SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings

Xin Jin[1] Kexin Pei[2] Jun Yeon Won[1] Zhiqiang Lin[1]

[1]The Ohio State University

[2]Columbia University

ACM CCS 2022

Predicting binary function names is extremely useful

- Function names: summary of function behavior and semantics.

Predicting binary function names is extremely useful

- Function names: summary of function behavior and semantics.

```
void FUN_001092f3(byte *param_1) {
    byte *local_48;
    ulong local_40;
    ...
    if (*local_48 == 10) {
        local_48 = local_48 + 1;
    }
    else if (*local_48 == 0x3a) {
        DAT_00119470 = '\0';
        bVar3 = *local_48;
    }
    ...
}
```

Predicting binary function names is extremely useful

- Function names: summary of function behavior and semantics.

```c
void FUN_001092f3(byte *param_1) {
    byte *local_48;
    ulong local_40;
    ...
    if (*local_48 == 10) {
        local_48 = local_48 + 1;
    }
    else if (*local_48 == 0x3a) {
        DAT_00119470 = '\0';
        bVar3 = *local_48;
    }
    ...
}
```

Function Name Prediction →

```c
void DNS_flood(byte *param_1) {
    byte *local_48;
    ulong local_40;
    ...
    if (*local_48 == 10) {
        local_48 = local_48 + 1;
    }
    else if (*local_48 == 0x3a) {
        DAT_00119470 = '\0';
        bVar3 = *local_48;
    }
    ...
}
```

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.

Missing Semantics

Names of **identifiers** and **function parameters** use the same words as function names [LWN21].

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics.** Very limited semantic information.
2. **Binary Variation.** Semantically similar code appearing differently.

openssl-1.0.1 libcrypto.a

```
<CMS_add0_cert>:
...
mov  eax, dword ptr  [rbp-0x2c]
add  eax, 1
mov  dword  ptr  [rbp-0x2c], eax
....
```

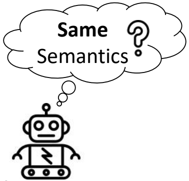
Add Obfuscation

openssl-1.0.1 libcrypto.a (obfuscated)

```
<CMS_add0_cert>:
...
xor  ecx, ecx
mov  eax, dword ptr  [rbp-0x2c]
sub  ecx, 1
sub  eax, ecx
mov  dword  ptr  [rbp-0x2c], eax
....
```

eax + 1

eax - (-1)



Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.
3. **Noisy Function Names**. Different developers naming functions differently.

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.
3. **Noisy Function Names**. Different developers naming functions differently.

Reasons

- Synonyms and abbreviations are ubiquitous in function names.
- Even single letters can be meaningful when probably used [BGOF17].
- Probability (two developers select same names for the same function) = **6.9%** [FMN+20].

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.
3. **Noisy Function Names**. Different developers naming functions differently.
4. **OOV Issues**. Out-of-vocabulary words widely used.

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.
3. **Noisy Function Names**. Different developers naming functions differently.
4. **OOV Issues**. Out-of-vocabulary words widely used.

OOV Words

	Category	**Ratio**	**Examples**
1	Abbreviation concatenation	29.9%	statinfo, streq
2	Clean word concatenation	22.3%	sharefile, startpoints
3	Misspelling	14.6%	anewer, tac, sb
4	Clean word	12.1%	dependent, specifer
5	Abbreviation	7.0%	utils, pred
6	Inflection	9.6%	addresses, using
7	Digits in word	4.5%	add32, merge2

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.
3. **Noisy Function Names**. Different developers naming functions differently.
4. **OOV Issues**. Out-of-vocabulary words widely used.
5. **Comprehensive semantic modeling**. Semantics preserved in calling context.

Predicting binary function names is very challenging

Challenges

1. **Missing Semantics**. Very limited semantic information.
2. **Binary Variation**. Semantically similar code appearing differently.
3. **Noisy Function Names**. Different developers naming functions differently.
4. **OOV Issues**. Out-of-vocabulary words widely used.
5. **Comprehensive semantic modeling**. Semantics preserved in calling context.

```
YY_BUFFER_STATE yy_scan_string (char *yystr) {
  size_t _yybytes_len;
  YY_BUFFER_STATE pyVar1;
  _yybytes_len = strlen(yystr);
  pyVar1 = yy_scan_bytes (yystr,_yybytes_len);
  return pyVar1;
}
```

Prior Works and Limitations

- Limited features, e.g.,
  1. **handcrafted features**: Debin [HIT+18] and Punstrip [PECK20].
  2. **partial function semantics**: NERO [DAY20] and NFRE [GCXZ21].

