Meaningful Variable Names for Decompiled Code: A Machine Translation Approach

Alan Jaffe, **Jeremy Lacomis**, Edward J. Schwartz*, Claire Le Goues, and Bogdan Vasilescu

STRUDEL

Carnegie Mellon University

* Software Engineering Institute

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```

```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```

```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```

```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

Decompiled C Code:

```
cp = buf;
(void)asxTab(level + 1);
for (n = asnContents(asn, buf, 512); n > 0; n--) {
  printf(" %02X ", *(cp++));
}
```

```
v14 = &v15;
asxTab(a2 + 1);
for (v13 = asnContents(a1, &v15, 512LL); v13 > 0; --v13) {
  v9 = (unsignedchar*)(v14++);
  printf(" %02X ", *v9);
}
```

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```

```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

Decompiled C Code:

```
cp = buf;
(void)asxTab(level + 1);
for (n = asnContents(asn, buf, 512); n > 0; n--) {
  printf(" %02X ", *(cp++));
}
```

```
v14 = &v15;
asxTab(a2 + 1);
for (v13 = asnContents(a1, &v15, 512LL); v13 > 0; --v13) {
  v9 = (unsignedchar*)(v14++);
  printf(" %02X ", *v9);
}
```

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```

```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

- Software is “natural” [Hindle et al., 2011].

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```



```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

- Software is “natural” [Hindle et al., 2011].
- Use large corpora + machine learning to predict better identifier names.
  - Corpora are easy to generate!

Problem: Obfuscated Variable Names in Code

Minified JavaScript:

```
function callback(error, response, body) {
 if (!error && response.statusCode == 200) {
   var info = JSON.parse(body);
…
```



```
function callback(o, s, a) {
 if (!o && s.statusCode == 200) {
   var c = JSON.parse(a);
…
```

- Software is “natural” [Hindle et al., 2011].

- Use large corpora + machine learning to predict better identifier names.
  - Corpora are easy to generate!

- Bavishi et al., Context2Name, 2017
- Vasilescu et al., JSNaughty, 2017
- Raychev et al., JSNice, 2015

Problem: Obfuscated Variable Names in Code

Can we use similar strategies for decompiled code?

Decompiled C Code:

```
cp = buf;
(void)asxTab(level + 1);
for (n = asnContents(asn, buf, 512); n > 0; n--) {
  printf(" %02X ", *(cp++));
}
```

```
v14 = &v15;
asxTab(a2 + 1);
for (v13 = asnContents(a1, &v15, 512LL); v13 > 0; --v13) {
  v9 = (unsignedchar*)(v14++);
  printf(" %02X ", *v9);
}
```

Statistical Machine Translation (SMT)

- Noisy channel model

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Go do some research!

Va faire de la recherche!

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Go do some research!

Va faire de la recherche!

$$argmax_e \, p(e \mid f)$$

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Go do some research!

Va faire de la recherche!

$$argmax_e\, p(e \mid f) = argmax_e \frac{p(f \mid e)\, p(e)}{p(f)}$$
$$= argmax_e\, p(f \mid e)\, p(e)$$

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Go do some research!

Va faire de la recherche!

$$argmax_e\, p(e \mid f) = argmax_e \frac{p(f \mid e)\, p(e)}{p(f)}$$

$$= argmax_e\, p(f \mid e)\, p(e)$$

Translation Model: Probability that *f* is a translation of *e*

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Go do some research!

Va faire de la recherche!

$$argmax_e\, p(e \mid f) = argmax_e \frac{p(f \mid e)\, p(e)}{p(f)}$$
$$= argmax_e\, p(f \mid e)\, p(e)$$

Language Model: "Fluency" of *e*

Statistical Machine Translation (SMT)

- Noisy channel model
- English → French:

Go do some research!

Va faire de la recherche!

