# Reverse C++ Like a Boss, Make LLM Reverse for You

CyCraft C++ 讀書會

2025/03/22

# 黃俊嘉 Ice1187

- 奧義智慧科技 資安研究員
- 臺灣大學資訊工程研究所 碩士
- 研究領域
  - 惡意程式自動化分析、模糊測試
- 研究發表
  - CyberSec、SECCON
  - ACM CCS (poster)

# 大綱

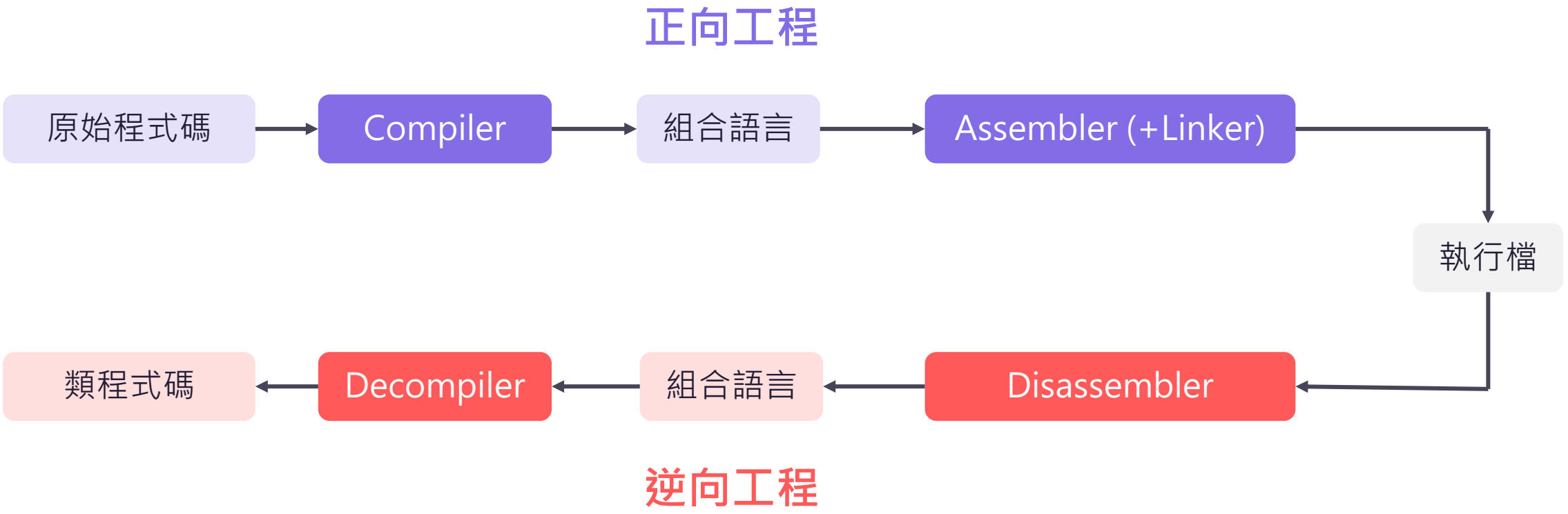
- C++ 逆向工程簡介
- LLM 應用於逆向工程的最新進展
- 使用 LLM 協助逆向

# C++ 逆向工程簡介

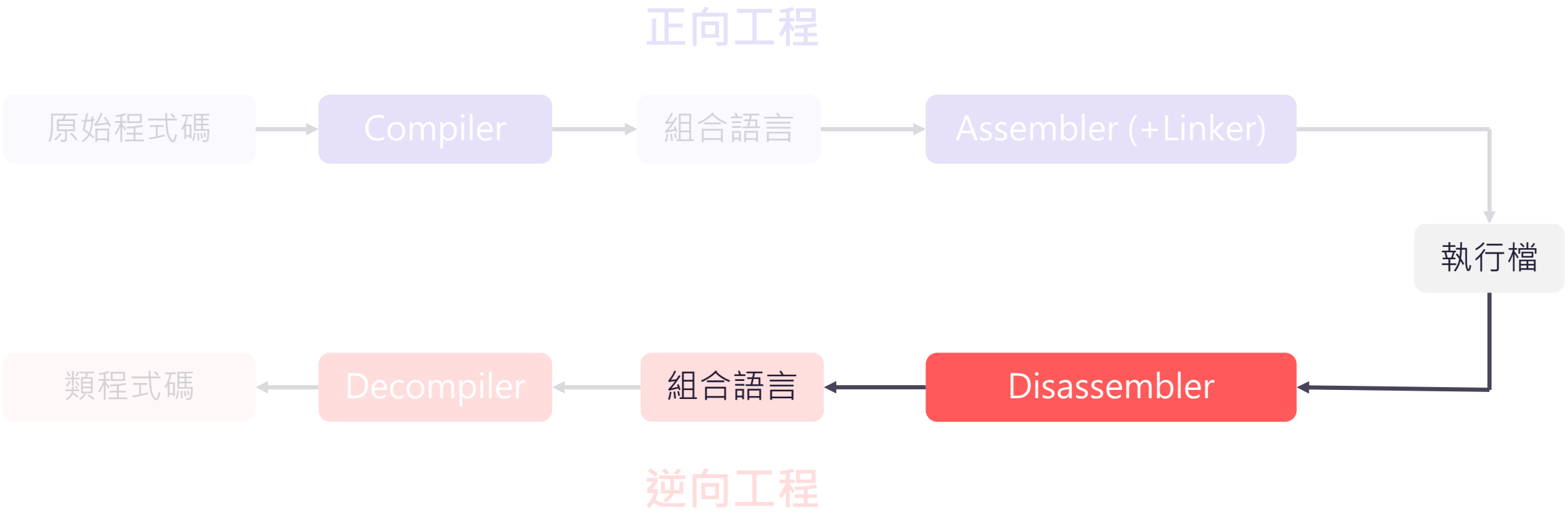
# 什麼是逆向工程？

- 在**缺少原始程式碼**的情況下，直接分析經過編譯的程式，以了解程式內部的設計與運作方式。
- 應用情境
  - 惡意程式分析
  - 漏洞挖掘
  - 產品分析
  - 周邊軟體開發
  - 軟體複製

