116 級畢業專題成果發表

利用大型語言模型引導之演化式搜尋進行自動化程式碼執行效能優化

LLM-Guided Evolutionary Search for Automated Code Runtime Performance Optimization

報告人

葉沁蓉

指導教授

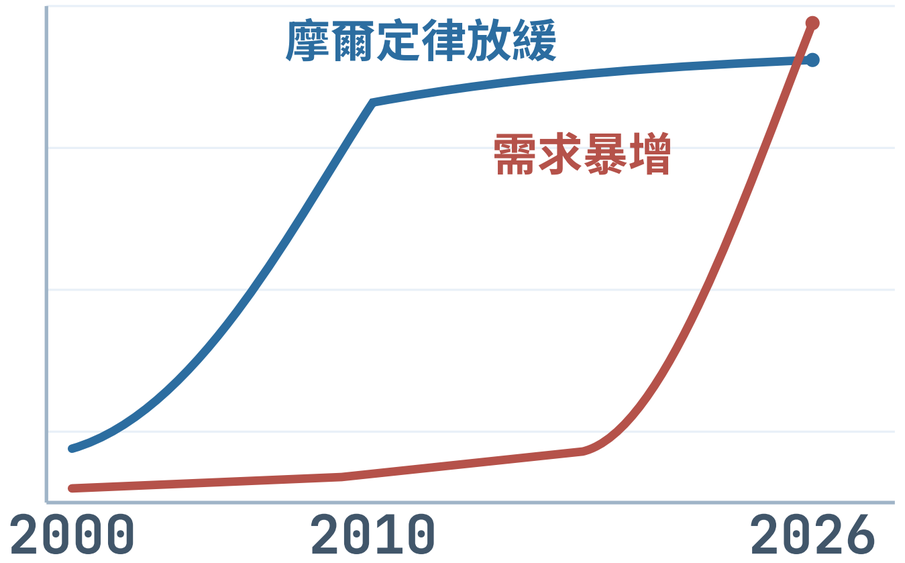
李信杰 教授

國立成功大學 資訊工程學系

01 研究動機：為什麼需要自動化優化

- **摩爾定律**放緩，硬體不再無限變快
- **軟體資料處理與運算**需求卻持續暴增
- 人工 profile 幾萬行程式碼 **無法規模化**
- 驅動我們走向 **自動化優化**

硬體效能 vs. 軟體系統運算需求



Data source: IEEE Spectrum / OpenAI Compute Trends

02 研究背景與挑戰：遺傳改進 (GI) 的演進與缺口

GI 演化流程

歷史演進與研究缺口

傳統隨機變異

依賴盲目的隨機刪除或替換，編譯通過率極低，搜尋效率如大海撈針。

引入大型語言模型

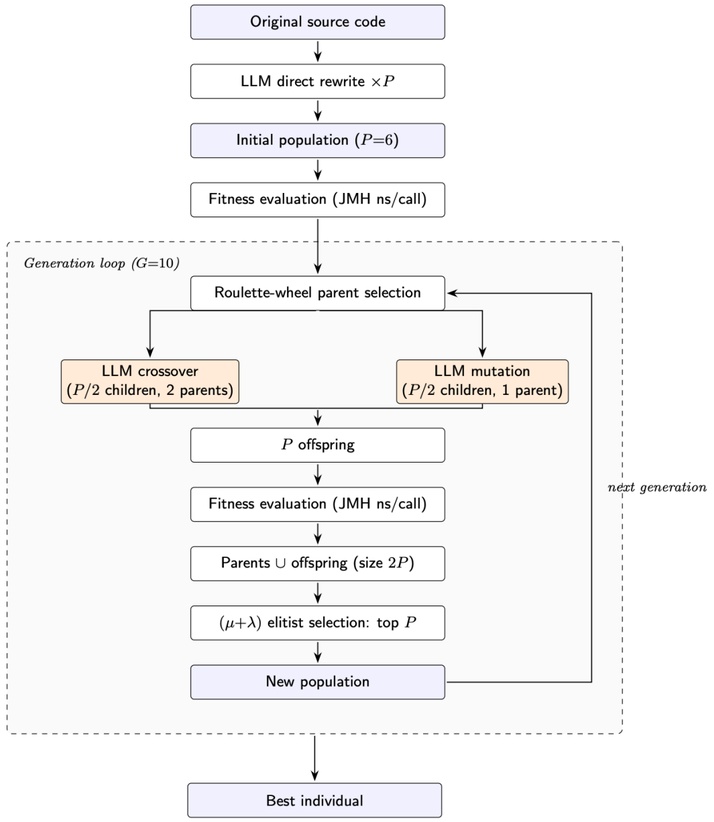
以 LLM 語意引導取代隨機盲猜；但多數研究僅停留在「可行性驗證」，缺乏量化的效能數據。