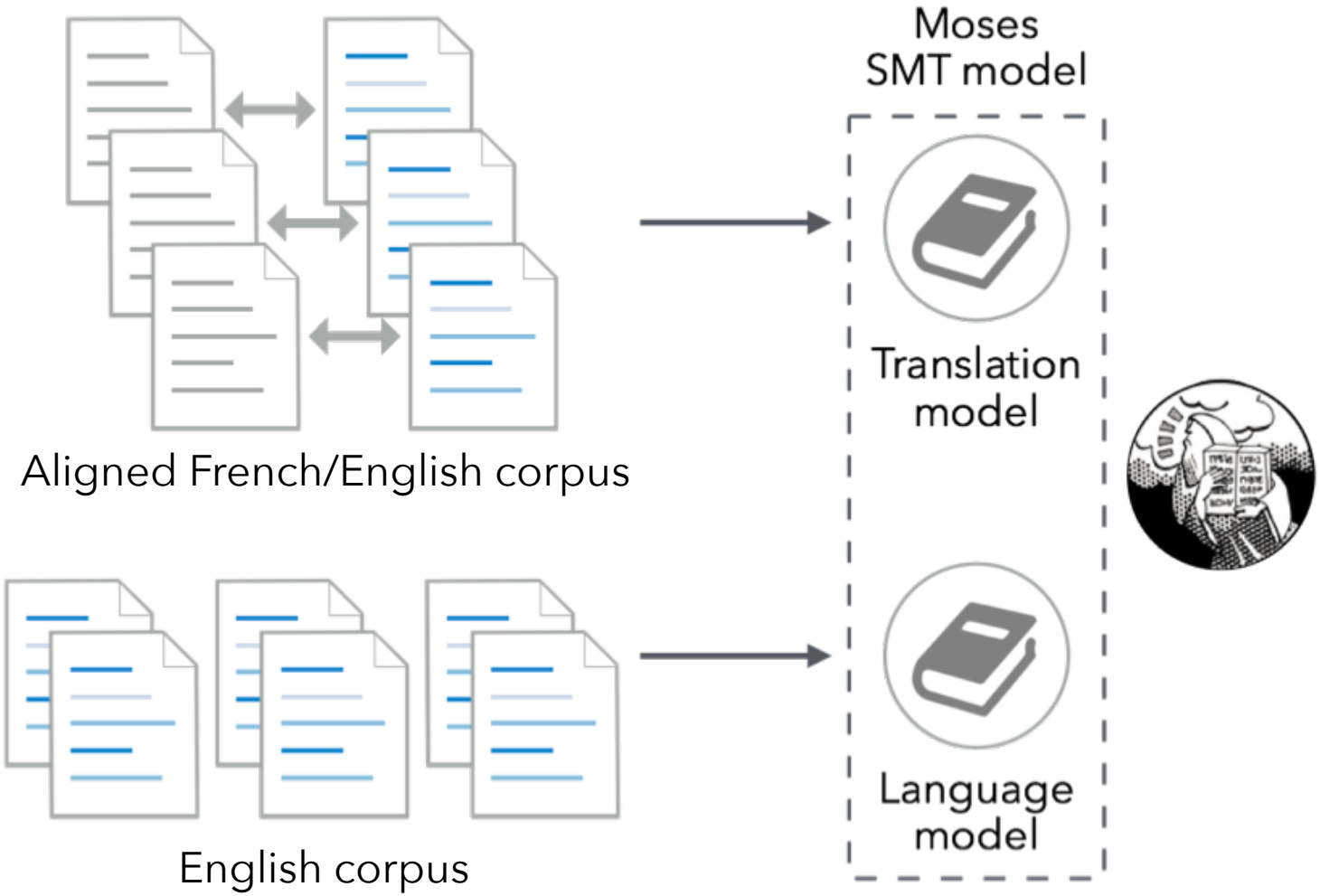
$$argmax_e\, p(e \mid f) = argmax_e \frac{p(f \mid e)\, p(e)}{p(f)}$$
$$= argmax_e\, p(f \mid e)\, p(e)$$

