1. Prompt 的設計與使用的大型語言模型

在本次作業中，我們利用 prompt 設計的方式，讓大型語言模型（LLM, Large Language Model）如 OpenAI GPT-4、ChatGPT 等進行程式碼品質提升與改寫。Prompt 的設計重點在於清楚說明現有的程式碼功能、待改善的面向（如可讀性、結構優化、去耦合、可維護性、錯誤處理），並要求生成結構更優雅的程式碼。同時，透過 prompt 明確提出後續報告的撰寫方向與內容，確保 LLM 生成的文字回答能同時兼顧技術與敘事的需求。

LLM 在此的角色不僅是程式碼重寫工具，也是協助使用者快速蒐集並整合優化建議的智能助理。由於 LLM 有強大的自然語言理解與生成能力，使用者能透過 prompt 明確引導模型產出更結構化與條理化的內容。

2. 優化後的程式碼與原本程式碼的比較

原始程式碼中，client 與 server 均將加密/解密邏輯直接嵌入主要的執行流程，導致程式碼可讀性較差，結構耦合度高，不利於維護。此外，錯誤處理與檢查相對簡化，若在網路、加密動作上出現異常狀況，debug 時較困難。

優化後的版本則將加密/解密功能抽出至獨立檔案 (crypto_utils.c/h)，讓主程式碼可專注於網路邏輯與客戶端管理。同時，增加錯誤處理步驟（如檢查 bind、listen、accept、read、send 的回傳值），並對讀寫動作加入更明確的註解。程式碼經過區塊劃分與命名規範後，更易於後續擴充與問題排查。這樣的重構讓程式碼具備可重複使用的函式，提高維護性與擴充性。

3. 評估 LLM 的有效性與局限性

使用 LLM 進行程式碼品質提升的過程中，我們觀察到其有效性在於：

- **快速提供建議**：LLM 可以即時給出多種重構建議，包括函式拆解、錯誤處理增強、變數命名改善及註解添加。
- **提升可讀性與結構**：透過自然語言描述，LLM 能輔助使用者將繁雜的程式碼邏輯改寫得更清晰。

然而，LLM 也有局限性：

- **專業領域知識限制**：若程式牽涉較特殊的網路協定或加密技術細節，LLM 給出的建議可能不夠深入，仍需要專業工程師審視與調整。
- **錯誤可能性**：LLM 可能生成不正確或不安全的程式碼建議，因此需由人類驗證其正確性。

- **缺乏實際執行測試**：LLM 不會主動測試程式碼，使用者仍需自行編譯、執行並確認行為正確。

4. 除了提升程式碼品質外，LLM 還可以如何應用

在本作業中，我們將 LLM 用於程式碼優化，但其應用範圍不僅止於此：

- **需求討論與系統設計**：在系統開發前期，LLM 能協助討論需求、評估系統架構、建議技術棧與設計模式。
- **程式碼審閱 (Code Review)**：LLM 可做為開發者的初步審閱工具，給出程式碼可改善的區域，協助降低人力審閱負擔。
- **教學與學習支援**：初學者可透過 LLM 來解釋程式碼片段、教導語言特性或演算法運作方式，加速學習曲線。
- **自動生成測試案例**：LLM 或可根據功能描述自動產生基礎測試案例，增加程式碼覆蓋率。

透過這些應用，我們能更全面地善用 LLM 的自然語言處理與生成能力，使其成為開發工作流程中多階段的輔助工具，而非僅侷限於程式碼層面的優化。

5. 心得

總結來說，本次透過 prompt 指引 LLM 不僅重構並優化了程式碼品質，同時也探索了 LLM 在程式開發過程中更廣泛且有潛力的應用方式。