本研究的核心提問

- 導入 LLM 反覆迭代，能帶來多少**實質效能效益**？
- 與「無迭代、等 Token 預算」的**直接重寫**相比，有無顯著差異？

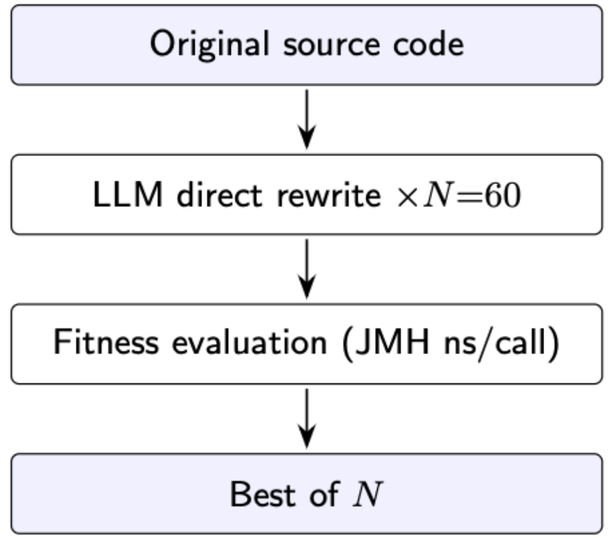
03 系統架構與對照組

實驗組 LLM 引導之演化搜尋 (GA)



LLM 擔任 **Crossover / Mutation** 算子；JMH 量測 **ns/call** 作為 Fitness 回饋，引導 (μ+λ) 菁英演化。

對照組 等預算直接重寫 (MOS)



相同 LLM 呼叫預算下，一次重寫 N 份，直接取最佳。

FAIR TEST = 60 LLM calls

在**相同預算**下對決 —— 驗證「演化搜尋機制」的真正價值

03b 演化算子解密

突變提示詞 (Mutation)

Rewrite the following Java code to create a functionally equivalent variant with a different implementation style or logic structure for performance comparison.

{code}

交配提示詞 (Crossover)

Combine the best performance optimizations and logic structures from the following two Java implementations into a single, improved Java method.

{parent_1} {parent_2}

交配行為解析：結構性重組

Parent 2 — Init & variable setup

```
private int[] estimateFullPix(Picture ref, byte[] patch, int mbX, int mbY, int mvpx, int mvpy) {
    exactCallCount_estimateFullPix++;

    final int mbX16 = mbX << 4;
    final int mbY16 = mbY << 4;
    final int planeW = ref.getPlaneWidth(0);
    final int planeH = ref.getPlaneHeight(0);
    final byte[] planeData = ref.getPlaneData(0);

    final int side = (maxSearchRange << 1) + 16;
    final byte[] searchPatch = new byte[side * side];

    int resMvX = 0, resMvY = 0, resScore = Integer.MAX_VALUE;
```

Parent 2 — Area loop & coordinate setup

```
    for (int area = 0; area < 2; area++) {
        final int startX = (area == 0) ? mbX16 : mbX16 + (mvpx >> 2);
        final int startY = (area == 0) ? mbY16 : mbY16 + (mvpy >> 2);
```

Parent 1 — Search patch & bounds calculation

```
        int patchTlX = Math.max(0, startX - maxSearchRange);
        int patchTlY = Math.max(0, startY - maxSearchRange);
        int patchBrX = Math.min(planeW, startX + maxSearchRange + 16);
        int patchBrY = Math.min(planeH, startY + maxSearchRange + 16);

        final int inPatchX = startX - patchTlX;
        final int inPatchY = startY - patchTlY;

        if (inPatchX < 0 || inPatchY < 0) continue;

        final int patchW = patchBrX - patchTlX;
        final int patchH = patchBrY - patchTlY;

        MBEncoderHelper.takeSafe(planeData, planeW, planeH, patchTlX, patchTlY, searchPatch, patchW, patchH);
```

