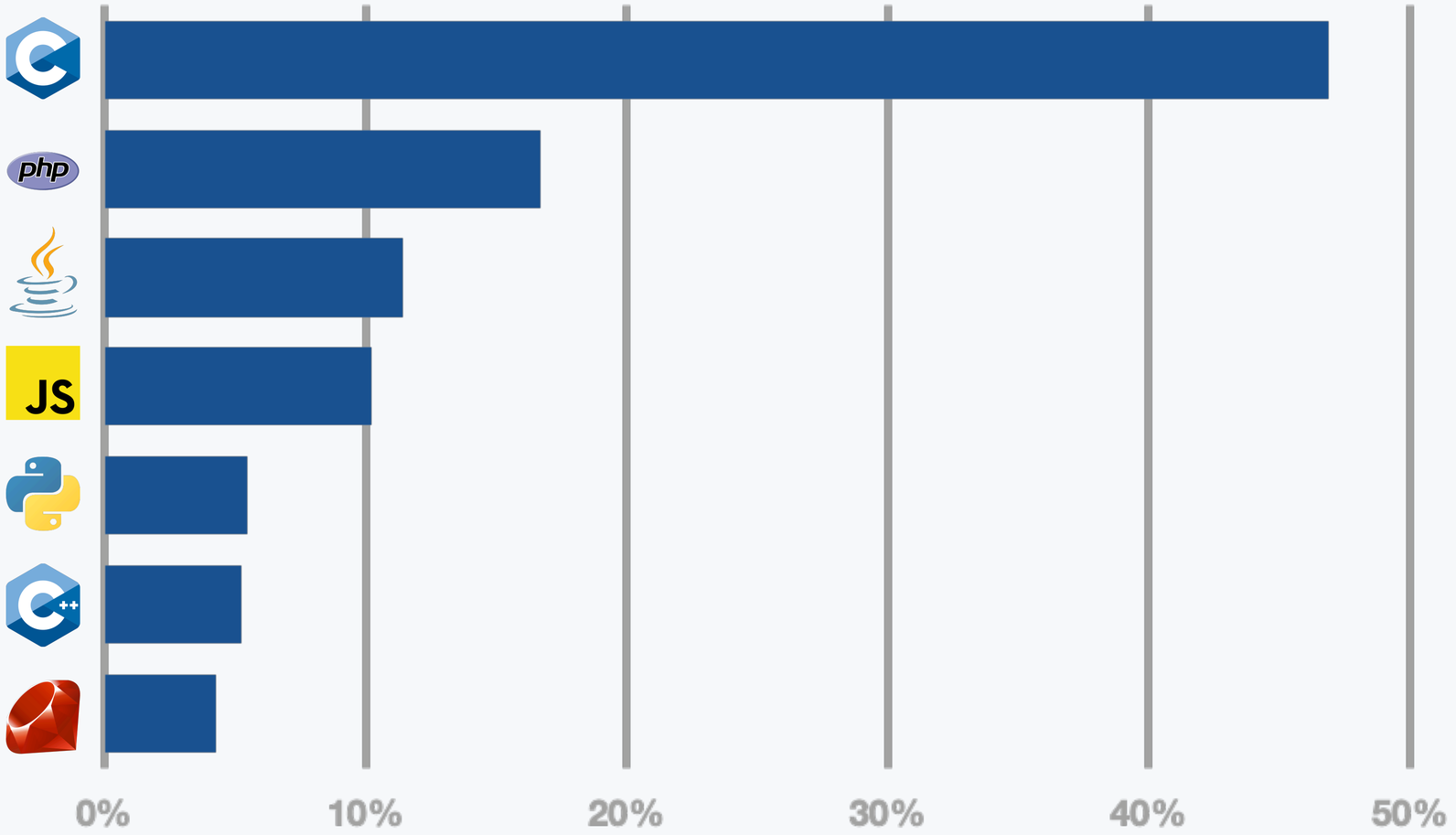
소프트웨어재난연구센터 2023 여름정기워크숍 주요연구발표

Simcrat:
A Signature-Modernizing C-to-Rust Translator using a Large Language Model

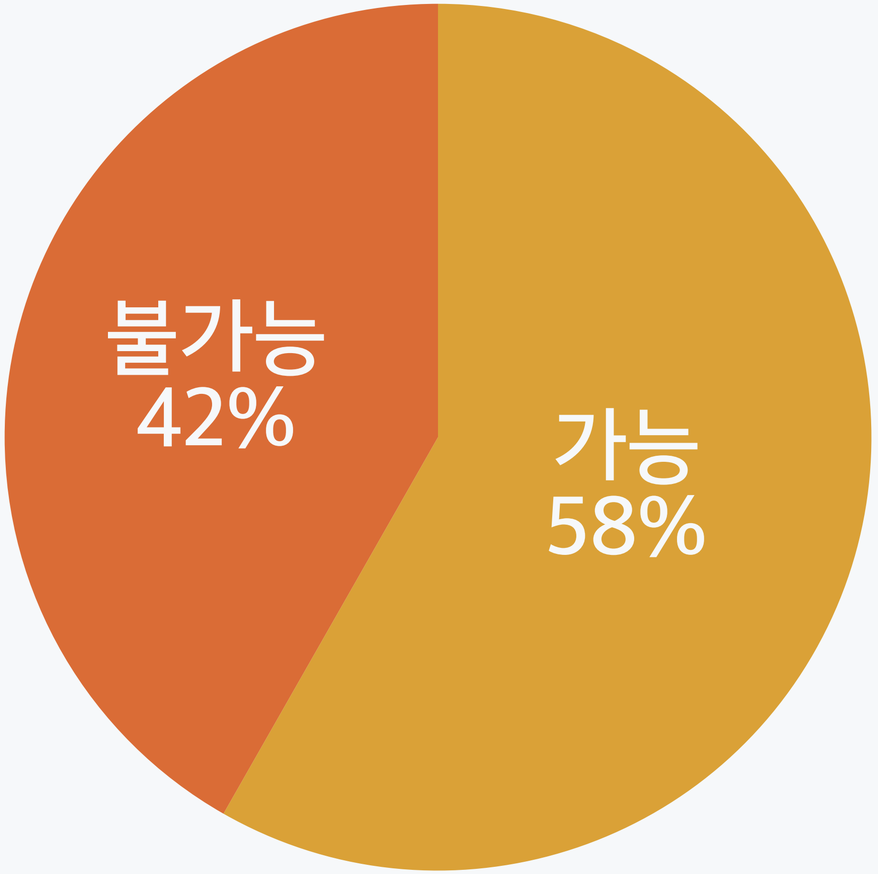
홍재민, 류석영 (KAIST)

