# Towards LLM-support for Deductive Verification of Java Programs

## Theorem Proving and Machine Learning in the Age of LLMs

***Results published at ISoLA 2024 and NSE 2025***

**Samuel Teuber**, Bernhard Beckert

Institute of Information Security and Dependability (KASTEL)

2025

# Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code?**

# Motivation: Can LLMs Generate Specifications?

```
/*@ normal_behavior
  @  ensures (\forall int j;j >= 0 && j < a.length; \result >= a[j]);
  @  ensures a.length > 0
  @      ==> (\exists int j;j >= 0 && j < a.length; \result == a[j]);
  @*/
public static /*@ pure */ int max(int[] a) {
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    /*@
      @
      @
      @
      @
      @*/
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

**Verification requires Loop Invariant**

- Holds before first loop iteration
- Preserved by loop iteration
- Implies post condition

**Additionally:**

- Loop *Variant*
- Assignable Heap Variables

# Motivation: Can LLMs Generate Specifications?

```
/*@ normal_behavior
  @  ensures (\forall int j;j >= 0 && j < a.length; \result >= a[j]);
  @  ensures a.length > 0
  @      ==> (\exists int j;j >= 0 && j < a.length; \result == a[j]);
  @*/
public static /*@ pure */ int max(int[] a) {
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    /*@  loop_invariant 0 <= i && i <= a.length;
      @  loop_invariant (\forall int k; 0 <= k && k < i; max >= a[k]);
      @  loop_invariant (\exists int k; 0 <= k && k < i; max == a[k]);
      @  decreases a.length - i;
      @  assignable max, i;
      @*/
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

# Motivation: Can LLMs Generate Specifications?

Large Language Models have seen tremendous success in recent years

GitHub Copilot & Co show: LLMs can **generate code**

But can they **specify code?**

Writing auxiliary spec yourself [ChatGPT]

# Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code?**

Having ChatGPT write auxiliary spec [ChatGPT]

# Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code?**

**Let's ask ChatGPT:**

Do you know JML, the Java Modeling Language?

Yes, I am familiar with JML (Java Modeling Language). JML is a formal specification language for Java programs. [...] JML is typically used in conjunction with formal verification tools, such as ESC/Java or **KeY**, to check that the code meets its specifications.

# The Program Verifier KeY

Deductive verification

100% Java Card

Java Modeling Language (JML)

**Numerous Case Studies**:

- TimSort (OpenJDK)
- `LinkedList` (OpenJDK)
- Super Scalar Sample Sort

Modular Reasoning

Verification Methodology:

Theorem Proving for **Java Dynamic Logic**

collaboration of KIT, TU Darmstadt, Chalmers University

Ahrendt u. a. 2016

# Java Dynamic Logic by Example

```java
class MyClass {
    int arr[];
    int maxOfFirstTwo() {
        if (this.arr[0] > this.arr[1]) {
            return this.arr[0];
        } else {
            return this.arr[1];
        }
    }}
```

JavaDL allows reasoning about **real-world Java** programs.
We can verify programs by **proving the validity of formulas via a calculus**:

[Beckert 2000; Beckert und Platzer 2006; Ahrendt u. a. 201[illegible]]

# Java Dynamic Logic by Example

```
class MyClass {
    int arr[];
    int maxOfFirstTwo() {
        if (this.arr[0] > this.arr[1]) {
            return this.arr[0];
        } else {
            return this.arr[1];
        }
    }}
```

JavaDL allows reasoning about **real-world Java** programs.
We can verify programs by **proving the validity of formulas via a calculus**:

```
res = maxOfFirstTwo();
```

Java Program

[Beckert 2000; Beckert und Platzer 2006; Ahrendt u. a. 2016]

# Java Dynamic Logic by Example

```java
class MyClass {
    int arr[];
    int maxOfFirstTwo() {
        if (this.arr[0] > this.arr[1]) {
            return this.arr[0];
        } else {
            return this.arr[1];
        }
}}
```