$p(f \mid e)$: Translation Model

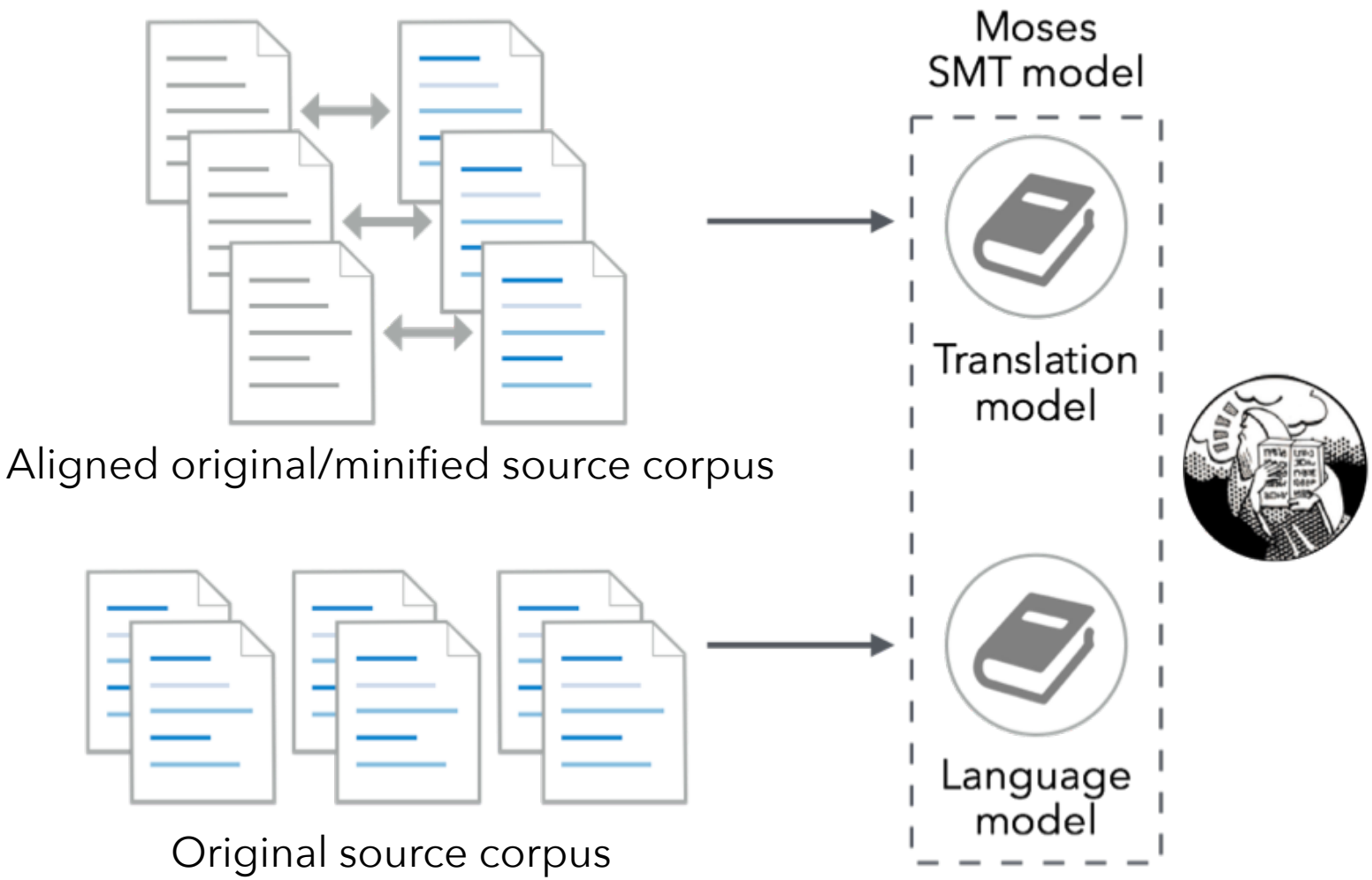
$p(e)$: Language Model

MOSES SMT:

SMT Model for Natural Language



SMT Model for Minified JavaScript



Problem: Obfuscated Identifiers in Code

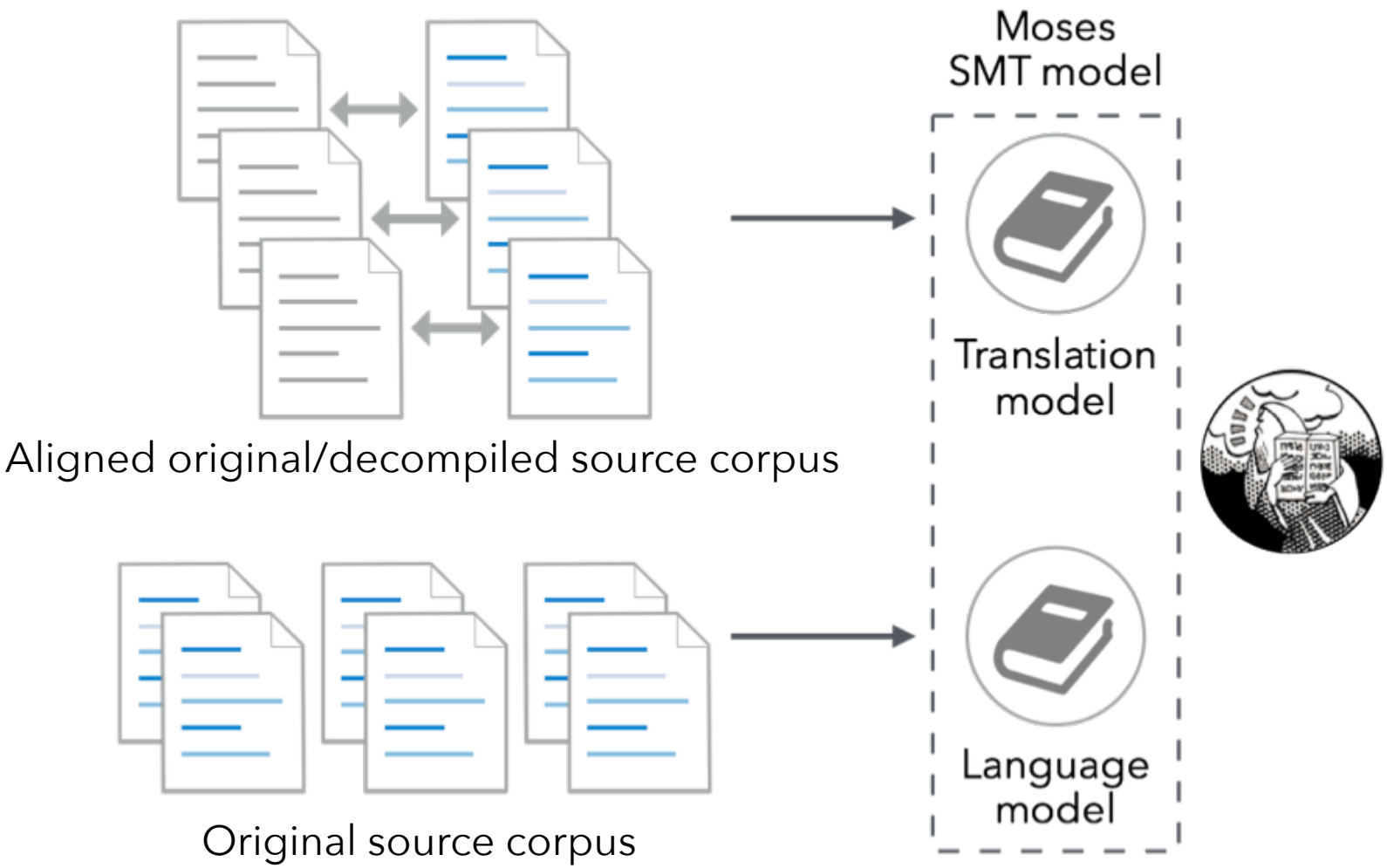
Can we use SMT for decompiled code?

Decompiled C Code:

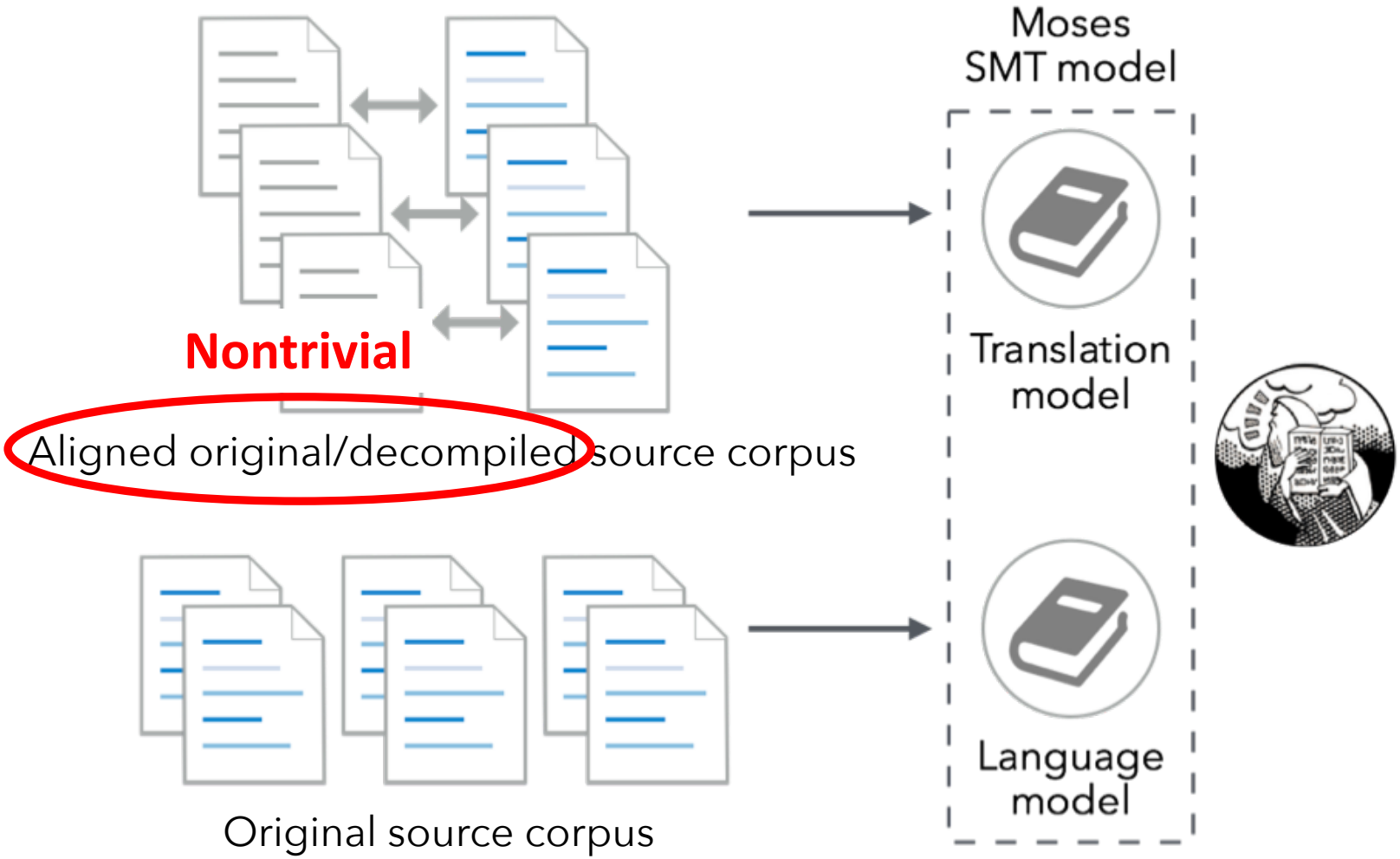
```
cp = buf;
(void)asxTab(level + 1);
for (n = asnContents(asn, buf, 512); n > 0; n--) {
  printf(" %02X ", *(cp++));
}
```

```
v14 = &v15;
asxTab(a2 + 1);
for (v13 = asnContents(a1, &v15, 512LL); v13 > 0; --v13) {
  v9 = (unsignedchar*)(v14++);
  printf(" %02X ", *v9);
}
```

SMT Model for Decompiled Code?



SMT Model for Decompiled Code?



Difficulty: Decompilation Changes Structure

Original Source

```c
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

Difficulty: Decompilation Changes Structure

9 Lines Original Source

```c
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

8 Lines Decompiled Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Different line count.

Difficulty: Decompilation Changes Structure

Original Source

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Different line count.
- Different numbers of variables.

Difficulty: Decompilation Changes Structure

Original Source

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Different line count.
- Different numbers of variables.
- Different types of loops.