Prior Works and Limitations

- Limited features, e.g.,
  1. **handcrafted features**: Debin [HIT+18] and Punstrip [PECK20].
  2. **partial function semantics**: NERO [DAY20] and NFRE [GCXZ21].

```
1   xor   ecx, ecx
2   mov   eax, dword ptr [rbp-0x2c]
3   sub   eax, ecx
4   mov   dword ptr [rbp-0x2c], eax
```

Normalization $\Longrightarrow$

```
1   INST_1534
2   INST_5741
3   INST_7745
4   INST_2573
```

Preprocessing Step of NFRE

Key Observations and Insights

Predicting function names

- Program semantics is manifested in execution behavior.

Key Observations and Insights

openssl-1.0.1 libcrypto.a

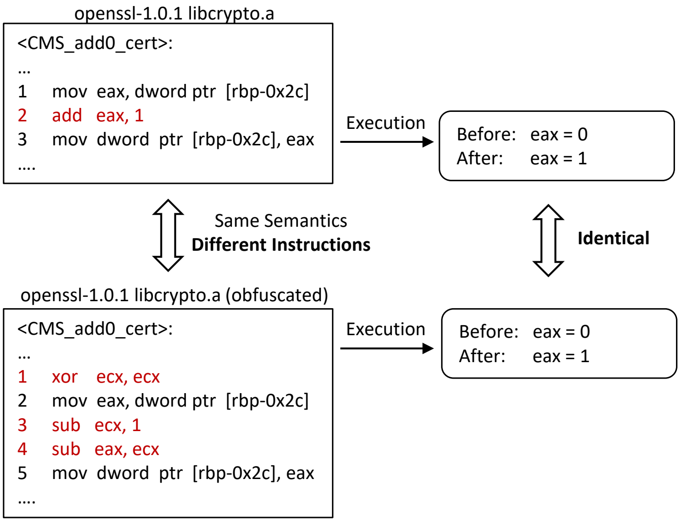
```
<CMS_add0_cert>:
...
mov  eax, dword ptr  [rbp-0x2c]
add   eax, 1
mov  dword  ptr  [rbp-0x2c], eax
....
```

Same Semantics
Different Instructions

openssl-1.0.1 libcrypto.a (obfuscated)

```
<CMS_add0_cert>:
...
xor    ecx, ecx
mov  eax, dword ptr  [rbp-0x2c]
sub   ecx, 1
sub   eax, ecx
mov  dword  ptr  [rbp-0x2c], eax
....
```

Key Observations and Insights



Key Observations and Insights

Predicting function names

- Program semantics is manifested in execution behavior.
- Learning semantics requires understanding both function instructions and calling context.

Key Observations and Insights

Predicting function names

- Program semantics is manifested in execution behavior.
- Learning semantics requires understanding both function instructions and calling context.
- Function name semantic similarity needs to be measured.