보고된 보안 취약점의 언어별 비중

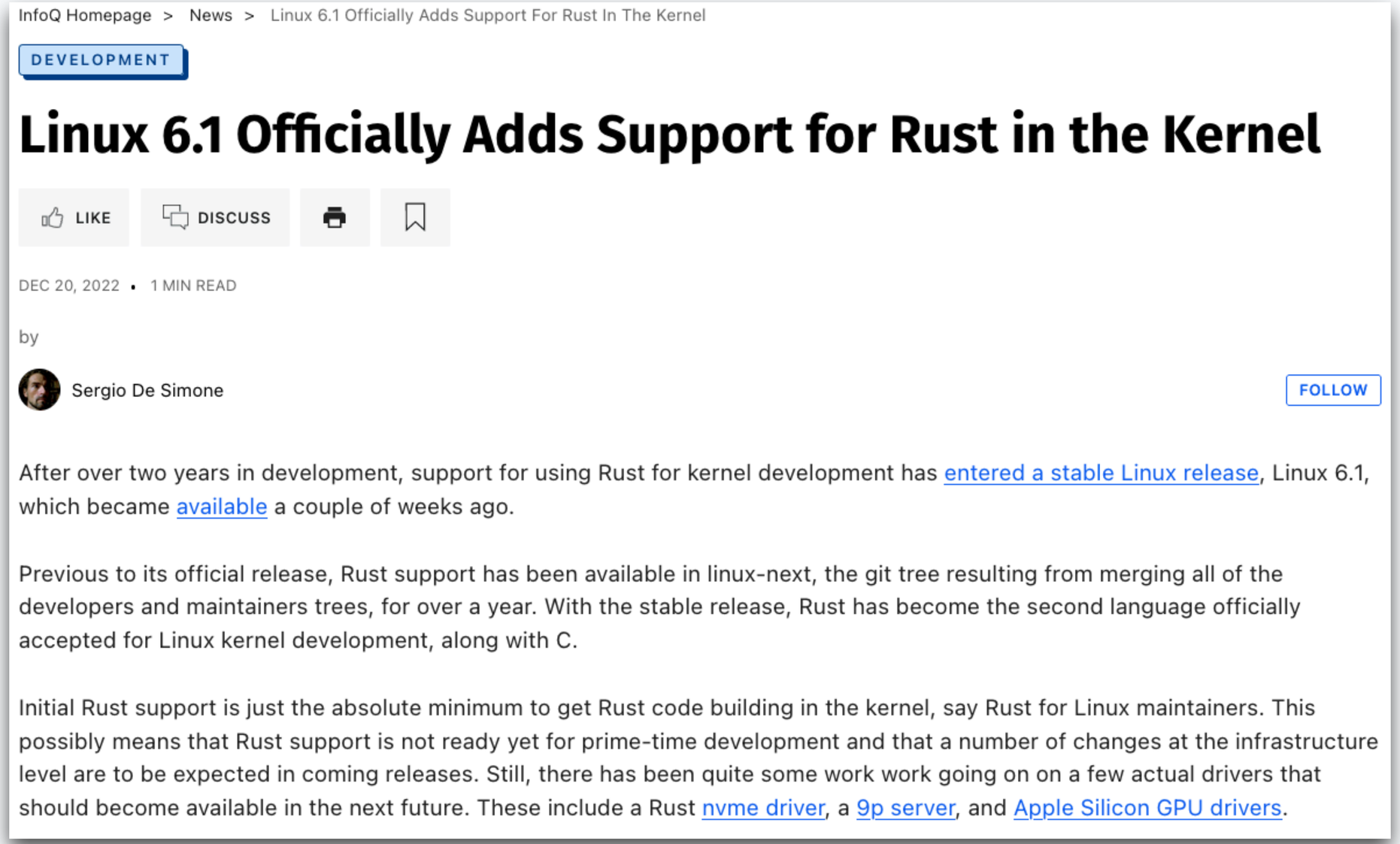


WhiteSource Report: What are the Most Secure Programming Languages

러스트가 cURL의 버그를 얼마나 많이 막을 수 있었을까?



Tim Hutt, Would Rust secure cURL?, https://blog.timhutt.co.uk/curl-vulnerabilities-rust/



DEVELOPMENT

Linux 6.1 Officially Adds Support for Rust in the Kernel

LIKE DISCUSS

DEC 20, 2022 • 1 MIN READ

by

Sergio De Simone

FOLLOW

After over two years in development, support for using Rust for kernel development has entered a stable Linux release, Linux 6.1, which became available a couple of weeks ago.

Previous to its official release, Rust support has been available in linux-next, the git tree resulting from merging all of the developers and maintainers trees, for over a year. With the stable release, Rust has become the second language officially accepted for Linux kernel development, along with C.

Initial Rust support is just the absolute minimum to get Rust code building in the kernel, say Rust for Linux maintainers. This possibly means that Rust support is not ready yet for prime-time development and that a number of changes at the infrastructure level are to be expected in coming releases. Still, there has been quite some work work going on on a few actual drivers that should become available in the next future. These include a Rust nvme driver, a 9p server, and Apple Silicon GPU drivers.

C에서 실패할 수 있는 함수를 구현하는 방법

```c
int div(int n, int d, int *q) {
  if (d == 0) { return 1; }
  *q = n / d;
  return 0;
}

int q;
if (div(10, 3, &q) == 0) {
  …
} else {
  …
}
```

함수 시그니처 현대화

현대화되지 않음

```
fn div(n: i32, d: i32, q: &mut i32) -> i32 {
  if d == 0 { return 1; }
  *q = n / d;
  0
}
```

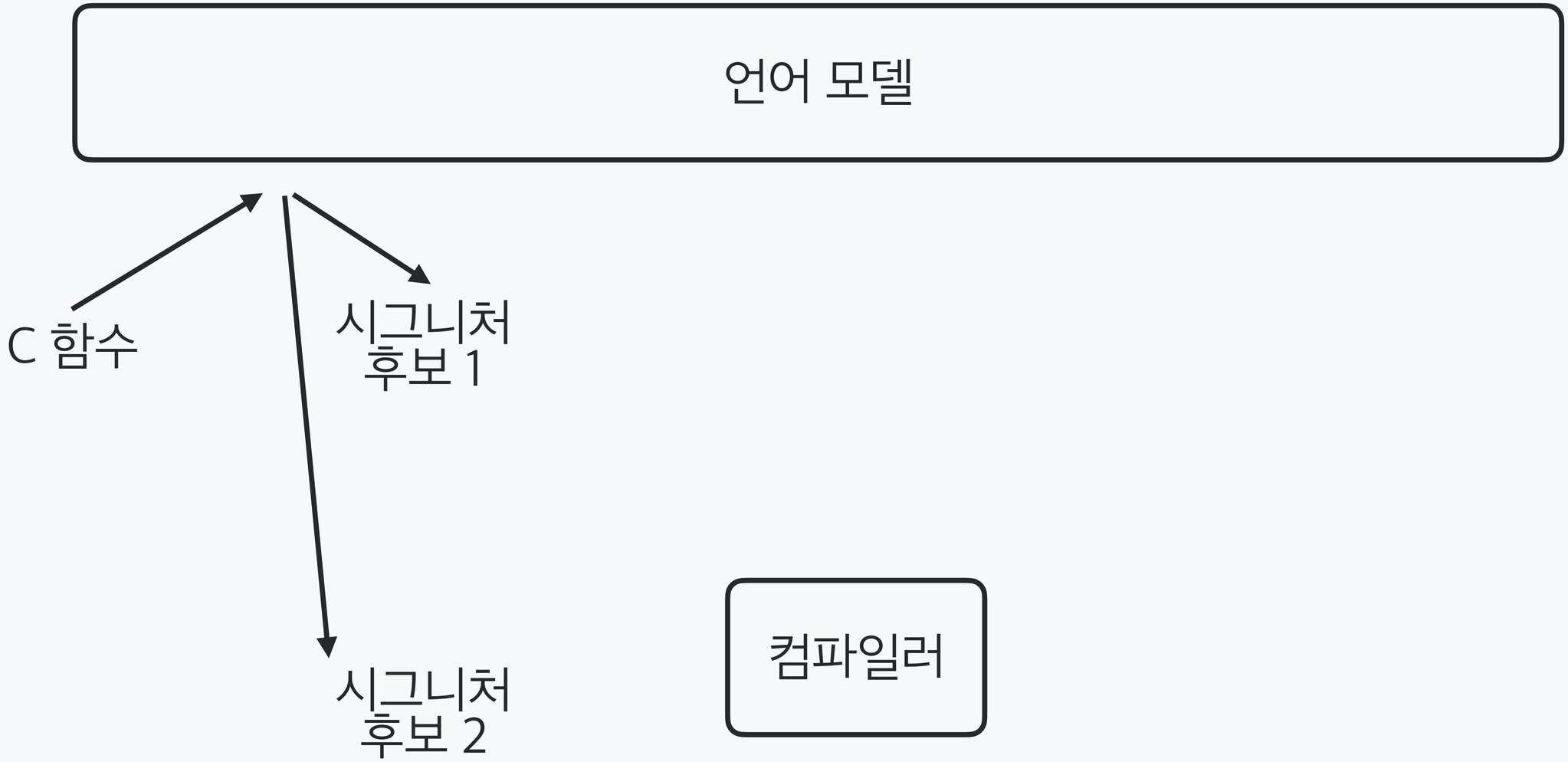
현대화됨

```
fn div(n: i32, d: i32) -> Option<i32> {
  if d == 0 { return None; }
  Some(n / d)
}
```

