Beyond Classification: Inferring Function Names in Stripped Binaries via Domain Adapted LLMs

Linxi Jiang Xin Jin Zhiqiang Lin

NDSS 2025

Binary function name prediction is extremely useful

```
1   void FUN_00001b20 (void) {
2     ...
3     if ((DAT_0020e389 == 0)  (DAT_0020e38c == -1)) {
4       iVar1 = rand();
5       DAT_0020e389 = (char)iVar1 + (char)(iVar1 / 0xff);
6       ...
7       DAT_0020e38c = '\0';
8       while (((((((DAT_0020e389 == 0
9                   (DAT_0020e389 == 10))
10                  ...
11                  (0xdf < DAT_0020e389)))))
12      {
13        iVar1 = rand();
14        DAT_0020e389 = (char)iVar1 + (char)(iVar1 / 0xff);
15        ...
16  }
```

Binary function name prediction is extremely useful

```
1   void get_random_ip (void) {
2     ...
3     if ((DAT_0020e389 == 0)  (DAT_0020e38c == -1)) {
4       iVar1 = rand();
5       DAT_0020e389 = (char)iVar1 + (char)(iVar1 / 0xff);
6       ...
7       DAT_0020e38c = '\0';
8       while (((((((DAT_0020e389 == 0
9                   (DAT_0020e389 == 10))
10                  ...
11                  (0xdf < DAT_0020e389)))))
12      {
13        iVar1 = rand();
14        DAT_0020e389 = (char)iVar1 + (char)(iVar1 / 0xff);
15        ...
16  }
```

Challenge I: Generalizable Task

Challenges

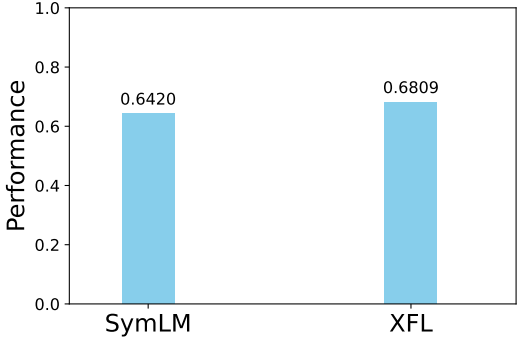
- Generalizable Task

Challenge I: Generalizable Task

- AsmDepictor [KBCK23], SymLM [JLYL23], and XFL [PEDK23]: **Classification Task**

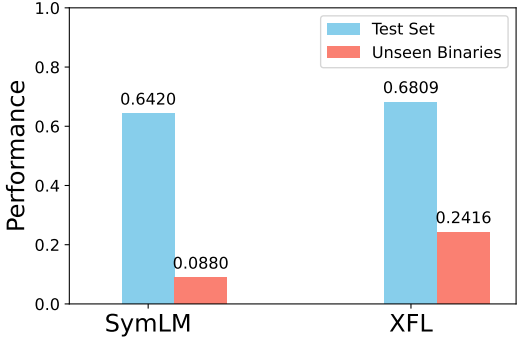
Challenge I: Generalizable Task

- AsmDepictor [KBCK23], SymLM [JLYL23], and XFL [PEDK23]: **Classification Task**



Challenge I: Generalizable Task

- AsmDepictor [KBCK23], SymLM [JLYL23], and XFL [PEDK23]: **Classification Task**

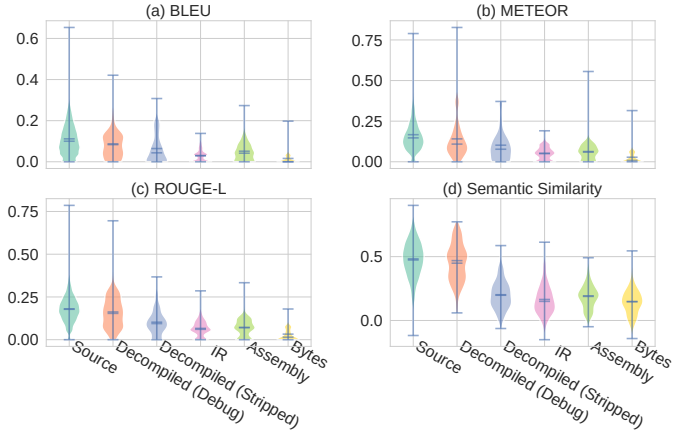


Challenge II: Knowledge Gaps

Challenges

- Generalizable Task
- **Knowledge Gaps**

Challenge II: Knowledge Gaps



(Source: [JLYL23])

