GNNIC: Finding Long-Lost Sibling Functions with Abstract Similarity

Qiushi Wu[1,2], Zhongshu Gu[2], Hani Jamjoom[2], Kangjie Lu[1]
Qiushi.Wu@ibm.com, zgu@us.ibm.com, jamjoom@us.ibm.com, kjlu@umn.edu
[1] University of Minnesota, [2] IBM Research

Motivation

Accurately identifying the indirect-call targets has always been a critical challenge.

It is the foundation of constructing an accurate call graph.

Motivation

Accurately identifying the indirect-call targets has always been a critical challenge.

It is the foundation of constructing an accurate call graph.

- Control-flow integrity
- Program static analysis
  - Slicing
  - Control-/data-flow analysis
- Program dynamic analysis
- …

Motivation

Different applications have different requirements

- Control-flow integrity
- Static analysis
- Dynamic analysis

- Can tolerate false indirect-call targets
- Cannot tolerate missing indirect-call targets
- Need scalability

Different applications have different requirements

- Control-flow integrity
- Static analysis
- Dynamic analysis

- Can tolerate some false positives and false negatives
- Need scalability

Motivation

High false-positive rate or poor performance for indirect-call targets identification

- Type-based analysis
- Alias analysis

Motivation

High false-positive rate or poor performance for indirect-call targets identification

- Type-based analysis
  - No false negative
  - Scalable
  - High false-positive rate

- Alias analysis

Motivation

High false-positive rate or poor performance for indirect-call targets identification

- Type-based analysis
  - No false negative
  - Scalable
  - High false-positive rate

- General types make type analysis less effective: int, void, long, etc.

- Alias analysis

High false-positive rate or poor performance for indirect-call targets identification

- Type-based analysis
  - No false negative
  - Scalable
  - High false-positive rate
- Alias analysis

- General types make type analysis less effective: int, void, long, etc.

E.g., Type-based approaches can find 1K+ target functions for the following function pointer:

```
int (*console_blank_hook) (int)
```

Motivation

High false-positive rate or poor performance for indirect-call targets identification

- Type-based analysis
  - No false negative
  - Scalable
  - High false-positive rate

- Alias analysis
  - Highly efficient and precise within limited scopes.
  - Not scalable
  - High FPR and FNR for large scopes analysis.

Different applications have different requirements

- Control-flow integrity
- Static analysis
- Dynamic analysis

We want to develop an indirect-call targets analysis approach having:

- Less false positives
- Less false negatives
- Scalable for large programs

Observation

Target functions of an indirect call share similarity of high-level semantics

Observation

Target functions of an indirect call share similarity of high-level semantics

- Given a function pointer: req→ns→file→f_op→read_iter(iocb, &iter)

Observation

Target functions of an indirect call share similarity of high-level semantics

- Given a function pointer: req→ns→file→f_op→read_iter(iocb, &iter)
- Its target functions should perform:
  file-, read-, and iterator-related operations

Observation

Target functions of an indirect call share similarity of high-level semantics

- Given a function pointer: req→ns→file→f_op→read_iter(iocb, &iter)
- Its target functions should perform:
  file-, read-, and iterator-related operations

```
ssize_t f2fs_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
ssize_t btrfs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
ssize_t v9fs_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
ssize_t f2fs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
```

A list of potential target functions identified by type analysis

Observation

Target functions of an indirect call share similarity of high-level semantics

- Given a function pointer: req→ns→file→f_op→read_iter(iocb, &iter)
- Its target functions should perform:
  file-, read-, and iterator-related operations

```
✓ ssize_t f2fs_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
✗ ssize_t btrfs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
✓ ssize_t v9fs_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
✗ ssize_t f2fs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
```

A list of potential target functions identified by type analysis

Intuition

How to measure the high-level semantics of the code?

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

GPIO device data structures

```
struct idi_48_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        spinlock_t ack_lock;
        unsigned char irq_mask[6];
        unsigned base;
        unsigned char cos_enb;
};
```

```
struct dio48e_gpio {
        struct gpio_chip chip;
        unsigned char io_state[6];
        unsigned char out_state[6];
        unsigned char control[2];
        raw_spinlock_t lock;
        unsigned base;
        unsigned char irq_mask;
};
```

```
struct idio_16_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        unsigned long irq_mask;
        unsigned base;
        unsigned out_state;
};
```

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

GPIO device data structures

```
struct idi_48_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        spinlock_t ack_lock;
        unsigned char irq_mask[6];
        unsigned base;
        unsigned char cos_enb;
};
```

```
struct dio48e_gpio {
        struct gpio_chip chip;
        unsigned char io_state[6];
        unsigned char out_state[6];
        unsigned char control[2];
        raw_spinlock_t lock;
        unsigned base;
        unsigned char irq_mask;
};
```

```
struct idio_16_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        unsigned long irq_mask;
        unsigned base;
        unsigned out_state;
};
```

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

GPIO device data structures

```
struct idi_48_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        spinlock_t ack_lock;
        unsigned char irq_mask[6];
        unsigned base;
        unsigned char cos_enb;
};
```

```
struct dio48e_gpio {
        struct gpio_chip chip;
        unsigned char io_state[6];
        unsigned char out_state[6];
        unsigned char control[2];
        raw_spinlock_t lock;
        unsigned base;
        unsigned char irq_mask;
};
```

```
struct idio_16_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        unsigned long irq_mask;
        unsigned base;
        unsigned out_state;
};
```

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

GPIO device data structures

```
struct idi_48_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        spinlock_t ack_lock;
        unsigned char irq_mask[6];
        unsigned base;
        unsigned char cos_enb;
};
```

```
struct dio48e_gpio {
        struct gpio_chip chip;
        unsigned char io_state[6];
        unsigned char out_state[6];
        unsigned char control[2];
        raw_spinlock_t lock;
        unsigned base;
        unsigned char irq_mask;
};
```

```
struct idio_16_gpio {
        struct gpio_chip chip;
        raw_spinlock_t lock;
        unsigned long irq_mask;
        unsigned base;
        unsigned out_state;
};
```