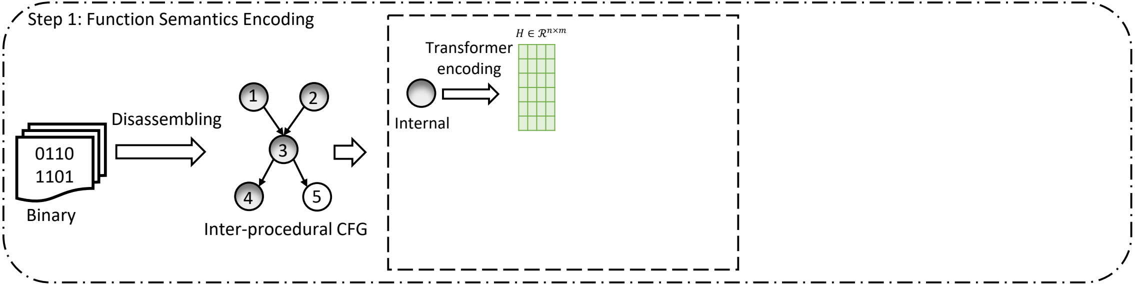
Key Observations and Insights

Predicting function names

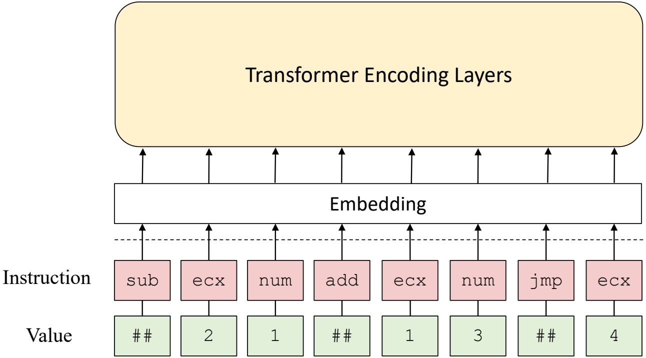
- Program semantics is manifested in execution behavior. $\Rightarrow$ C1 and C2
- Learning semantics requires understanding both function instructions and calling context. $\Rightarrow$ C5
- Function name semantic similarity needs to be measured. $\Rightarrow$ C3 and C4