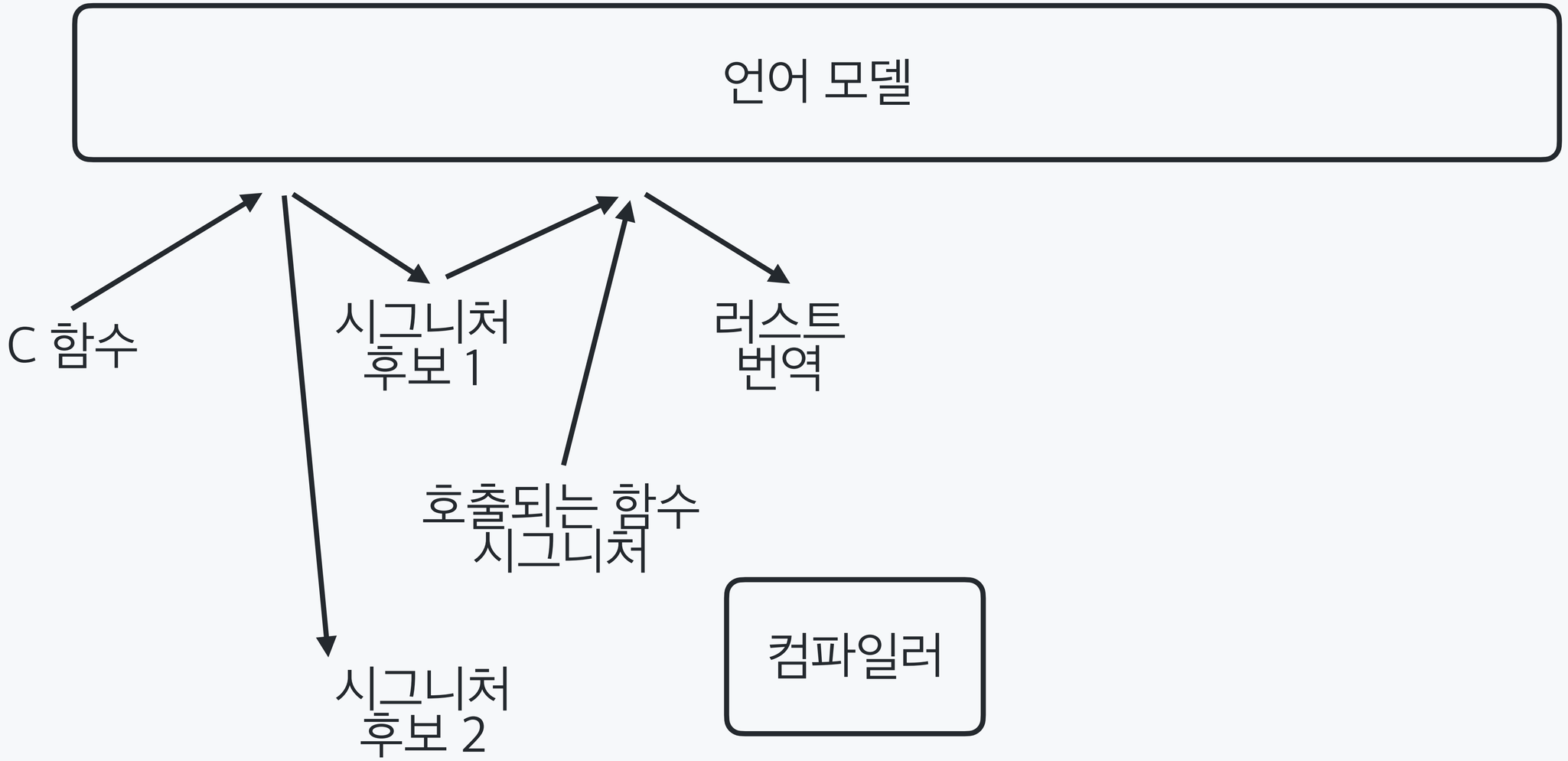
Simcrat의 구조

C 함수

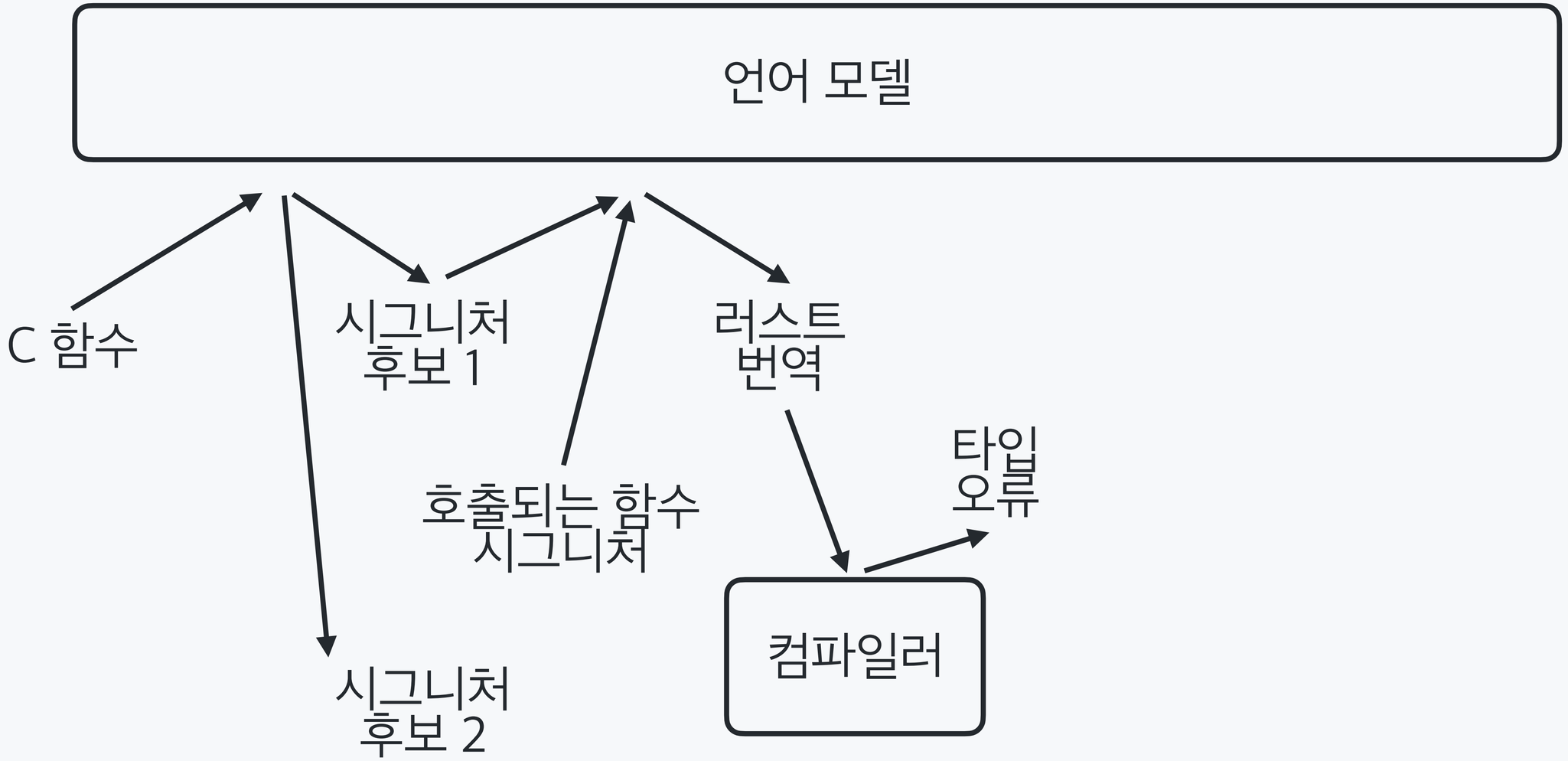
Simcrat의 구조



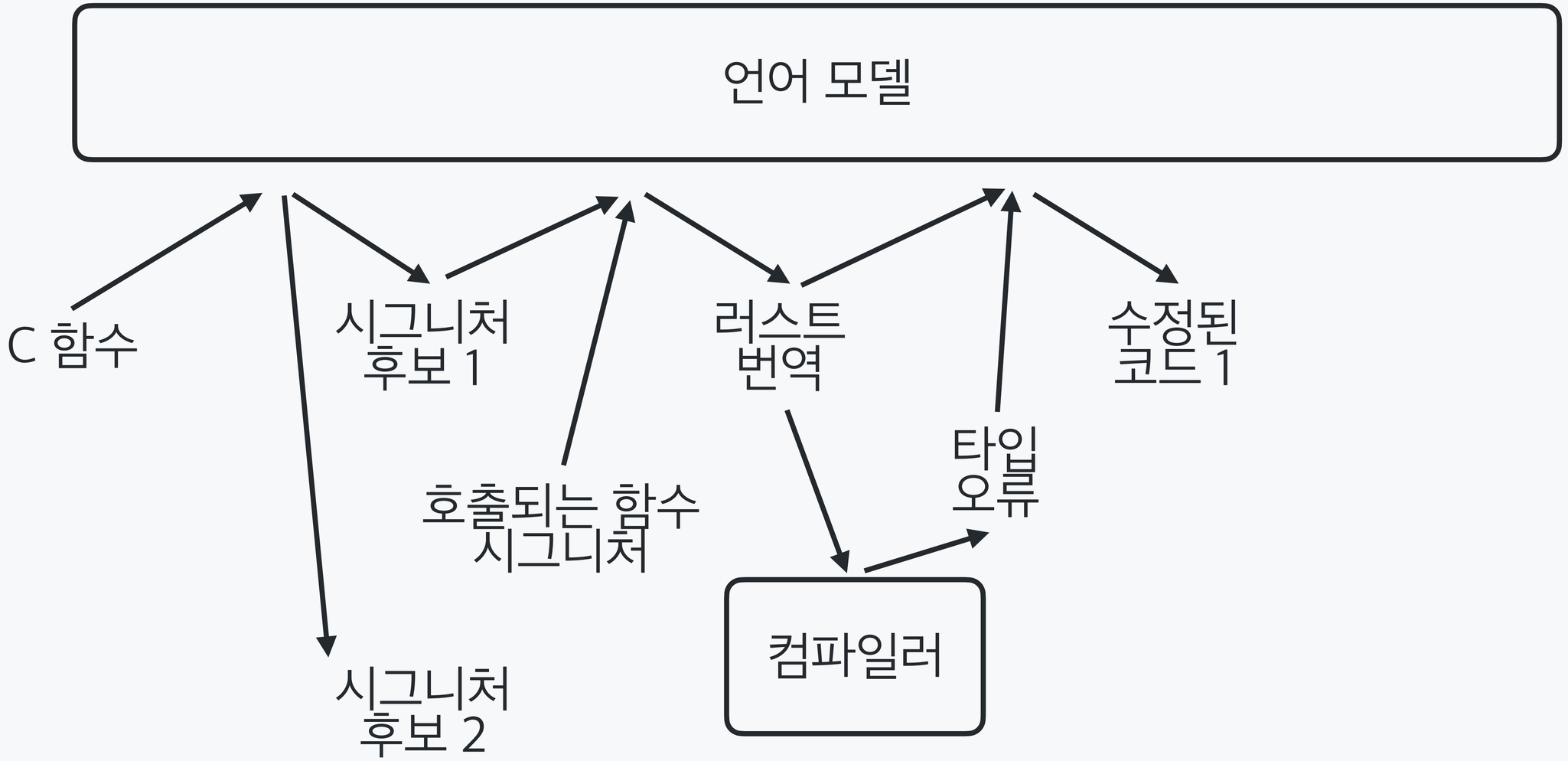
Simcrat의 구조



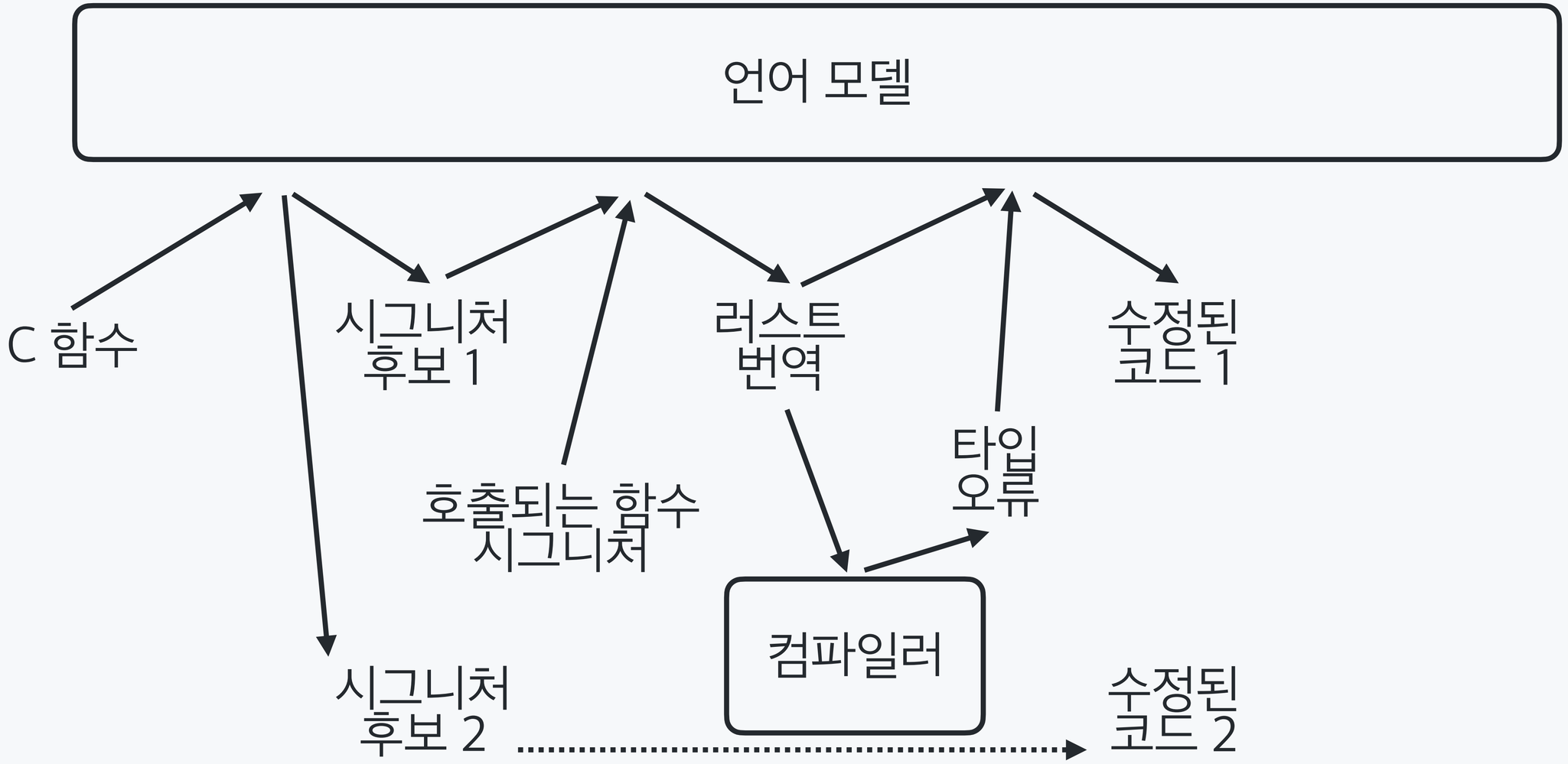
Simcrat의 구조