# 逆向工程的流程



# 逆向工程的流程 — Disassembler



# Disassembler 的功能

- Disassembler 將 CPU 指令轉換成等價的組合語言

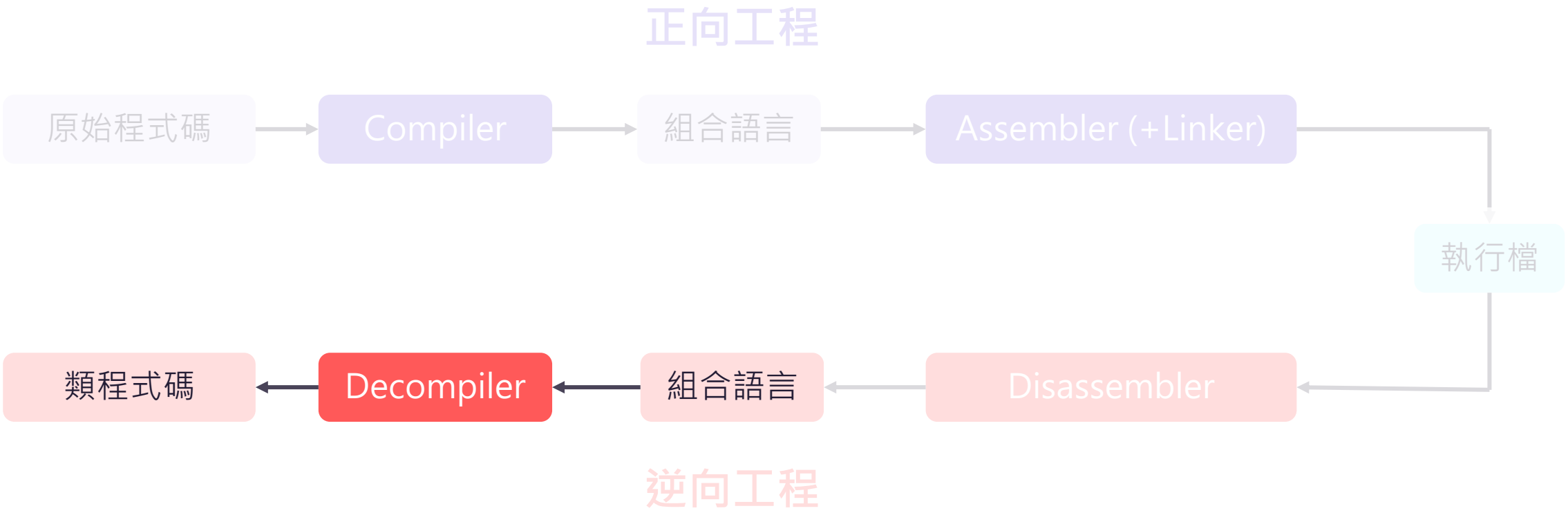
```
001440: ffff ffe9 a700 0000 8b45 d88d 1400 488b
001450: 45f8 8b40 188b 4ddc 89ce 89c7 e8e4 feff
001460: ff89 c248 8b45 f889 d648 89c7 e80b ffff
001470: ffeb 7c48 8b45 f88b 4018 3945 f47e 3766
001480: 0fef c0f3 0f2a 45d8 f30f 5945 d4f3 0f2c
001490: d048 8b45 f88b 4018 8b4d dc89 ce89 c7e8
0014a0: a1fe ffff 89c2 488b 45f8 89d6 4889 c7e8
0014b0: c8fe ffff eb39 660f efc0 f30f 2a45 d8f3
0014c0: 0f59 45d4 f30f 58c0 f30f 2cd0 488b 45f8
0014d0: 8b40 188b 4ddc 89ce 89c7 e866 feff ff89
0014e0: c248 8b45 f889 d648 89c7 e88d feff ff83
0014f0: 45f0 018b 45f0 4898 4839 45e0 0f87 e9fe
001500: ffff 9090 c9c3 f30f 1efa 5548 89e5 4154
001510: 5348 83ec 10bf a000 0000 e8a1 fbff ff49
001520: 89c4 4c89 e341 b800 0000 00b9 d007 0000
```

**Binary**

```
eb 7c                   jmp    14ef <_ZNSolsEi@plt+0x3bf>
48 8b 45 f8             mov    rax,QWORD PTR [rbp-0x8]
8b 40 18                mov    eax,DWORD PTR [rax+0x18]
39 45 f4                cmp    DWORD PTR [rbp-0xc],eax
7e 37                   jle    14b6 <_ZNSolsEi@plt+0x386>
66 0f ef c0             pxor   xmm0,xmm0
f3 0f 2a 45 d8          cvtsi2ss xmm0,DWORD PTR [rbp-0x28]
f3 0f 59 45 d4          mulss  xmm0,DWORD PTR [rbp-0x2c]
f3 0f 2c d0             cvttss2si edx,xmm0
48 8b 45 f8             mov    rax,QWORD PTR [rbp-0x8]
8b 40 18                mov    eax,DWORD PTR [rax+0x18]
8b 4d dc                mov    ecx,DWORD PTR [rbp-0x24]
89 ce                   mov    esi,ecx
89 c7                   mov    edi,eax
e8 a1 fe ff ff          call   1345 <_ZNSolsEi@plt+0x215>
89 c2                   mov    edx,eax
48 8b 45 f8             mov    rax,QWORD PTR [rbp-0x8]
89 d6                   mov    esi,edx
```

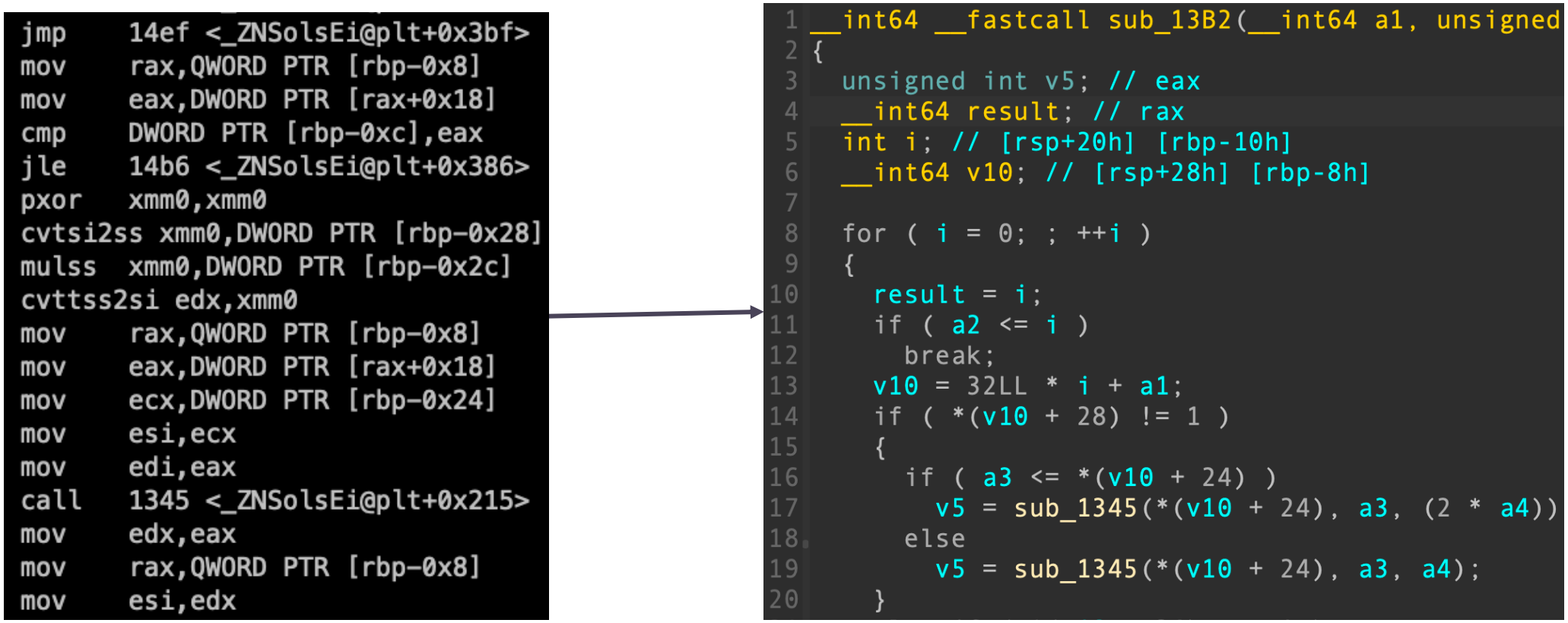
**組合語言**

# 逆向工程的流程 — Decompiler



# Decompiler 的功能

- Decompiler 將組合語言轉換成**語意近似的**類程式碼




組合語言　　類程式碼



CYCRAFT

# Decompile / 逆向工程時的難題

- 編譯過程中，區域變數名稱、函式名稱和型別等資訊被移除
- 編譯最佳化會模糊程式碼原有的邏輯與架構
  - `if (a>5) then (a*b) else (a+b)`
    - `((-((a > 5) & 1)) & a*b) | (~(-((a > 5) & 1)) & a+b)`
  - Inline function
  - Loop unrolling
- 不同編譯器可能使用不同的組合語言去實作同一份程式碼