System Workflow: Step 1

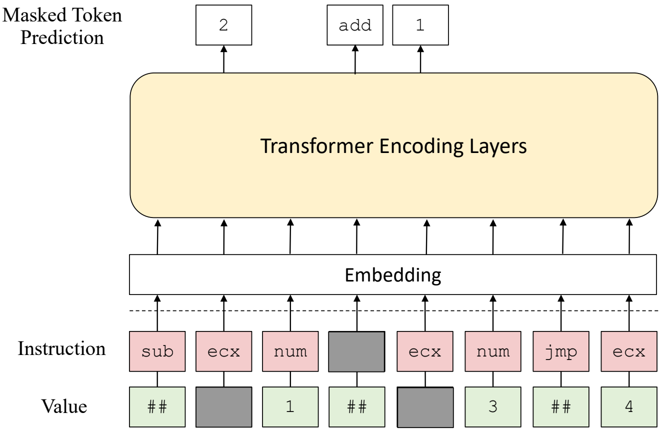
System Workflow: Step 1



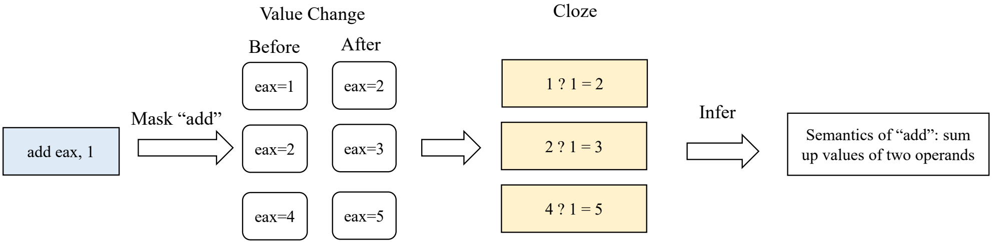
System Workflow: Step 1



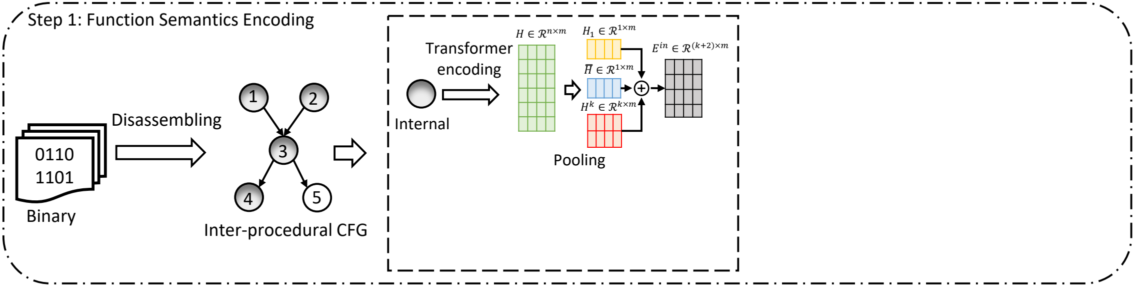
System Workflow: Step 1



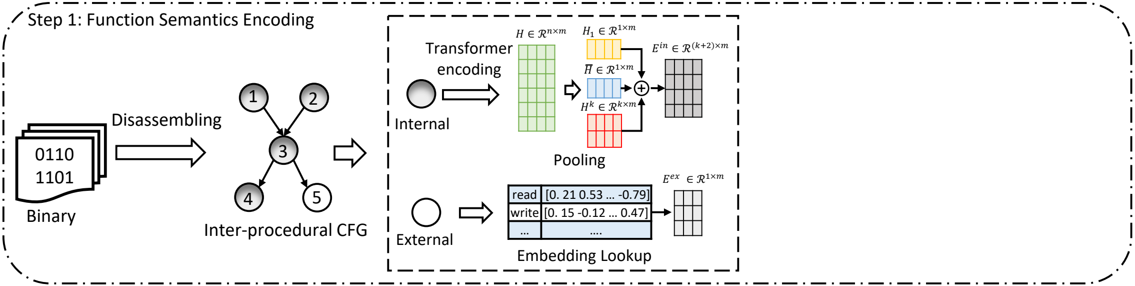
System Workflow: Step 1



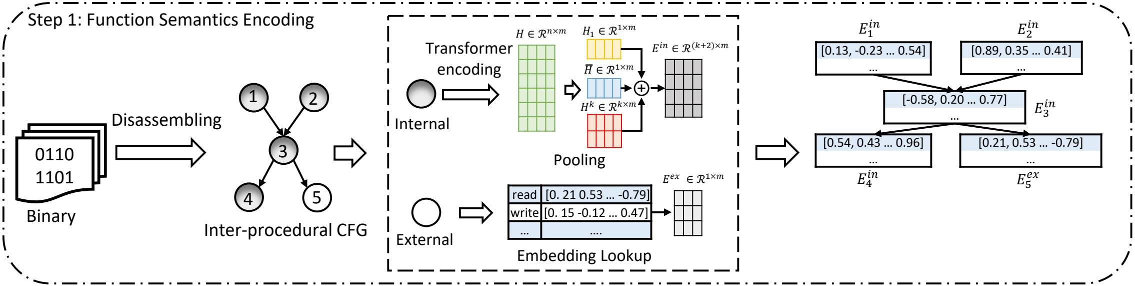
System Workflow: Step 1



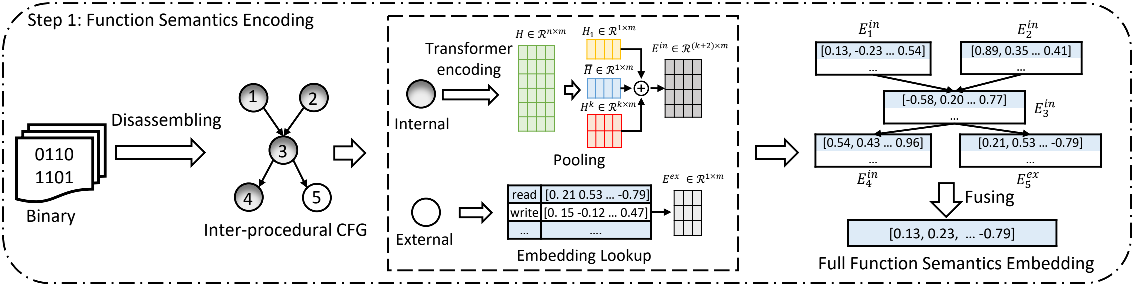
System Workflow: Step 1



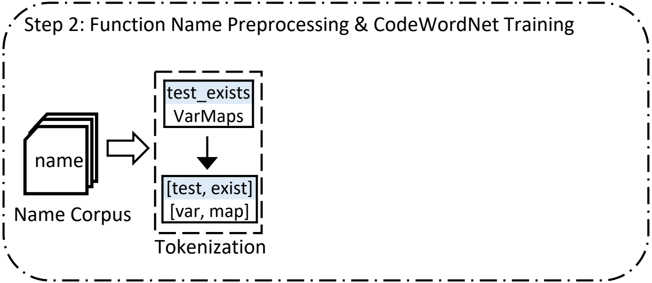
System Workflow: Step 1



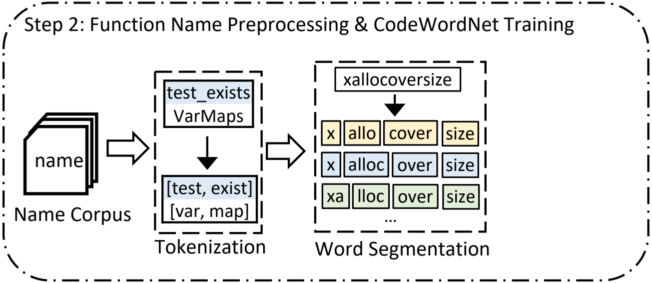
System Workflow: Step 1



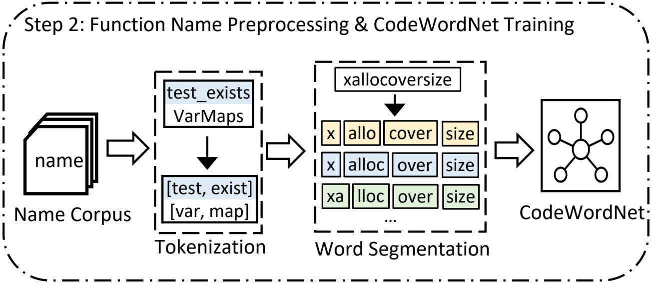