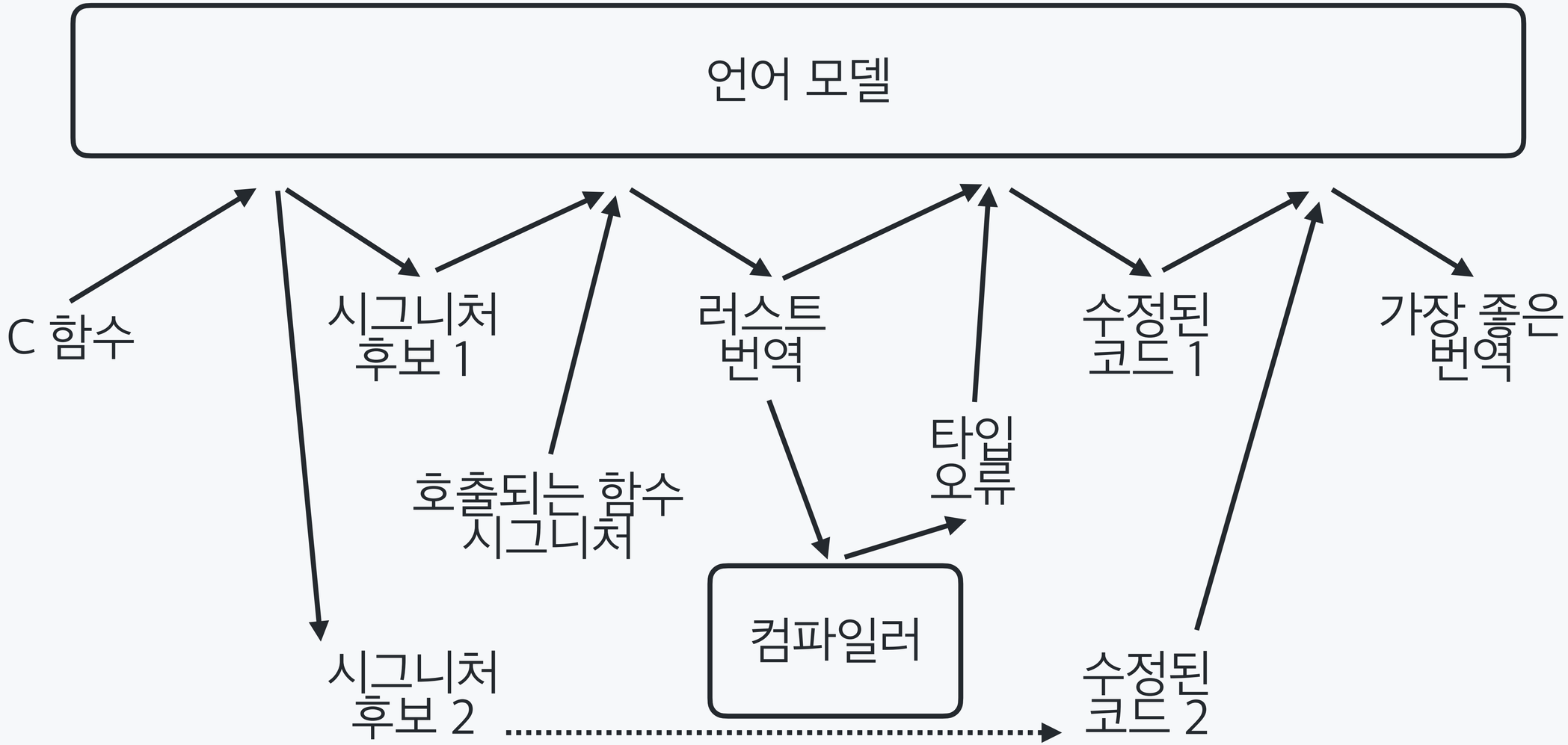
Simcrat의 구조


언어 모델
C 함수
시그니처
후보 1
시그니처
후보 2
호출되는 함수
시그니처
러스트
번역
타입
오류
수정된
코드 1
컴파일러

Simcrat의 구조


언어 모델
C 함수
시그니처
후보 1
시그니처
후보 2
호출되는 함수
시그니처
러스트
번역
타입
오류
컴파일러
수정된
코드 1
수정된
코드 2

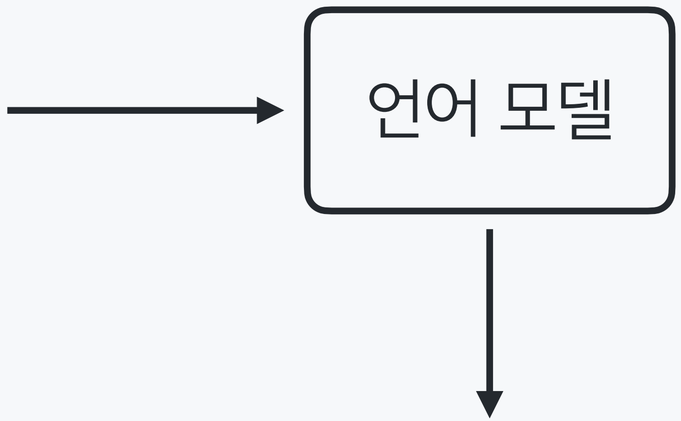
Simcrat의 구조


언어 모델
C 함수
시그니처
후보 1
시그니처
후보 2
호출되는 함수
시그니처
러스트
번역
타입
오류
컴파일러
수정된
코드 1
수정된
코드 2
가장 좋은
번역

시그니처 후보 없이 번역하기

```
Translate to Rust:


int div(int n, int d, int *q) {
  if (d == 0) { return 1; }
  *q = n / d;
  return 0;
}
```

