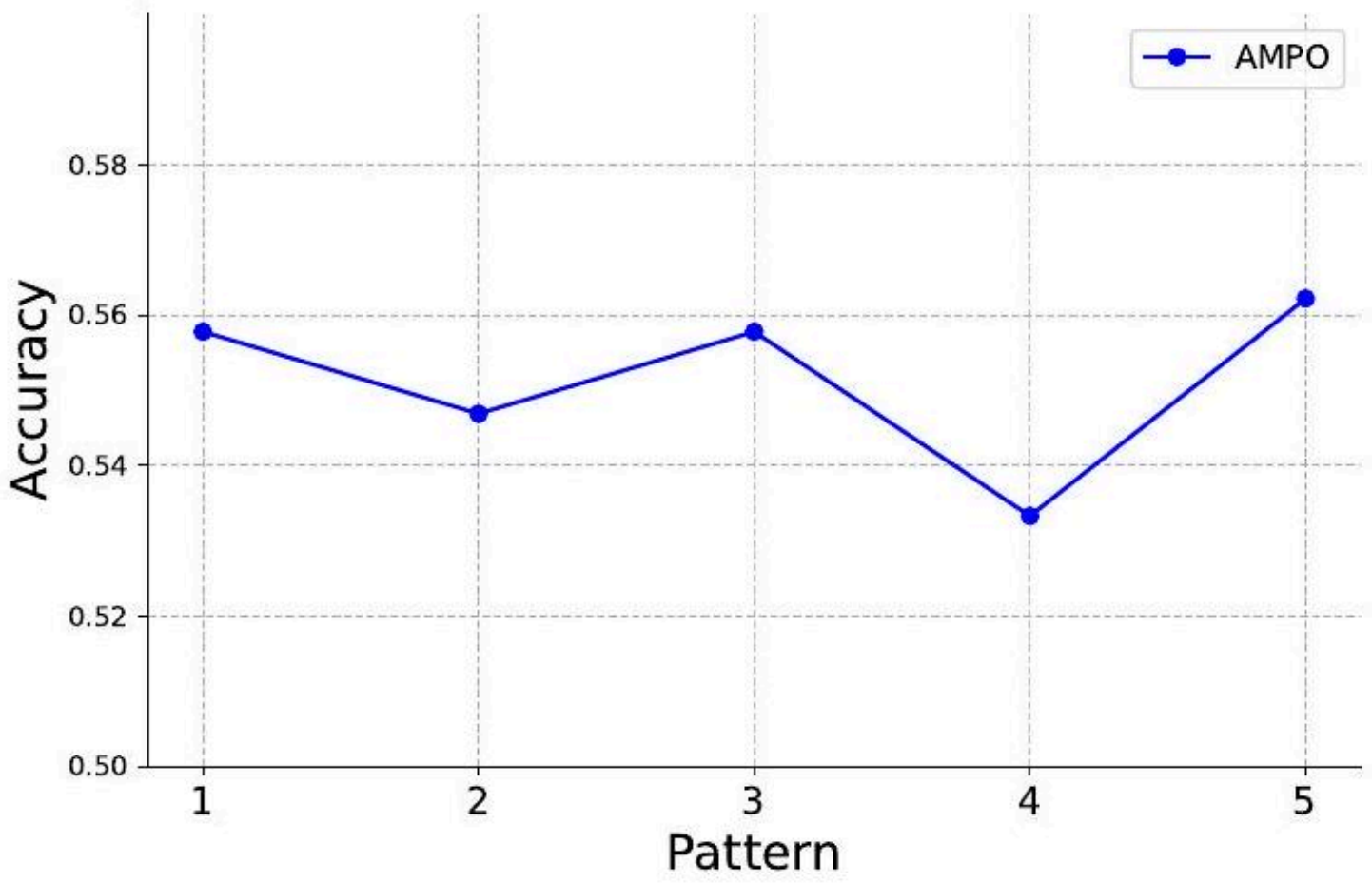
A. 2 種不同的模式結果

我們探討了選擇前 N 個模式對結果的影響。正如表中所示，

類型	任務	訓練	評估	測試
一般自然語言理解 (NLU)	SST-5	100	50	450
	TREC	100	50	400
	RACE	100	50	400
領域知識	MEDQA	100	50	400
	MEDMCQA	100	50	400