# 人工重新命名符號與標註型別

```
__int64 __fastcall sub_13B2(__int64 a1, unsigned
{
  unsigned int v5; // eax
  __int64 result; // rax
  int i; // [rsp+20h] [rbp-10h]
  __int64 v10; // [rsp+28h] [rbp-8h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( a2 <= i )
      break;
    v10 = 32LL * i + a1;
    if ( *(v10 + 28) != 1 )
    {
      if ( a3 <= *(v10 + 24) )
        v5 = sub_1345(*(v10 + 24), a3, (2 * a4));
      else
        v5 = sub_1345(*(v10 + 24), a3, a4);
    }
```

類程式碼

```
void __fastcall sub_13B2(
        Person *persons,
        unsigned __int64 num_person,
        unsigned int basis,
        int factor,
        float remote_factor)
{
  unsigned int new_salary; // edx
  int i; // [rsp+20h] [rbp-10h]
  Person *person; // [rsp+28h] [rbp-8h]

  for ( i = 0; num_person > i; ++i )
  {
    person = &persons[i];
    if ( !person->remote )
    {
      if ( basis <= person->salary )
        new_salary = calc_new_salary(person->salary,
      else
        new_salary = calc_new_salary(person->salary,
```

重新命名變數、函式和標註型別

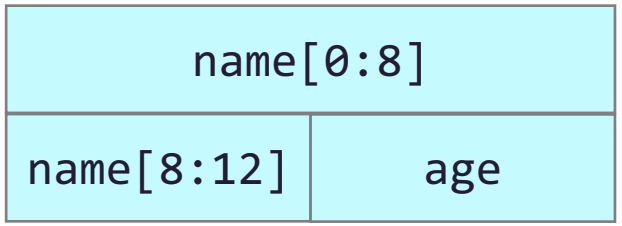
# C++ 為逆向帶來的難題

- 繼承
- 虛擬函式覆寫 (virtual function, override)
- ~~模板~~
- ~~其他語言特性~~

# 物件在記憶體中的儲存結構

```
class Person {
    char name[12];
    int age;


};
```

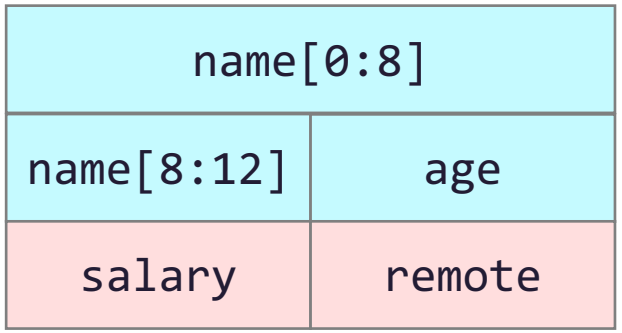
Person object



# 子類別物件的記憶體結構

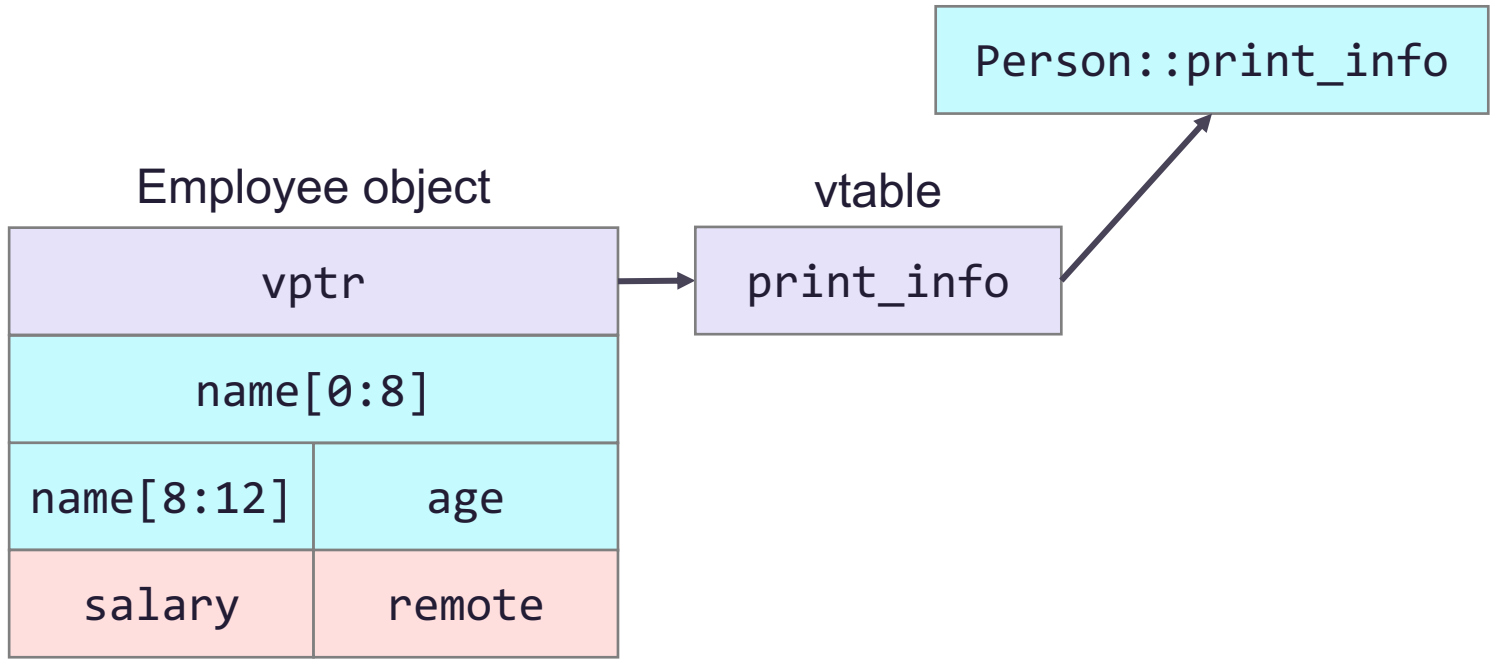
```
class Person {
    char name[12];
    int age;


};

class Employee : Person {
    int salary;
    bool remote;


};
```