Decompiled Code Corpus Generation

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

Decompiled Code Corpus Generation

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Original Code

Decompiled Code Corpus Generation

Decompiled Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

```c
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Original Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

Decompiled Code Corpus Generation

Decompiled Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

```c
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Original Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

Decompiled Code Corpus Generation

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Original Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int __;
 for (__ = 0; __ < 10; ++__)
   printf("%d\n", __);
 return v1;
}
```

Decompiled Code Corpus Generation

Decompiled Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

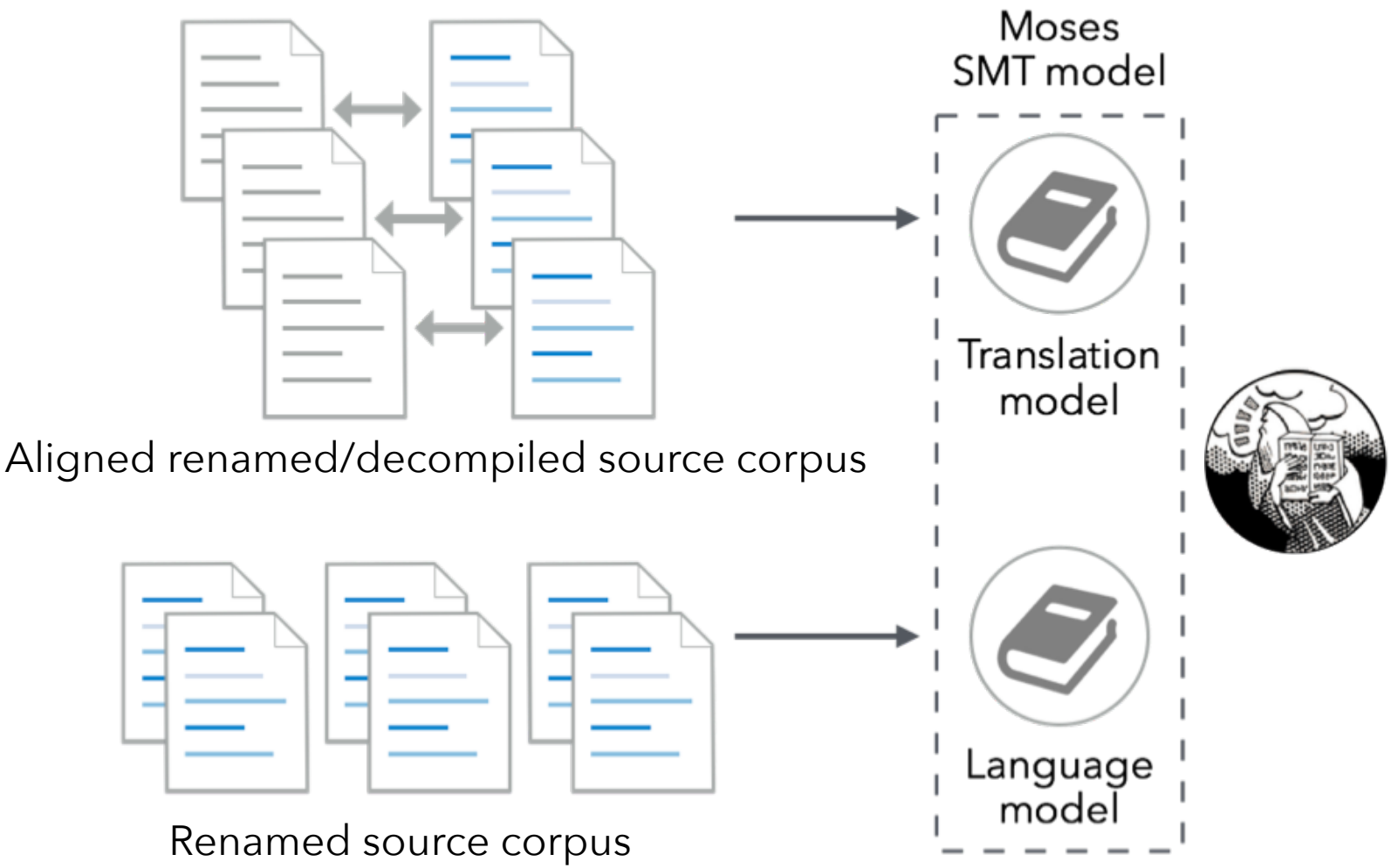
```c
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Original Code

```c
#include <stdio.h>
int main() {
 int v1 = 0;
 int cur;
 for (cur = 0; cur < 10; ++cur)
   printf("%d\n", cur);
 return v1;
}
```

Renamed Decompiled Code

Better SMT Model for Decompiled Code



Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Not used as the return value.

Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Not used as the return value.
- Used inside of a loop.

Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Not used as the return value.
- Used inside of a loop.
- Used in a function call.

Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int v2;
 for (v2 = 0; v2 < 10; ++v2)
   printf("%d\n", v2);
 return v1;
}
```

- Not used as the return value.
- Used inside of a loop.
- Used in a function call.
- Same operations.

Choosing Renamings

Original Code

```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int __;
 for (__ = 0; __ < 10; ++__)
   printf("%d\n", __);
 return v1;
}
```

- Not used as the return value.
- Used inside of a loop.
- Used in a function call.
- Same operations.

Choosing Renamings

Original Code

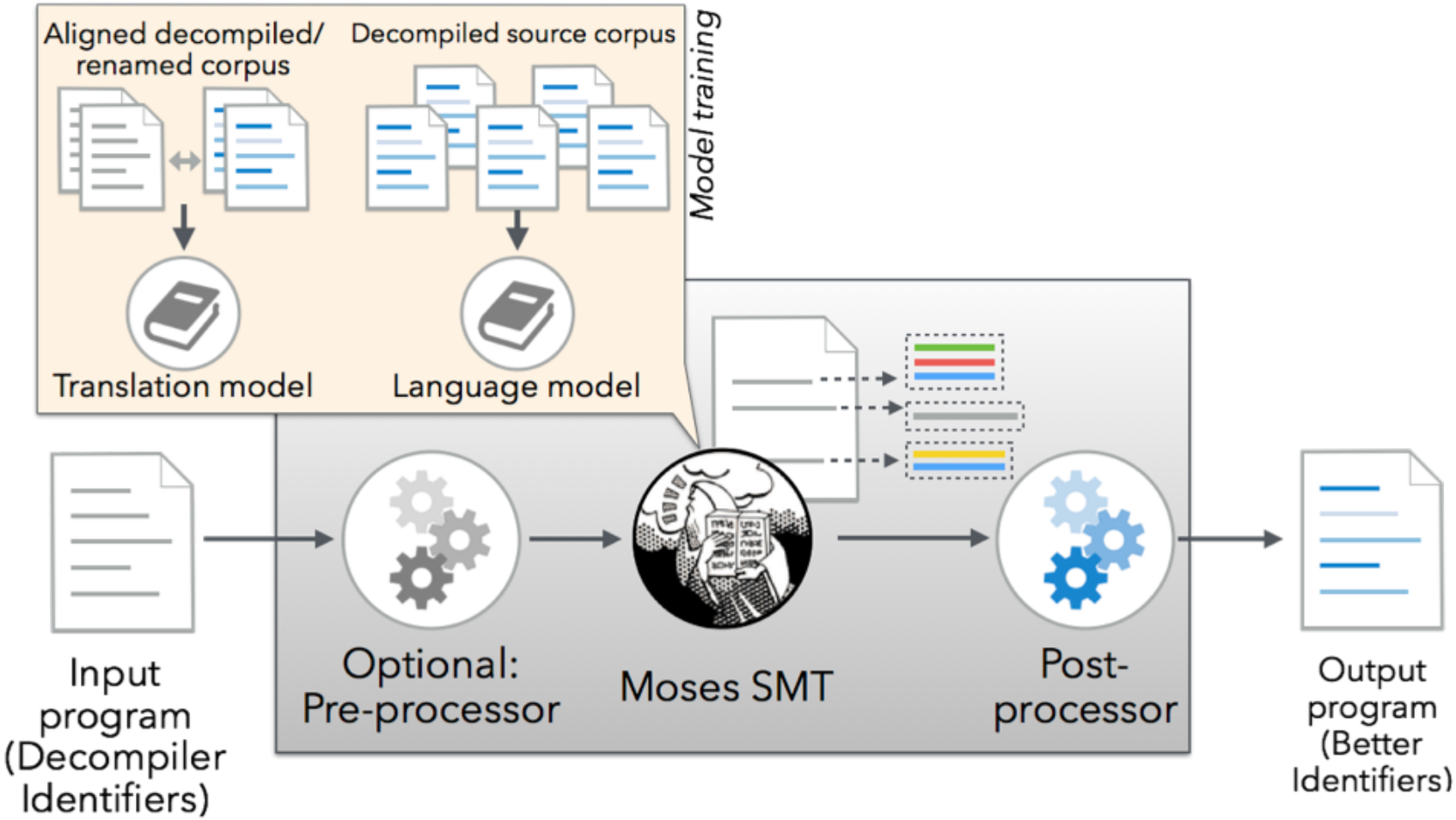
```
#include <stdio.h>
int main() {
 int cur = 0;
 while (cur <= 9) {
   printf("%d\n", cur);
   ++cur;
 }
 return 0;
}
```

Decompiled Code