Parent 1 — Core hill-climbing loop

```
        int bestMvX = inPatchX;
        int bestMvY = inPatchY;
        int bestScore = sad(searchPatch, patchW, patch, bestMvX, bestMvY);

        for (int i = 0; i < maxSearchRange; i++) {
            if (bestScore == 0) break;

            // s1..s4 = SAD at 4-neighbor of (bestMvX, bestMvY) or INT_MAX if out-of-bounds

            // minVal = min(s1, s2, s3, s4)

            if (minVal > bestScore) break;

            bestScore = minVal;
            if (s1 == minVal) --bestMvX;
            else if (s2 == minVal) ++bestMvX;
            else if (s3 == minVal) --bestMvY;
            else ++bestMvY;
        }
```

Parent 2 — Result aggregation & early exit

```
        if (bestScore < resScore) {
            resScore = bestScore;
            resMvX = (bestMvX + patchTlX - mbX16) << 2;
            resMvY = (bestMvY + patchTlY - mbY16) << 2;
        }

        if (resScore == 0) break;
    }
```

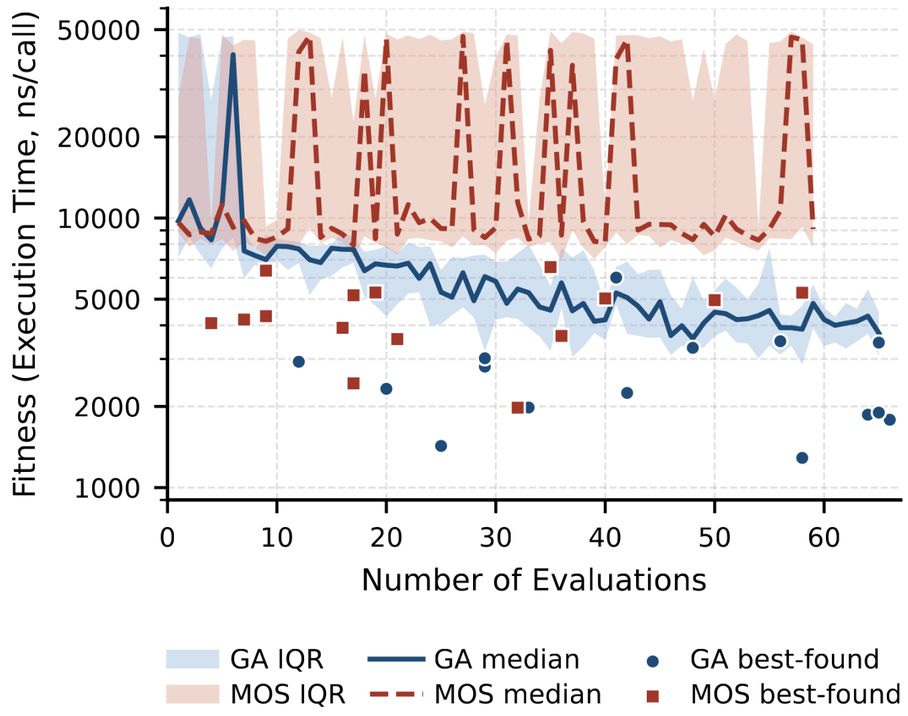
Parent 2 — Return statement

```
    return new int[] { resMvX, resMvY };
}
```

紅色區塊源自 **Parent 2**、藍色區塊源自 **Parent 1**。LLM 並非盲目拼接，而是**精準抽取雙親優勢邏輯**進行結構性重組（run 3 / gen 6 實例）。

04 實驗結果

收斂行為：GA 中位數穩定下行，與 MOS 完全分離



執行時間（**ns/call**，log scale，愈低愈快）對評估次數。約第 10 次後兩條 IQR 帶完全分離。

46.2%
中位數加速幅度
vs. 直接重寫對照組 (MOS)