Employee object



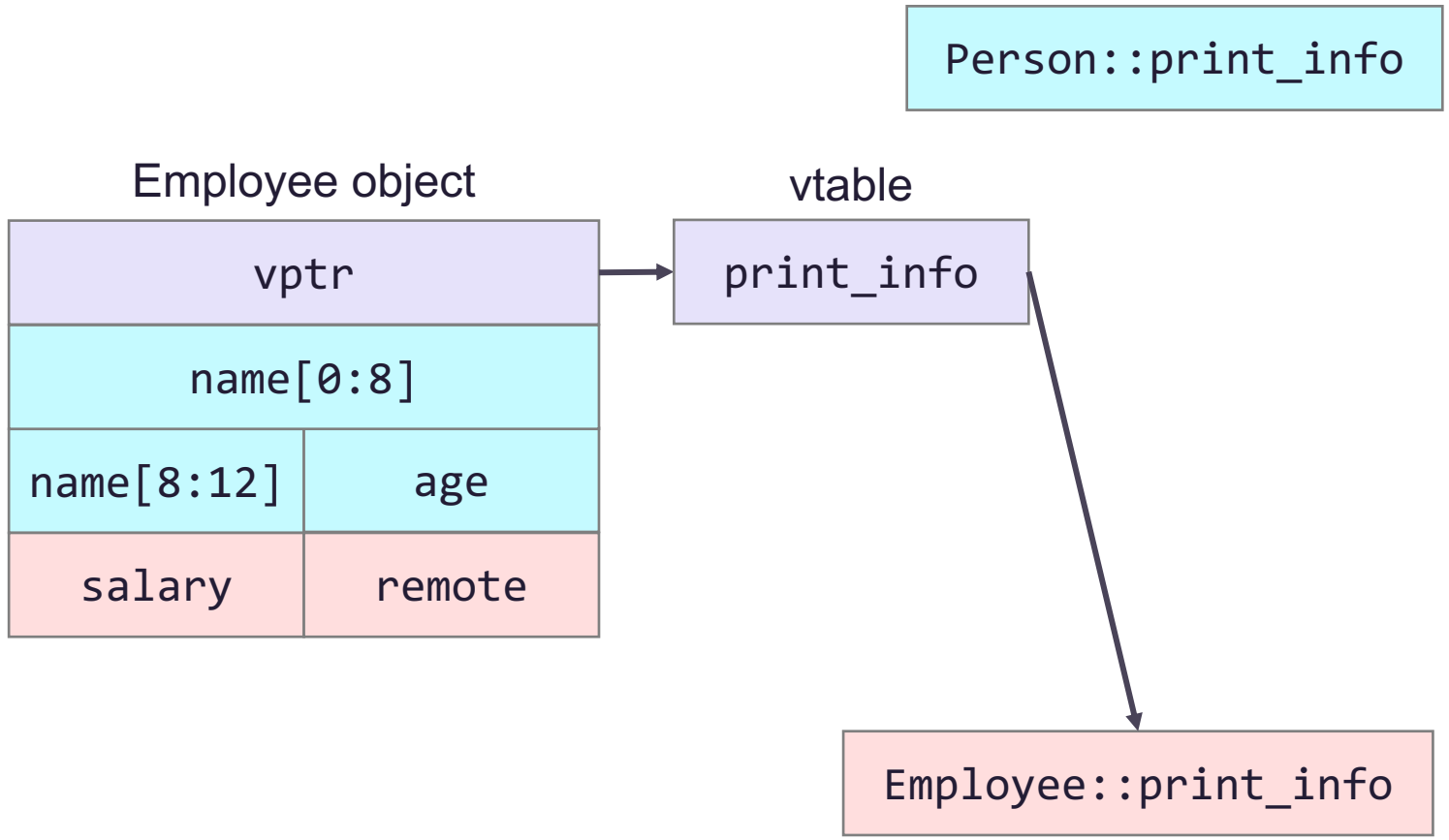
# 帶有虛擬函式的子類別物件的記憶體結構

```
class Person {
    char name[12];
    int age;

    virtual void print_info();
};

class Employee : Person {
    int salary;
    bool remote;


};
```

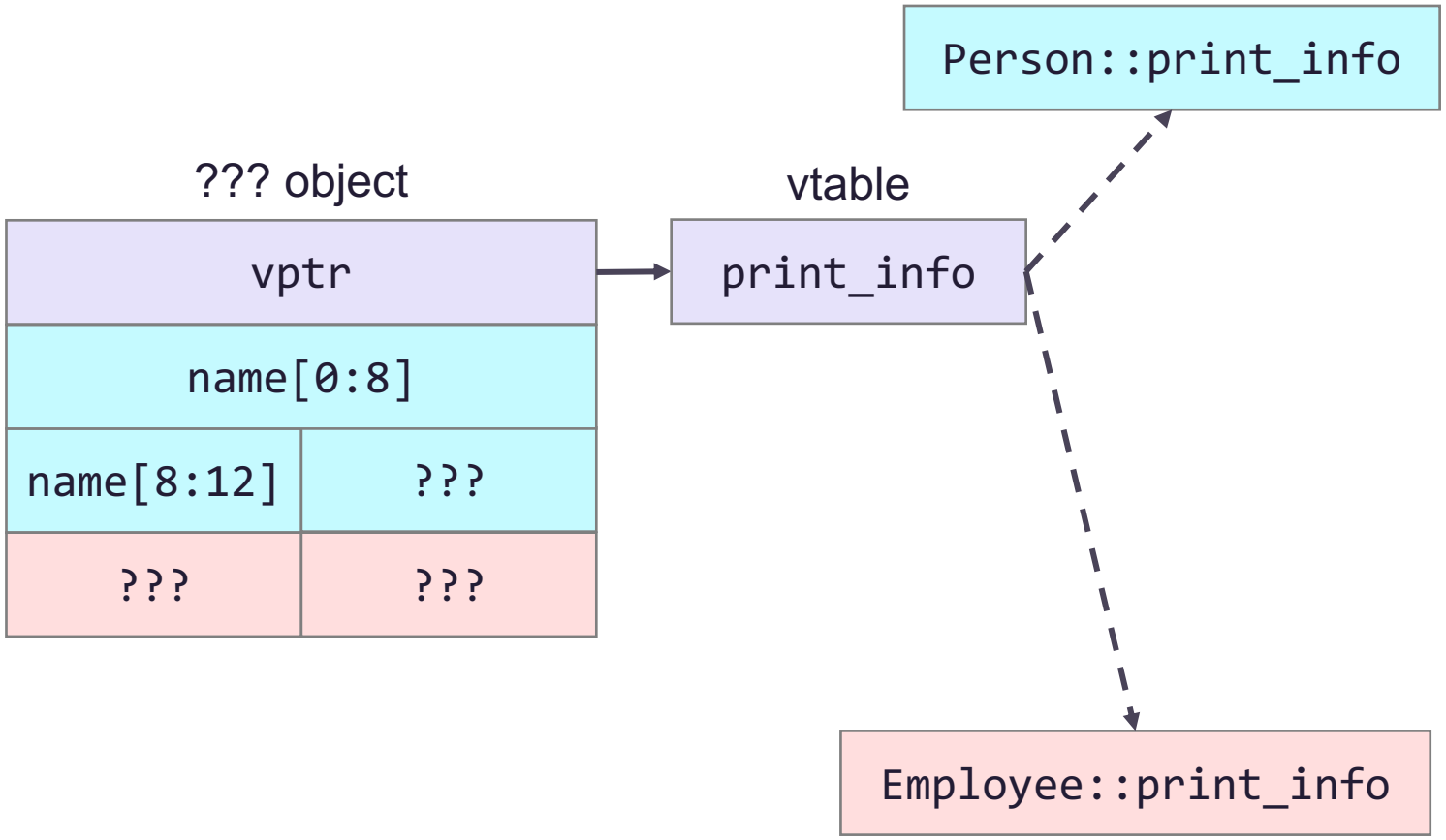


# 子類別物件的虛擬函式覆寫

```
class Person {
    char name[12];
    int age;

    virtual void print_info();
};

class Employee : Person {
    int salary;
    bool remote;

    void print_info() override;
};
```