Challenge III: Data Leakage

Challenges

- Generalizable Task
- Knowledge Gaps
- **Data Leakage**

Challenge III: Data Leakage

```
1 void [MASK](vbuf *vb)
2 {
3   vb->avail = 0;
4   vb->used = vb->avail;
5   vb->buf = (byte *)0x0;
6   return;
7 }
```

Challenge III: Data Leakage

```
1 void [MASK](vbuf *vb)
2 {
3   vb->avail = 0;
4   vb->used = vb->avail;
5   vb->buf = (byte *)0x0;
6   return;
7 }
```

```
1 void adns__vbuf_init(vbuf *vb)
2 {
3   vb->avail = 0;
4   vb->used = vb->avail;
5   vb->buf = (byte *)0x0;
6   return;
7 }
```

Challenge III: Data Leakage

```
1 void [MASK](vbuf *vb)
2 {
3   vb->avail = 0;
4   vb->used = vb->avail;
5   vb->buf = (byte *)0x0;
6   return;
7 }
```

```
1 void adns__vbuf_init(vbuf *vb)
2 {
3   vb->avail = 0;
4   vb->used = vb->avail;
5   vb->buf = (byte *)0x0;
6   return;
7 }
```

Challenge III: Data Leakage

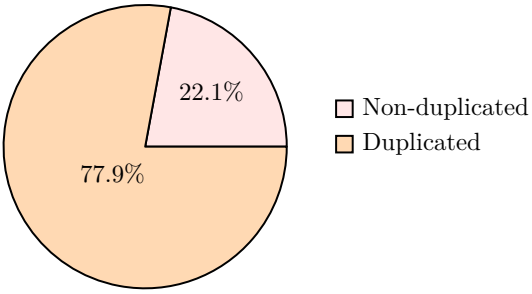
```
...
mov_esi_ 0x40c78c ,
mov_rdi_rax,
call_ 0x2e24 ,
mov_qword_ptr_[rbp-0x10]_rax,
cmp_qword_ptr_[rbp-0x10]_0,
...
```

(a) AsmDepictor's Training Sample

```
...
mov_esi_ 0x41ce4c ,
mov_rdi_rax,
call_ 0xfffffffffffb75b ,
mov_qword_ptr_[rbp-0x10]_rax,
cmp_qword_ptr_[rbp-0x10]_0,
...
```

(b) AsmDepictor's Test Sample

Challenge III: Data Leakage



Challenge VI: Resource-expensive Training

Challenges

- Generalizable Task
- Knowledge Gaps
- Data Leakage
- **Resource-expensive Training**

Challenge VI: Resource-expensive Training

	GPU Type	GPU Power consumption	GPU-hours	Total power consumption	Carbon emitted (tCO_2eq)
OPT-175B	A100-80GB	400W	809,472	356 MWh	137
BLOOM-175B	A100-80GB	400W	1,082,880	475 MWh	183
LLaMA-7B	A100-80GB	400W	82,432	36 MWh	14
LLaMA-13B	A100-80GB	400W	135,168	59 MWh	23
LLaMA-33B	A100-80GB	400W	530,432	233 MWh	90
LLaMA-65B	A100-80GB	400W	1,022,362	449 MWh	173

(Source: [TLI+23])

Insights

Key Insights

- **S1**: Function Name Inference with Generative LLMs. $\Rightarrow$ **C1**

Insights

Key Insights

- **S1**: Function Name Inference with Generative LLMs. $\Rightarrow$ **C1**
- **S2**: Domain-adaptive Learning with Function Summaries. $\Rightarrow$ **C2**

