

Large Language Models for Verified Programs

Omkar Zade

Supervised by:

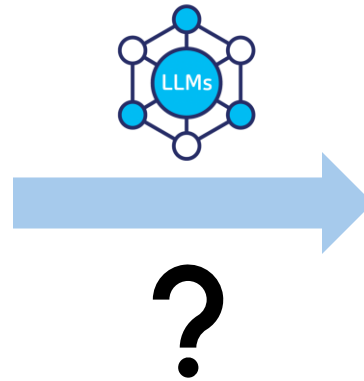
Prof. Dr. Peter Müller, Nicolas Klose, Jingxuan He

D INFK

ETH zürich

Motivation

```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    if head.next is None:
        n = Node(val)
        head.next = n
    else:
        append(head.next, val)
```



```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    Requires(is_list(head))
    Ensures(is_list(head))
    Unfold(is_list(head))
    if head.next is None:
        n = Node(val)
        Fold(is_list(n))
        head.next = n
    else:
        append(head.next, val)
    Fold(is_list(head))
```

Few-shot prompting

Ability of LLMs to adapt to a new task *without* gradient updates^[1]

Circulation revenue has increased by 5%
in Finland. // Positive

Panostaja did not disclose the purchase
price. // Neutral

Paying off the national debt will be
extremely painful. // Negative

The company anticipated its operating
profit to improve. // _____

LM

Circulation revenue has increased by
5% in Finland. // Finance

They defeated ... in the NFC
Championship Game. // Sports

Apple ... development of in-house
chips. // Tech

The company anticipated its operating
profit to improve. // _____

LM

Example from:
[https://ai.stanford.edu/blog/
understanding-incontext/](https://ai.stanford.edu/blog/understanding-incontext/)

[1] Language models are few-shot learners (OpenAI, 2020)

Our few-shot prompt

- Each example is:

Input: Unverified program
Verification error

Output: Verified program

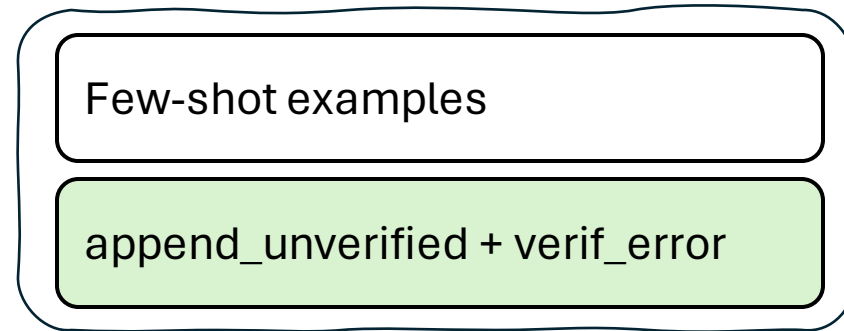
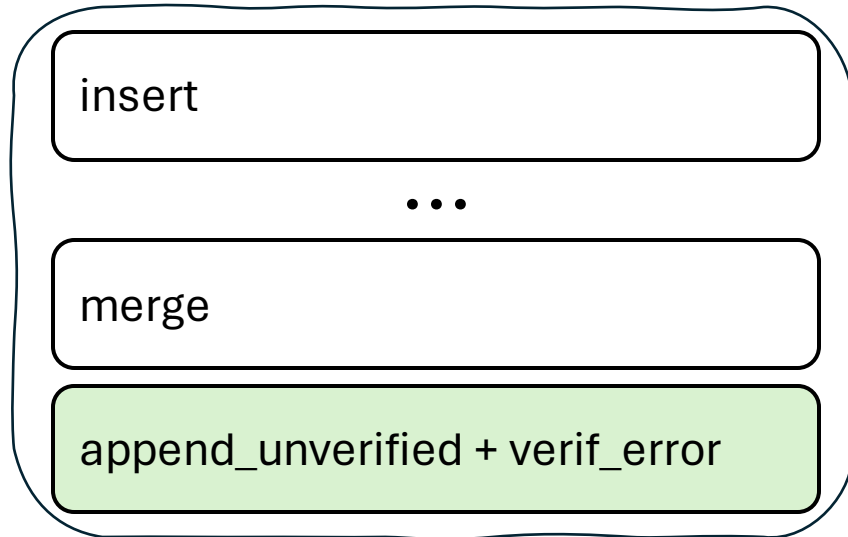
append

```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    if head.next is None:
        n = Node(val)
        head.next = n
    else:
        append(head.next, val)
```