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

Module-specific release functions under drivers/media/dvb-frontends module

```
static void cx22702_release(struct dvb_frontend *fe)
{
        struct cx22702_state *state = fe->demodulator_priv;
        kfree(state);
}
```

```
static void cx24113_release(struct dvb_frontend *fe)
{
        struct cx24113_state *state = fe->tuner_priv;
        dprintk("\n");
        fe->tuner_priv = NULL;
        kfree(state);
}
```

```
static void cx24110_release(struct dvb_frontend* fe)
{
        struct cx24110_state* state = fe->demodulator_priv;
        kfree(state);
}
```

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

Module-specific release functions under drivers/media/dvb-frontends module

```
static void cx22702_release(struct dvb_frontend *fe)
{
        struct cx22702_state *state = fe->demodulator_priv;
        kfree(state);
}
```

```
static void cx24113_release(struct dvb_frontend *fe)
{
        struct cx24113_state *state = fe->tuner_priv;
        dprintk("\n");
        fe->tuner_priv = NULL;
        kfree(state);
}
```

```
static void cx24110_release(struct dvb_frontend* fe)
{
        struct cx24110_state* state = fe->demodulator_priv;
        kfree(state);
}
```

Intuition

High-level semantics are frequently hidden within nested structures of types and functions.

Module-specific release functions under drivers/media/dvb-frontends module

```
static void cx22702_release(struct dvb_frontend *fe)
{
        struct cx22702_state *state = fe->demodulator_priv;
        kfree(state);
}
```

```
static void cx24113_release(struct dvb_frontend *fe)
{
        struct cx24113_state *state = fe->tuner_priv;
        dprintk("\n");
        fe->tuner_priv = NULL;
        kfree(state);
}
```

```
static void cx24110_release(struct dvb_frontend* fe)
{
        struct cx24110_state* state = fe->demodulator_priv;
        kfree(state);
}
```

Intuition

We would like to use graph to capture such nested information.

Approach

Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)

Approach

Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)
  - Call graph of the program
  - Nested type graphs

Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)

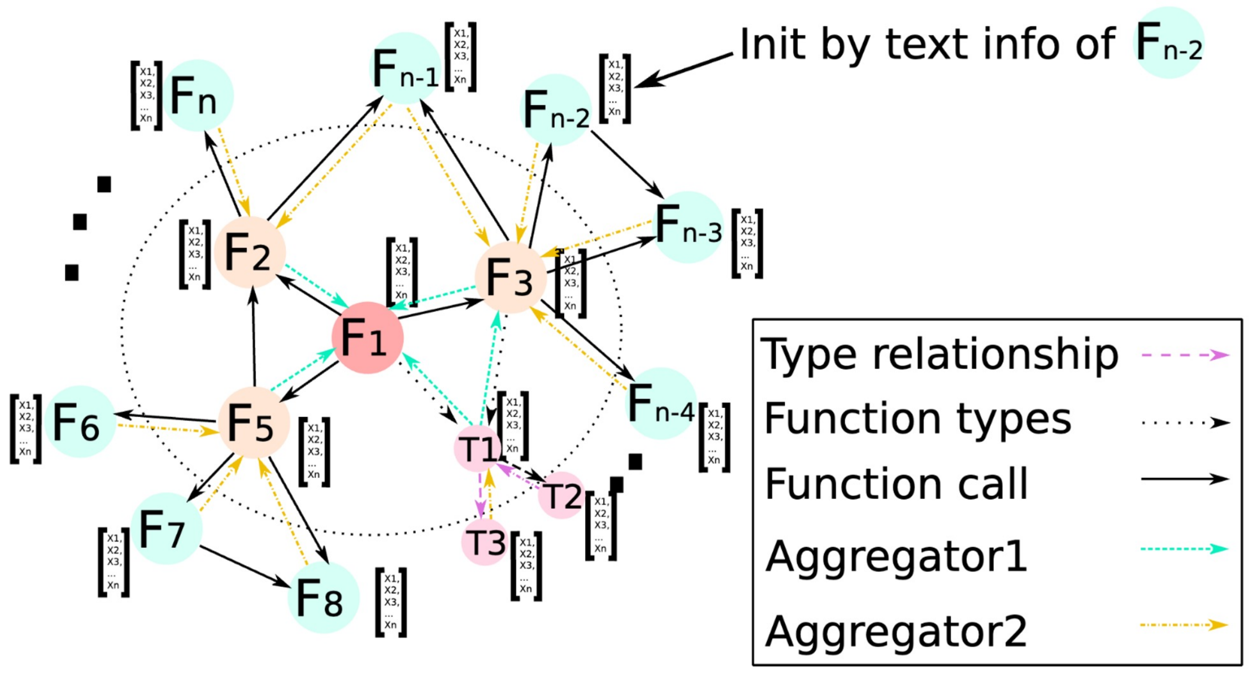
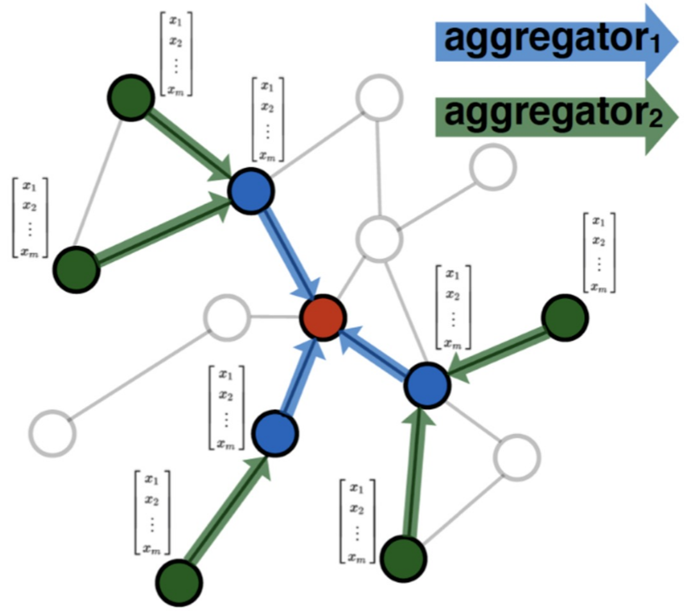


Fig. 4: Structure of GNN on representative abstraction graph. F=function, T=type.

Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Aggregate features information from the neighbors.

[7]. GraphSAGE: Inductive Representation Learning on Large Graphs
https://snap.stanford.edu/graphsage/#:~:text=GraphSAGE%20is%20a%20framework%20for,Code

Approach

Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)

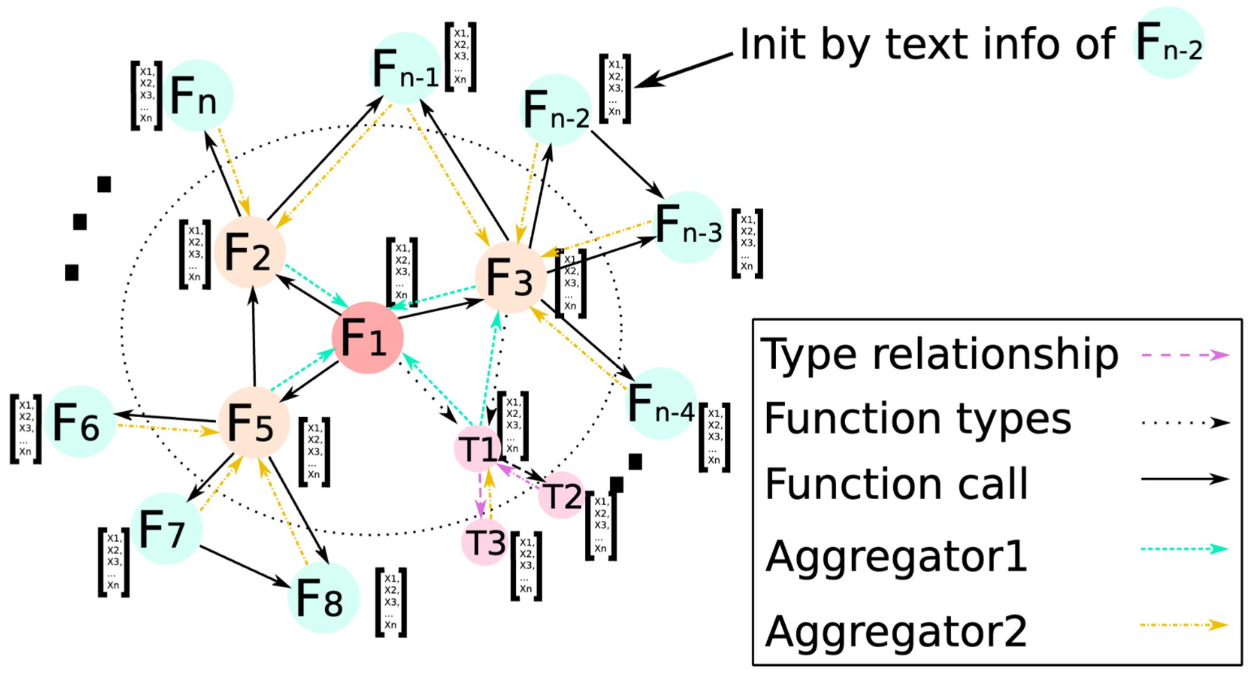
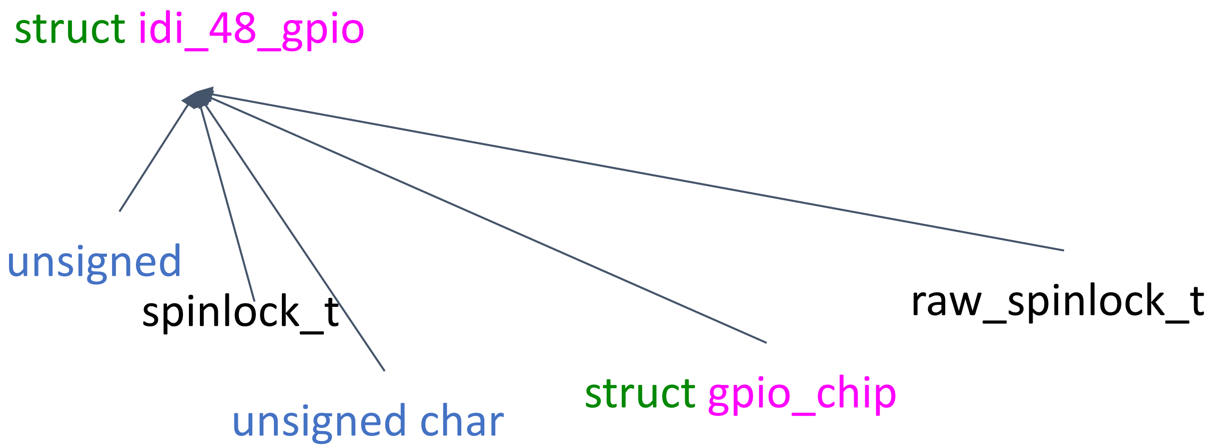


Fig. 4: Structure of GNN on representative abstraction graph. F=function, T=type.

