CCS 2024, Salt Lake City, Utah

Distinguished Paper Award

ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries

Danning Xie, Zhuo Zhang, Nan Jiang, Xiangzhe Xu, Lin Tan, Xiangyu Zhang

PURDUE UNIVERSITY

Background: Stripped Binary and Decompiled Code

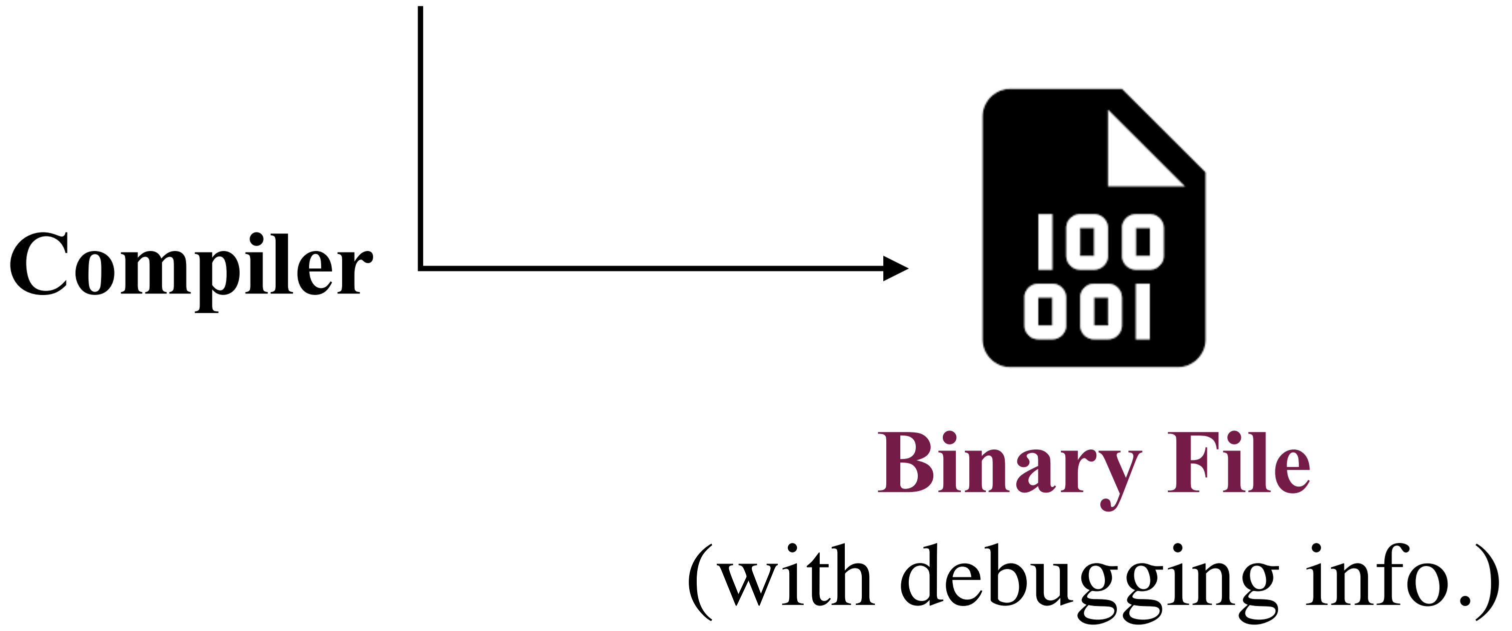
Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```



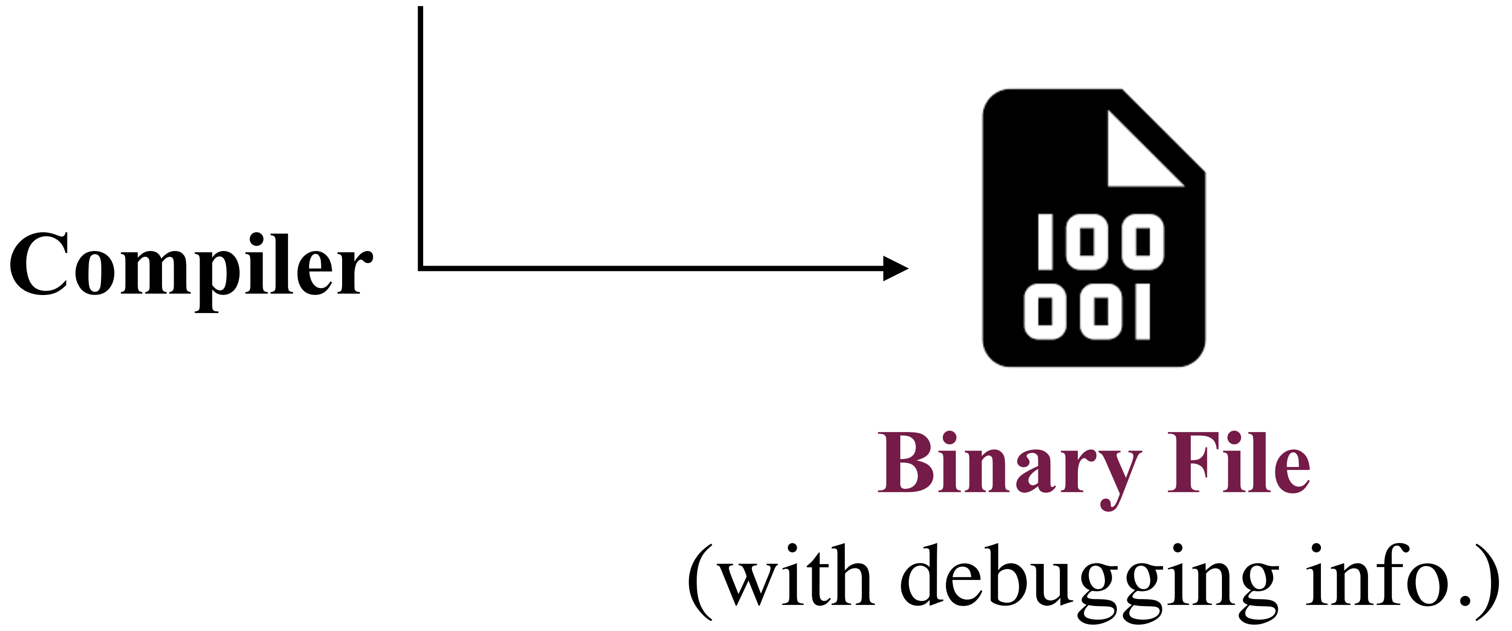
Compiler

Binary File
(with debugging info.)

Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```



Compiler

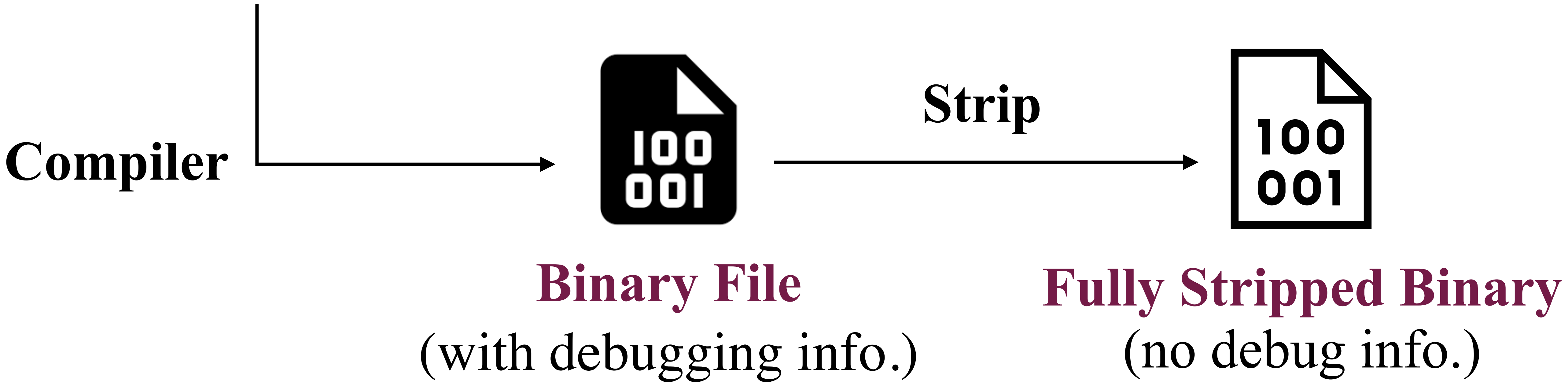
Binary File
(with debugging info.)

Debugging Information: locations, sizes, and layout of functions and objects

Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```



Binary File
(with debugging info.)

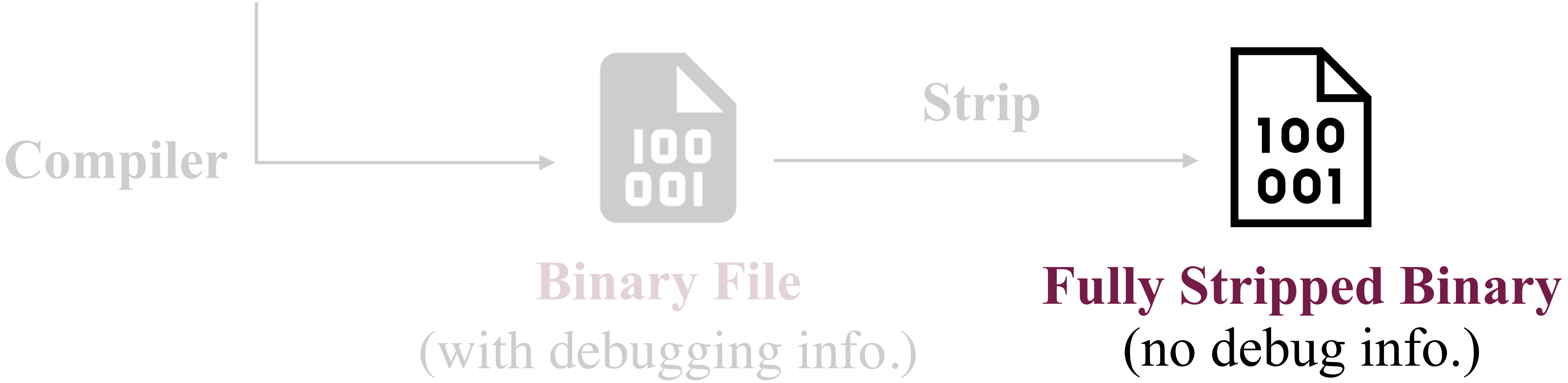
Fully Stripped Binary
(no debug info.)

Debugging Information: locations, sizes, and layout of functions and objects

Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```



Binary File
(with debugging info.)