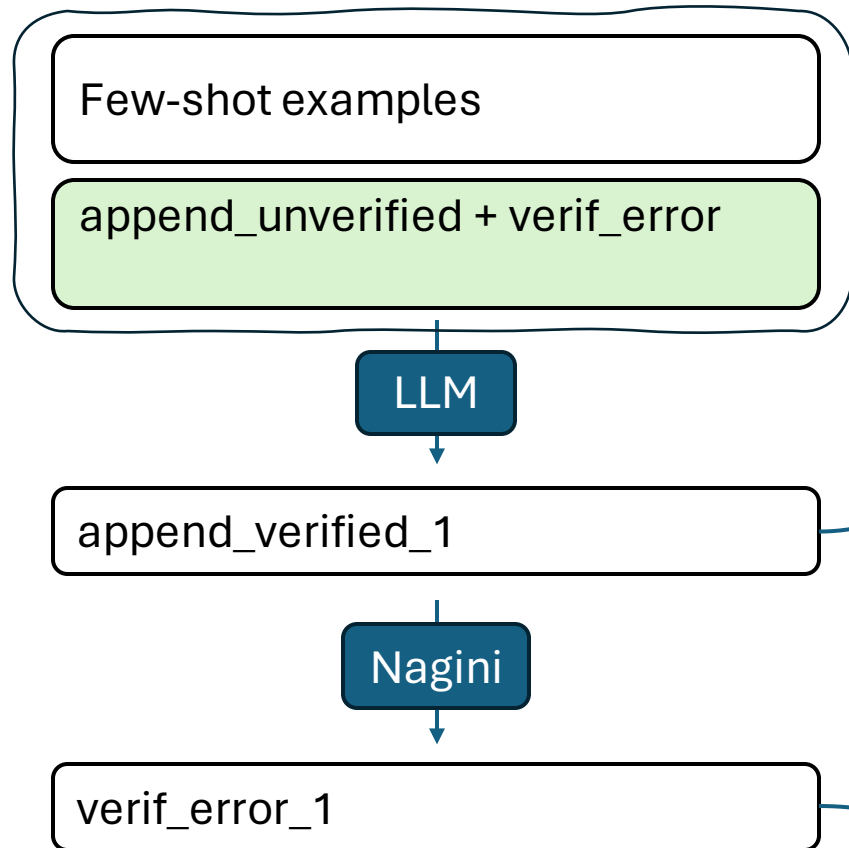
Conditional statement might fail. There might be insufficient permission to access head.next. at line 3.7

```
def append(head: Node, val: int) -> None:
    """Append a new node with the given value
    to the end of the list."""
    Requires(is_list(head))
    Ensures(is_list(head))
    Unfold(is_list(head))
    if head.next is None:
        n = Node(val)
        Fold(is_list(n))
        head.next = n
    else:
        append(head.next, val)
    Fold(is_list(head))
```

Our few-shot prompt



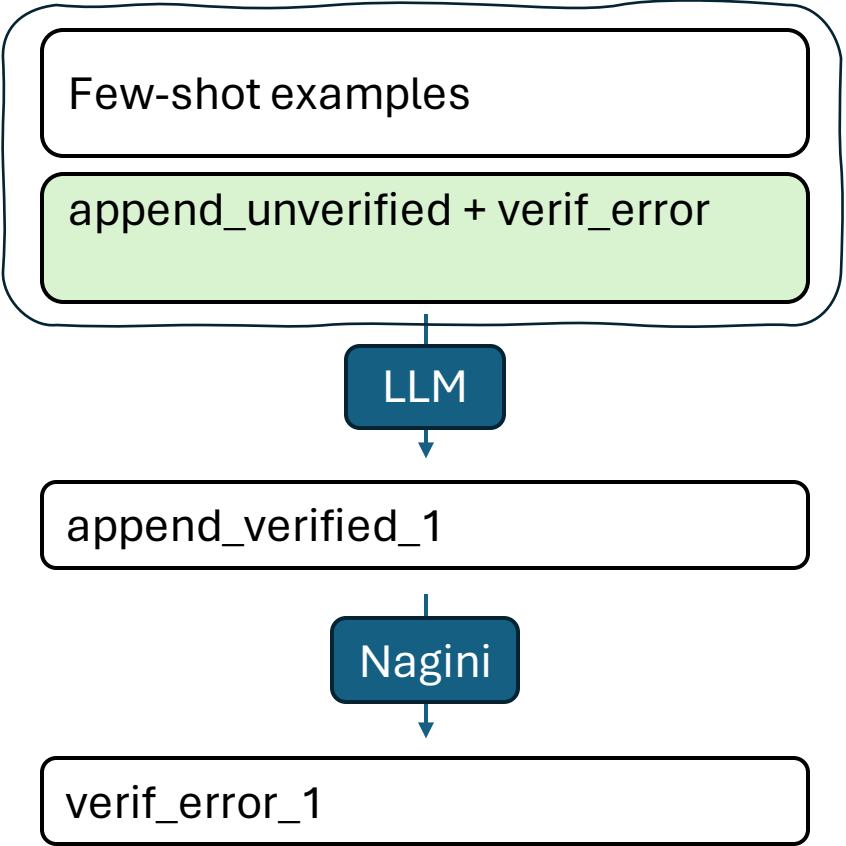
TryVerify



```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    Requires(is_list(head))
    Ensures(is_list(head))
    Unfold(is_list(head))
    if head.next is None:
        n = Node(val)
        Fold(is_list(n))
        head.next = n
    else:
        append(head.next, val)
        Fold(is_list(head))
```