```
#include <stdio.h>
int main() {
 int v1 = 0;
 int cur;
 for (cur = 0; cur < 10; ++cur)
   printf("%d\n", cur);
 return v1;
}
```

- Not used as the return value.
- Used inside of a loop.
- Used in a function call.
- Same operations.

System Architecture



Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Decompiled

```
my_rc base2_string(base2_handle a1, char* a2,
                   size_t a3)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Decompiled

```
my_rc base2_string(base2_handle a1, char* a2,
                   size_t a3)
```

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Exact

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Approx

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Not a match

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

- 12.7% Exact
- 16.2% Exact + Approx

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Not a match

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

- 12.7% Exact
- 16.2% Exact + Approx

Results and Evaluation

Original

```
my_rc base2_string(base2_handle base2_h, char* buffer,
                   size_t buffer_size)
```

Decompiled

```
my_rc base2_string(base2_handle a1, char* a2,
                   size_t a3)
```

Renamed Decompiled

```
my_rc base2_string(base2_handle base2_h, char* buf,
                   size_t len)
```

- 12.7% Exact
- 16.2% Exact + Approx

Preliminary Investigation: Human Study

- Presented users with short snippets (<50 lines) of decompiled code, asked to perform various maintenance tasks, graded and timed:

Preliminary Investigation: Human Study

- Presented users with short snippets (<50 lines) of decompiled code, asked to perform various maintenance tasks, graded and timed:

```
1 int x = 1;
2 int y = 0;
3 while (x <= 5) {
4   y += 2;
5   x += 1;
6 }
7 printf("%d", y);
```

- What is the value of the variable y on line 7?

Preliminary Investigation: Human Study

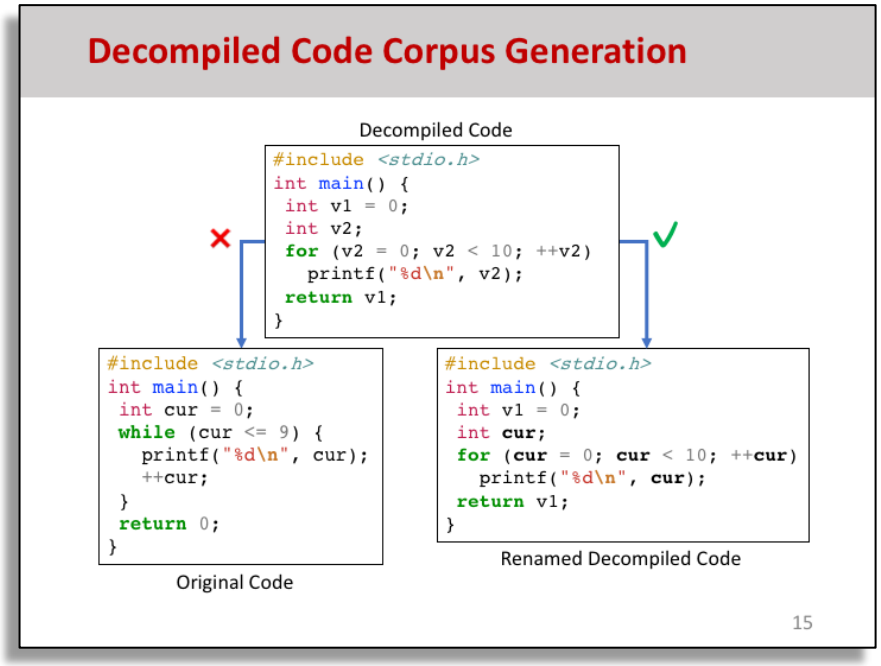
- Presented users with short snippets (<50 lines) of decompiled code, asked to perform various maintenance tasks, graded and timed:

```
1 int x = 1;
2 int y = 0;
3 while (x <= 5) {
4   y += 2;
5   x += 1;
6 }
7 printf("%d", y);
```

- What is the value of the variable y on line 7?

- For correct answers, the time to answer using our renamings was statistically significantly lower than when using the decompiler names.

Conclusion



- Questions?
- Suggestions?