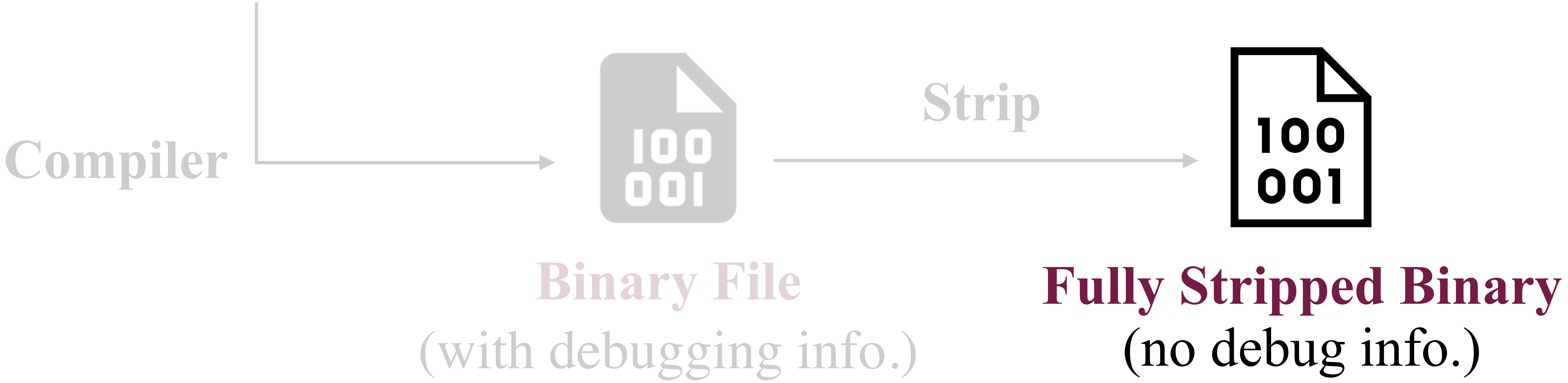
Fully Stripped Binary
(no debug info.)

Debugging Information: locations, sizes, and layout of functions and objects

Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```



Binary File

(with debugging info.)

Fully Stripped Binary

(no debug info.)

Debugging Information: locations, sizes, and layout of functions and objects

Understanding stripped binaries is essential to ensure software security.

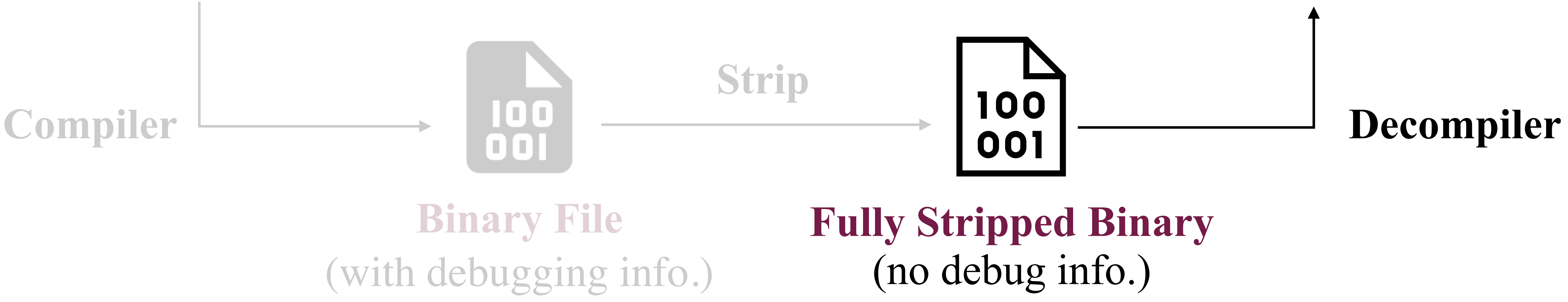
Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```



Binary File
(with debugging info.)

Fully Stripped Binary
(no debug info.)

- Decompilation aims to recover the source code form of a binary executable.

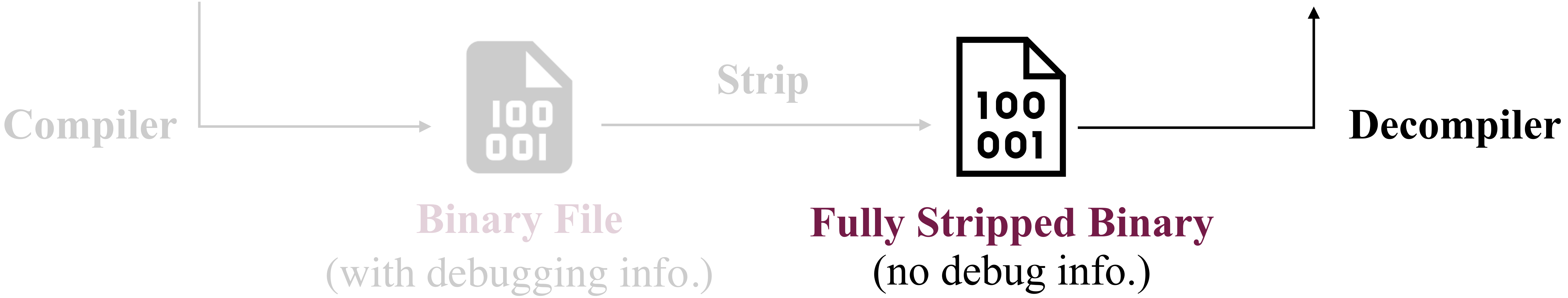
Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```



- Decompilation aims to recover the source code form of a binary executable.

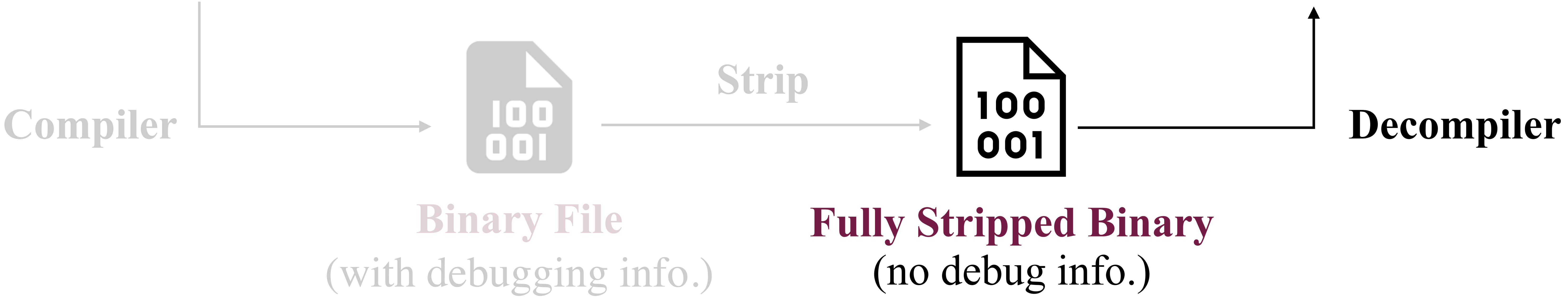
Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```



Binary File
(with debugging info.)

Fully Stripped Binary
(no debug info.)

- Decompilation aims to recover the source code form of a binary executable.

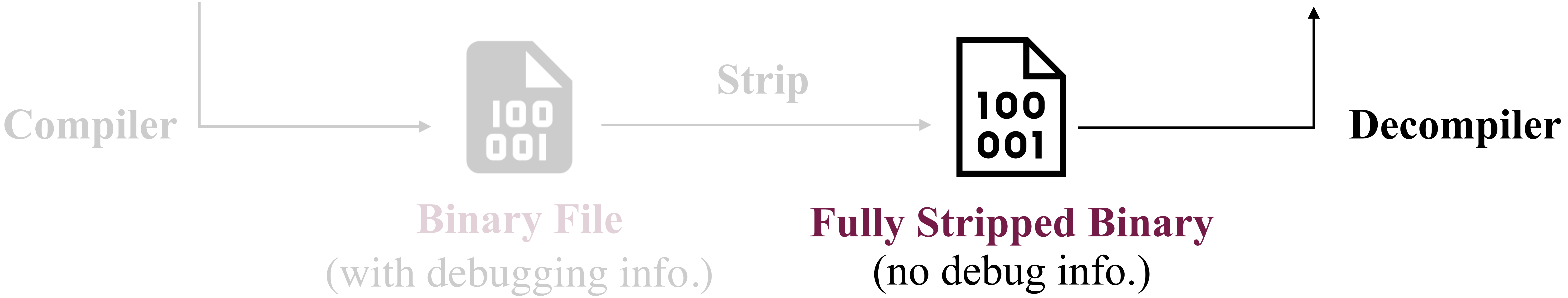
Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```



- Decompilation aims to recover the source code form of a binary executable.

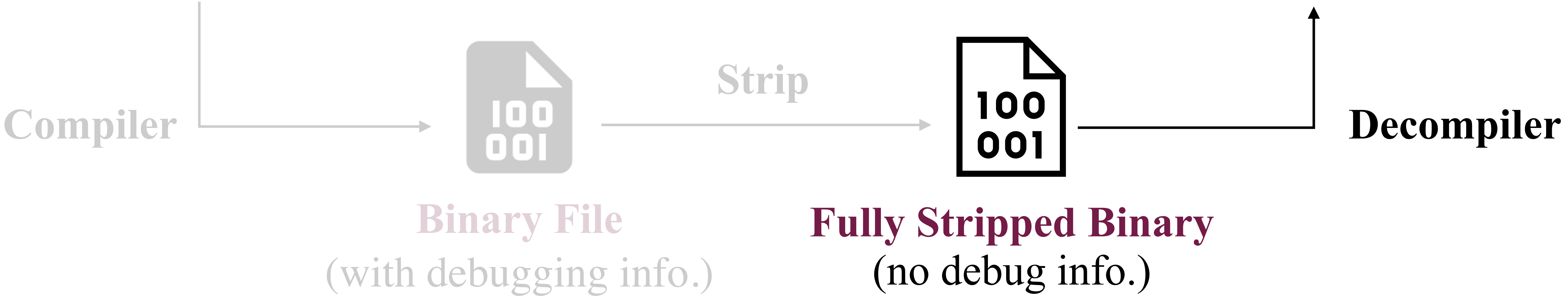
Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```



- Decompilation aims to recover the source code form of a binary executable.