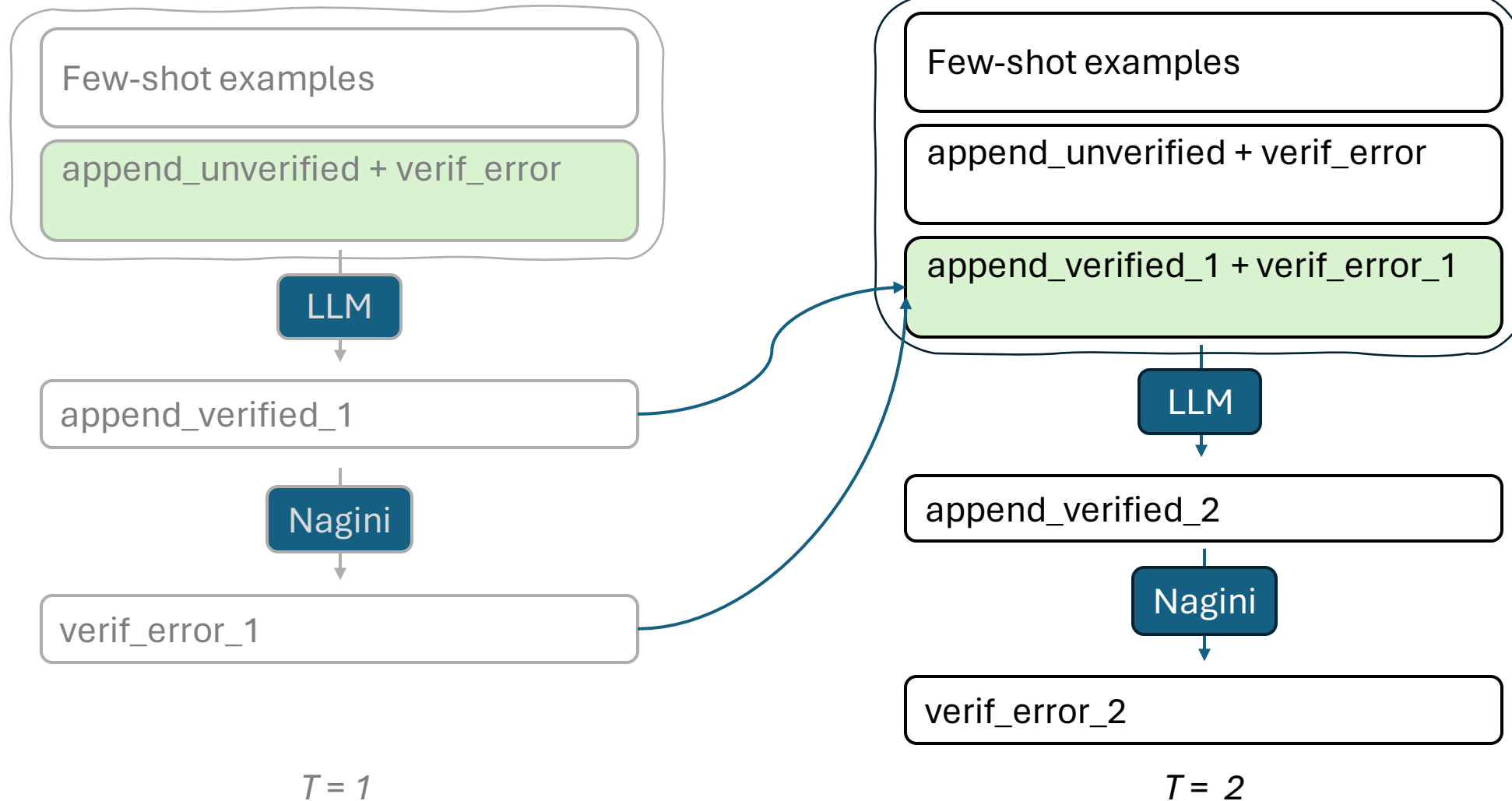
Fold might fail. There might be insufficient permission to access `is_list(head.next)` on line 11.4

TryVerify



$T = 1$

TryVerify



Experiment 1: Few-shot prompting

- Given examples of verified programs, can the model verify an unseen program?

dataset = [append, insert, prepend, ..., merge]