Approach

Graph Neural Network for Code Structure Understanding

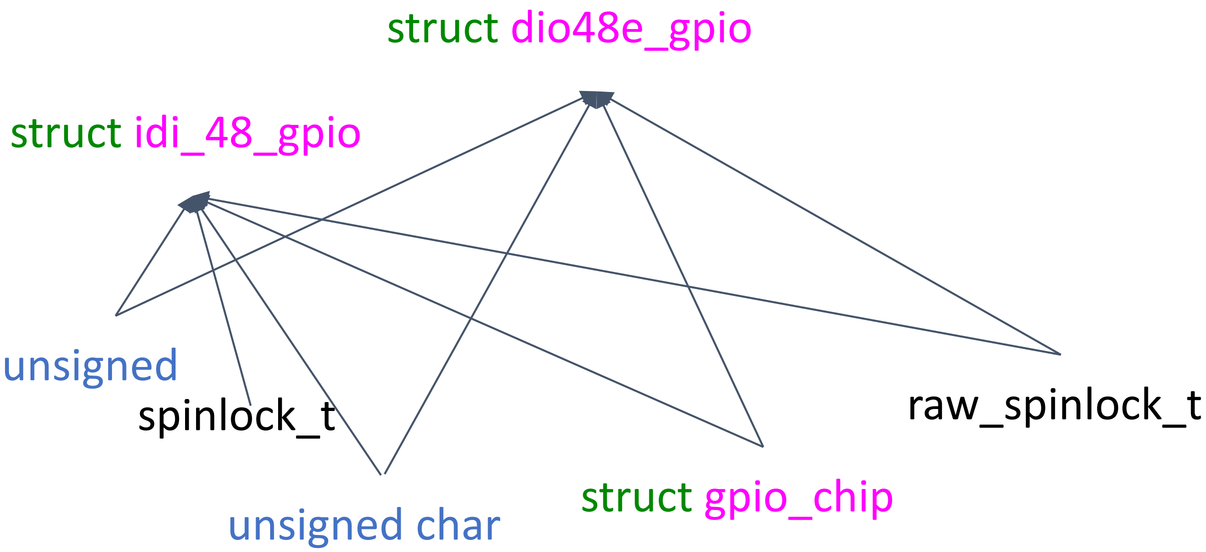
- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

Graph Neural Network for Code Structure Understanding

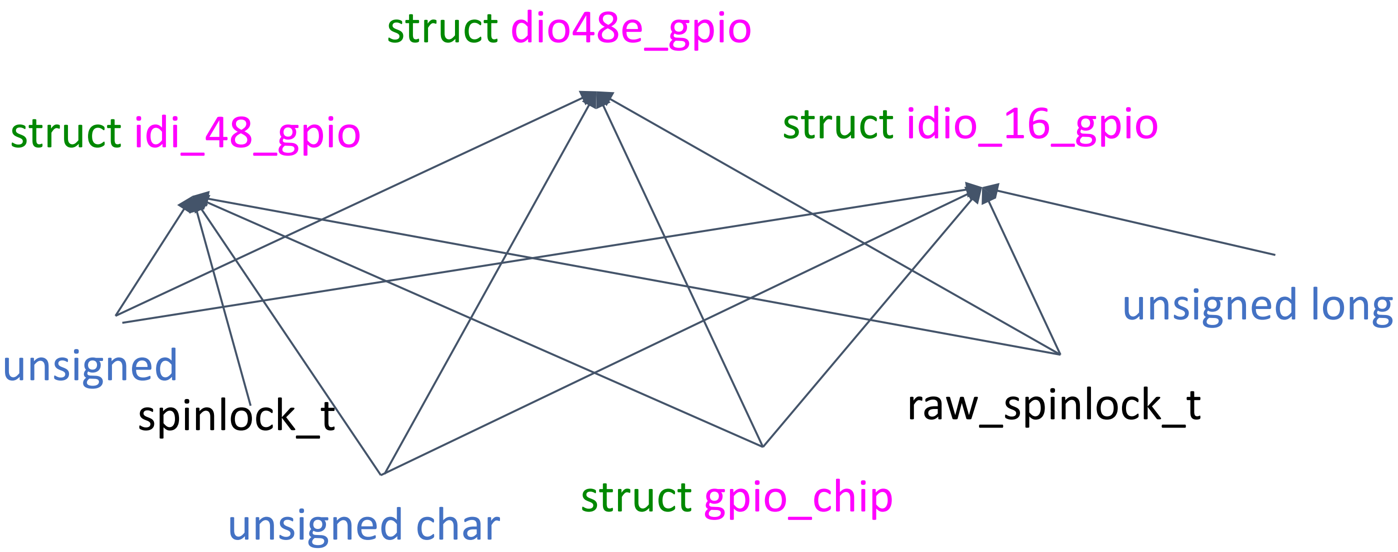
- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

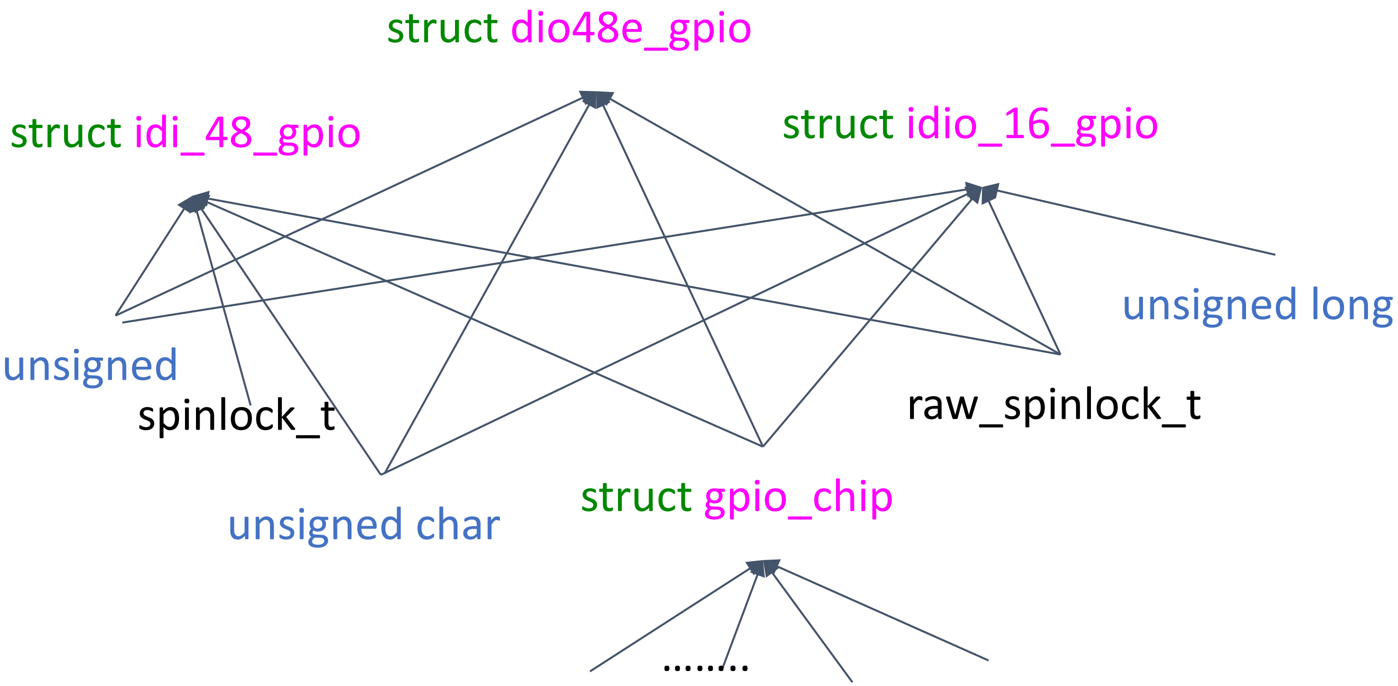
Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Graph Neural Network for Code Structure Understanding