# 分析具有虛擬函式物件時的問題



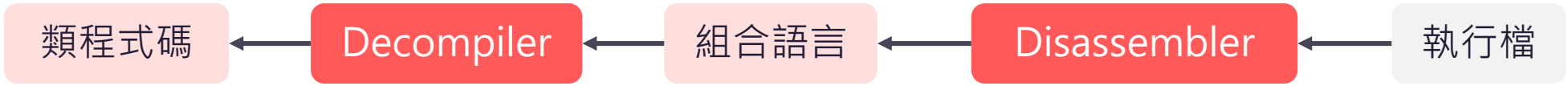
# 小結 Summary

- 什麼是逆向工程
- 逆向工程的挑戰
  - 符號名稱、型別、程式碼架構等資訊被移除
  - [C++] 識別子類別物件虛擬函式的實際實作

# 應用 LLM 於逆向工程的最新進展

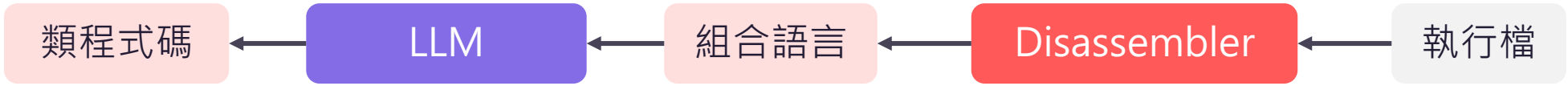
# 應用 LLM 於逆向工程的方法

- 做 decompile
- 改善可讀性
- 解釋程式碼



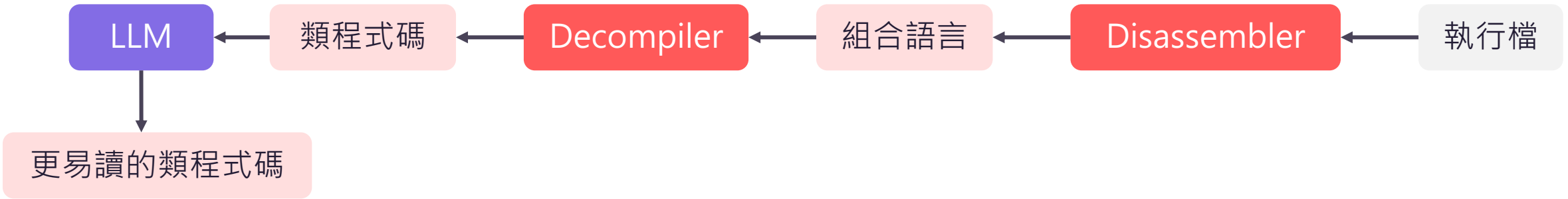
# 應用 LLM 於逆向工程的方法

- 做 decompile
- 改善可讀性
- 解釋程式碼



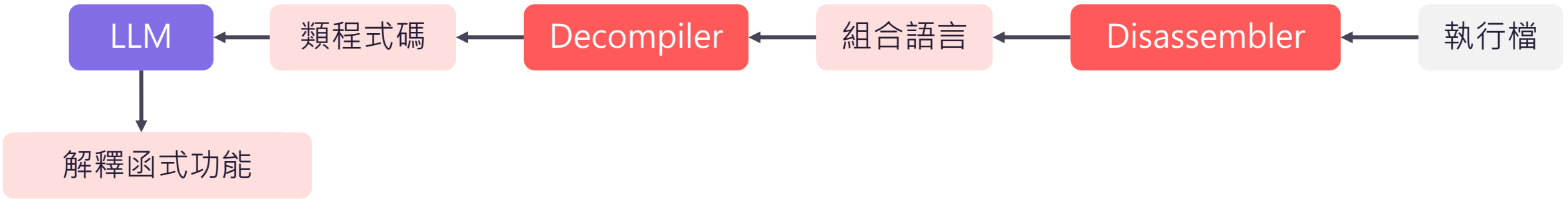
# 應用 LLM 於逆向工程的方法

- 做 decompile
- 改善可讀性
- 解釋程式碼



# 應用 LLM 於逆向工程的方法

- 做 decompile
- 改善可讀性
- 解釋程式碼



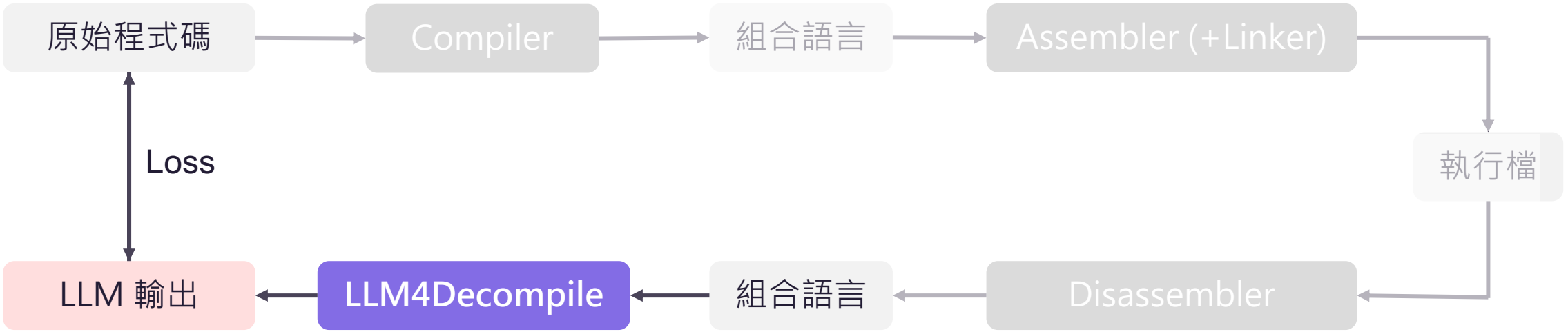
# 應用 LLM 於逆向工程的相關研究

- 做 decompile (end-to-end)
  - LLM4Decompile (EMNLP'24), sc2dec (EMNLP'24), ReF (arXiv'25)
- 改善可讀性 (refine)
  - LLM4Decompile (EMNLP'24), ReSym (CCS'24), DeGPT (NDSS'24)
  - LmPa (arXiv'23), DecGPT (arXiv'23)
- 解釋程式碼
  - IDA Pro, Ghidra 插件

# LLM4Decompile (EMNLP'24)

- 第一個針對 decompile 任務進行微調的開源 LLM
- 參數量：1.3B, 6.7B, 33B

# 訓練架構



# 訓練資料

- 預訓練模型：DeepSeek-Coder
- 微調資料集：1.6M functions in C, 572M tokens from ExeBench
  - compiler: gcc
  - platform: x86 Linux
- Prompt：

> This is the assembly code:
> [ASM code]
>
> What is the source code?