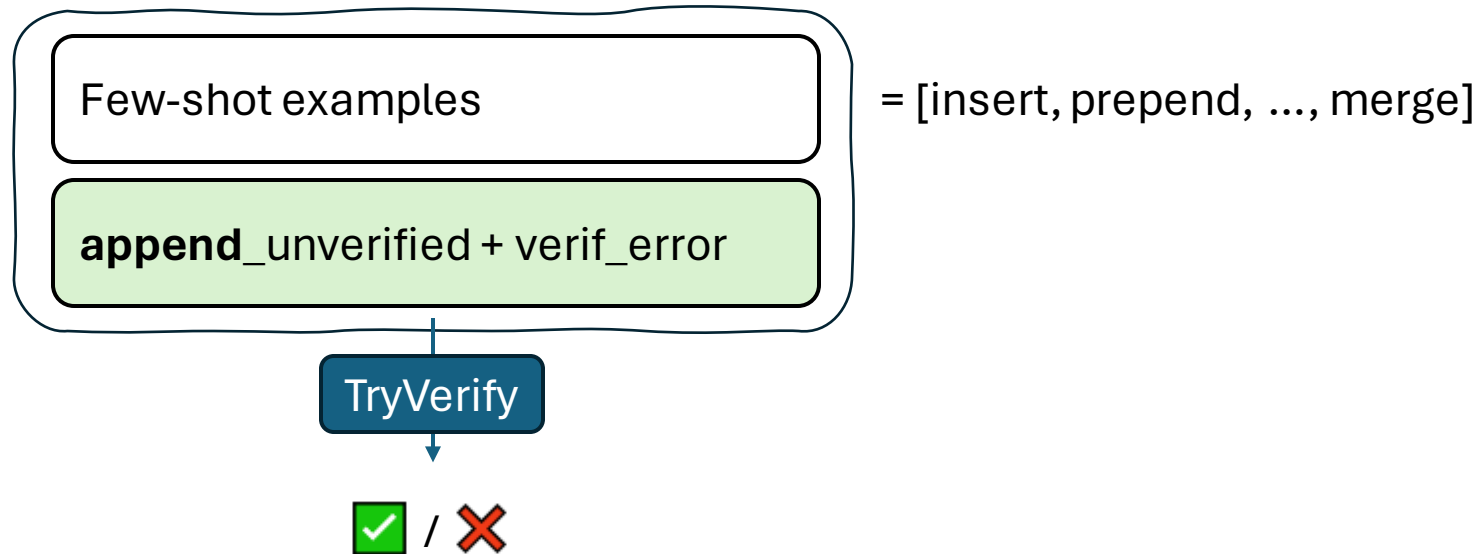
- Few-shot examples include all but the example we are verifying

Experiment 1: Few-shot prompting

- Given examples of verified programs, can the model verify an unseen program?

dataset = [**append**, insert, prepend, ..., merge]

- Few-shot examples include all but the example we are verifying

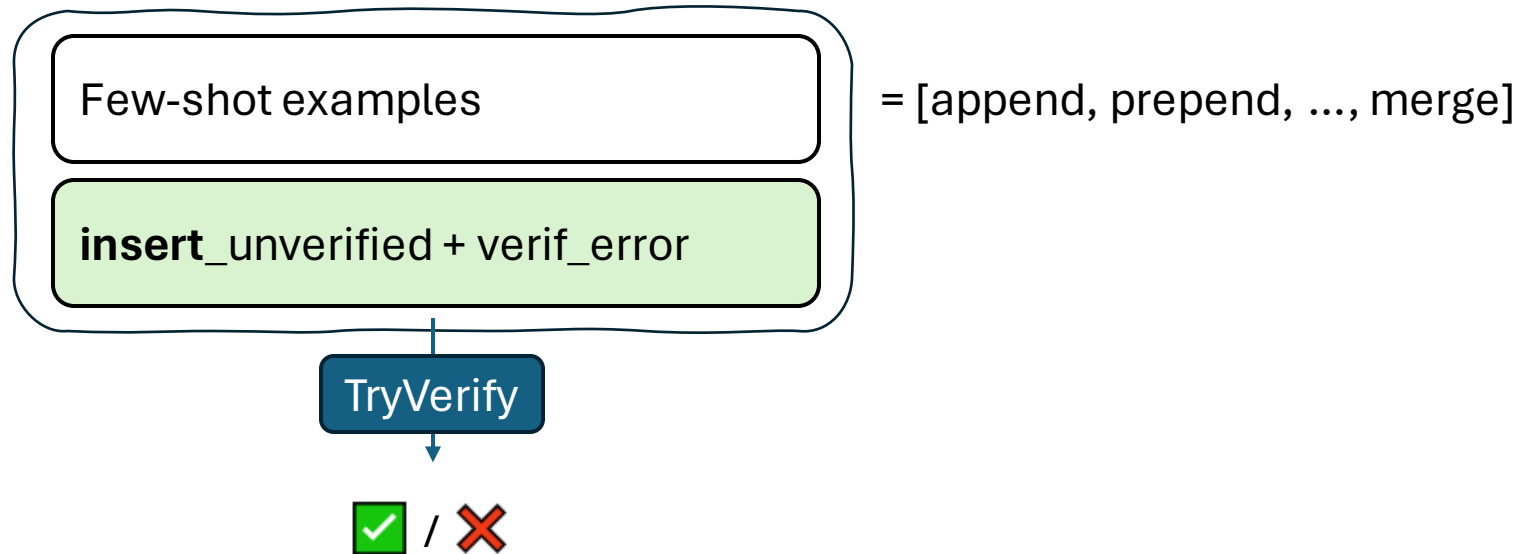


Experiment 1: Few-shot prompting

- Given examples of verified programs, can the model verify an unseen program?