JavaDL allows reasoning about **real-world Java** programs.
We can verify programs by **proving the validity of formulas via a calculus**:

$$[\texttt{res = maxOfFirstTwo();}]\ \texttt{res} = \texttt{arr[0]}$$

Java Program — postcondition — after every program run

[Beckert 2000; Beckert und Platzer 2006; Ahrendt u. a. 201]

# Java Dynamic Logic by Example

```
class MyClass {
    int arr[];
    int maxOfFirstTwo() {
        if (this.arr[0] > this.arr[1]) {
            return this.arr[0];
        } else {
            return this.arr[1];
        }
    }}
```

JavaDL allows reasoning about **real-world Java** programs.
We can verify programs by **proving the validity of formulas via a calculus**:

$$\texttt{heap\_assumptions}() \land \cdots \land \texttt{arr[0]} > \texttt{arr[1]} \rightarrow [\, \texttt{res = maxOfFirstTwo();} \,]\; \texttt{res} = \texttt{arr[0]}$$

precondition — Java Program — after every program run — postcondition

[Beckert 2000; Beckert und Platzer 2006; Ahrendt u. a. 2016]

# Java Dynamic Logic by Example

```java
class MyClass {
    int arr[];
    int maxOfFirstTwo() {
        if (this.arr[0] > this.arr[1]) {
            return this.arr[0];
        } else {
            return this.arr[1];
        }
    }}
```

**Proof Interaction:**

- Interactive (manual) application of proof rules

JavaDL allows reasoning about **real-world Java** programs.
We can verify programs by **proving the validity of formulas via a calculus**:

$\texttt{heap\_assumptions}() \land \cdots \land \texttt{arr[0]} > \texttt{arr[1]} \rightarrow [\texttt{res = maxOfFirstTwo();}]\ \texttt{res} = \texttt{arr[0]}$

precondition — Java Program — after every program run — postcondition

[Beckert 2000; Beckert und Platzer 2006; Ahrendt u. a. 2016]

# Java Dynamic Logic by Example

```java
class MyClass {
    int arr[];
    int maxOfFirstTwo() {
        if (this.arr[0] > this.arr[1]) {
            return this.arr[0];
        } else {
            return this.arr[1];
        }
}}
```

**Proof Interaction:**

- Interactive (manual) application of proof rules
- Annotation of source code
  **auto-active verification**
  (Leino und Moskal 2010)

JavaDL allows reasoning about **real-world Java** programs.
We can verify programs by **proving the validity of formulas via a calculus**:

$$\texttt{heap\_assumptions()} \land \cdots \land \texttt{arr[0]} > \texttt{arr[1]} \rightarrow [\,\texttt{res = maxOfFirstTwo();}\,]\ \texttt{res} = \texttt{arr[0]}$$

precondition — Java Program — after every program run — postcondition

[Beckert 2000; Beckert und Platzer 2006; Ahrendt u. a. 201[illegible]]

# Java Modelling Language

- Specification Language for Java
- Design by Contract Paradigm
- Rich set of possible first-order annotations:
  - Hoare-Style pre- and post-conditions
  - Invariants
  - Asserts
  - Class-Invariants
- Supported by numerous tools for Java verification

```
/*@ normal_behavior
  @  ensures (\forall int j;j >= 0 && j <
  @  ensures a.length > 0
  @        ==> (\exists int j;j >= 0 && j <
  @*/
public static /*@ pure */ int max(int[] a)
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    /*@  loop_invariant 0 <= i && i <= a.l
      @  loop_invariant (\forall int k; 0
      @  loop_invariant (\exists int k; 0
      @  decreases a.length - i;
      @  assignable max, i;
      @*/
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

# LLMs for Deductive Java Verification

**Large Language Models**

- May produce output that **is not correct**
- “Reasoning” is **not rigorous**
- **Inconsistent Answers**

# LLMs for Deductive Java Verification

**Large Language Models**

- May produce output that **is not correct**
- “Reasoning” is **not rigorous**
- **Inconsistent Answers**

**Deductive Verifiers**

- Lack “common sense”
- Symbolic techniques:
  Not good at **“guessing”** annotations from context

# LLMs for Deductive Java Verification

**Large Language Models**

- May produce output that **is not correct**
- "Reasoning" is **not rigorous**
- **Inconsistent Answers**

**Deductive Verifiers**

- Lack "common sense"
- Symbolic techniques:
  Not good at **"guessing"** annotations from context

**Objective: An Intersymbolic AI approach to Program Verification**

Combine LLMs and Deductive Verification so that **weaknesses cancel out**

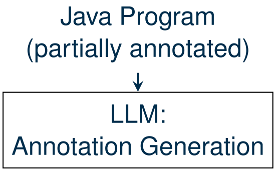
# LLM-based generation of JML: Approach

Java Program
(partially annotated)