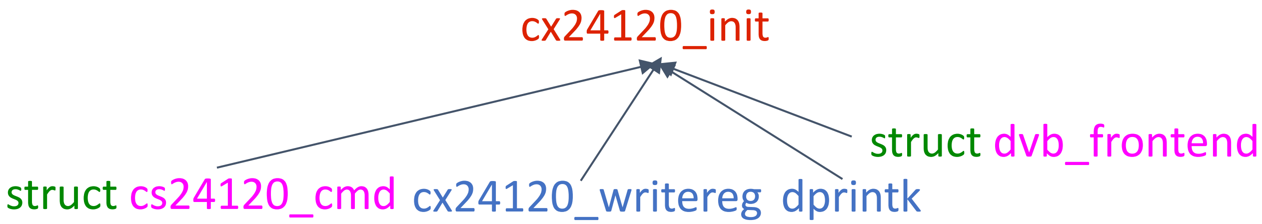
- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

Graph Neural Network for Code Structure Understanding

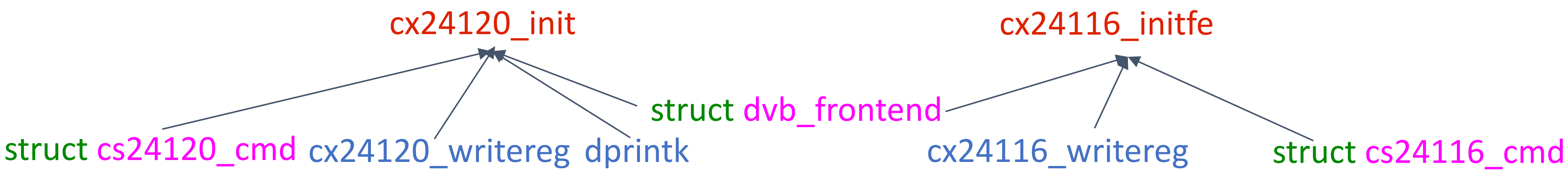
- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

Graph Neural Network for Code Structure Understanding

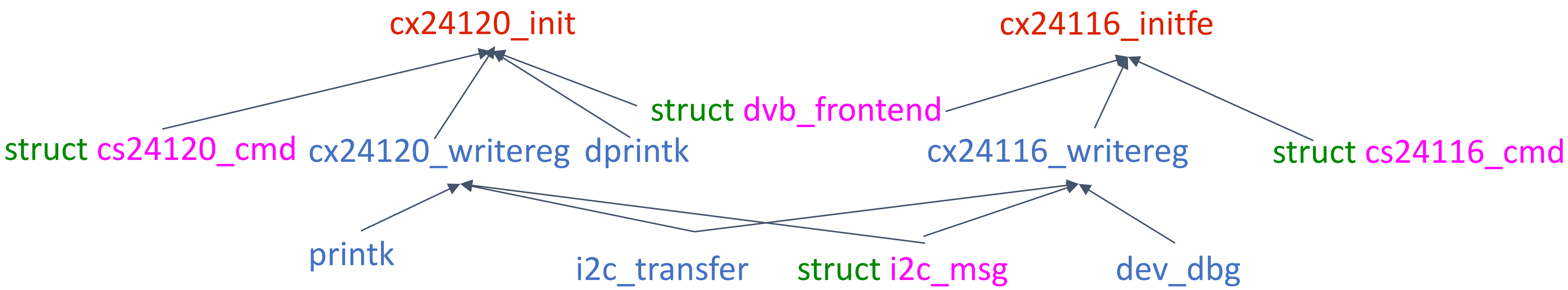
- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