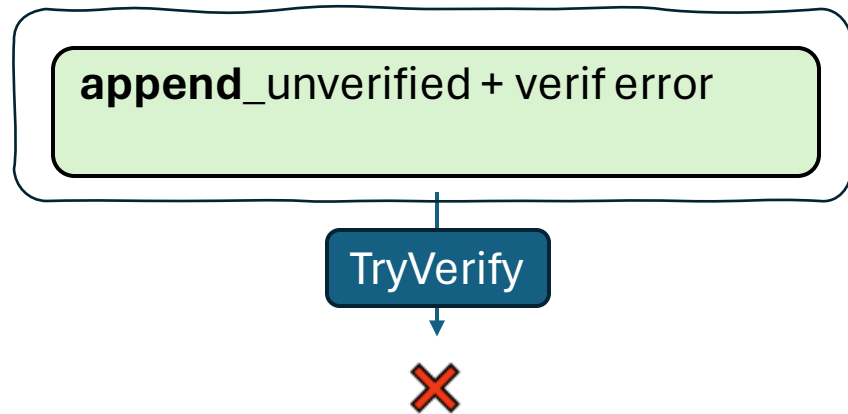
dataset = [append, **insert**, prepend, ..., merge]

- Few-shot examples include all but the example we are verifying



Experiment 2: Incremental verification

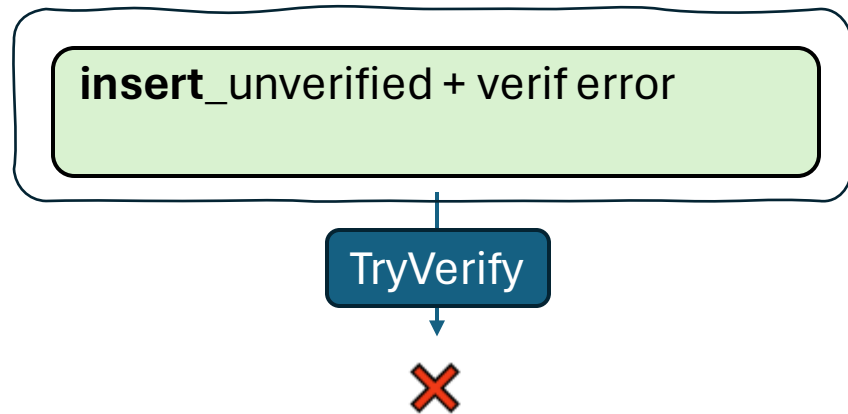
- Can we start from scratch and build up the set of verified programs?



unverified = [**append**, insert, prepend, ... , merge]
verified = []

Experiment 2: Incremental verification

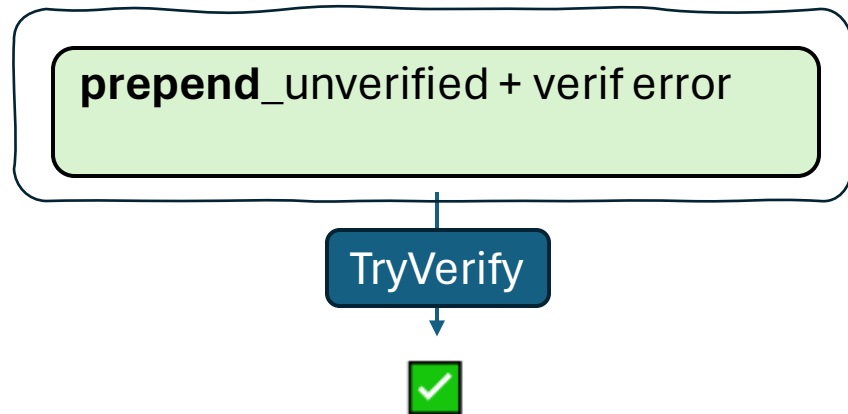
- Can we start from scratch and build up the set of verified programs?



unverified = [append, **insert**, prepend, ... , merge]
verified = []