Insights

Key Insights

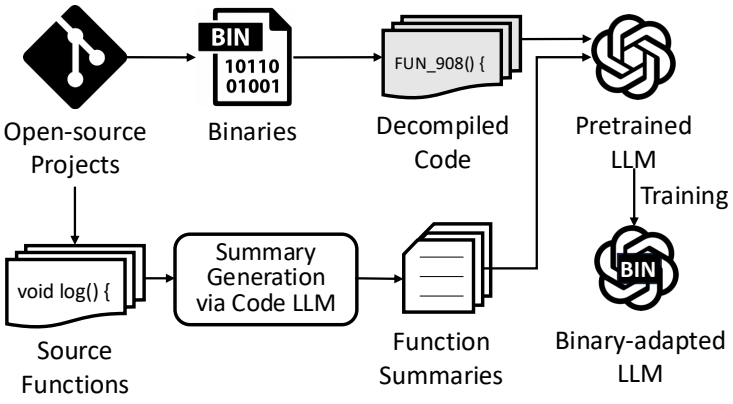
- **S1**: Function Name Inference with Generative LLMs. $\Rightarrow$ **C1**
- **S2**: Domain-adaptive Learning with Function Summaries. $\Rightarrow$ **C2**
- **S3**: Data Deduplication and Leakage Mitigation. $\Rightarrow$ **C3**

Insights

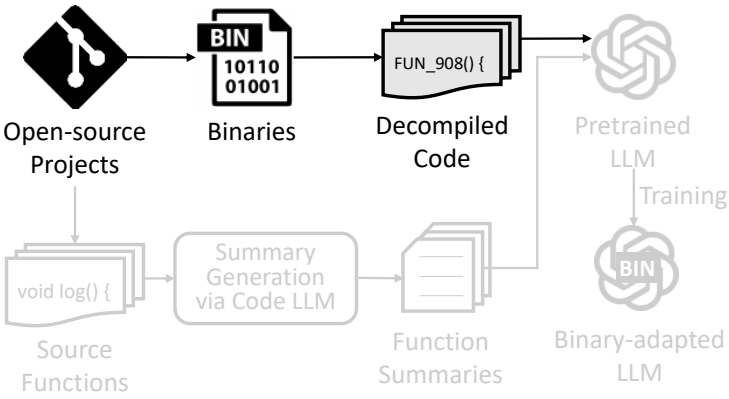
Key Insights

- **S1**: Function Name Inference with Generative LLMs. $\Rightarrow$ **C1**
- **S2**: Domain-adaptive Learning with Function Summaries. $\Rightarrow$ **C2**
- **S3**: Data Deduplication and Leakage Mitigation. $\Rightarrow$ **C3**
- **S4**: Parameter-efficient Learning. $\Rightarrow$ **C4**

System Overview: SymGen



Workflow Step 1: Data Processing



Workflow Step 1: Data Processing

- Decompilation


```
1 void FUN_0000c42e(long param_1)
2
3 {
4   *(undefined4 *)(param_1 + 0xb8) = 0;
5   FUN_0000c663(param_1);
6   FUN_0000c663(param_1 + 8);
7   return;
8 }
```

Workflow Step 1: Data Processing

- Function Name Substitution

```
1 void FUN_0000c42e(long param_1)
2
3 {
4   *(undefined4 *)(param_1 + 0xb8) = 0;
5   FUN_0000c663(param_1);
6   FUN_0000c663(param_1 + 8);
7   return;
8 }
```

```
1 void [MASK](long param_1)
2
3 {
4   *(undefined4 *)(param_1 + 0xb8) = 0;
5   FUN_0000c663(param_1);
6   FUN_0000c663(param_1 + 8);
7   return;
8 }
```