Graph Neural Network for Code Structure Understanding

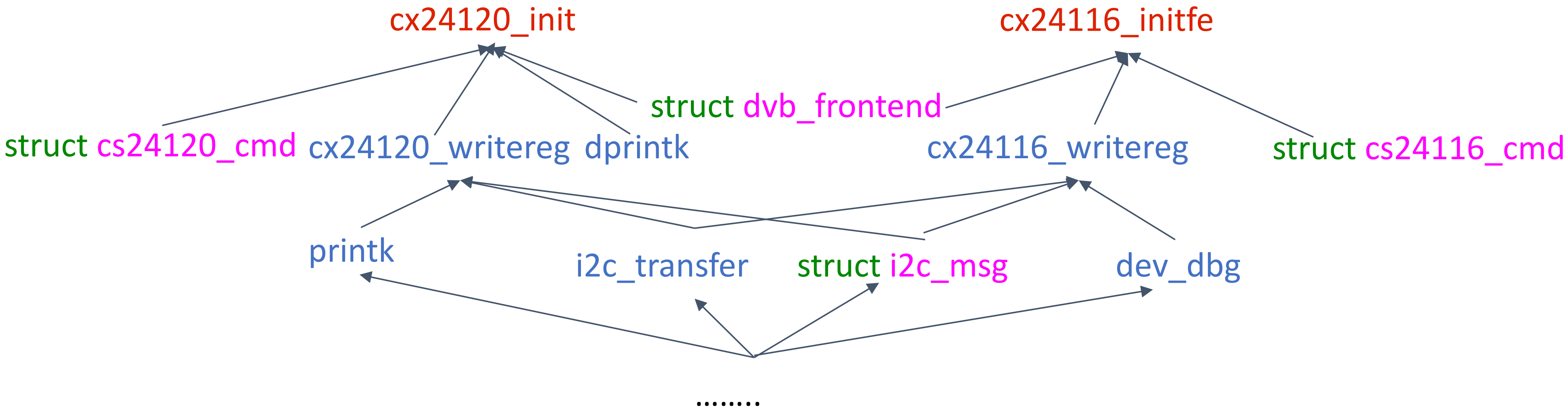
- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

Graph Neural Network for Code Structure Understanding

- Representing code structure with representative abstraction graph (RAG)
- Training the RAG with existing GNN framework: GraphSAGE



Approach

Target functions of an indirect call share similarity against high-level semantics

- Given a function pointer: req->ns->file->f_op->read_iter(iocb, &iter)
- And a list of potential target functions

```
ssize_t f2fs_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
ssize_t btrfs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
ssize_t v9fs_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
ssize_t f2fs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
```

A list of potential target functions identified by type analysis

Approach

Target functions of an indirect call share similarity against high-level semantics

- Given a function pointer: req->ns->file->f_op->read_iter(iocb, &iter)
- And a list of potential target functions

```
ssize_t f2fs_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
ssize_t btrfs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
ssize_t v9fs_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
ssize_t f2fs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
```

How can we assess the likelihood of these functions being a real target?

A list of potential target functions identified by type analysis

Approach

Target functions of an indirect call share similarity against high-level semantics

- Given a function pointer: req->ns->file->f_op->read_iter(iocb, &iter)
- And a list of potential target functions

```
ssize_t ext4_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
```

Anchor function

```
ssize_t f2fs_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
ssize_t btrfs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
ssize_t v9fs_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
ssize_t f2fs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
```

A list of potential target functions identified by type analysis

Approach

Target functions of an indirect call share similarity against high-level semantics

- Given a function pointer: req->ns->file->f_op->read_iter(iocb, &iter)
- And a list of potential target functions

```
ssize_t ext4_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
```

Anchor function

```
✓ ssize_t f2fs_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
✗ ssize_t btrfs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
✓ ssize_t v9fs_file_read_iter(struct kiocb *iocb, struct iov_iter *to)
✗ ssize_t f2fs_direct_IO(struct kiocb *iocb, struct iov_iter *iter)
```

A list of potential target functions identified by type analysis

Overview

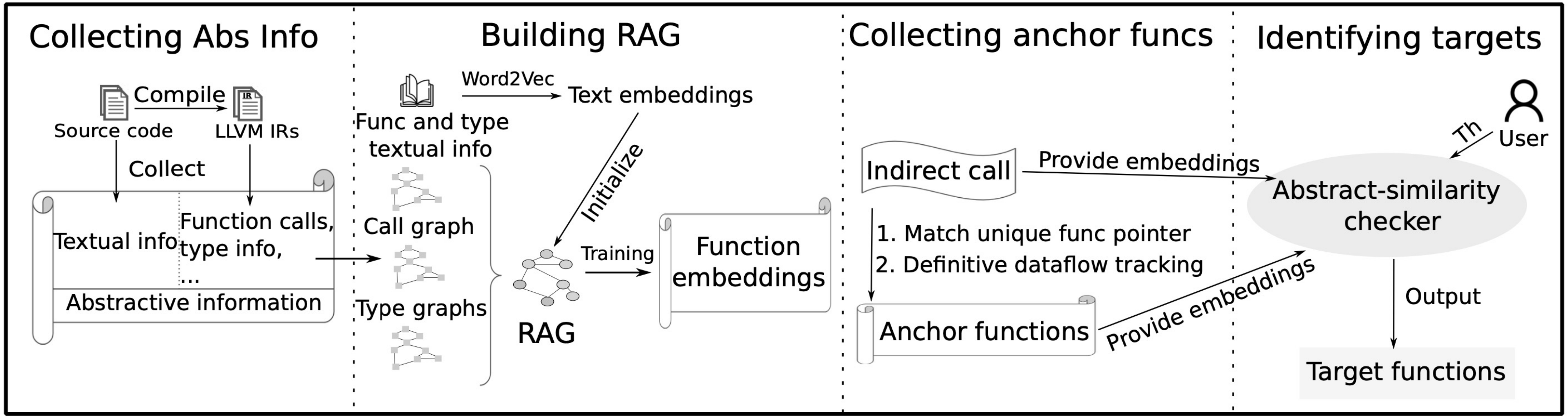


Fig. 2: Overview of GNNIC. RAG=representative abstraction graph, Abs Info=abstractive information, Th = threshold specified by user.

Overview

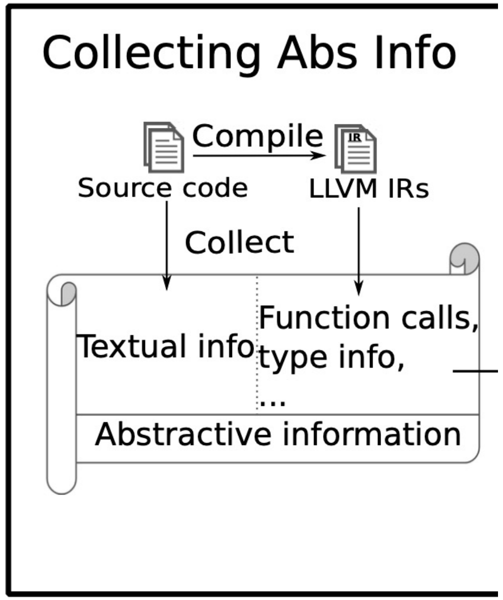


Fig. 2: Overview of GNNIC. RAG=representative abstraction graph, Abs Info=abstractive information, Th = threshold specified by user.