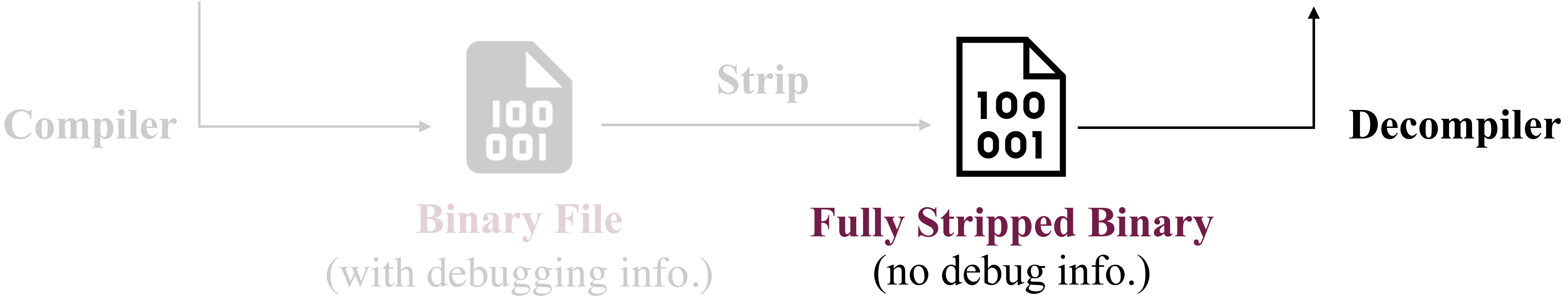
Background: Stripped Binary and Decompiled Code

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

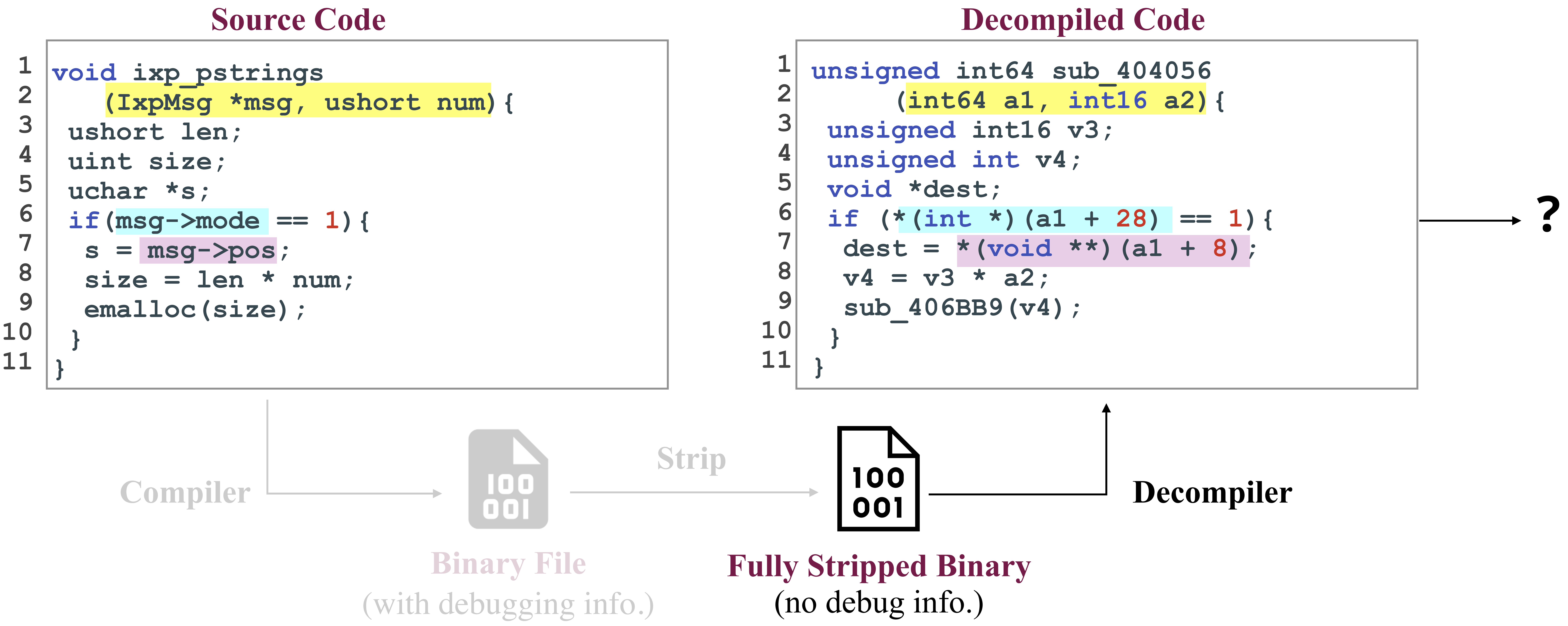


Binary File (with debugging info.)

Fully Stripped Binary (no debug info.)

- Decompilation aims to recover the source code form of a binary executable
- Decompiled code from stripped binaries loses symbol information.

Background: Stripped Binary and Decompiled Code



Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

- Decompilation aims to recover the source code form of a binary executable
- Decompiled code from stripped binaries loses symbol information.

Existing Techniques are Limited on Recovering User-defined Data Structures

Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Existing Techniques are Limited on Recovering User-defined Data Structures

Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Existing Techniques are Limited on Recovering User-defined Data Structures

Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

```
struct IxpMsg {
 char*     data;
 char*     pos;
 char*     end;
 _ixpuint size;
 _ixpuint mode;
};
```

Ground Truth

Existing Techniques are Limited on Recovering User-defined Data Structures

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

```
struct IxpMsg {
 char*     data;
 char*     pos;
 char*     end;
 _ixpuint size;
 _ixpuint mode;
};
```

Ground Truth

```
struct sha256_ctx {
 uint32_t H[8];
 uint32_t total[2];
 uint32_t buflen;
 char buffer[128];
};
```

DIRTY

Uses a multi-classification model

```
struct struct0{
 int8*  s_0,
 int8*  s_1,
 int8*  s_2,
 int64  s_3,
 int64  s_4
};
```

OSPREY

Only recovers layout

Opportunities: LLMs' Power of Code Comprehension

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

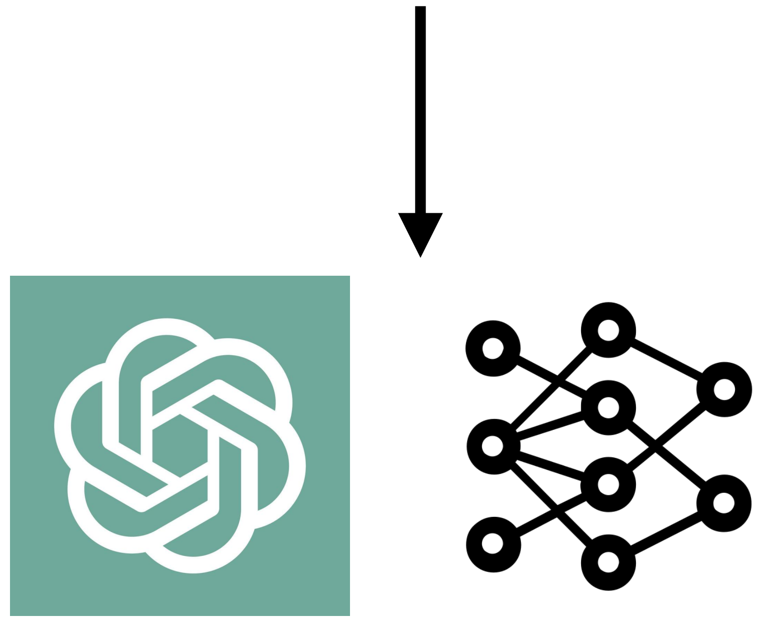
Opportunities: LLMs' Power of Code Comprehension

Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```



Opportunities: LLMs' Power of Code Comprehension

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

ChatGPT-4 Output

```
unsigned long long sub_404056
   (long long a1, unsigned short a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if(*(unsigned int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 28);

  v4 = v3 * a2;
  sub_406BB9(v4);
  }
 }
```

Direct Prompting Pre-trained/Fine-tuned LLMs is not Ideal

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

ChatGPT-4 Output

```
unsigned long long sub_404056
   (long long a1, unsigned short a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if(*(unsigned int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 28);

  v4 = v3 * a2;
  sub_406BB9(v4);
  }
}
```

Fine-tuned StarCoder-3B Model Output

```
void process
  (state *s, unsigned short n){
 unsigned short len;
 int size;
 char *tmp;
 if (s->last == 1) {
  tmp = s->tmp;
  size = len * n;
  min(len);
  }
}
```

Direct Prompting Pre-trained/Fine-tuned LLMs is not Ideal

Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

ChatGPT-4 Output

```
unsigned long long sub_404056
   (long long a1, unsigned short a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if(*(unsigned int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 28);

  v4 = v3 * a2;
  sub_406BB9(v4);
  }
}
```

Fine-tuned StarCoder-3B Model Output

```
void process
  (state *s, unsigned short n){
 unsigned short len;
 int size;
 char *tmp;
 if (s->last == 1) {
  tmp = s->tmp;
  size = len * n;
  min(len);
  }
}
```

Direct Prompting Pre-trained/Fine-tuned LLMs is not Ideal

Source Code

```
void ixp_pstrings
    (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

ChatGPT-4 Output

```
unsigned long long sub_404056
   (long long a1, unsigned short a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if(*(unsigned int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 28);
                    // should be `8`
  v4 = v3 * a2;
  sub_406BB9(v4);
  }
 }
```

Fine-tuned StarCoder-3B Model Output

```
void process
  (state *s, unsigned short n){
 unsigned short len;
 int size;
 char *tmp;
 if (s->last == 1) {
  tmp = s->tmp;
  size = len * n;
  min(len); // should be `size`
  }
}
```

Direct Prompting Pre-trained/Fine-tuned LLMs is not Ideal

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
```

Challenges:

- A **general-purpose LLM** (e.g., ChatGPT) **struggles** to produce readable decompiled code.
- LLMs' direct outputs may have **incorrect semantics**.
- Accurate recovery requires a **global view**, while LLMs have token limits.

```
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if(*(unsigned int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 28);
                   // should be `8`
  v4 = v3 * a2;
  sub_406BB9(v4);
  }
}
```

```
 unsigned short len;
 int size;
 char *tmp;
 if (s->last == 1) {
  tmp = s->tmp;
  size = len * n;
  min(len); // should be `size`
  }
}
```

Our Technique: **ReSym**

- A *hybrid approach* synergizes insights from LLMs and program analysis to recover variable and data structure symbols from stripped binaries
- Leverages two fine-tuned LLMs
- Replicates the reverse engineering process used by human experts