```
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}


int g(int x) {
    return x+x;
}
```

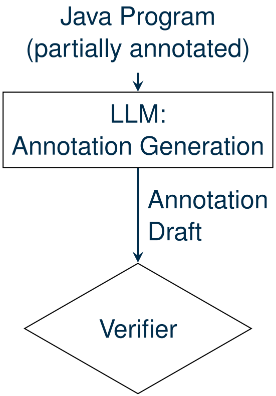
# LLM-based generation of JML: Approach




```java
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}


int g(int x) {
    return x+x;
}
```

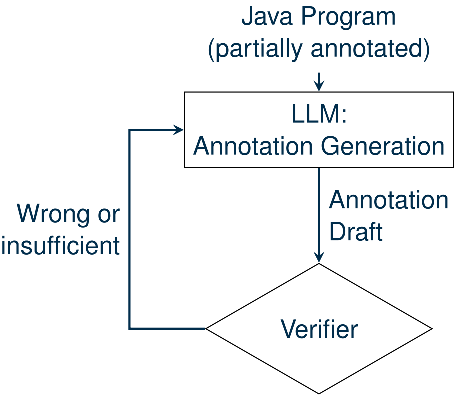
# LLM-based generation of JML: Approach




```java
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}

//@ ensures x == 2 ==> \result == 4;
int g(int x) {
    return x+x;
}
```

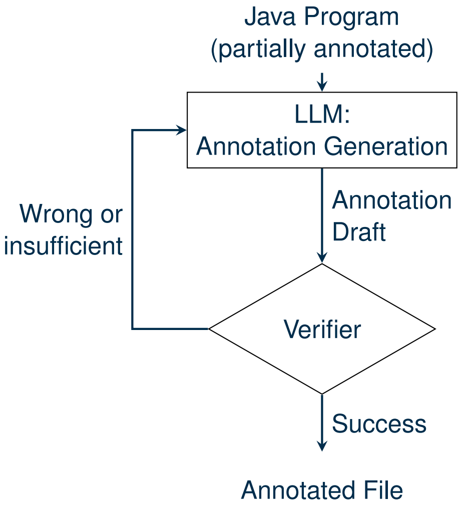
# LLM-based generation of JML: Approach




```
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}

//@ ensures x == 2 ==> \result == 4;
int g(int x) {
    return x+x;
}
```

# LLM-based generation of JML: Approach