Overview

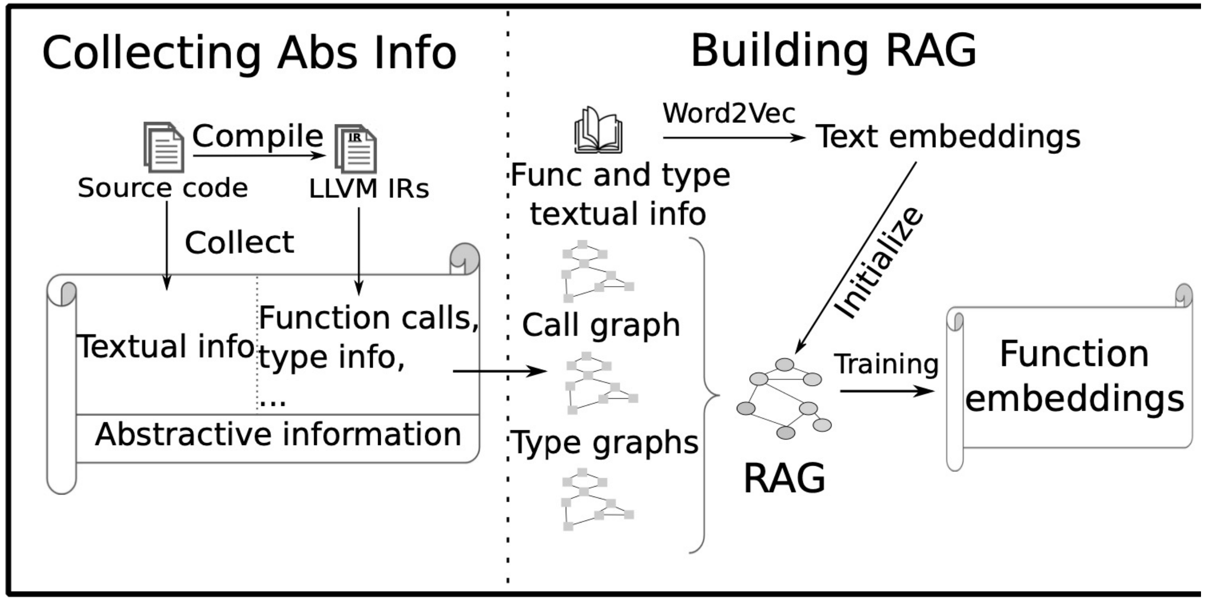


Fig. 2: Overview of GNNIC. RAG=representative abstraction graph, Abs Info=abstractive information, Th = threshold specified by user.

Overview

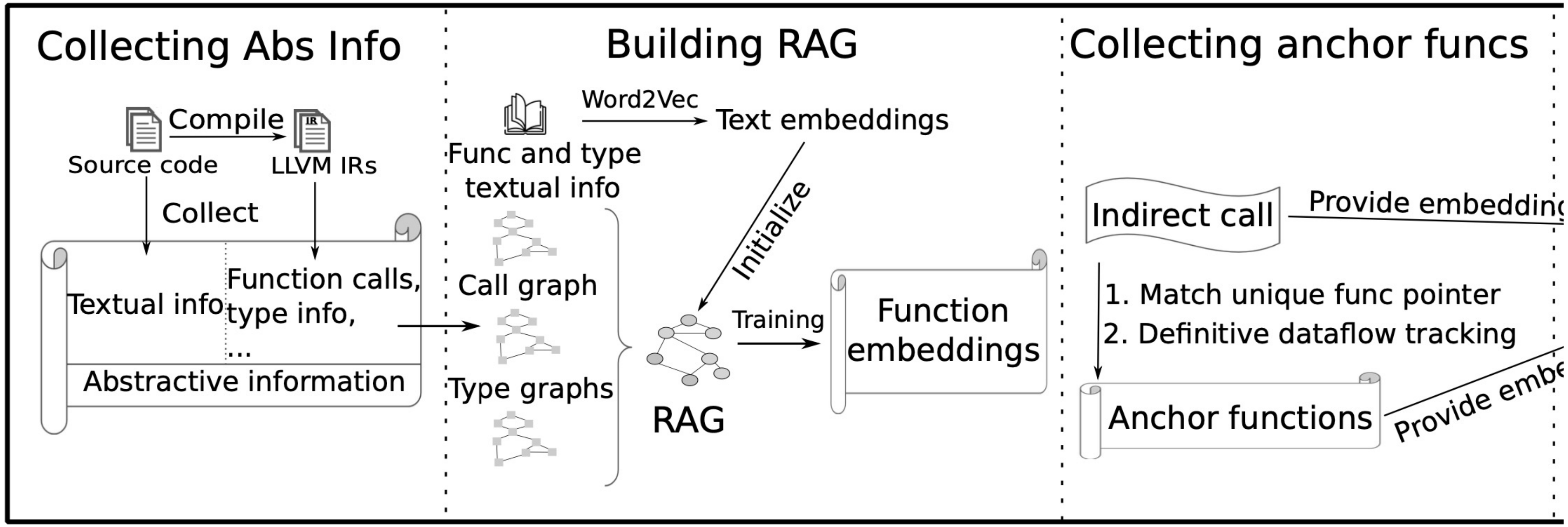


Fig. 2: Overview of GNNIC. RAG=representative abstraction graph, Abs Info=abstractive information, Th = threshold specified by user.