Our Technique: **ReSym**

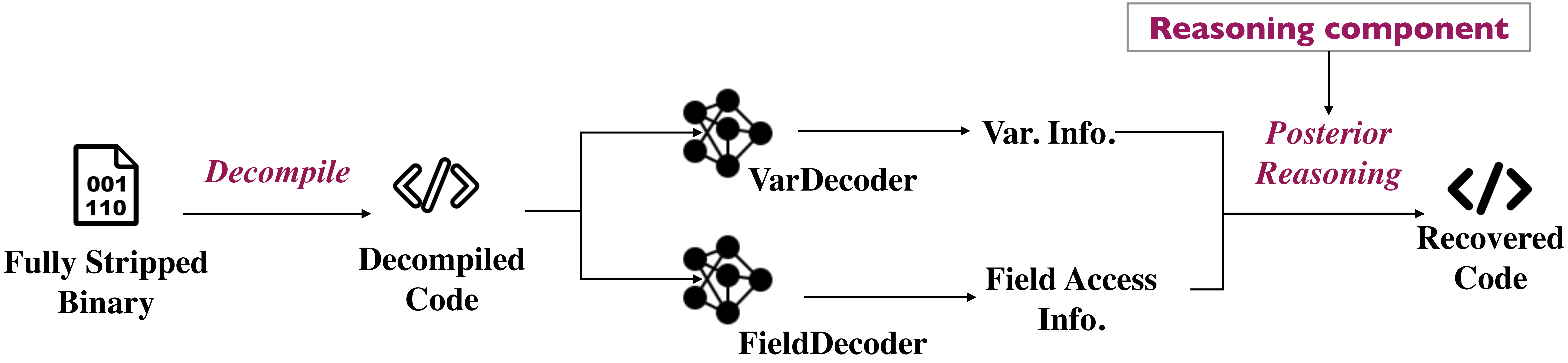
- A *hybrid approach* synergizes insights from LLMs and program analysis to recover variable and data structure symbols from stripped binaries
- Leverages two fine-tuned LLMs
- Replicates the reverse engineering process used by human experts

① **Break** the task into smaller manageable subtasks

② Focus on one piece at a time and then **aggregate** insights from multiple code snippets