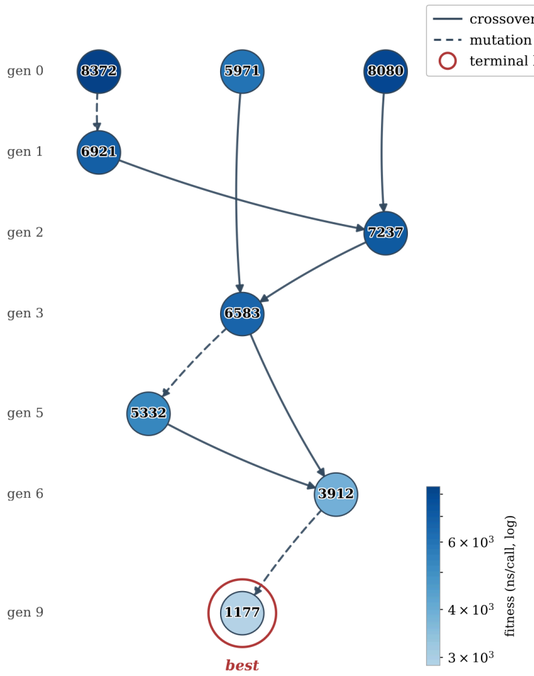
p = 0.0005
Mann–Whitney U檢定 具高度統計顯著性

Â₁₂ = 0.876
達大效應量等級
Large effect

2,326 ns vs. MOS 4,325 ns
GA 中位執行時間 —— 顯著超越對照組

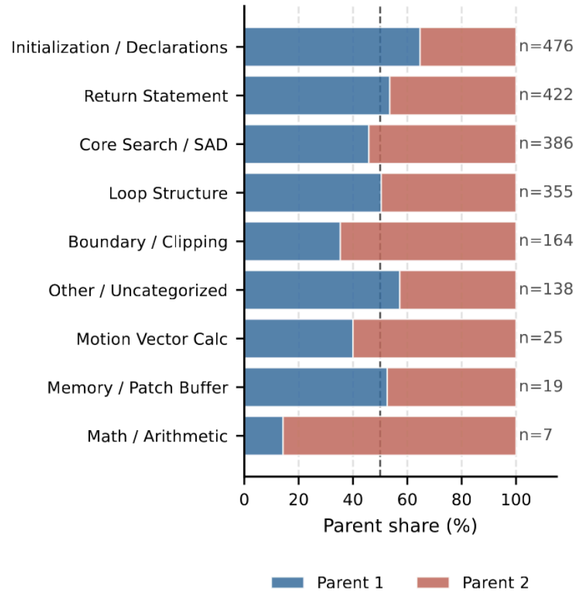
05 演化行為分析：解構 LLM 優化邏輯

算子交替優化



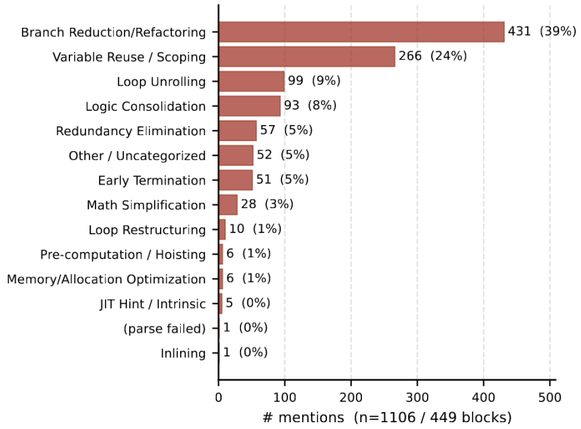
效能突破仰賴**「交配」與「突變」**算子頻繁交織接力。證實兩者在語意引導下具備**高度協同性**，缺一不可。

無位置偏誤



雙親代碼對子代貢獻比例趨近 **50：50**。成功證明 LLM 克服了常見的**「提示詞順序偏誤」**（Prompt Bias）。

突變類型統計



LLM 傾向執行**「控制流簡化」**。但受限於自迴歸特性，**較難執行**逆向的大規模全域重構。

06 結論與未來展望

結論

成功結合 LLM 與演化搜尋

提出以 LLM 為算子的 **GA 框架**，證實**「交配」與「突變」**機制具備**高度協同效應**，兩者缺一不可。

突破性效能提升

在嚴格隔離 **Token 預算**的前提下，GA 較純突變對照組 (MOS) 達成中位數 **46.2% 顯著加速**。

未來展望

突破逆向分析瓶頸

針對 LLM 難以執行**逆向分析**的結構性弱點，未來將設計**專屬 Prompt 或生成機制**予以補強。

外部效度擴展

將本框架推廣至**更多專案**、**其他程式語言**，以及**更新型的 LLM 模型**。

Target Projects:

Jsoup · PMD · JavaParser · Joda-Time · Gson · Jackson · Jcodec · Apache Lucene · LMAX Disruptor · Karate