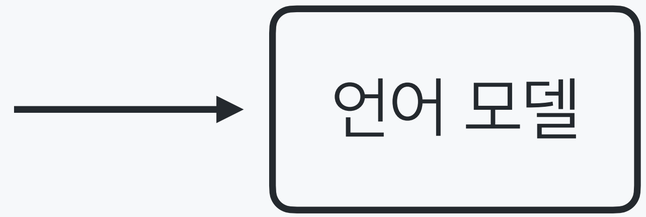



```
fn div(n: i32, d: i32, q: &mut i32) -> i32 {
  if d == 0 { return 1; }
  *q = n / d;
  0
}
```

시그니처 후보 생성

```
Give 2 idiomatic Rust signatures:


int div(int n, int d, int *q) {
  if (d == 0) { return 1; }
  *q = n / d;
  return 0;
}
```




```
fn div(n: i32, d: i32, q: &mut i32) -> i32
fn div(n: i32, d: i32) -> Option<i32>
```

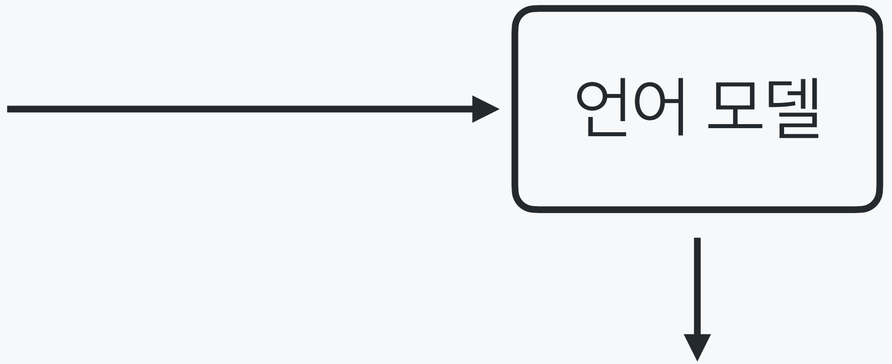
시그니처 후보를 사용하여 번역하기

Translate to Rust:

```
int div(int n, int d, int *q) {
  if (d == 0) { return 1; }
  *q = n / d;
  return 0;
}
```

Start with:

```
fn div(n: i32, d: i32)
    -> Option<i32>
```




```
fn div(n: i32, d: i32)
    -> Option<i32> {
  if d == 0 { return None; }
  Some(n / d)
}
```

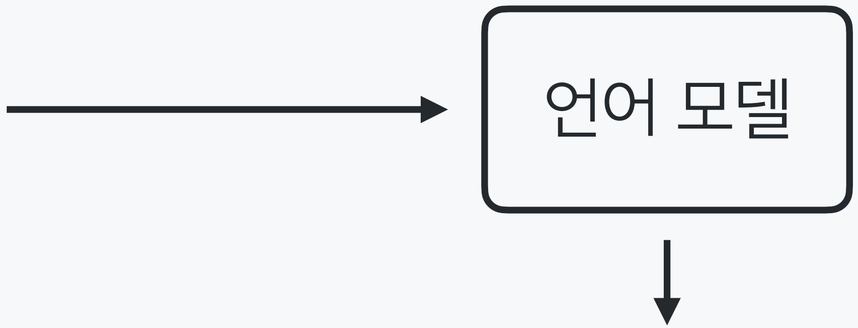
호출되는 함수의 시그니처 없이 번역하기

```
Translate to Rust:

int div5(int d, int *q) {
  return div(5, d, q);
}

Start with:

fn div5(d: i32) -> Option<i32>
```



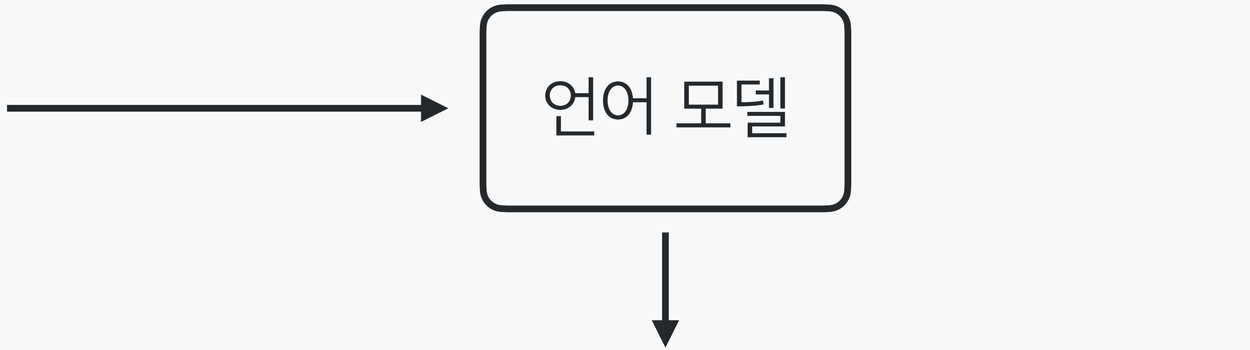

```
fn div5(d: i32) -> Option<i32> {
  let mut q = 0;
  let res = div(5, d, &mut q);
  …
}
```

호출되는 함수의 시그니처를 사용하여 번역하기

```
Already translated:
fn div(n: i32, d: i32)
    -> Option<i32>

Translate to Rust:
int div5(int d, int *q) {
  return div(5, d, q);
}

Start with:
fn div5(d: i32) -> Option<i32>
```



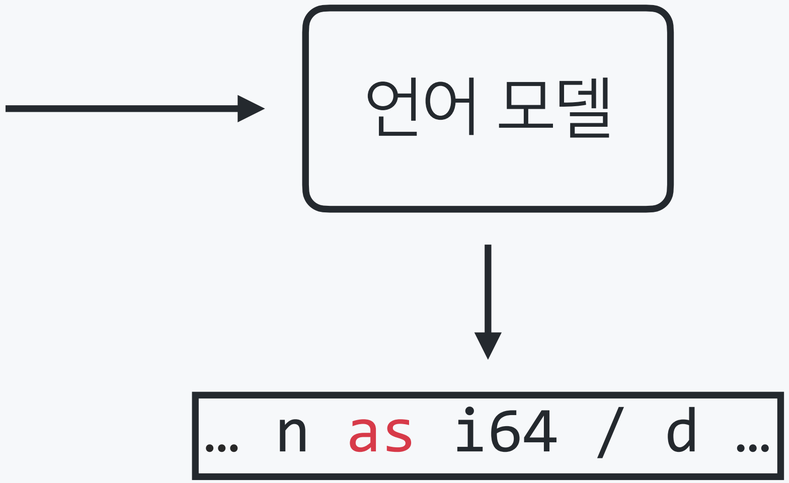

```
fn div5(d: i32) -> Option<i32> {
  div(5, d)
}
```

타입 오류 수정

```
Fix the code:

fn div(n: i32, d: i64) -> Option<i64> {
  if d == 0 { return None; }
  Some(n / d)
}

cannot divide `i32` by `i64`
  Some(n / d)
```