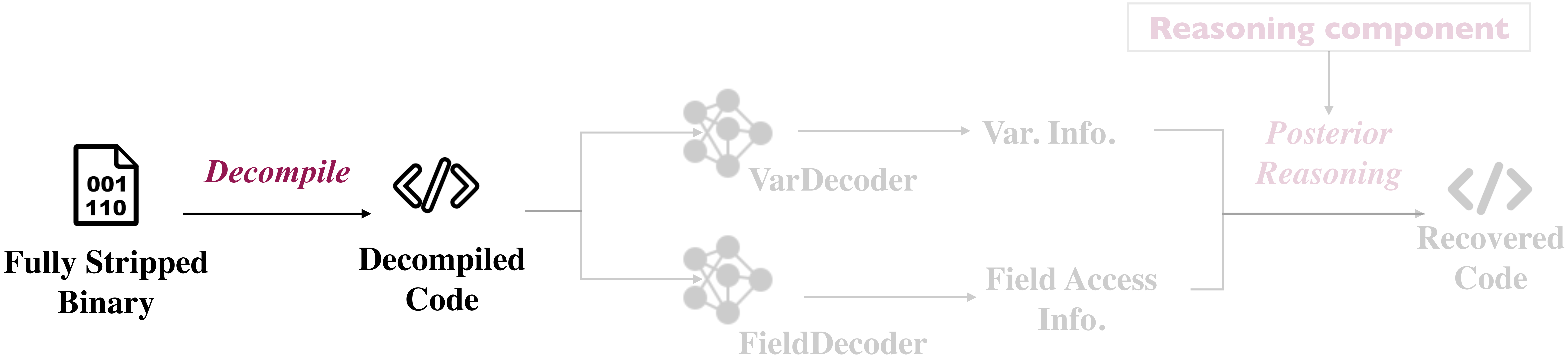
ReSym Pipeline

Key idea: Break and Aggregate



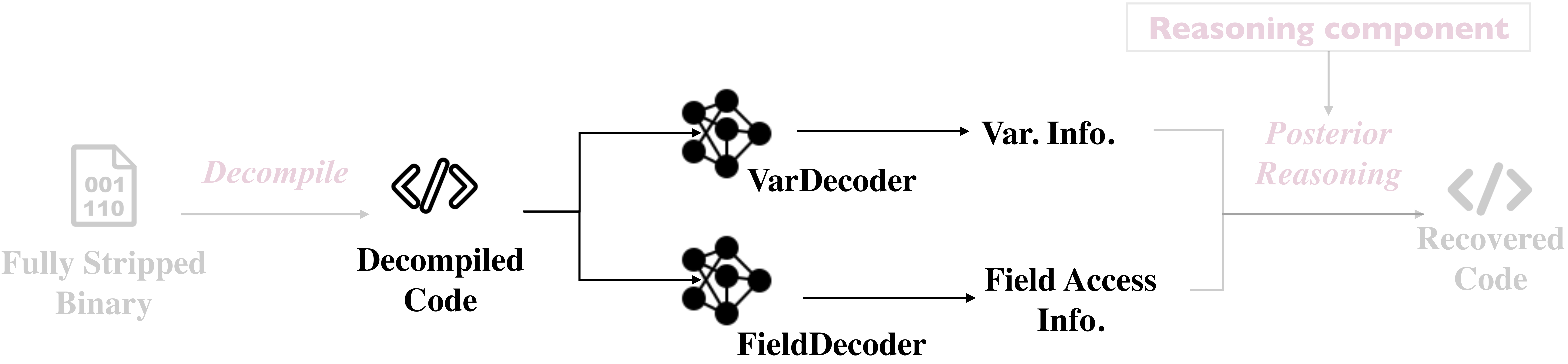
ReSym Pipeline

Key idea: Break and Aggregate



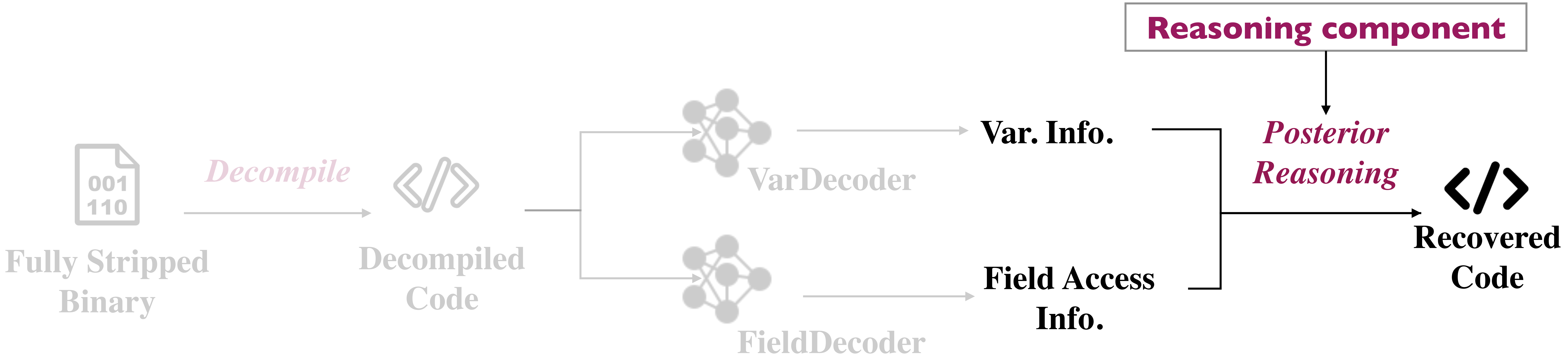
ReSym Pipeline

Key idea: Break and Aggregate



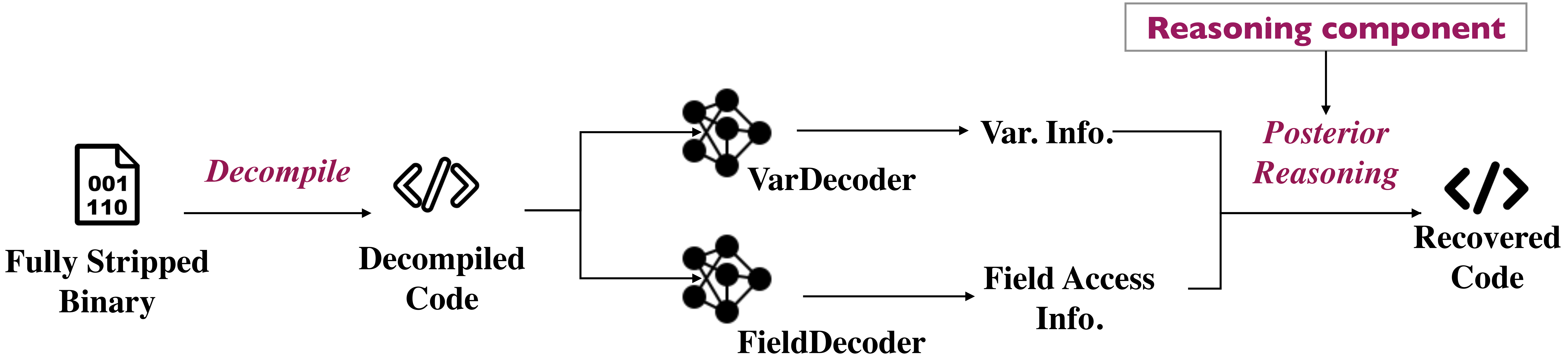
ReSym Pipeline

Key idea: Break and Aggregate



ReSym Pipeline

Key idea: Break and Aggregate



VarDecoder: Recover Variable Information

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

VarDecoder: Recover Variable Information

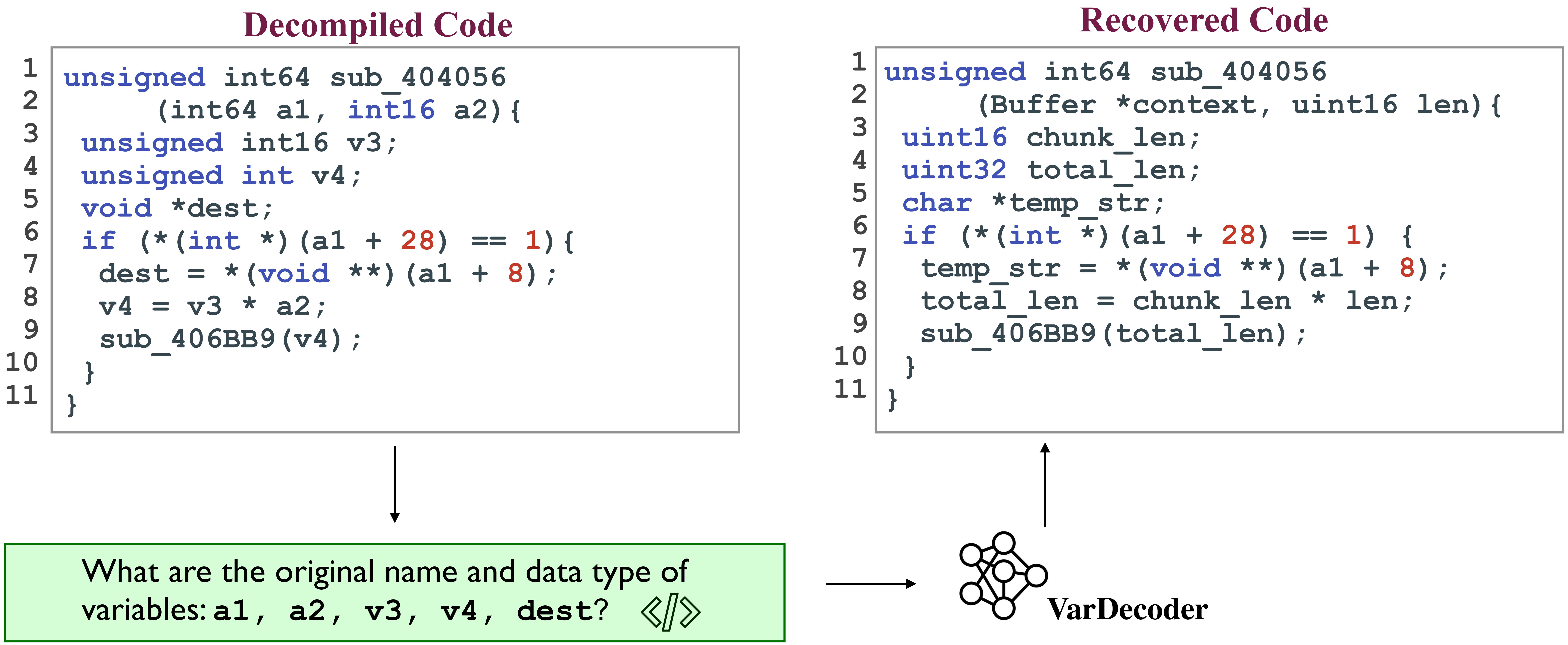
Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

What are the original name and data type of variables: `a1, a2, v3, v4, dest`?

VarDecoder

VarDecoder: Recover Variable Information



Decompiled Code

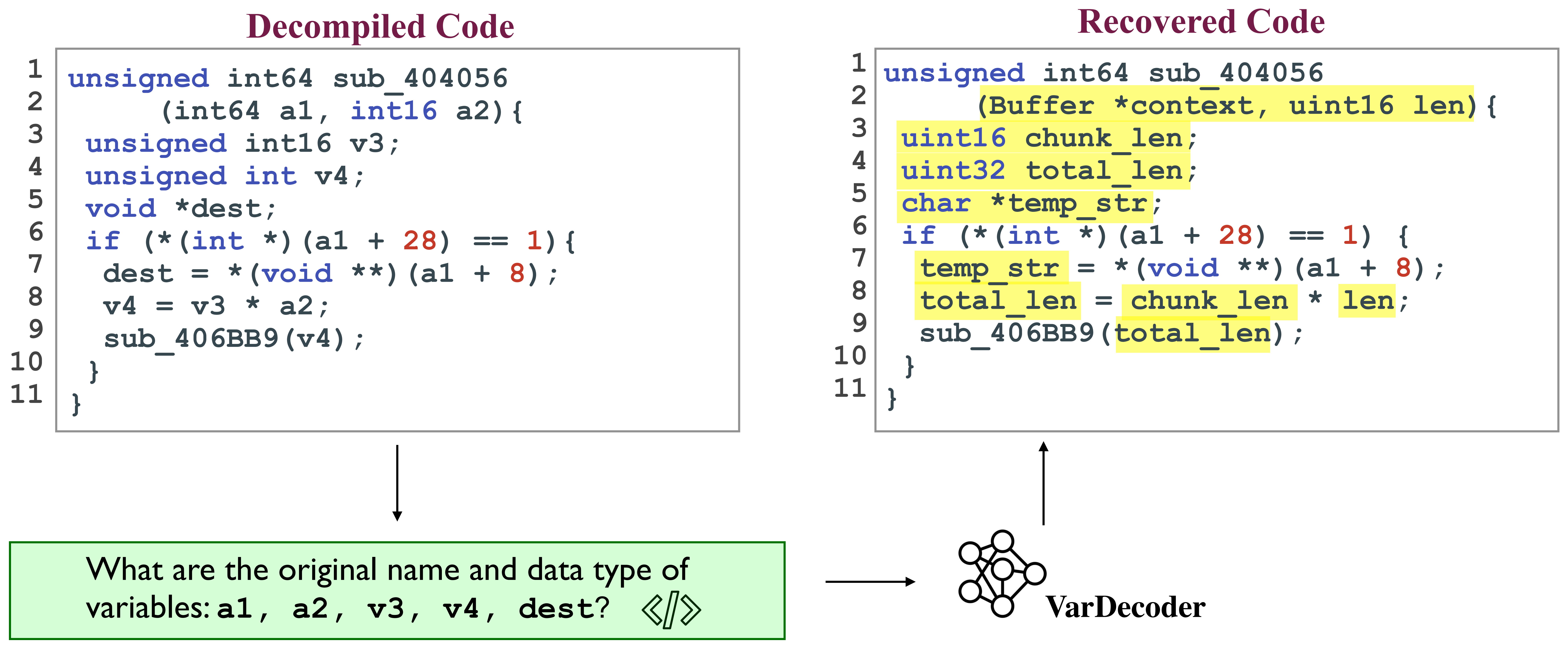
```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

What are the original name and data type of variables: `a1, a2, v3, v4, dest`? </>

Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (*(int *)(a1 + 28) == 1) {
  temp_str = *(void **)(a1 + 8);
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

VarDecoder: Recover Variable Information



Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

What are the original name and data type of variables: `a1, a2, v3, v4, dest`?

VarDecoder

Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (*(int *)(a1 + 28) == 1) {
  temp_str = *(void **)(a1 + 8);
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

FieldDecoder: Recover Field Access Information

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

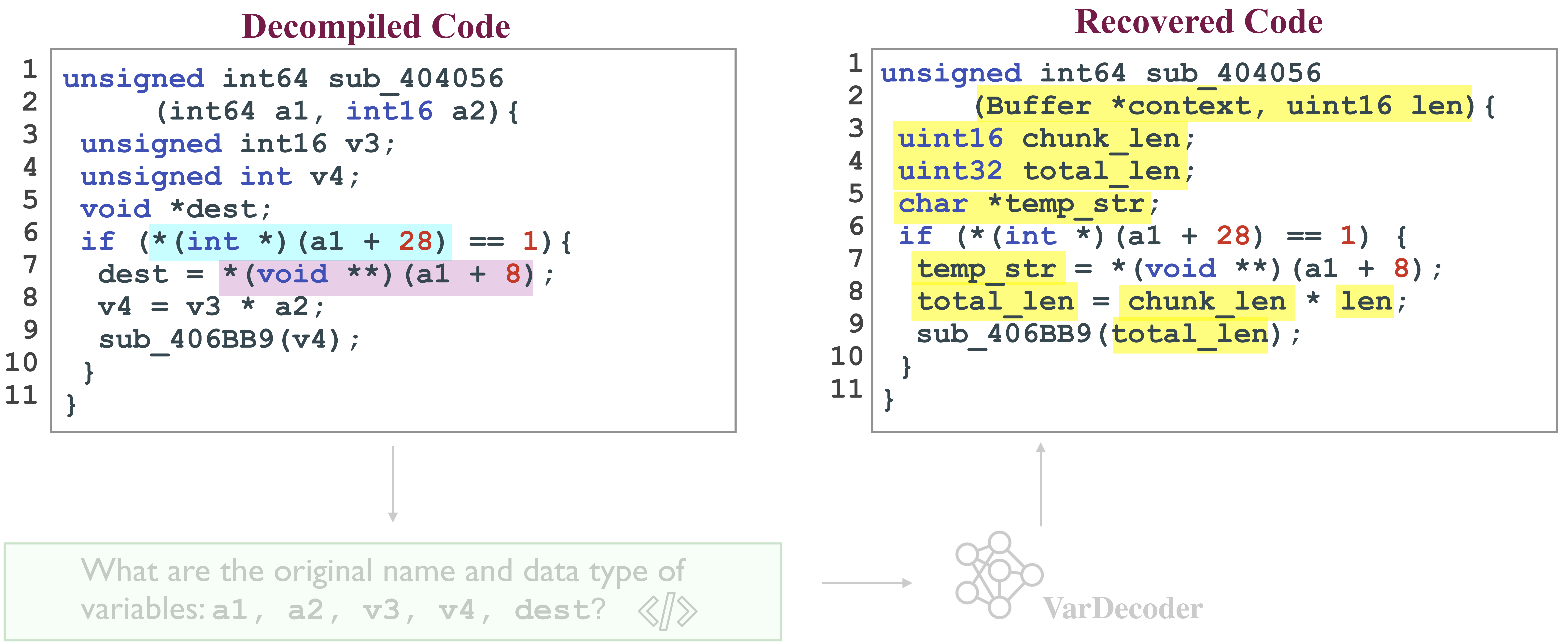
Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (*(int *)(a1 + 28) == 1) {
  temp_str = *(void **)(a1 + 8);
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

What are the original name and data type of variables: `a1, a2, v3, v4, dest`?

VarDecoder

FieldDecoder: Recover Field Access Information



Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (*(int *)(a1 + 28) == 1) {
  temp_str = *(void **)(a1 + 8);
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

What are the original name and data type of variables: a1, a2, v3, v4, dest?

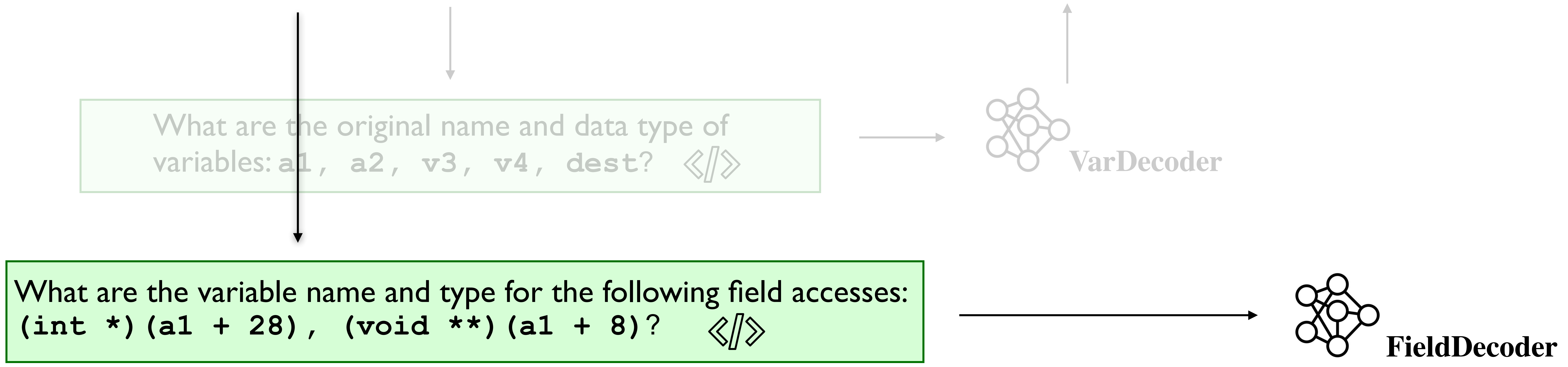
FieldDecoder: Recover Field Access Information

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (*(int *)(a1 + 28) == 1) {
  temp_str = *(void **)(a1 + 8);
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```



What are the original name and data type of variables: `a1, a2, v3, v4, dest?` </>

What are the variable name and type for the following field accesses: `(int *)(a1 + 28), (void **)(a1 + 8)?` </>

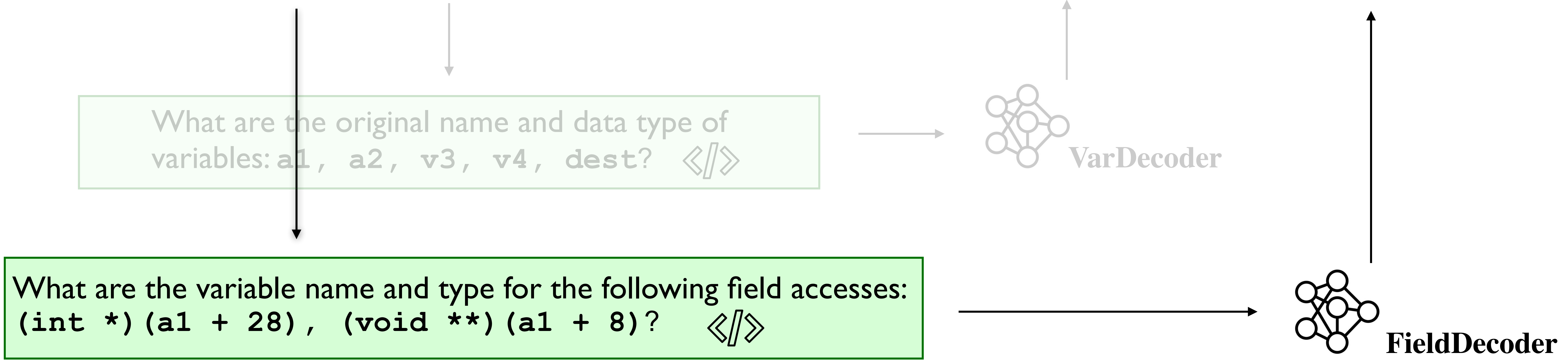
FieldDecoder: Recover Field Access Information

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (context->type == 1) {
  temp_str = context->pos;
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```



What are the original name and data type of variables: `a1, a2, v3, v4, dest?` </>

What are the variable name and type for the following field accesses: `(int *)(a1 + 28), (void **)(a1 + 8)?` </>

Posterior Reasoning:

Aggregating Field Access from Multiple Functions

Decompiled Code