System Workflow: Step 2



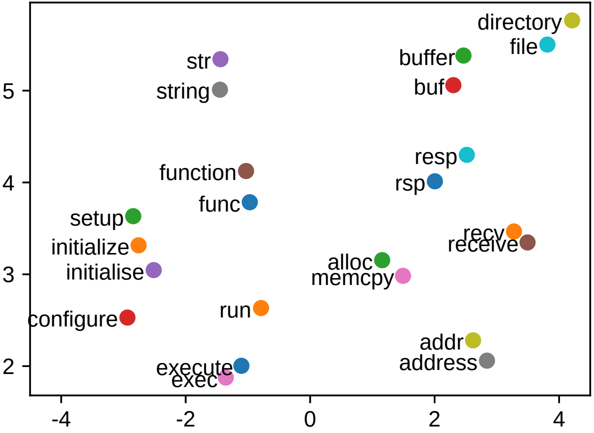
System Workflow: Step 2



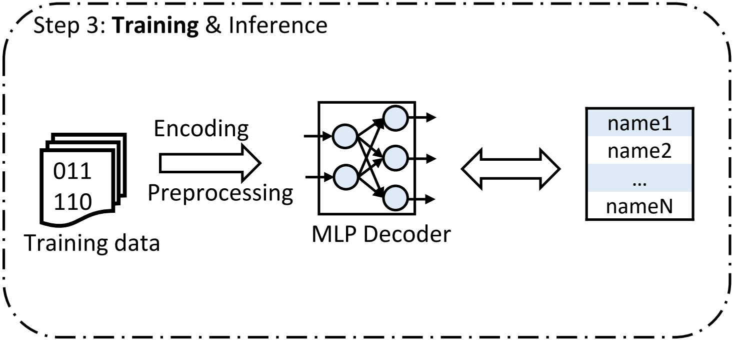
System Workflow: Step 2



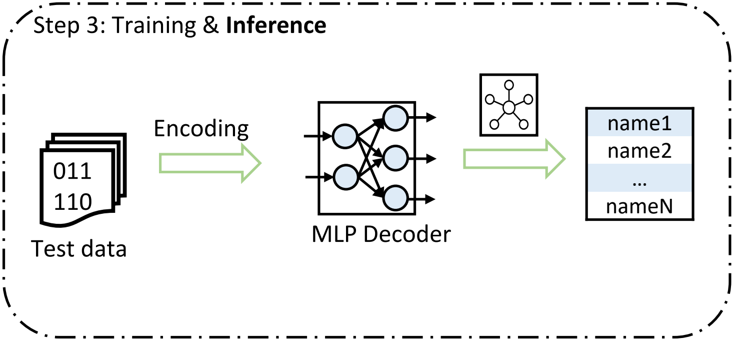
System Workflow: Step 2



System Workflow: Step 3



System Workflow: Step 3



Evaluation Setup

1. **Dataset**: 1,431,169 functions

Evaluation Setup

1. **Dataset**: 1,431,169 functions
   - 27 open-source projects.
   - 4 architectures: x86, x64, ARM, and MIPS.
   - 4 optimization levels: O0, O1, O2, and O3 (GCC-7.5).
   - 4 obfuscation options: bcf, cff, sub, and split (LLVM obfuscator).