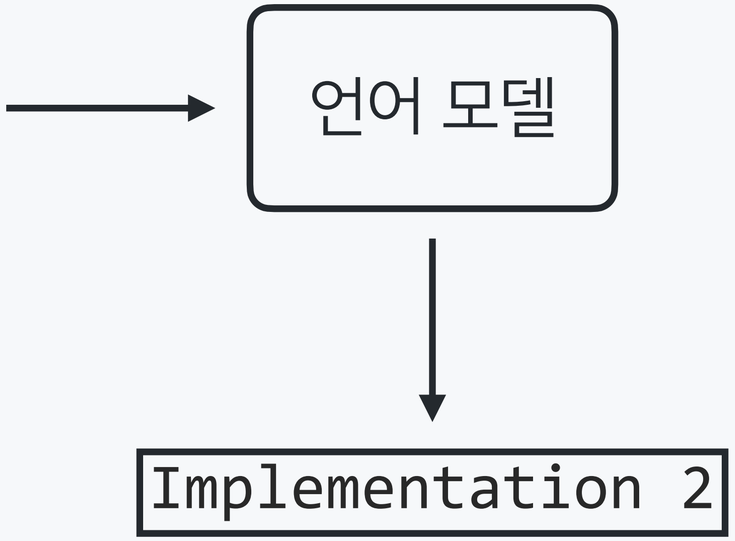


가장 좋은 번역 선택하기

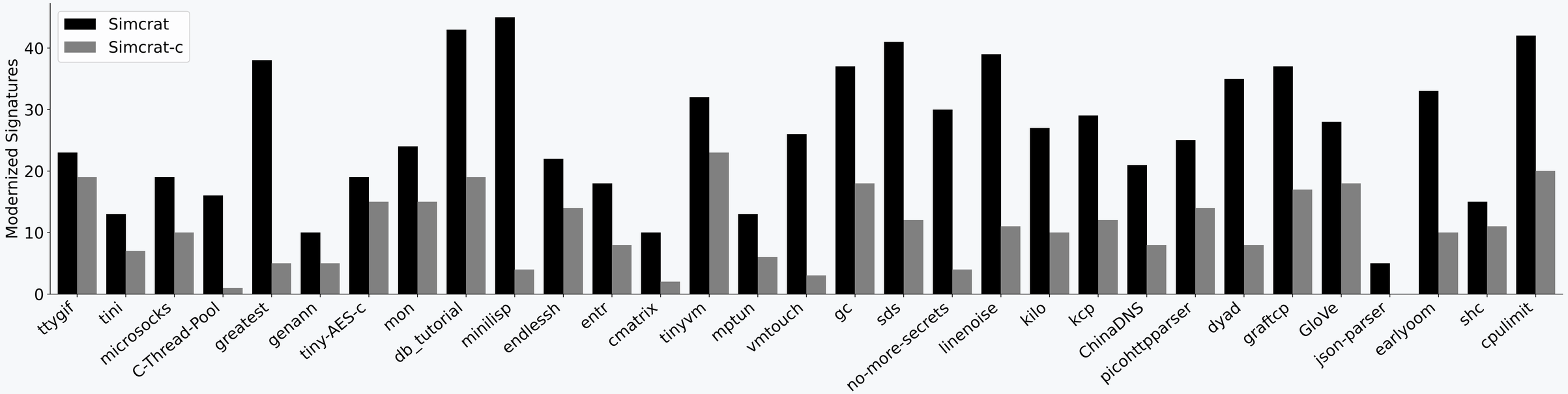
Which one is more Rust-idiomatic?

```rust
fn div(n: i32, d: i32, q: &mut i32) -> i32 {
  if d == 0 { return 1; }
  *q = n / d;
  0
}

fn div(n: i32, d: i32) -> Option<i32> {
  if d == 0 { return None; }
  Some(n / d)
}
```



시그니처를 효과적으로 현대화하는가?



2.77배 더 많이 시그니처를 현대화

시그니처를 효과적으로 현대화하는가?

C 함수:

```
int get_cursor_position(int *rows, int *cols) {
  if (…) { return 1; } …
  *rows = …; *cols = …;
  return 0;
}
```

시그니처 후보 없이 번역:

```
fn get_cursor_position(rows: &mut i32, cols: &mut i32) -> i32;
```

시그니처 후보를 사용하여 번역:

```
fn get_cursor_position() -> Option<(i32, i32)>;
```

시그니처를 효과적으로 현대화하는가?

C 함수:

```
void check_auth_method(char *buf, size_t n) {
  …
}
```

시그니처 후보 없이 번역:

```
fn check_auth_method(buf: &[u8], n: size);
```

시그니처 후보를 사용하여 번역:

```
fn check_auth_method(buf: &[u8]);
```

시그니처를 효과적으로 현대화하는가?

C 함수:

```
int cmp_net_mask(const void *a, const void *b) {
  NetMaskT *neta = (NetMaskT *) a;
  NetMaskT *netb = (NetMaskT *) b; …
}
```

시그니처 후보 없이 번역:

```
fn cmp_net_mask(a: *const c_void, b: *const c_void) -> c_int;
```

시그니처 후보를 사용하여 번역:

```
fn cmp_net_mask(a: &NetMaskT, b: &NetMaskT) -> Ordering;
```

시그니처를 효과적으로 현대화하는가?

C 함수:

```
void make_env(Obj **vars) {
  r->vars = *vars; …
}
```

시그니처 후보 없이 번역:

```
fn make_env(vars: &mut *mut Obj);
```

시그니처 후보를 사용하여 번역:

```
fn make_env(vars: &Obj);
```

시그니처를 효과적으로 현대화하는가?

C 함수:

```
void system_exec(Options o) {
  …
}
```

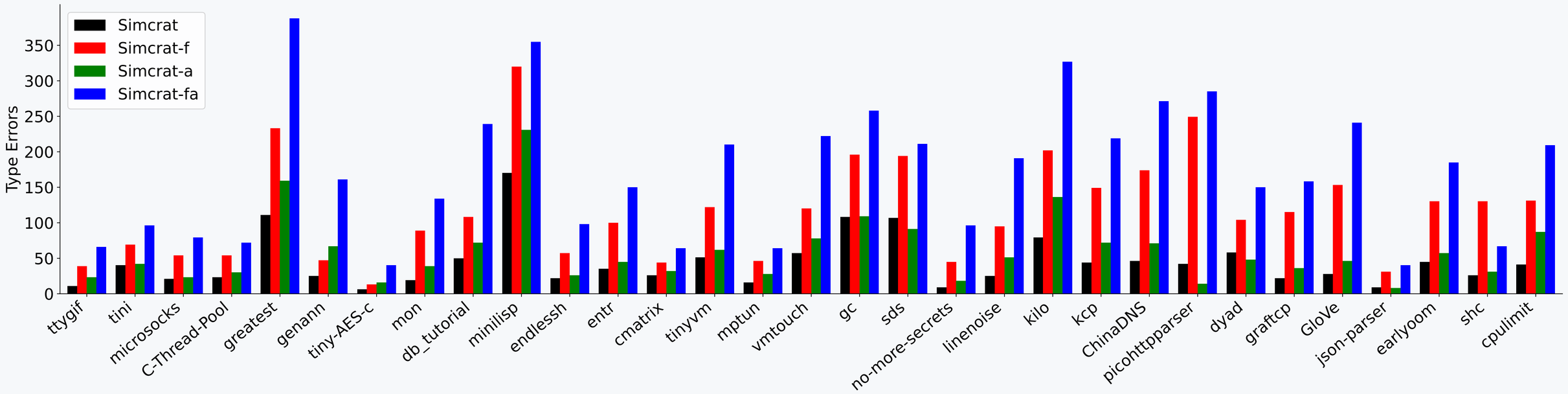
시그니처 후보 없이 번역:

```
fn system_exec(o: Options);
```