Overview

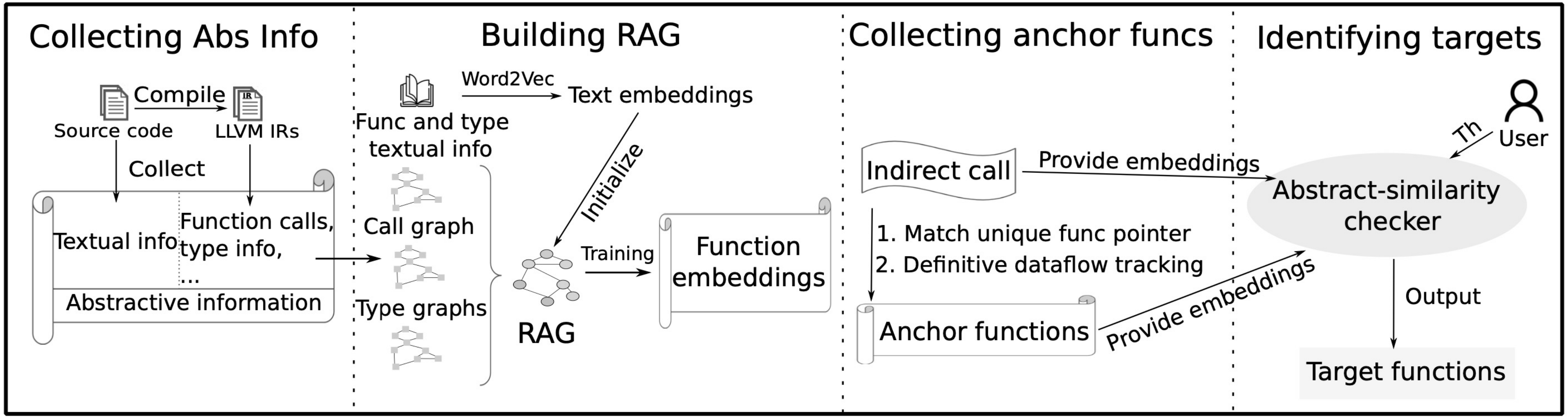


Fig. 2: Overview of GNNIC. RAG=representative abstraction graph, Abs Info=abstractive information, Th = threshold specified by user.

Evaluation: Compare with type-based approach

- Function-pointer analysis

TABLE I: Distribution of indirect calls that have a number of targets in the range (specified on the first row). Type-based: Original type-based approach on the system; Sim=0.9: The results of GNNIC based on the similarity at 0.9.

System/ # of targets	**<10**	**10-100**	**100-1000**	**>= 1000**	**TotalTargets**	**Total Icalls**	**Mean**	**MAX**
Linux (Type-based)	77.0%	15.8%	5.4%	1.8%	4734762	55921	84.7	8862
Linux (GNNIC, Sim = 0.9)	85.7%	13.0%	1.3%	0.0%	545452	55921	9.7	2436
Android (Type-based)	82.8%	10.0%	5.6%	1.6%	4769716	62618	76.1	9056
Android (GNNIC, Sim = 0.9)	94.9%	4.5%	0.6%	0.0%	297284	62618	4.7	2645
FreeBSD (Type-based)	85.5%	11.7%	1.3%	1.5%	251307	7578	33.2	1960
FreeBSD (GNNIC, Sim = 0.9)	88.5%	11.3%	0.2%	0.0%	34669	7578	4.5	217

Evaluation: Performance & Accuracy

- Function-pointer analysis

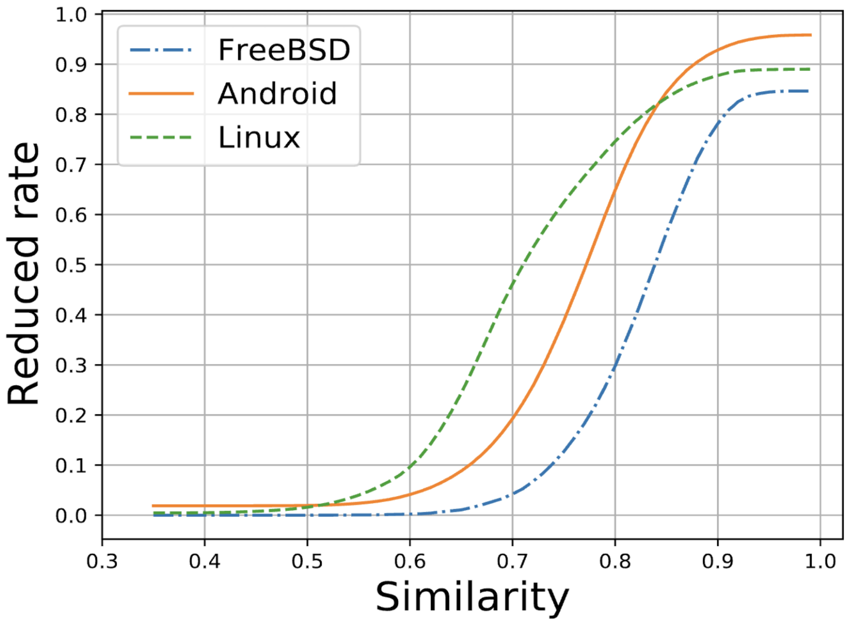


Fig. 6: Percentage of refined indirect-call targets for different OS kernels.

- At a FPR of 0.33%, GNNIC achieves a recall of 84.8%, indicating 15.2% of FNs.
- When recall reaches 99.6%, the corresponding FPR rises to 60.7%.
- It takes about 7 hours to train the model and analyze the whole kernel.

Evaluation: Enhancing program analysis with GNNIC and abstract similarity

- Function-pointer analysis
- Other security applications

- Find similar bugs caused by similar functions
- Enhancing vulnerability-reachability analysis.
- Improving directed fuzzing and concolic execution.

Conclusion

- Analyzed abstract similarity of functions.
- Developed graph-based techniques for indirect call identification.
- Evaluated on a spectrum of security applications.