Experiment 2: Incremental verification

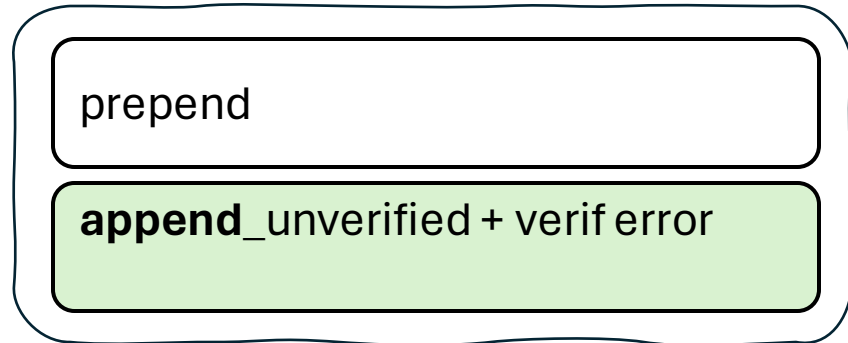
- Can we start from scratch and build up the set of verified programs?



unverified = [append, insert, **prepend**, ... , merge]
verified = []

Experiment 2: Incremental verification

- Can we start from scratch and build up the set of verified programs?



unverified = [**append**, insert, ... , merge]
verified = [prepend]

- Bootstrap verification with some simple methods, build up to more complex methods

Dataset

- 50 methods across 3 predicates verified for *memory safety* in Nagini

list: insert, remove, reverse, merge, merge_sort ... (N = 18, N_{iter} = 2)

tree: insert, contains, height ... (N = 11)

lseg: list methods + iterative versions (N=21, N_{iter} = 9)

Few-shot prompting

Percentage of methods verified: GPT-4:

	System prompt 1
list	94.4
tree	90.9
lseg	76.2
Average	84.0

System prompt 1

Few-shot examples

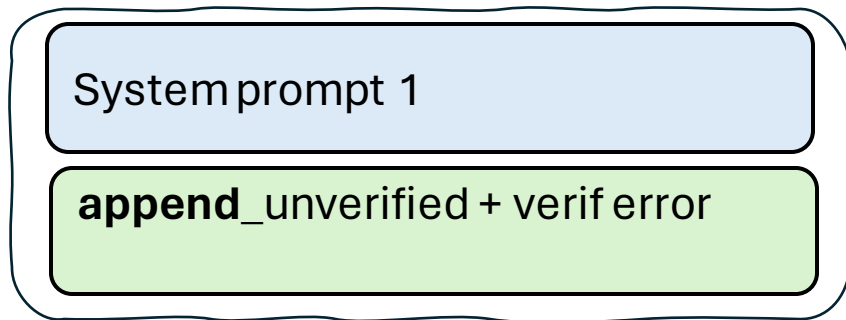
append_unverified + **verif_error**

- Brief explanation of the verification task
- Available constructs: Fold(), Unfold(), Invariant()
- The **list** predicate definition

Incremental Verification

Percentage of methods verified: GPT-4:

	System prompt 1
list	24.0
tree	36.3
Average	28.7



Without access to few-shot examples, the model struggles to bootstrap

System prompt 2

Problem:

- The model tries to infer functional specs, makes syntax errors, uses @ContractOnly
- Struggles to infer the first iterative method
- Some methods easier to infer using *Unfolding* rather than *Unfold / Fold*

Add to the system prompt:

- Basic example for an *unrelated* predicate
- *Semantics* of the available constructs incl. Invariant
- Example demonstrating equivalence of *Unfolding* and *Unfold/Fold*

```

def reverse(head: Node) -> Optional[Node]:
    """Reverse the list and return the new head."""
    Requires(is_list(head))
    Ensures(Implies(Result() is not None, is_list(Result()))))
    Unfold(is_list(head))
    if head.next is None:
        Fold(is_list(head))
        return head
    ## missing Fold(is_list(head)) here
    prev = None # type: Optional[Node]
    ptr = head # type: Optional[Node]
    while ptr != None:
        Invariant(Implies(ptr is not None, is_list(ptr)))
        ...

```

✘ Loop invariant might not hold on entry. There might be insufficient permission to access `is_list(ptr)`

```

def reverse(head: Node) -> Optional[Node]:
    """Reverse the list and return the new head."""
    Requires(is_list(head))
    Ensures(Implies(Result() is not None, is_list(Result()))))

    if Unfolding(is_list(head), head.next) is None:
        return head

    prev = None # type: Optional[Node]
    ptr = head # type: Optional[Node]
    while ptr != None:
        Invariant(Implies(ptr is not None, is_list(ptr)))
        ...