시그니처 후보를 사용하여 번역:

```
fn system_exec(o: &Options);
```

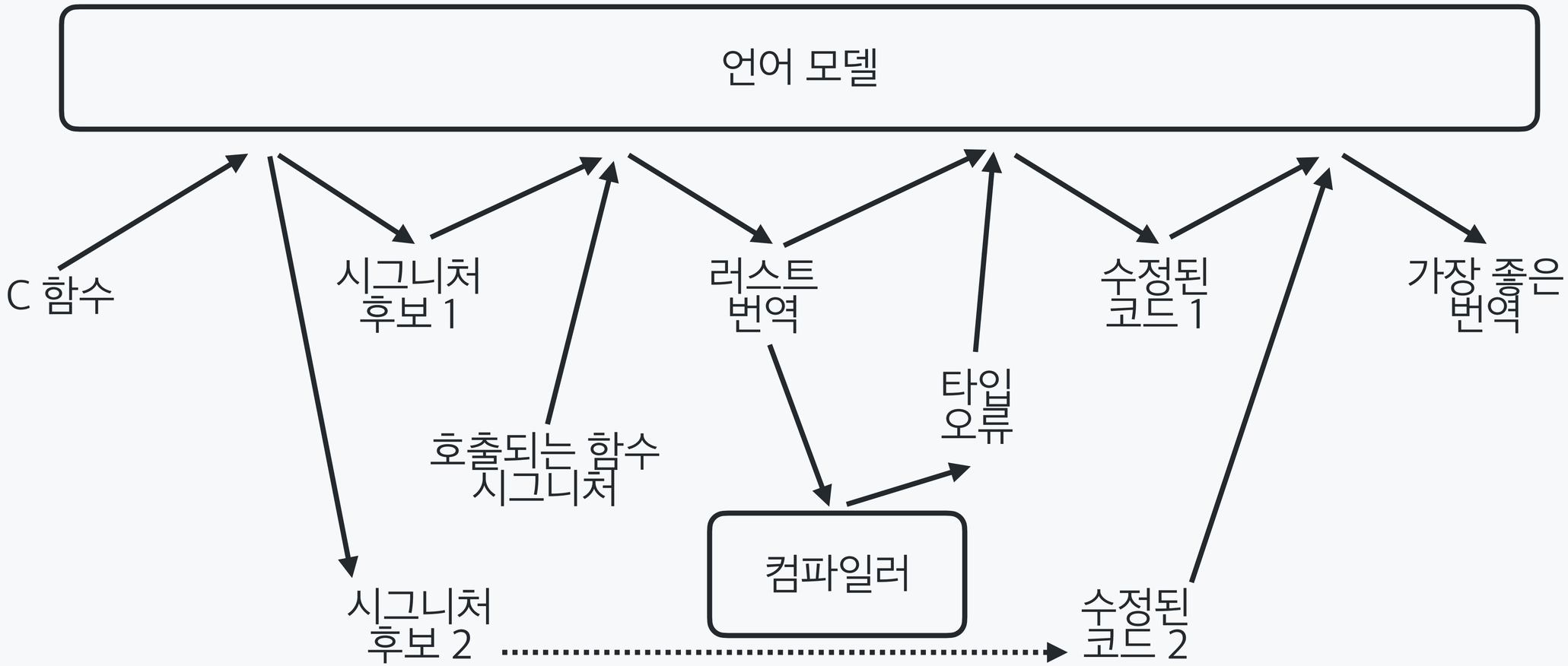
타입 오류를 효과적으로 줄이는가?



호출되는 함수 시그니처 제공: 28%

타입 오류 수정: 65% 적은 타입 오류

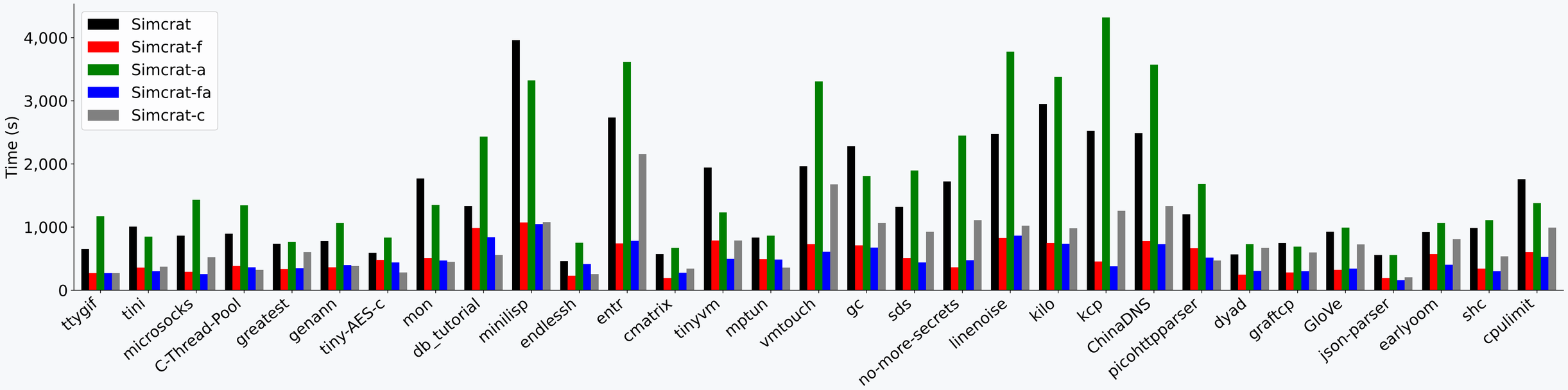
함께: 77%



Simcrat:

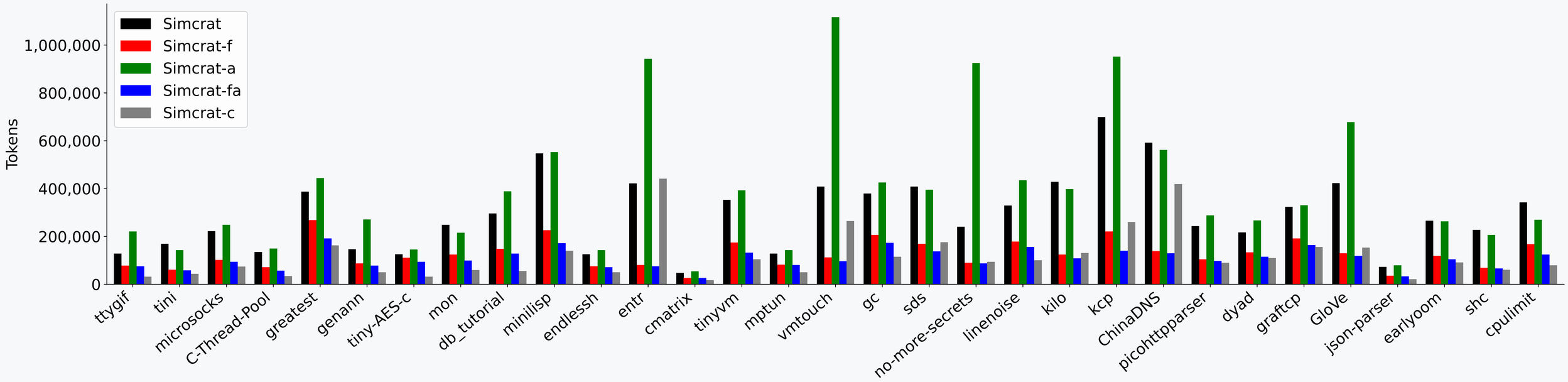
A Signature-Modernizing C-to-Rust Translator using a Large Language Model

번역 시간의 증가가 합리적인가?



호출되는 함수 시그니처 제공: 0.83배
타입 오류 수정: 2.64배 더 소요
함께: 2.75배

번역 비용의 증가가 합리적인가?



호출되는 함수 시그니처 제공: 0.81배

타입 오류 수정: 2.23배 더 많은 토큰 사용

함께: 2.57배