Evaluation Setup

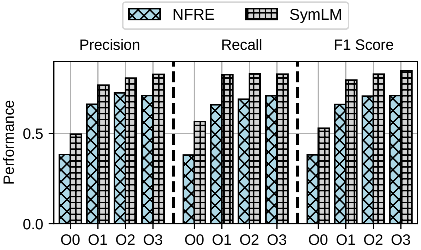
1. **Dataset**: 1,431,169 functions
   - 27 open-source projects.
   - 4 architectures: x86, x64, ARM, and MIPS.
   - 4 optimization levels: O0, O1, O2, and O3 (GCC-7.5).
   - 4 obfuscation options: bcf, cff, sub, and split (LLVM obfuscator).
2. Baselines: NERO [DAY20] and NFRE [GCXZ21].

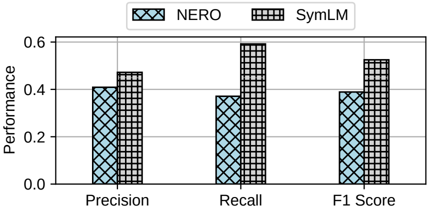
Evaluation Setup

1. **Dataset**: 1,431,169 functions
   - 27 open-source projects.
   - 4 architectures: x86, x64, ARM, and MIPS.
   - 4 optimization levels: O0, O1, O2, and O3 (GCC-7.5).
   - 4 obfuscation options: bcf, cff, sub, and split (LLVM obfuscator).
2. Baselines: NERO [DAY20] and NFRE [GCXZ21].
3. Metrics: precision, recall, and F1-score.

Baseline Comparison

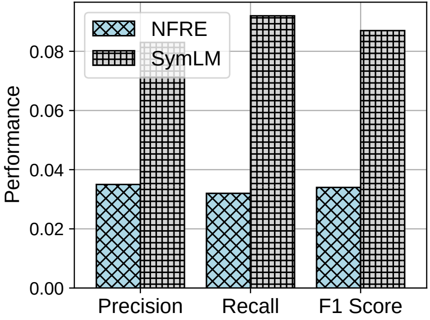
- SymLM outperforms the state-of-the-art works (up to 35% improvement on F1 score).

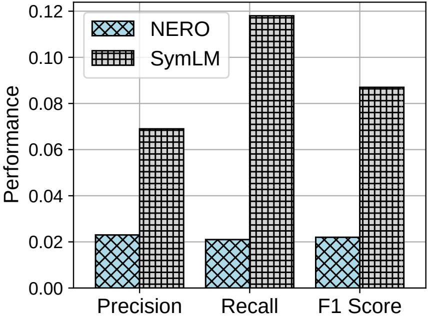




Generalizability

- SymLM is more generalizable to unseen binary functions (295.5% better F1 score).





Use Case

- SymLM successfully infers function semantics of IoT firmware image [Gat].

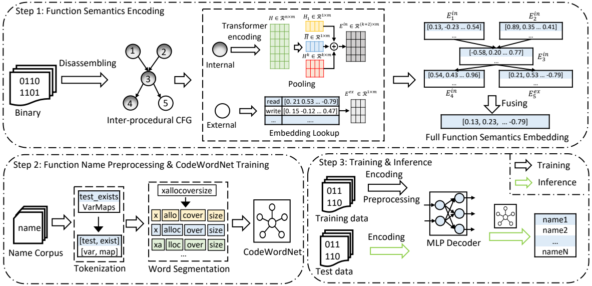
```c
uint32_t analogRead(uint32_t ulPin){
  ...
  if (pin == NC) uVar3 = 0;
  else {
    uVar2 = adc_read_value(pin);
    uVar3 = (uint32_t)uVar2;
    if (uVar4 != 0xc) {
      if ((uint) uVar4 < 0xc)
        return (uint)(uVar2 >> (0xcU - uVar4 & 0xff));
      return uVar3 << (uVar4 - 0xcU & 0xff);
    }
  }
  return uVar3;
}
```