\captionsetup{labelformat=empty}

表 3：表 3：實驗數據分佈



\captionsetup{labelformat=empty}
圖 6：圖 7：在 SST-5 任務上選擇不同數量的最佳模式的表現。我們使用 GPT-3.5turbo 作為目標模型。

當 $N = 5$ 時準確率達到峰值，而在 $N = 4$ 時最低，整體呈現在合理範圍內的震盪趨勢。因此，為了提升效率，我們選擇了 $N = 1$ 。

A.3 LLM-Analyzer Meta-Prompt

A.4 LLM-Summarizer Meta-Prompt

A. 5 LLM-Revisor 元提示

—問題開始—
我有一些針對特定問題的指示：
—指示開始—
{{initial_prompt}}
—InstructionsEnd—

但它以下列情況判斷錯誤：
—BadCasesStart—
{{bad_examples}}
—BadCasesEnd—

你的任務是作為分析者，針對我的 [# Instructions] 找出根本成因。請依照以下步驟執行：
(1) 請辨識在我的問題中應考量的觀點有哪些。請盡可能全面地思考，涵蓋所有面向。
(2) 根據你識別出的這些潛在觀點，分析失敗案例的模式。
(3) 仔細檢視我 [# Instructions] 的每個步驟，找出哪些步驟忽略了該模式中的關鍵資訊，導致失敗。
(4) 寫出你的理由，並用 <START> 與 <END> 包住每個理由。

表 4：LLM-Analyzer

—ProblemStart—
我有一些針對特定問題的指示：
—InstructionsStart—
{{initial_prompt}}
—InstructionsEnd—

以下是我目前指示無法解決某些問題的原因：
—Reasons—
{{Reasons}}
—原因—

您的任務是將上面提供的眾多原因總結為幾個主要類別，並為每個類別指定一個重要性分數。請小心刪除重複與相似的原因。每個總結後的模式應以 <START> 和 <END> 包裹。

表 5：LLM-Summarizer
—問題開始—
您有一些關於特定任務的指示：
—InstructionsStart—
{{initial_prompt}}
—InstructionsEnd—

然而，由於真實世界情況的複雜性，單一流程的指示（即順序指示）無法適用於所有情況。因此，您應將指示轉換為條件式方法，也就是針對不同的模式採用不同的指示。

值得注意的是，這個過程的關鍵在於建立一個自適應的提示結構，以容納難度各異的任務。為達成此目標，您應在針對新模式新增分支與根據任務難度與已識別模式分佈為現有分支提供更多細節之間，找到一個黃金中間點。

一位專家指出了一些您之前未考慮納入指示的模式：
—ExpertAnalysisStart—
{{patterns}}
—ExpertAnalysisEnd—

請根據專家分析逐步優化您的[# Instructions]：
(1) 逐步仔細檢查您指示中的每個步驟。
(2) 辨識因缺乏專家分析中提到的關鍵資訊而出錯的步驟。
(3) 對於每個次優步驟，您有以下選項：

3.1 考慮改進該步驟以包含關鍵資訊。
3.2 否則，您也可以考慮使用 **if** 或 **if-else** 結構新增 **子步驟** 來處理 **新** 的模式。確保每個子步驟具體且避免含糊不清的指示。
請注意，若某一步需考慮多種情況，請將該步拆分為子步驟以便更易遵循。
(4) 包含提示或注意事項：若僅透過像 if-else 的分支來優化現有步驟仍無法充分涵蓋所有面向，請在目前指示中新增提示或注意事項以處理不同模式。
(5) 保持其他主要步驟與初始提示一致，以免遺失資訊。
(6) 最後，檢視整體步驟並修剪分支，以避免指示過度擬合。

請只輸出經過優化的提示，不要包含其他任何內容。
表 6：LLM-Revisor

---

*同等貢獻。
† 本研究於 Microsoft 實習期間完成。
‡ 通訊作者。
[1] 我們的方法也能優化人類撰寫的提示。在本文中，我們使用基於 LLM 的提示初始化來減輕人類的工作負擔。