```
unsigned int64 sub_404056
      (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Recovered Code

```
unsigned int64 sub_404056
      (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (context->type == 1) {
  temp_str = context->pos;
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

Recovered Data Structure

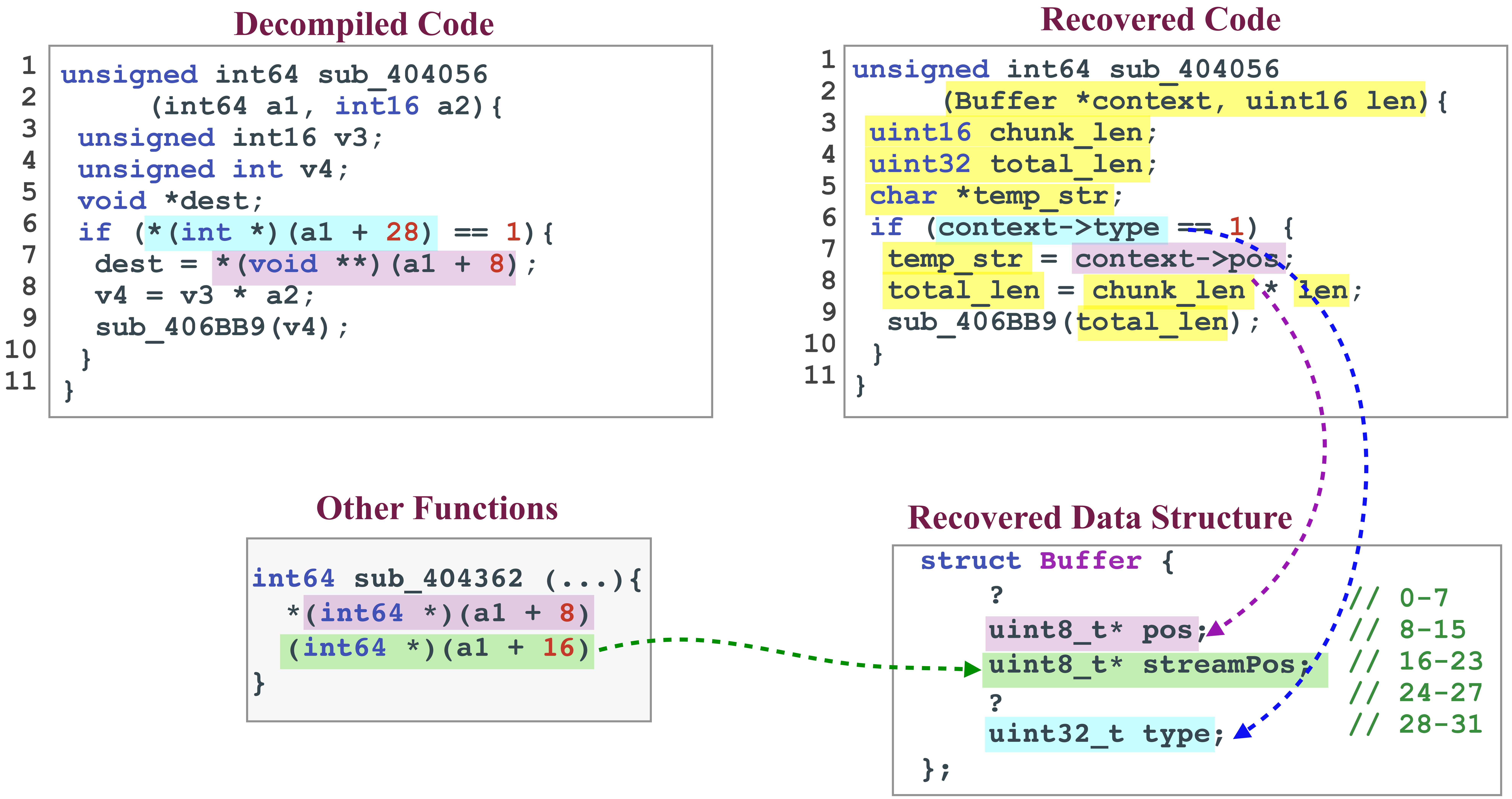
```
struct Buffer {
    ?                          // 0-7
    uint8_t* pos;              // 8-15

    ?                          ?

    uint32_t type;             // 28-31
};
```

Posterior Reasoning:

Aggregating Field Access from Multiple Functions



Decompiled Code

```
unsigned int64 sub_404056
      (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Recovered Code

```
unsigned int64 sub_404056
      (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (context->type == 1) {
  temp_str = context->pos;
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

Other Functions

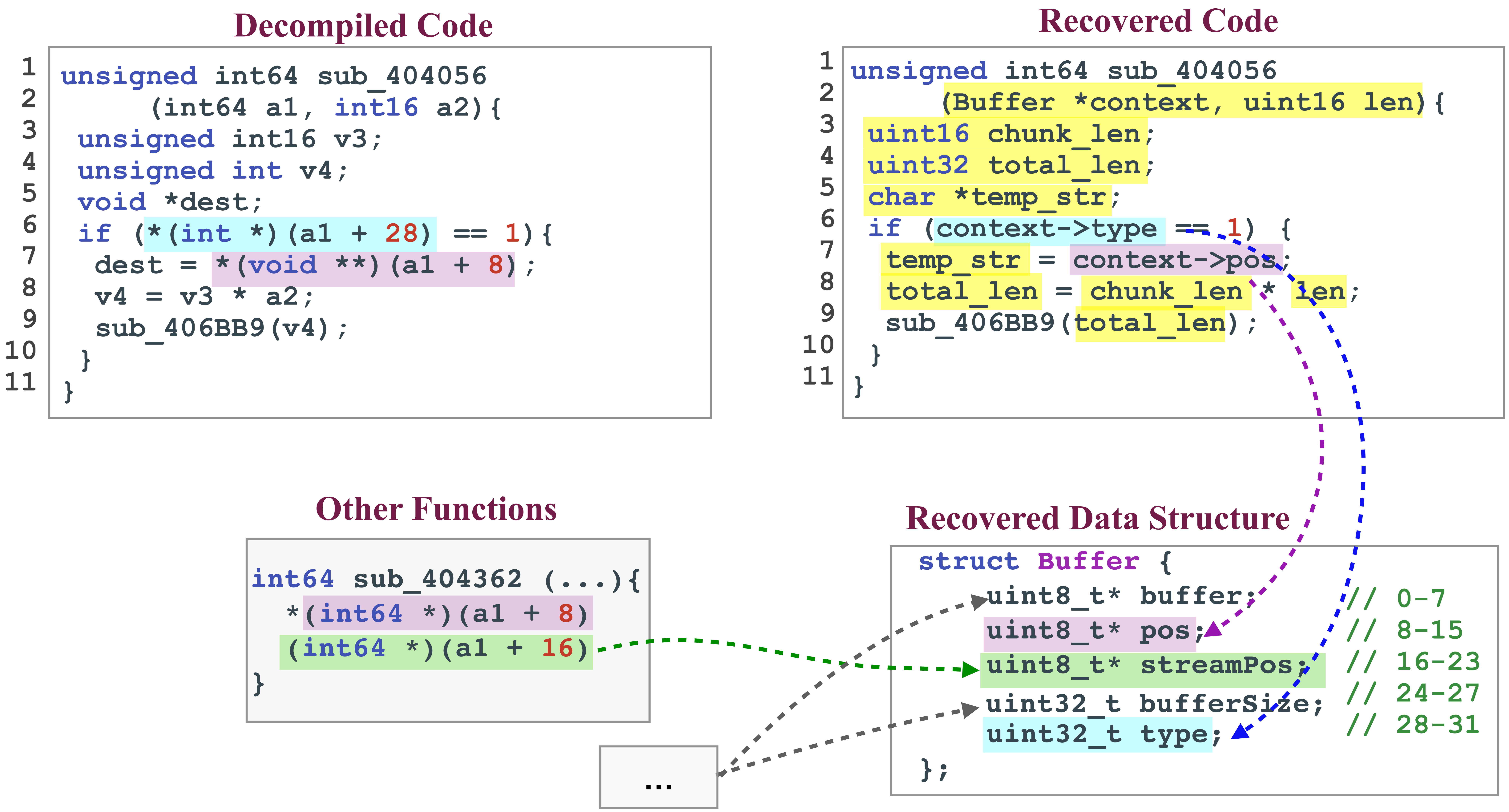
```
int64 sub_404362 (...){
  *(int64 *)(a1 + 8)
  (int64 *)(a1 + 16)
}
```

Recovered Data Structure