## Source Code

```
int func0(float num[], int size,
  float threshold) {
  int i, j;
  for (i = 0; i < size; i++)
    for (j = i + 1; j < size; j++)
      if (fabs(num[i] - num[j])
          < threshold)
          return 1;
  return 0;}
```

## LLM4Decompile-End-6.7B

```
int func0(float *x, int n,
  float eps){
  int i, j;
  for (i = 0; i < n; i++)
    for (j = i + 1; j < n; j++)
      if (fabsf(x[i] - x[j]) < eps)
        return 1;
  return 0;}
```

## Ghidra

```
undefined4 func0(float param_1,long param_2,int param_3){
  int local_28;
  int local_24;
  local_24 = 0;
  do {
    local_28 = local_24;
    if (param_3 <= local_24) {
      return 0;}
  while (local_28 = local_28 + 1, local_28 < param_3) {
    if ((double)((ulong)(double)(*(float *)(param_2 + (long)local_24 * 4) -
        *(float *)(param_2 + (long)local_28 * 4)) &
        SUB168(_DAT_00402010,0)) < (double)param_1) {
      return 1;}}
    local_24 = local_24 + 1;
  } while( true );}
```

## Ghidra+GPT-4o

```
typedef unsigned int undefined4;
undefined4 func0(float param_1,
  long param_2, int param_3)
  {
  int local_28;
  int local_24;
  local_24 = 0;
  while (1) {
    local_28 = local_24;
    if (param_3 <= local_24) {
      return 0;
    }
    while (local_28 + 1 < param_3) {
      local_28++;
      if ((double)((*(float *)
        (param_2 + (long)local_24 * 4)
        - *(float *)(param_2 +
        (long)local_28 * 4)))
        < (double)param_1) {
          return 1;
      }
    }
    local_24++;}}
```

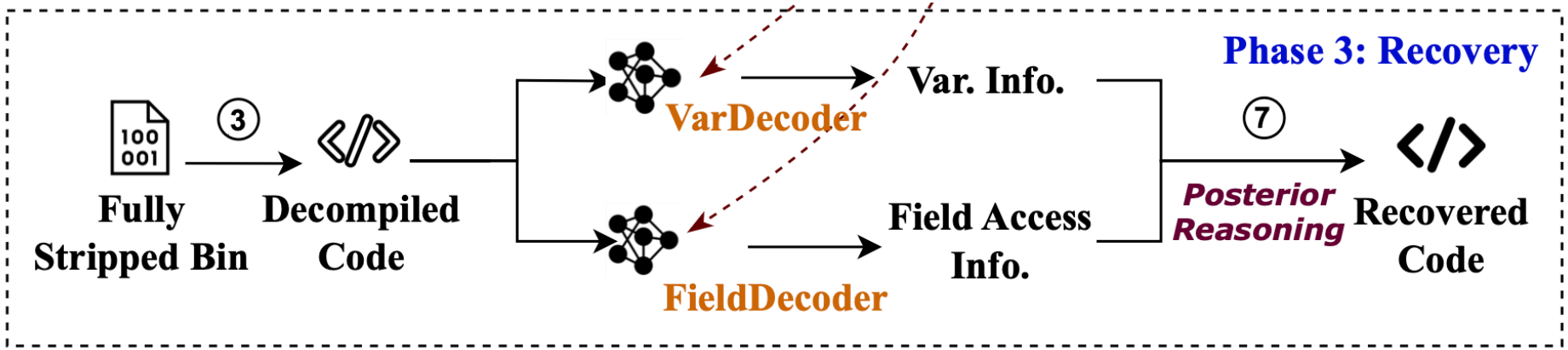
Tan et al. (2024)

# End-to-end 的缺點 / 改善可讀性的動機

- Decompile 出來的類程式碼會有幻覺問題
- 重新命名與標註型別需要全局理解
- 無法還原未出現在訓練資料裡的 user-defined structures

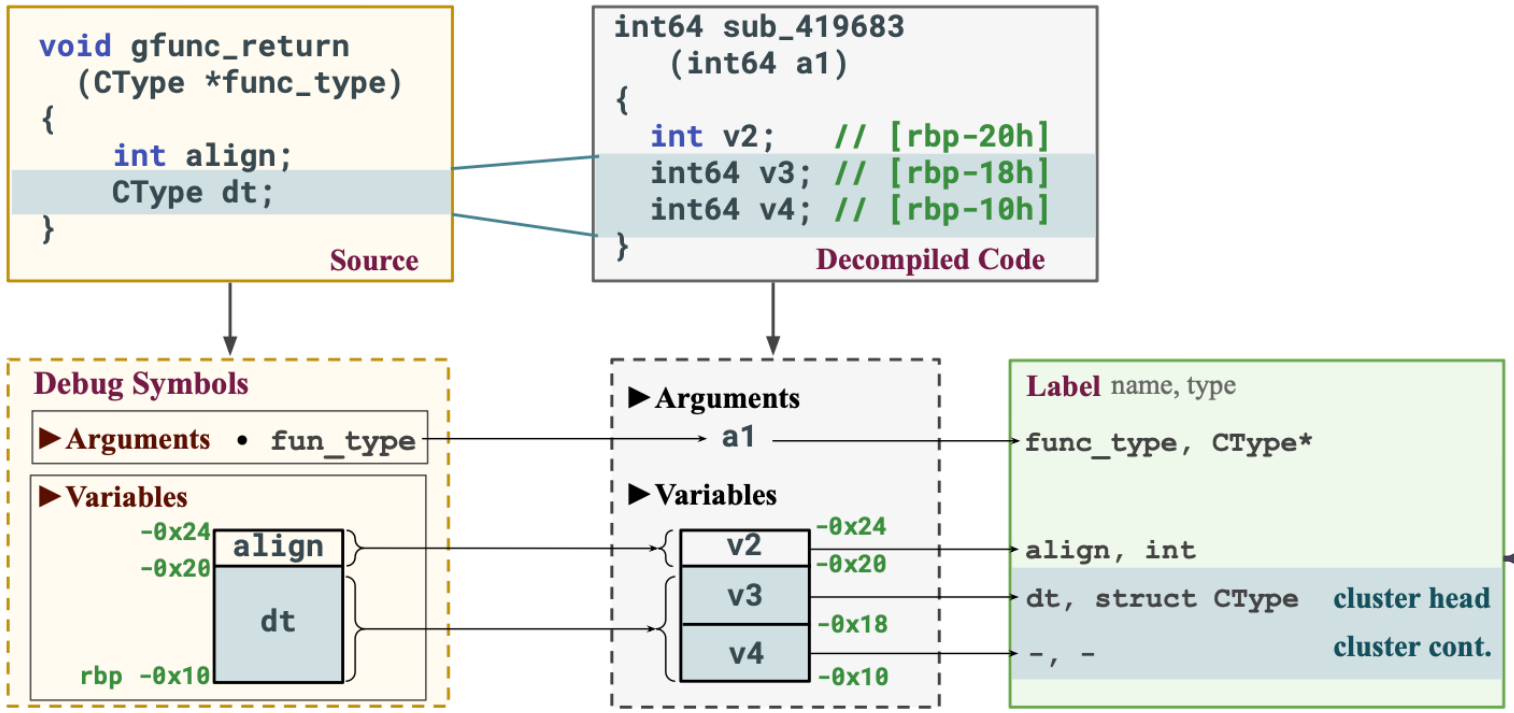
# ReSym (CCS'24)

- VarDecoder：還原區域變數的名稱與型別
- FieldDecoder：還原物件與成員變數的名稱與型別
- 後處理：找出跨函式變數最適合的名稱與型別




Xie et al. (2024)

# VarDecoder 的訓練架構




Xie et al. (2024)

Loss

What are the original name and data type of variables:
`a1, v2, v3, v4?`

[decompile code]