Workflow Step 1: Data Processing

- Data Deduplication

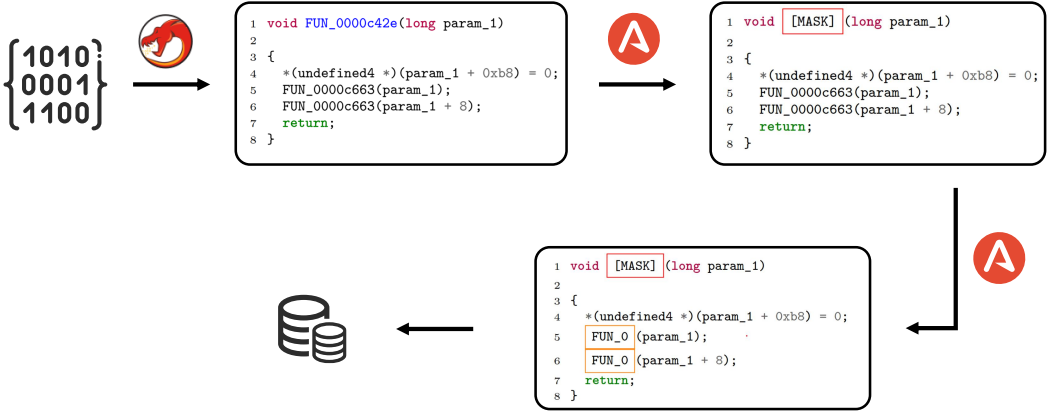
```
1 void FUN_0000c42e(long param_1)
2
3 {
4   *(undefined4 *)(param_1 + 0xb8) = 0;
5   FUN_0000c663(param_1);
6   FUN_0000c663(param_1 + 8);
7   return;
8 }
```

```
1 void [MASK] (long param_1)
2
3 {
4   *(undefined4 *)(param_1 + 0xb8) = 0;
5   FUN_0000c663(param_1);
6   FUN_0000c663(param_1 + 8);
7   return;
8 }
```

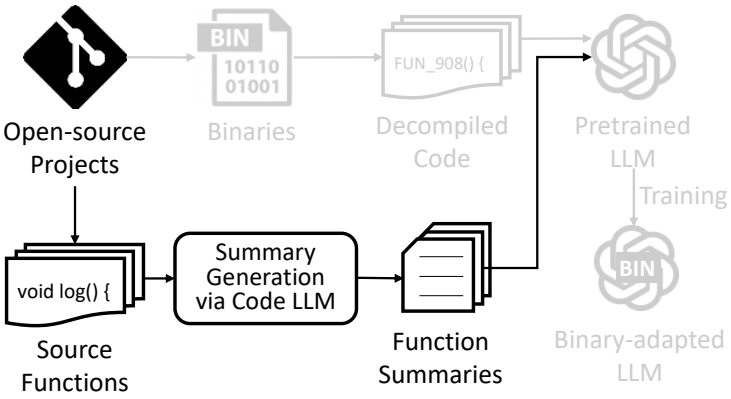
```
1 void [MASK] (long param_1)
2
3 {
4   *(undefined4 *)(param_1 + 0xb8) = 0;
5   FUN_0 (param_1);
6   FUN_0 (param_1 + 8);
7   return;
8 }
```