```
struct Buffer {
      ?                       // 0-7
      uint8_t* pos;           // 8-15
      uint8_t* streamPos;     // 16-23
      ?                       // 24-27
      uint32_t type;          // 28-31
};
```

Posterior Reasoning:

Aggregating Field Access from Multiple Functions



Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

Recovered Code

```
unsigned int64 sub_404056
     (Buffer *context, uint16 len){
 uint16 chunk_len;
 uint32 total_len;
 char *temp_str;
 if (context->type == 1) {
  temp_str = context->pos;
  total_len = chunk_len * len;
  sub_406BB9(total_len);
 }
}
```

Other Functions

```
int64 sub_404362 (...){
  *(int64 *)(a1 + 8)
  (int64 *)(a1 + 16)
}
```

```
...
```

Recovered Data Structure

```
 struct Buffer {
     uint8_t* buffer;      // 0-7
     uint8_t* pos;         // 8-15
     uint8_t* streamPos;   // 16-23
     uint32_t bufferSize;  // 24-27
     uint32_t type;        // 28-31
 };
```

Existing Techniques are Limited on Recovering User-defined Data Structures

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

```
struct IxpMsg {
 char*     data;
 char*     pos;
 char*     end;
 _ixpuint size;
 _ixpuint mode;
};
```

Ground Truth

```
struct sha256_ctx {
 uint32_t H[8];
 uint32_t total[2];
 uint32_t buflen;
 char buffer[128];
};
```

DIRTY

Uses a multi-classification model

```
struct struct0{
 int8*   s_0,
 int8*   s_1,
 int8*   s_2,
 int64   s_3,
 int64   s_4
};
```

OSPREY

Only recovers layout

Existing Techniques are Limited on Recovering User-defined Data Structures

Source Code

```
void ixp_pstrings
   (IxpMsg *msg, ushort num){
 ushort len;
 uint size;
 uchar *s;
 if(msg->mode == 1){
  s = msg->pos;
  size = len * num;
  emalloc(size);
 }
}
```

Decompiled Code

```
unsigned int64 sub_404056
     (int64 a1, int16 a2){
 unsigned int16 v3;
 unsigned int v4;
 void *dest;
 if (*(int *)(a1 + 28) == 1){
  dest = *(void **)(a1 + 8);
  v4 = v3 * a2;
  sub_406BB9(v4);
 }
}
```

```
struct IxpMsg {
 char*     data;
 char*     pos;
 char*     end;
 _ixpuint size;
 _ixpuint mode;
};
```

Ground Truth

```
struct sha256_ctx {
 uint32_t H[8];
 uint32_t total[2];
 uint32_t buflen;
 char buffer[128];
};
```

DIRTY

Uses a multi-classification model

```
struct struct0{
 int8*   s_0,
 int8*   s_1,
 int8*   s_2,
 int64   s_3,
 int64   s_4
};
```

OSPREY

Only recovers layout

```
struct Buffer {
 uint8_t* buffer;
 uint8_t* pos;
 uint8_t* streamPos;
 uint32_t bufferSize;
 uint32_t type;
};
```

ReSym

Experimental Setup

- **3,058** C/C++ real-world projects collected from GitHub
- **16,217** binary files
  - average size: **116 KB**; maximum size: **8.9 MB**
- Split train/test set **by project** with a ratio of 0.95
- Fine-tune two **StarCoder 3B** models for VarDecoder and FieldDecoder

Research Questions

- How good is ReSym at recovering variable names and types?
- How good is ReSym at recovering user-defined data structures?

Evaluation: Name and Type Recovery

Perfect match accuracy (%)

Method	Overall	
	name	type
ReSym	56.7 8.0↑	60.7 4.9↑
DIRTY	48.7	55.8

Evaluation: Name and Type Recovery

Perfect match accuracy (%)

Method	Overall	
	name	type
ReSym	56.7 8.0↑	60.7 4.9↑
DIRTY	48.7	55.8

ReSym is effective in recovering variable names and types.

ReSym outperforms DIRTY by 4.9 — **8.0%**.

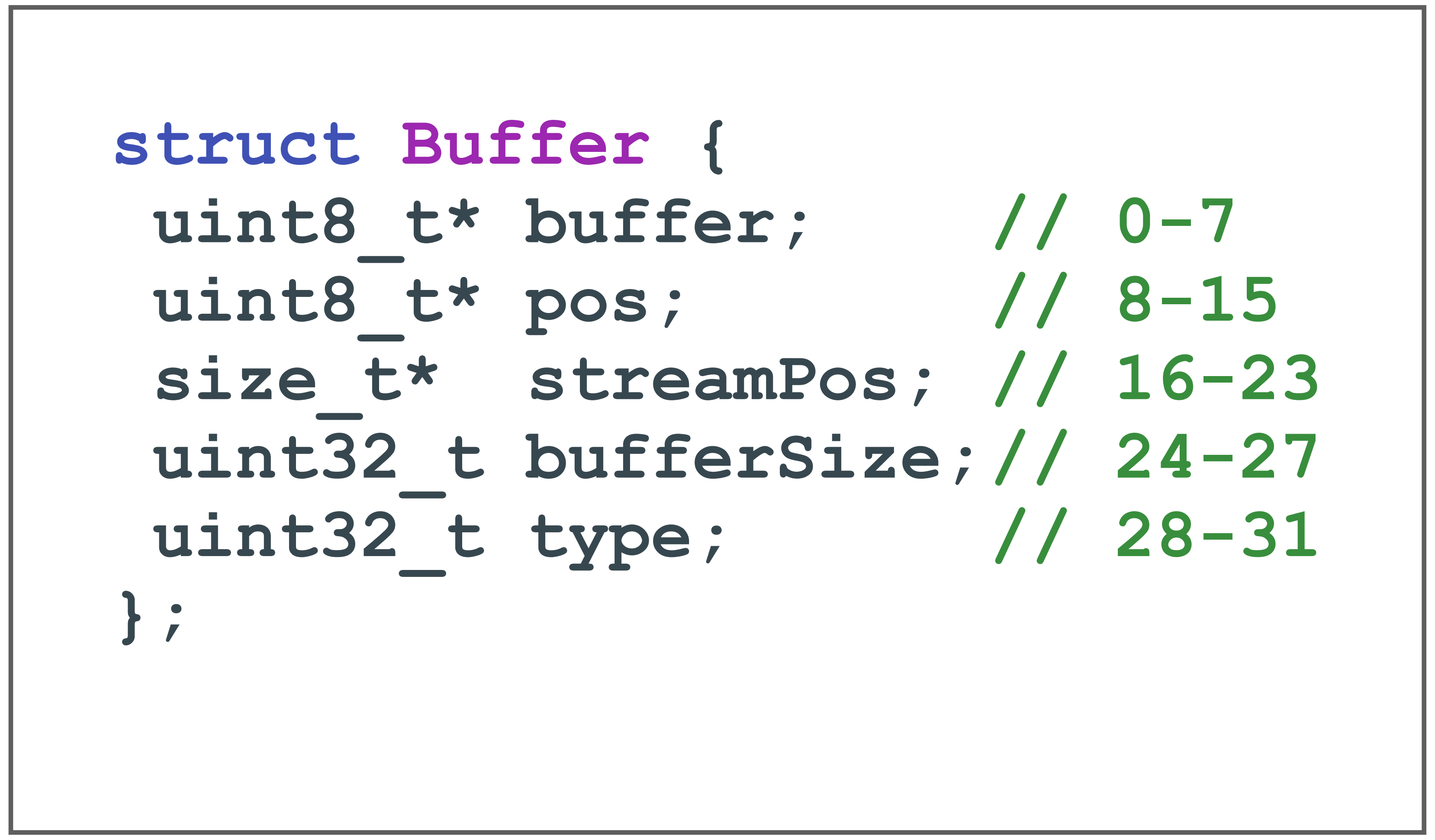
Evaluation: User-Defined Data Structure Recovery

Method	**Struct Layout**			**Struct Annotation (Accuracy)**		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5

Evaluation: User-Defined Data Structure Recovery

Method	**Struct Layout**			**Struct Annotation (Accuracy)**		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5

Recovered Data Structure