```

✔ Verification successful

Incremental Verification

Percentage of methods verified: GPT-4:

	System prompt 1	System prompt 2
list	24.0	93.3
tree	36.3	87.8
Average	28.7	90.6

Few-shot prompting

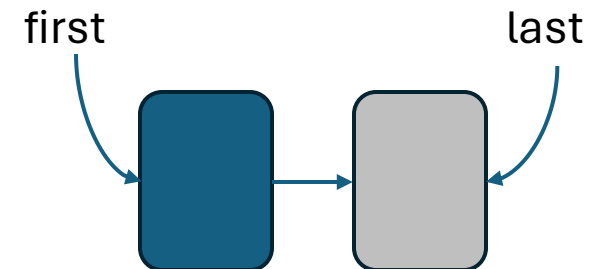
Percentage of methods verified: GPT-4:

	System prompt 1	System prompt 2
list	94.4	100.0
tree	90.9	100.0
lseg	76.2	80.1
Average:	84.0	90.0

Difficulty of lseg

```
lseg(first, last) =  
  first != last ⇒ acc(first.val) and acc(first.next) and lseg(first.next, last)
```

```
def remove_last(first: Optional[Node], last: Node) ->  
Optional[Node]:  
  """Remove the last node from the list and return the new last"""  
  Requires(lseg(first, last))  
  Ensures(lseg(first, Result()))  
  ...  
  Unfold(lseg(first, last))  
  if first.next is last:  
    Fold(lseg(first, first))  
    return first  
  last = remove_last(first.next, last)  
  ...
```



Fine-tuning

Dataset generation and fine-tuning an open-source LLM

Fine-tuning

- Use data to *update* the weights of a pre-trained model to improve its performance on our task
- Motivation:
 - Explore the use of verifier to assist in fine-tuning
 - Zero-shot inference (no few shot examples needed)
 - Token limit
- Pre-trained model: CodeLlama-7B^[1]
 - Initialized from Llama-2-7B and fine-tuned on 500B code tokens

[1] Code Llama: Open Foundation Models for Code (Meta AI, 2023)

Dataset generation

- The root dataset is insufficient
 - Contains too few examples
 - Contains no *partially verified* examples
- Hence, generate a larger dataset
 - Remove all combinations of spec statements

m specs $\Rightarrow 2^m$ examples

```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    Requires(is_list(head))
    Ensures(is_list(head))
    Unfold(is_list(head))
    if head.next is None:
        n = Node(val)
        Fold(is_list(n))
        head.next = n
    else:
        append(head.next, val)
        Fold(is_list(head))
```

Fold might fail. There might be insufficient permission to access `is_list(head.next)` on line 11.4

Dataset generation cont'd

1. Remove combination of specs

```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    Requires(is_list(head))
    Ensures(is_list(head))
    Unfold(is_list(head))
    if head.next is None:
        n = Node(val)
        Fold(is_list(n))
        head.next = n
    else:
        append(head.next, val)
    Fold(is_list(head))
```

2. Apply prompt format:

```
{System prompt 1}

### Unverified program:
{unverified program}

### Verification error:
{error}

### Verified program:
{verified program}
```

Fine-tuned model

- Performance of pre-trained model vs. our fine-tuned model on the *test set*
- N_train = 33 (4611)
N_test = 17 (list = 6, tree = 3, lseg = 8)

	Pre-trained	Fine-tuned 1
list	0.0	66.7
tree	0.0	33.3
lseg	0.0	25.0
Average (N = 17)	0.0	41.2

Pitfalls

- All examples in the training data are of the form fewer specs => more specs
So the model never *deletes* an extraneous spec statement

```
if Unfolding(lseg(ptr, last), ptr.val) == val:  
    Fold(lseg(ptr, last))  
    join(first, ptr, last)
```


Fold might fail...

- Inserts Invariant(...) in recursive methods

```
def insert(node: TreeNode, key: int) -> None:  
    """Insert a node with given key into a binary tree."""  
    ...  
    if key < node.key:  
        Invariant(tree(node))  
    ...
```

Adding spurious specs

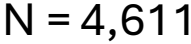
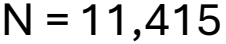
```
def append(head: Node, val: int) -> None:
    """Append a new node with the given
    value to the end of the list."""
    Requires(is_list(head))
    Ensures(is_list(head))
    Unfold(is_list(head))
    Fold(is_list(head))
    if head.next is None:
        n = Node(val)
        Fold(is_list(n))
        head.next = n
    else:
        append(head.next, val)
    Fold(is_list(head))
```



- After adding spurious specs to some examples, and running dataset generation, we fine-tuned a second model

Fine-tuned model

- Performance of pre-trained model vs. our fine-tuned models on the *test set*
- N_train = 33
N_test = 17 (list = 6, tree = 3, lseg = 8)

	Pre-trained	Fine-tuned 1 	Fine-tuned 2 
list	0.0	66.7	66.7
tree	0.0	33.3	100.0
lseg	0.0	25.0	62.5
Average	0.0	41.2	70.5

- Both fine-tuned models perform 100% on training data
- Fine-tuned 2 learns to delete specs, but not in all cases