Workflow Step 1: Data Processing

- Data Deduplication



Workflow Step 2: Summary Generation



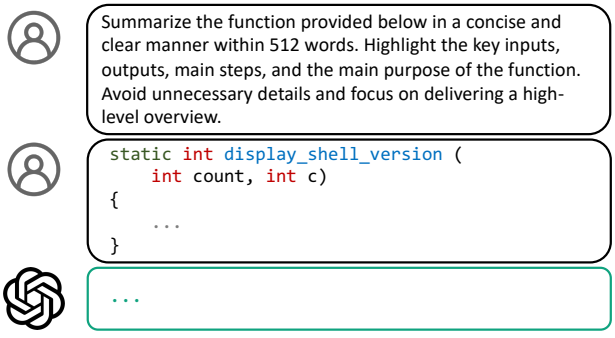
Workflow Step 2: Summary Generation

- We collect two types of summaries:
  1. High-quality comments

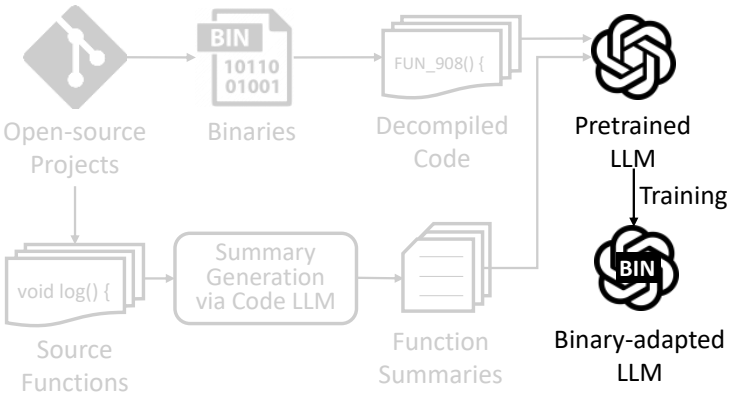
```
1   /*
2   FUNCTION
3       bfd_set_error_program_name
4
5   SYNOPSIS
6       void bfd_set_error_program_name (const char *);
7
8   DESCRIPTION
9       Set the program name to use when printing a BFD error.  This
10      is printed before the error message followed by a colon and
11      space.  The string must not be changed after it is passed to
12      this function.
13  */
14
15  void
16  bfd_set_error_program_name (const char *name)
```

Workflow Step 2: Summary Generation

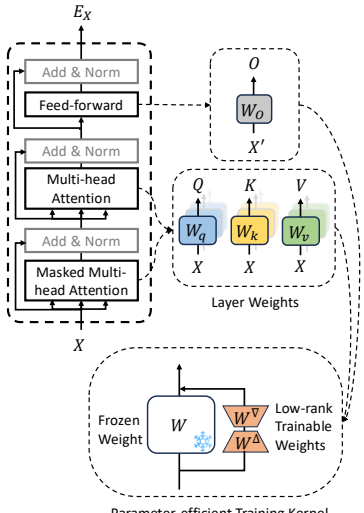
- We collect two types of summaries:
  1. High-quality comments
  2. Summary generated by code LLMs



Workflow Step 3: Model Training



Workflow Step 3: Model Training



Workflow Step 3: Model Training

Suppose you are an expert in software reverse engineering. Here is a piece of decompiled code. Please infer the code semantics and tell me the original function name from the contents of the function to replace [MASK]. Now, the decompiled code is as follows:

```
void [MASK] (undefined4 *param_1 …)
{
    *param_1 = param_2;
    return;
}
```

The predicted function name is ……

Evaluation Setup

1. **Dataset**:
   - 33 open-sourced C projects.
   - 4 computer architectures (x86-32, x86-64, ARM, and MIPS) and 4 optimization levels (O0, O1, O2, and O3).
   - 3 obfuscation options (bcf, cff, sub).

Evaluation Setup

1. **Dataset**:
   - 33 open-sourced C projects.
   - 4 computer architectures (x86-32, x86-64, ARM, and MIPS) and 4 optimization levels (O0, O1, O2, and O3).
   - 3 obfuscation options (bcf, cff, sub).
2. **Baselines**: AsmDepictor [KBCK23], SymLM [JLYL23], and XFL [PEDK23].

Evaluation Setup

1. **Dataset**:
   - 33 open-sourced C projects.
   - 4 computer architectures (x86-32, x86-64, ARM, and MIPS) and 4 optimization levels (O0, O1, O2, and O3).
   - 3 obfuscation options (bcf, cff, sub).
2. **Baselines**: AsmDepictor [KBCK23], SymLM [JLYL23], and XFL [PEDK23].
3. **Metric**: Precision, Recall, and F1-score