```
struct Buffer {
 uint8_t* buffer;     // 0-7
 uint8_t* pos;        // 8-15
 size_t*  streamPos;  // 16-23
 uint32_t bufferSize;// 24-27
 uint32_t type;       // 28-31
};
```

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5

Recovered Data Structure

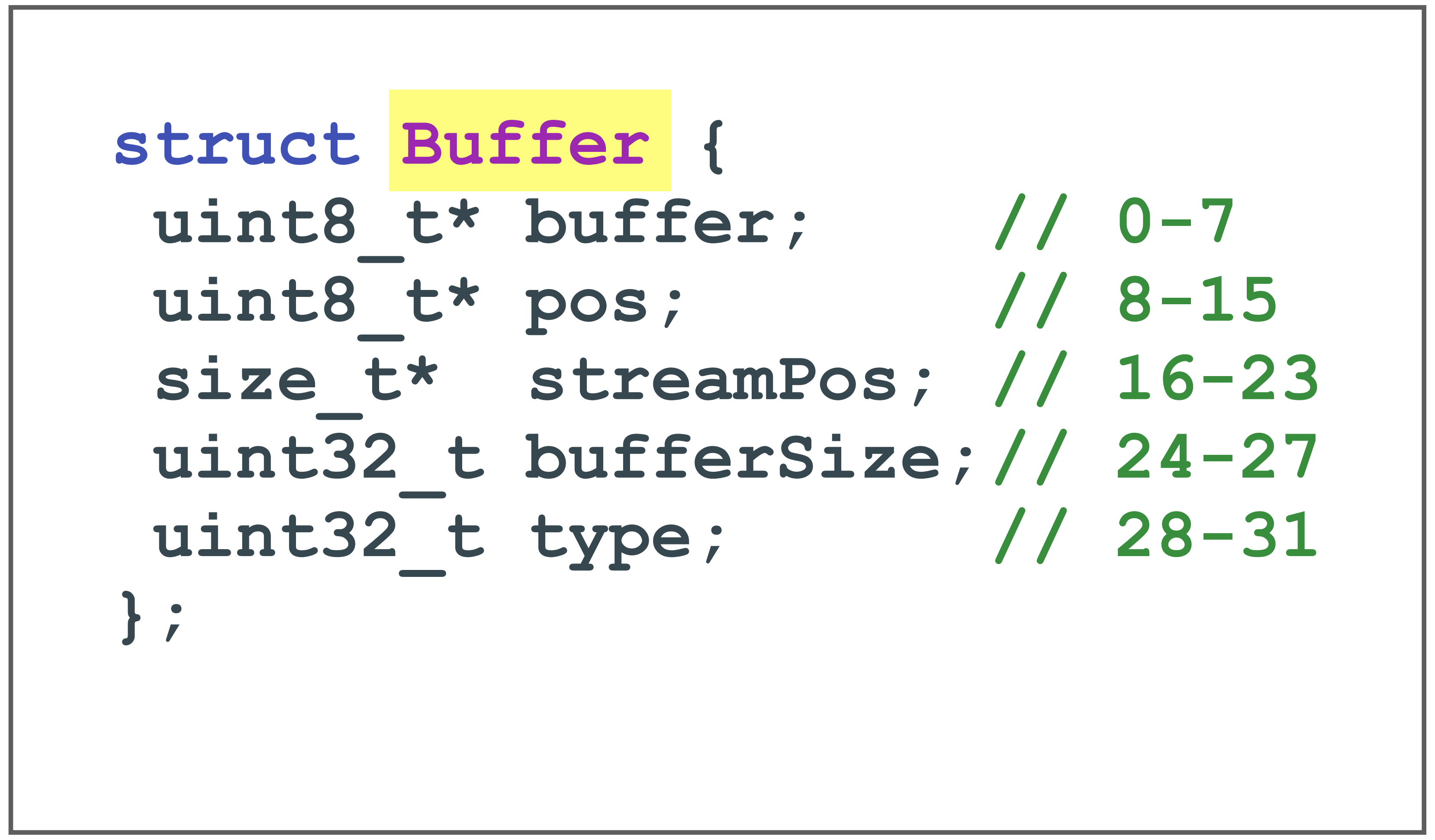
```
struct Buffer {
 uint8_t* buffer;     // 0-7
 uint8_t* pos;        // 8-15
 size_t*  streamPos;  // 16-23
 uint32_t bufferSize;// 24-27
 uint32_t type;       // 28-31
};
```

predicted offsets and sizes

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	**Struct Type**	Field Name	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5

Recovered Data Structure



```
struct Buffer {
 uint8_t* buffer;     // 0-7
 uint8_t* pos;        // 8-15
 size_t*  streamPos;  // 16-23
 uint32_t bufferSize; // 24-27
 uint32_t type;       // 28-31
};
```

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	Struct Type	**Field Name**	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5

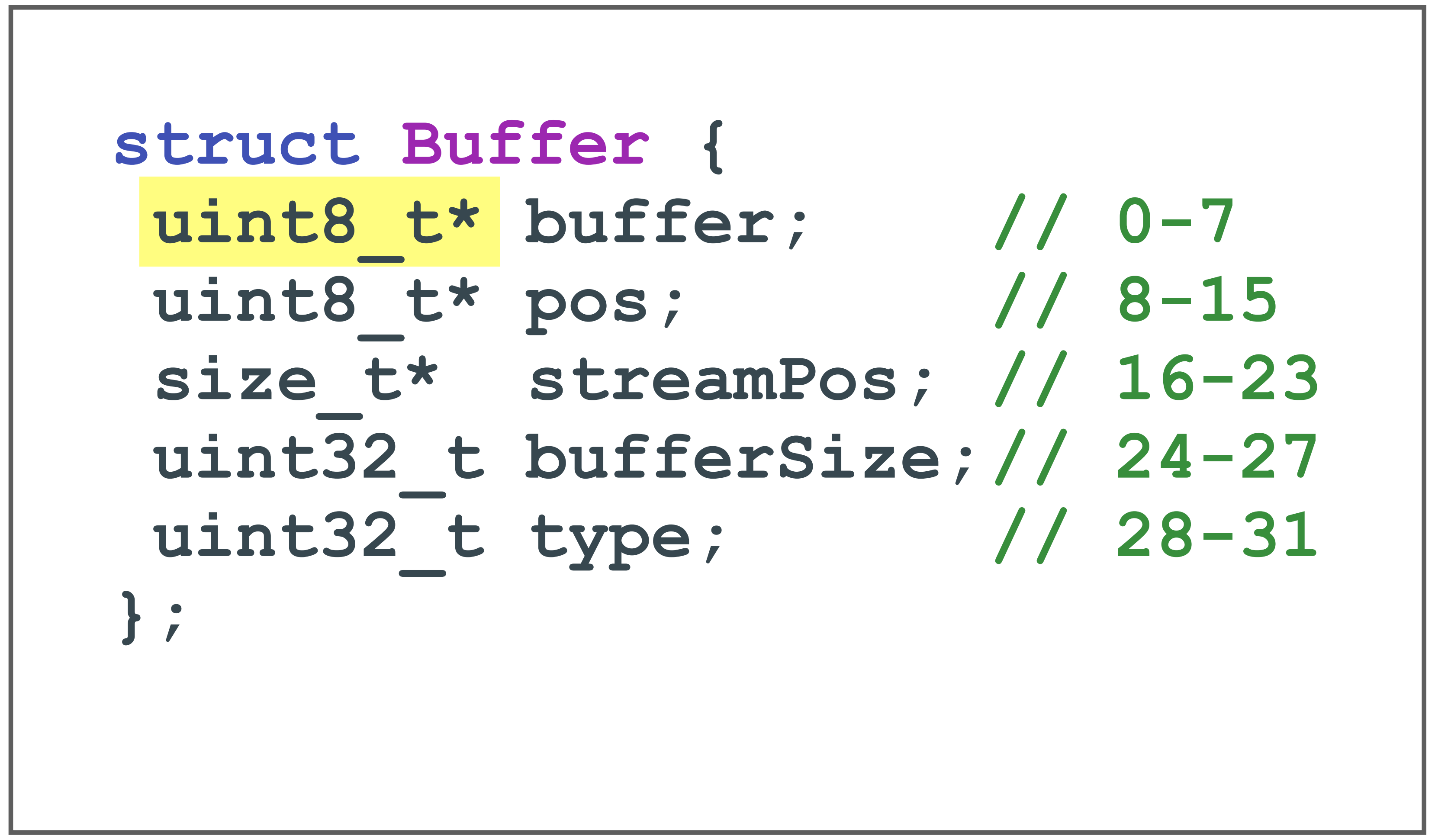
Recovered Data Structure

```
struct Buffer {
 uint8_t* buffer;      // 0-7
 uint8_t* pos;         // 8-15
 size_t*  streamPos;   // 16-23
 uint32_t bufferSize;// 24-27
 uint32_t type;        // 28-31
};
```

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	Struct Type	Field Name	**Field Type**
ReSym	81.9	34.6	48.6	44.4	14.4	15.5

Recovered Data Structure



```
struct Buffer {
 uint8_t* buffer;     // 0-7
 uint8_t* pos;        // 8-15
 size_t*  streamPos;  // 16-23
 uint32_t bufferSize;// 24-27
 uint32_t type;       // 28-31
};
```

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5
ReSymLight	73.3	29.8	42.4	40.2	7.9	8.3

ReSym without posterior reasoning

Evaluation: User-Defined Data Structure Recovery

Method	**Struct Layout**			**Struct Annotation (Accuracy)**		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	81.9	34.6	**48.6**	44.4	14.4	15.5
ReSym$_{Light}$	73.3	29.8	**42.4**	40.2	7.9	8.3

ReSym without posterior reasoning

Posterior Reasoning is effective and improves the F1 score by 6.2%.

Evaluation: User-Defined Data Structure Recovery

Method	**Struct Layout**			**Struct Annotation (Accuracy)**		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	81.9	34.6	48.6	44.4	14.4	15.5
ReSym$_{Light}$	73.3	29.8	42.4	40.2	7.9	8.3
OSPREY	38.1	60.2	46.7	-	-	-
DIRTY	54.6	3.3	6.2	0.5	0.8	0.9

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	**81.9**	34.6	**48.6**	**44.4**	**14.4**	**15.5**
$\text{ReSym}_{\text{Light}}$	73.3	29.8	42.4	40.2	7.9	8.3
OSPREY	38.1	**60.2**	46.7	-	-	-
DIRTY	54.6	3.3	6.2	0.5	0.8	0.9

ReSym's lower recall:

1. Discarding functions due to token limit
2. Data-flow analysis is inherently undecidable

Evaluation: User-Defined Data Structure Recovery

Method	**Struct Layout**			**Struct Annotation (Accuracy)**		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	**81.9**	34.6	**48.6**	**44.4**	**14.4**	**15.5**
ReSym$_{\text{Light}}$	73.3	29.8	42.4	40.2	7.9	8.3
OSPREY	38.1	**60.2**	46.7	-	-	-
DIRTY	54.6	3.3	6.2	0.5	0.8	0.9

Evaluation: User-Defined Data Structure Recovery

Method	**Struct Layout**			**Struct Annotation (Accuracy)**		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	**81.9**	34.6	**48.6**	**44.4**	**14.4**	**15.5**
ReSym$_{\text{Light}}$	73.3	29.8	42.4	40.2	7.9	8.3
OSPREY	38.1	**60.2**	46.7	-	-	-
DIRTY	54.6	3.3	6.2	0.5	0.8	0.9

ReSym achieves the highest F1 score and accurately recovers structure annotation.

Evaluation: User-Defined Data Structure Recovery

Method	Struct Layout			Struct Annotation (Accuracy)		
	Precision	Recall	F1	Struct Type	Field Name	Field Type
ReSym	**81.9**	34.6	**48.6**	**44.4**	**14.4**	**15.5**
ReSym$_{\text{Light}}$	73.3	29.8	42.4	40.2	7.9	8.3
OSPREY	38.1	**60.2**	46.7	-	-	-
DIRTY	54.6	3.3	6.2	0.5	0.8	0.9

ReSym achieves the highest F1 score and accurately recovers structure annotation.

ReSym analyzes each binary file in 3.4s, while OSPREY takes 528.24s.

Conclusions

- Propose a prototype, **ReSym**, that harnesses LLMs to effectively recover variable and data structure symbols from stripped binaries.
- **Divide** the difficult symbol recovery problem into two manageable sub-problems.
- Develop a rigorous reasoning component to **aggregate** and cross-check local results, enabling the recovery of **comprehensive data structures**.
- ReSym **outperforms** state-of-the-art techniques, DIRTY and OSPREY.
- Build a **large-scale public dataset** of C/C++ projects containing binaries annotated with corresponding symbols, together with the automatic annotation pipeline, to facilitate future research.