Ground Truth

```c
uint32_t read(uint32_t ulPin){
  ...
  if (pin == NC) uVar3 = 0;
  else {
    uVar2 = read_value(pin);
    uVar3 = (uint32_t)uVar2;
    if (uVar4 != 0xc) {
      if ((uint) uVar4 < 0xc)
        return (uint)(uVar2 >> (0xcU - uVar4 & 0xff));
      return uVar3 << (uVar4 - 0xcU & 0xff);
    }
  }
  return uVar3;
}
```

Prediction

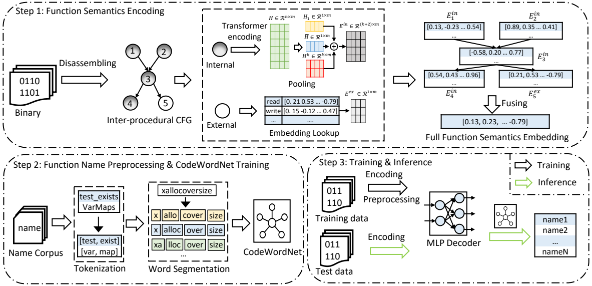
Takeaway



SymLM

- A novel neural architecture that generates execution-aware context-sensitive code embeddings.
- Effective modules, function name preprocessing and CodeWordNet, to calculate function name similarity.
- Advancing the state-of-the-art and practical use cases.

Takeaway



SymLM

- A novel neural architecture that generates execution-aware context-sensitive code embeddings.
- Effective modules, function name preprocessing and CodeWordNet, to calculate function name similarity.
- Advancing the state-of-the-art and practical use cases.

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

References I

- Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G Feitelson, *Meaningful identifier names: the case of single-letter variables*, 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), IEEE, 2017, pp. 45–54.
- Kevin Burk, Fabio Pagani, Christopher Kruegel, and Giovanni Vigna, *Decomperson: How humans decompile and what we can learn from it*, 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 2765–2782.
- Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang, *Selectivetaint: Efficient data flow tracking with static binary rewriting*, 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, August 2021.
- Yaniv David, Uri Alon, and Eran Yahav, *Neural reverse engineering of stripped binaries using augmented control flow graphs*, Proceedings of the ACM on Programming Languages **4** (2020), no. OOPSLA, 1–28.
- Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk, *Binsec/rel: Efficient relational symbolic execution for constant-time at binary-level*, 2020 IEEE Symposium on Security and Privacy (SP), IEEE, 2020, pp. 1021–1038.
- Dror Feitelson, Ayelet Mizrahi, Nofar Noy, Aviad Ben Shabat, Or Eliyahu, and Roy Sheffer, *How developers choose names*, IEEE Transactions on Software Engineering (2020).
- *Gateway*, `https://github.com/RiS3-Lab/p2im-real_firmware/blob/master/binary/Gateway`, Accessed: 2022-04-26.
- Han Gao, Shaoyin Cheng, Yinxing Xue, and Weiming Zhang, *A lightweight framework for function name reassignment based on large-scale stripped binaries*, Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 607–619.

References II

- Masoud Ghaffarinia and Kevin W Hamlen, *Binary control-flow trimming*, Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1009–1022.
- Jingxuan He, Pesho Ivanov, Petar Tsankov, Veselin Raychev, and Martin Vechev, *Debin: Predicting debug information in stripped binaries*, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1667–1680.
- Yi Li, Shaohua Wang, and Tien N Nguyen, *A context-based automated approach for method name consistency checking and suggestion*, 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 574–586.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, *Distributed representations of words and phrases and their compositionality*, Advances in neural information processing systems **26** (2013).
- James Patrick-Evans, Lorenzo Cavallaro, and Johannes Kinder, *Probabilistic naming of functions in stripped binaries*, Annual Computer Security Applications Conference, 2020, pp. 373–385.