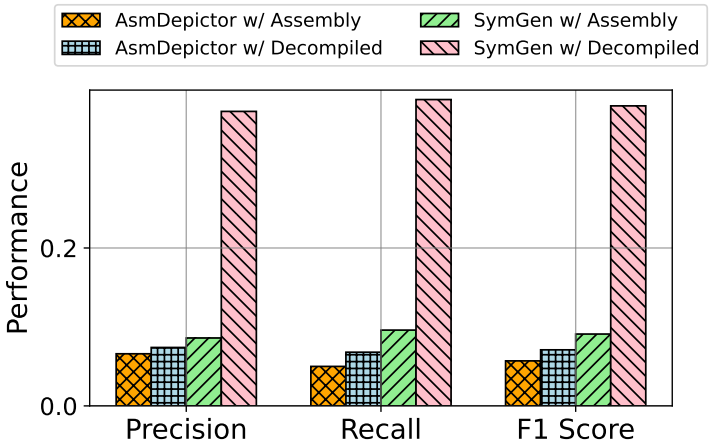
Baseline Comparison

Arch	Model	Precision	Recall	F1 Score
			O0	
x86-64	SymGen	0.346	0.356	0.351
	SymLM	0.099	0.100	0.100
	AsmDepictor	0.059	0.043	0.050
	XFL *	0.169	0.140	0.153
	SymGen v.s. AsmDepictor	(↑486.4%)	(↑727.9%)	(↑602.0%)
	SymGen v.s. SymLM	(↑249.5%)	(↑256.0%)	(↑251.0%)
	SymGen v.s. XFL	(↑104.7%)	(↑154.3%)	(↑129.4%)

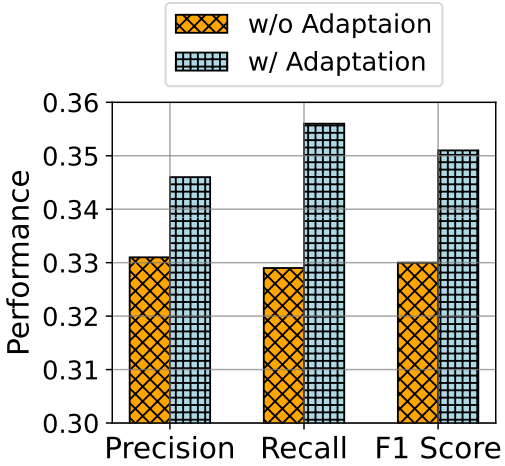
Baseline Comparison

Arch	Model	Precision	Recall	F1 Score
			O0	
x86-64	SymGen	0.346	0.356	0.351
	SymLM	0.099	0.100	0.100
	AsmDepictor	0.059	0.043	0.050
	XFL *	0.169	0.140	0.153
	SymGen v.s. AsmDepictor	(↑486.4%)	(↑727.9%)	(↑602.0%)
	SymGen v.s. SymLM	(↑249.5%)	(↑256.0%)	(↑251.0%)
	SymGen v.s. XFL	(↑104.7%)	(↑154.3%)	(↑129.4%)

Baseline Comparison



Ablation Study



Ablation Study

```
1  undefined8 FUN_0001be92(undefined8 param_1,
2      undefined8 param_2,undefined8 param_3,
3      char *param_4)
4  {
5    size_t sVar1;
6    undefined8 uVar2;
7
8    sVar1 = strlen(param_4);
9    if (sVar1 == 1) {
10     FUN_0002345c(param_1,*param_4);
11     uVar2 = 0;
12   }
13   else {
14     uVar2 = 3;
15   }
16   return uVar2;
17 }
```

Ablation Study

```
1  undefined8 FUN_0001be92(undefined8 param_1,
2      undefined8 param_2,undefined8 param_3,
3      char *param_4)
4  {
5    size_t sVar1;
6    undefined8 uVar2;
7
8    sVar1 = strlen(param_4);
9    if (sVar1 == 1) {
10     FUN_0002345c(param_1,*param_4);
11     uVar2 = 0;
12   }
13   else {
14     uVar2 = 3;
15   }
16   return uVar2;
17 }
```

```
1  undefined8 FUN_0001be92(undefined8 param_1,
2      undefined8 param_2,undefined8 param_3,
3      char *param_4)
4  {
5    size_t sVar1;
6    undefined8 uVar2;
7
8    sVar1 = strlen(param_4);
9    if (sVar1 == 1) {
10     FUN_0002345c(param_1,*param_4);
11   }
12   else {
13     FUN_0002345c(param_1, param_2 + param_3);
14     uVar2 = param_2 + param_3;
15   }
16   return uVar2;
17 }
```