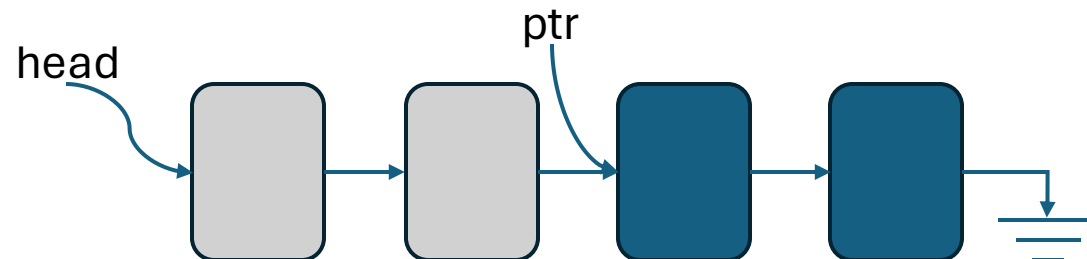
Data imbalance

```
Fold(is_list(head))  
while ptr is not None:  
    Invariant(is_list(head))  
    Invariant(is_list(ptr))  
    ...  
    join(head, tmp, ptr)
```

Code produced by fine-tuned model for
list::drop_iter

```
Fold(lseg(head, ptr))  
while ptr is not None:  
    Invariant(lseg(head, ptr))  
    Invariant(lseg(ptr, None))  
    ...  
    join(head, tmp, ptr)
```

Invariant and structure common to
many iterative **lseg** methods



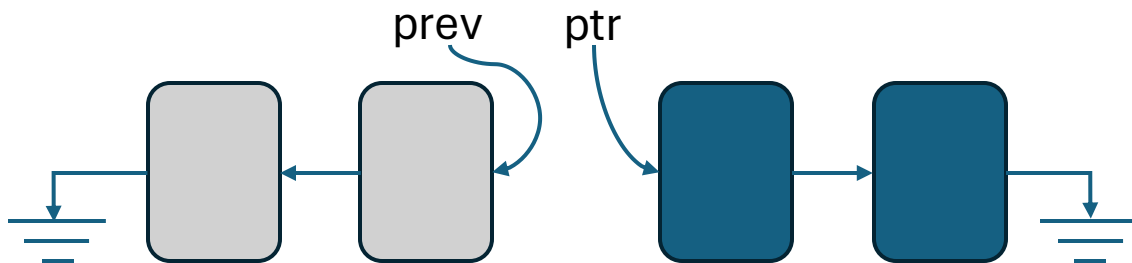
Invariant(Implies(ptr is not None, is_list(ptr)))

Training data: iterative methods
list: 1
lseg: 6

Generalization

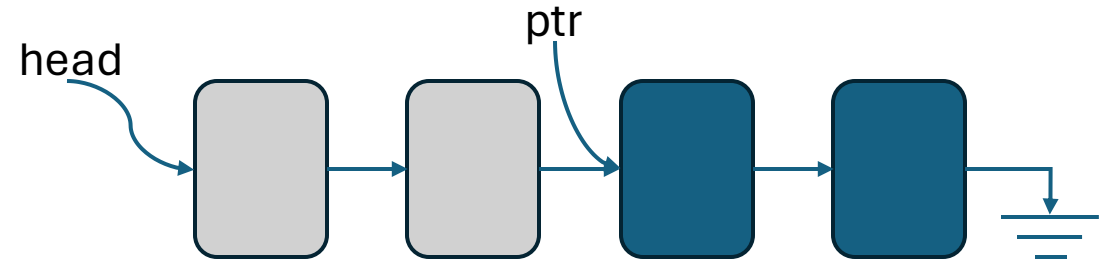
```
while ptr is not None:  
    Invariant(lseg(prev, None))  
    Invariant(lseg(ptr, None))
```

Code produced by *fine-tuned* model for **lseg::reverse**



```
while ptr is not None:  
    Invariant(lseg(head, ptr))  
    Invariant(lseg(ptr, None))
```

Invariant common to most **lseg** methods



Dataset

- 50 methods across 3 predicates verified for *memory safety* in Nagini

list: insert, remove, reverse, merge, merge_sort ... (N = 18, N_{iter} = 2)

tree: insert, contains, height ... (N = 11)

lseg: list methods + iterative versions (N=21, N_{iter} = 9)

16

Few-shot prompting

Percentage of methods verified: GPT-4:

	System prompt 1	System prompt 2
list	94.4	100.0
tree	90.9	100.0
lseg	76.2	80.1
Average:	84.0	90.0

22

Incremental Verification


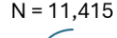
Percentage of methods verified: GPT-4:

	System prompt 1	System prompt 2
list	24.0	93.3
tree	36.3	87.8
Average	28.7	90.6

21

Fine-tuned model

- Performance of pre-trained model vs. our fine-tuned models on the *test set*
- N_{train} = 33
N_{test} = 17 (list = 6, tree = 3, lseg = 8)

	Pre-trained	Fine-tuned 1 	Fine-tuned 2 
list	0.0	66.7	66.7
tree	0.0	33.3	100.0
lseg	0.0	25.0	62.5
Average	0.0	41.2	70.5

- Both fine-tuned models perform 100% on training data

31