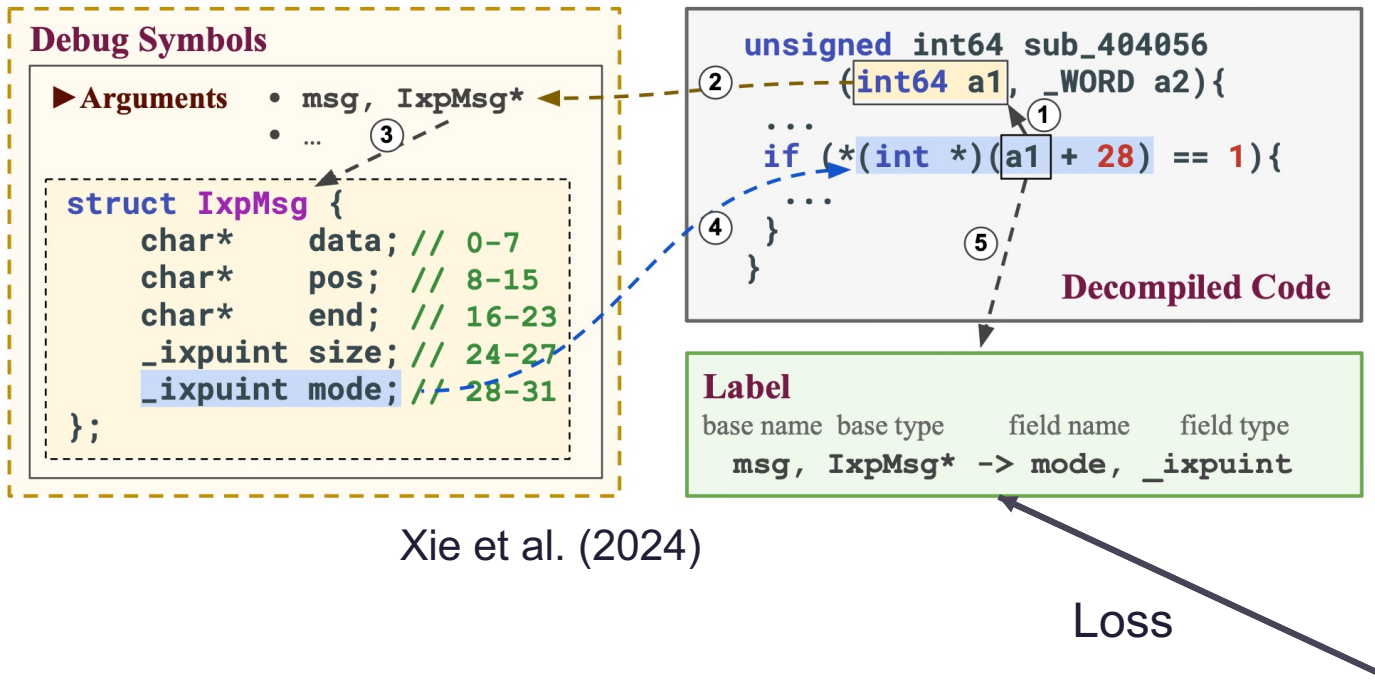
VarDecoder

```
a1: func_type, CType*
v2: align    , uint32
v3: dt       , struct CType
v4: -        , -
```



CYCRAFT

# FieldDecoder 的訓練架構




What are the variable name and type for the following field accesses:
`(int *)(a1 + 28)`?

[decompile code]

FieldDecoder

```
(int *)(a1 + 28): msg , IxpMsg*
                  -> mode, _ixpuint
```

# 推論與投票跨函式變數的預測結果

- 定義規則來找到跨函式、有衝突的變數
  - caller 和 callee 的參數型別應該要相同
  - 同 struct 成員變數的名稱與型別應該要相同
- 從多個選項中，投票決定跨函式變數的名稱 / 型別

# 訓練資料

- 預訓練模型：StarCoder 3B
- 微調資料集：16217 程式
  - 約 6.5% 來自 C++ projects，其餘為 C
  - compiler: GHCC (GitHub Cloner & Compiler)
- Decompiler: IDA Pro

# Evaluation

**Source Code**

```
1  void attack_udp_dns
2      (uint8_t targs_len, ...) {
3   int domain_len;
4   struct iphdr *iph;
5   struct udphdr *udph;
6   ...
7   for (i = 0; i < targs_len; i++){
8    if ( ... )
9     udph->source = rand_next();
10   if ( ... )
11    udph->dest = rand_next();
12    udph->check = checksum_tcpudp(
13      iph, udph,
14      udph->len,
15      data_len + domain_len + 27);
16   ...
17  } ...
18 }
```

**Decompiled Code**

```
1  unsigned int64 sub_40C09E
2    (unsigned int8 a1, ...) {
3   int v56;              // [rbp-88h]
4   unsigned int16 *v67; // [rbp-38h]
5   unsigned int16 *v68; // [rbp-30h]
6   ...
7   for ( i = 0; i<a1; ++i ) {
8    if ( ... )
9     *v68 = sub_410F98();
10   if ( ... )
11    *(_WORD*)(v68+2) = sub_410F98();
12   *(_WORD*)(v68+6) = sub_40D5B2(
13     (__int64)v67, v68,
14     *(_WORD*)(v68+4),
15     (unsigned int)v48 + v56 + 27);
16   ...
17  } ...
18 }
```

**ReSym Recovered Code**

```
1  unsigned int64 sub_40C09E
2     (size_t pkt_size, ...){
3   int domain_len;         // [rbp-88h]
4   struct ip_hdr* iphdr; // [rbp-38h]
5   struct udp_hdr* udp;  // [rbp-30h]
6   ...
7   for ( i = 0; i< pkt_size; ++i ) {
8    if ( ... )
9      udp->src_port = sub_410F98();
10   if ( ... )
11     udp->dst_port = sub_410F98();
12   udp->chksum = sub_40D5B2(
13      ip_hdr, udp,
14      udp->len,
15      proto + domain_len + 27);
16   ...
17  } ...
18 }
```

**Ground Truth Structure**

```
struct udphdr {
 __be16 source;      // 0-1
 __be16 dest;        // 2-3
 __be16 len;         // 4-5
 __sum16 check;      // 6-7
};
```

**ReSym Recovered Structure**

```
struct udp_hdr {
 uint16_t src_port;// 0-1
 uint16_t dst_port;// 2-3
 uint16_t len;     // 4-5
 uint16_t chksum;  // 6-7
};
```

**Figure 9: Case study: a recovered function and data structure in a real-world malware Mirai.**

Xie et al. (2024)

# 使用 LLM 協助逆向

# LLM4Decompile

- Case Study: 帶有符號資訊的 aria2c
  - 符號資訊：函式名稱、全域變數名稱...

Source

```
int main(int argc, char** argv) {
  aria2::error_code::Value r;

  aria2::global::initConsole(false);
  aria2::Platform platform;
  r = aria2::main(argc, argv);

  return r;
}
```

IDA Pro

```
int main(int argc, const char **argv, const char **envp) {
  char **v3;
  int v4;
  aria2::Platform v6;

  aria2::global::initConsole(0LL, argv);
  aria2::Platform::Platform(&v6);
  v4 = aria2::main(argc, argv, v3);

  aria2::Platform::~Platform(&v6);
  return v4;
}
```

LLM4Decompile (demangled)

```
int main(int argc, char *argv[]) {
  int ret;
  char cwd[256];
  char *p;

  aria2::global::initConsole(0);
  aria2::Platform::Platform(cwd);
  ret = aria2::main(argc, argv);

  aria2::Platform::~Platform(cwd);
  return ret;
}
```

# LLM4Decompile (stripped)

```
__int64 main(unsigned int a1, char **a2, char **a3)
{
  unsigned int v3;
  int v4;
  int v5;
  int v6;
  int v7;
  char v9;

  sub_593F0(0);
  sub_87C50();
  v3 = sub_545A0(a1, a2);
  sub_87EB0(&v9, a2, v4, v5, v6, v7);

  return v3;
}
```