Ablation Study

- CodeBLEU score: 0.098

```
if (sVar1 == 1) {
  FUN_0002345c(param_1,*param_4);
  uVar2 = 0;
}
else {
  uVar2 = 3;
}
```

```
if (sVar1 == 1) {
  FUN_0002345c(param_1,*param_4);
}
else {
  FUN_0002345c(param_1, param_2 + param_3);
  uVar2 = param_2 + param_3;
}
```

Effectiveness on Real-world Binaries

Table: Performance of SymGen, SymLM, and XFL on Obfuscated Binaries

Obfuscation	SymGen	SymLM	XFL
w/o obfuscation	0.302	0.071	0.139
bcfobf	0.288 (-4.6%)	0.063 (-11.3%)	0.044 (-215.9%)
cffobf	0.271 (-10.3%)	0.056 (-21.1%)	0.034 (-308.8%)
subobf	0.303 (+0.3%)	0.062 (-12.7%)	0.039 (-256.4%)

Takeaway

SymGen

- Propose a novel framework based on LLMs for binary function name inference.
- Present an advanced domain adaptation approach and a parameter-efficient learning strategy for tuning the pretrained LLMs to binary semantic modeling.
- Outperforms existing solutions in function name prediction, setting new generalizability benchmarks.

Takeaway

SymGen

- Propose a novel framework based on LLMs for binary function name inference.
- Present an advanced domain adaptation approach and a parameter-efficient learning strategy for tuning the pretrained LLMs to binary semantic modeling.
- Outperforms existing solutions in function name prediction, setting new generalizability benchmarks.

The source code is available at https://github.com/OSUSecLab/SymGen.

References I

Xin Jin, Jonathan Larson, Weiwei Yang, and Zhiqiang Lin, *Binary code summarization: Benchmarking chatgpt/gpt-4 and other large language models*, arXiv preprint arXiv:2312.09601 (2023).

Hyunjin Kim, Jinyeong Bak, Kyunghyun Cho, and Hyungjoon Koo, *A transformer-based function symbol name inference model from an assembly language for binary reversing*, Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, 2023, pp. 951–965.

James Patrick-Evans, Moritz Dannehl, and Johannes Kinder, *Xfl: Naming functions in binaries with extreme multi-label learning*, 2023 IEEE Symposium on Security and Privacy (SP), IEEE, 2023, pp. 2375–2390.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al., *Llama: Open and efficient foundation language models*, arXiv preprint arXiv:2302.13971 (2023).

Exploration of In-dataset Deduplication

Table: Performance of SymGen and Baselines on Training Set with and without Duplicates

Model	Training Set	Precision	Recall	F1 Score
AsmDepictor	w/ duplicates	0.019	0.015	0.017
	w/o duplicates	0.059	0.043	0.050
SymLM	w/ duplicates	0.117	0.068	0.086
	w/o duplicates	0.099	0.100	0.100
XFL	w/ duplicates	0.159	0.144	0.151
	w/o duplicates	0.169	0.140	0.153
SymGen	w/ duplicates	0.353	0.329	0.341
	w/o duplicates	0.346	0.356	0.351

Exploration of In-dataset Deduplication

Table: Performance of SymGen and Baselines on Test Set with and without Duplicates

Model	Test Set	Precision	Recall	F1 Score
AsmDepictor	w/ duplicates	0.057	0.042	0.048
	w/o duplicates	0.059	0.043	0.050
SymLM	w/ duplicates	0.103	0.093	0.097
	w/o duplicates	0.099	0.100	0.100
XFL	w/ duplicates	0.152	0.138	0.144
	w/o duplicates	0.169	0.140	0.153
SymGen	w/ duplicates	0.348	0.369	0.358
	w/o duplicates	0.346	0.356	0.351

Exploration of In-dataset Deduplication

Conclusion

1. Compared with data leakage, in-dataset duplication does not significantly affect performance.
2. The duplication cause the model to overfit on the training set and memorize these repeated data rather than learning generalizable features