IDA Pro

```
int main(int argc, char *argv[])
{
  char buf[15];
  char *p = buf;
  int i;
  int ret;

  std::ostream& std::ostream::_M_insert<long
long>(std::ostream& endl(std::ostream& os)@plt + 0);
 std::ctype<char>::do_widen(char) const@@Base(p);
  ret = std::ostream& std::ostream::_M_insert<long
long> (argc, argv);
 std::ctype<char>::do_widen(char) const@@Base(p);
  return ret;
}
```

Bug：未使用的區域變數

幻覺：錯誤的型別與函式

LLM4Decompile (demangled)

# LLM4Decompile (stripped)

幻覺：誤認為 CPython

```
__int64 sub_545A0(unsigned int a1, __int64 a2) {
  unsigned int v2;
  _QWORD vars0[2];
  _QWORD vars10[7];

  memset(vars10, 0, 24);
  sub_59FB0(vars0, 1LL, a1, a2, vars10);
  sub_591B0(vars10);
  if ( vars0[0] )
    v2 = sub_80310();
  else
    v2 = 0;
  sub_59B40(vars0);
  return v2;
}
```

IDA Pro

```
PyObject *sub_545A0(PyObject *self, PyObject
*args) {
    PyObject *return_value = NULL;
    Py_ssize_t _pos = 0;
    PyObject *item;
    _ssize_t pos = _pos;

    if (PyTuple_GET_SIZE(args) < 2) {
        goto skip_optional;
    }
    item = PyTuple_GET_ITEM(args, 1);
    if (_PyTuple_UnpackSequence(item, &pos,
NULL)) {
        goto skip_optional;
    }
    ...
```

LLM4Decompile

# ReSym (stripped)

```
__int64 main(unsigned int a1, char **a2, char **a3)
{
  unsigned int v3;
  int v4;
  int v5;
  int v6;
  int v7;
  char v9;

  sub_593F0(0);
  sub_87C50();
  v3 = sub_545A0(a1, a2);
  sub_87EB0(&v9, a2, v4, v5, v6, v7);

  return v3;
}
```

IDA Pro

```
__int64 main(unsigned int argc, char **argv, char **envp
{
  int ret;
  // int v4;
  // int v5;
  // int v6;
  // int v7;
  struct ucall uc;

  sub_593F0(0);
  sub_87C50();
  ret = sub_545A0(argc, argv);
  sub_87EB0(&uc, argv);

  return ret;
}
```

正確移除不存在的區域變數

幻覺：未知的型別

ReSym

# ReSym (stripped)

```
__int64 sub_545A0(unsigned int a1, __int64 a2) {
  unsigned int v2;
  _QWORD vars0[2];
  _QWORD vars10[7];

  memset(vars10, 0, 24);
  sub_59FB0(vars0, 1LL, a1, a2, vars10);
  sub_591B0(vars10);
  if ( vars0[0] )
    v2 = sub_80310();
  else
    v2 = 0;
  sub_59B40(vars0);
  return v2;
}
```

IDA Pro

明顯不合適的名稱

幻覺：不存在的型別

```
__int64 sub_545A0(int fd, const void* key) {
  int ret;
  struct COVER_map_t map;
  struct COVER_state_t state;

  memset(state, 0, 24);
  sub_59FB0(map, 1LL, fd, key, state);
  sub_591B0(state);
  if ( map[0] )
    ret = sub_80310();
  else
    ret = 0;
  sub_59B40(map);
  return ret;
}
```

ReSym

# 未微調超大型 LLM

- GPT-4o, GPT-o3-mini-high, Claude 3.7 Sonnet
- 輸入
  - 組合語言，同 LLM4Decompile
- 有符號資訊
  - 重新命名: 效果接近 IDA pro，帶有備註
  - 標註型別: 不如 IDA pro
- 沒有符號資訊 (stripped)：嚴重幻覺

```cpp
#include <iostream>

int main(int argc, char* argv[]) {
    std::cout << argc;
    return argc;
}
```

GPT-o3-mini-high 的結果

# Case Study：虛擬函式覆寫

- GPT-4o, GPT-o3-mini-high, Claude 3.7 Sonnet
- 輸入
  - 虛擬函式呼叫、子類別的 ctor、母類別的 ctor 的組合語言
- 結果：皆能正確推論

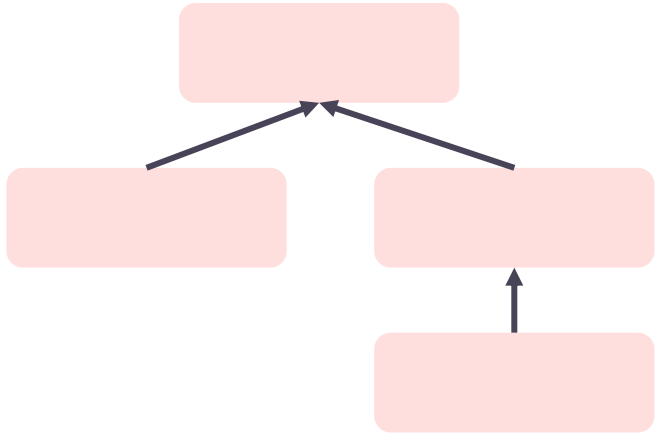
> Because the `Employee` constructor sets the vtable pointer to point to `Employee__say_my_name`, the function called at `0x1396` is **Employee__say_my_name** (the Employee's override of the virtual function `say_my_name`).
>
> Thus, the function that will be called at `0x1396 call rdx` is **Employee::say_my_name (Employee__say_my_name)**.

# 不同 LLM-based 方法的優缺點

- 做 decompile
  - 適合內容單純的小函式
  - 若產生幻覺會導致逆向無法進行
- 改善可讀性
  - 重新命名：可用（common pattern、人工提供上下文）
  - 標註型別：效果不佳
- 解釋程式碼
  - 容易過於籠統或瑣碎，但適合拿來認 common patterns

# 使用 LLM 協助逆向的建議

- 選擇重新命名變數、解釋程式碼的方法
- 以 bottom-up 的方式逆向
- Human in the loop
  - 1. 從 CFG 最底層的函式開始
  - 2. 使用 explain 方法協助辨認 common patterns 或小函式
  - 3. 人工逆向，提供上下文資訊
  - 4. refine 協助重新命名變數、函式
  - 5. 往上一層函式進行逆向



# 其他用法

- 快速解釋一段組合語言
- 快速生成分析腳本
- [新手] 協助認識 common patterns
  - 編碼
  - 加解密
  - 常見演算法

# 具潛力的專案和研究題目

- 1. LLM 與現有逆向工具整合度不足
- 2. 針對單一程式語言微調、資料集更大的逆向專用 LLM
- 3. 如何有效擷取跨函式上下文，使 LLM 有更多資訊協助判斷
- 4. LLM 具有小範圍推論的潛力，如何開發這個能力協助逆向
- 5. 如何利用 LLM 協助處理高階語言 (C++, Go, Rust) 的複雜結構
- 6. AI 逆向分析師（？）

# 結論

- 認識什麼是逆向工程，並對 C++ 的底層運作有新的了解
- 掌握 LLM 於協助逆向工程的最新發展和成果
- 對於資安工作者/學生，了解現階段 LLM 可以如何協助我們逆向

# 推薦閱讀

- Unleashing AI: The Future of Reverse Engineering with Large Language Models (ReCon 2024)

# Q&A

Thanks for your participation.