```java
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}

//@ ensures \result == 2*x;
int g(int x) {
    return x+x;
}
```

# Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

# Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

**Benchmark Categories:**

- Generate top-level contract for isolated method
  (Java $\Rightarrow$ JML)

# Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

**Benchmark Categories:**

- Generate top-level contract for isolated method
  (Java $\Rightarrow$ JML)
- Generate auxilliary annotations for given top-level spec
  (Java+JML $\Rightarrow$ JML)
  - Loop invariant (given method contract)
  - Contract of callee-method (given caller contract)

# Evaluation Benchmark: Features

| | Isolated Methods | Sub-Methods | Invariants | Total | |
|---|---|---|---|---|---|
| **No. of benchmarks** | 36 | 27 | 14 | 77 | |
| **JML features** | | | | | |
| Quantifiers | 32 | 24 | 14 | 70 | 91% |
| Non-empty assignable | 24 | 17 | 11 | 52 | 68% |
| Reference to pre-state (\old) | 18 | 11 | 8 | 37 | 48% |
| (Pure) Method calls in spec | 10 | 7 | 7 | 24 | 31% |
| **Java features** | | | | | |
| Arrays | 30 | 23 | 13 | 66 | 86% |
| Field access | 11 | 7 | 3 | 21 | 27% |

# Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

**Benchmark Categories:**

- Generate top-level contract for isolated method
  (Java $\Rightarrow$ JML)
- Generate auxilliary annotations for given top-level spec
  (Java+JML $\Rightarrow$ JML)
  - Loop invariant (given method contract)
  - Contract of callee-method (given caller contract)

| Category | # Benchmarks | $\mu \pm \sigma$ of success rate (%) | |
|---|---|---|---|
| | | **GPT 3.5** ($n = 10$) | **GPT 4o** ($n = 3$) |
| Isolated Method | 36 | $52.2 \pm 4.3$ | $\mathbf{62.0 \pm 1.6}$ |
| Submethods | 14 | $19.3 \pm 12.1$ | $\mathbf{40.5 \pm 4.1}$ |
| Invariants | 27 | $37.0 \pm 7.4$ | $\mathbf{67.9 \pm 5.7}$ |

# Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

**Benchmark Categories:**

- Generate top-level contract for isolated method
  (Java ⇒ JML)
- Generate auxilliary annotations for given top-level spec
  (Java+JML ⇒ JML)
  - Loop invariant (given method contract)
  - Contract of callee-method (given caller contract)

| **Category** | **# Benchmarks** | **$\mu \pm \sigma$ of success rate (%)** | |
|---|---|---|---|
| | | **GPT 3.5** ($n$ = 10) | **GPT 4o** ($n$ = 3) |
| Isolated Method | 36 | $52.2 \pm 4.3$ | **$62.0 \pm 1.6$** |
| Submethods | 14 | $19.3 \pm 12.1$ | **$40.5 \pm 4.1$** |
| Invariants | 27 | $37.0 \pm 7.4$ | **$67.9 \pm 5.7$** |

**No elaborate prompt engineering yet!**

- **Objective:** Evaluate baseline performance
- Not enough benchmarks

# Evaluation: Isolated Methods

| Category | # Benchmarks | $\mu \pm \sigma$ of success rate (%) | |
|---|---|---|---|
| | | GPT 3.5 ($n$ = 10) | GPT 4o ($n$ = 3) |
| Isolated Method | 36 | $52.2 \pm 4.3$ | $\mathbf{62.0 \pm 1.6}$ |

**Success criterion: KeY proves code satisfies generated spec**

**Manual Inspection:** Spec **adequately specifies** code behavior
Sometimes incomplete

# Evaluation: Isolated Methods — Errors

| Error Category | Share of error (%) | |
|---|---|---|
| | GPT 3.5 | GPT 4o |
| **Syntactic Errors** | | |
| Loop Invariant Generation | 4.3 | 1.3 |
| Unknown variable names | 11.8 | 2.2 |
| Incorrect usage of `\result` | 1.1 | 0.3 |
| Other parsing errors | 17.1 | 10.7 |
| **Semantic Errors** | | |
| Incomplete proof | 49.5 | 70.3 |
| Timeout of Verifier | 16.0 | 15.1 |

**Observation**

GPT 4o tends to make **semantic** errors (much less syntactic errors).

# Demo

Backup

# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

**Concrete Counterexamples**

Via **bounded model checking**

# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

**Concrete Counterexamples**
Via **bounded model checking**

The provided invariant **does not hold for the following instantiations of variables**:
k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.
...
Here's the corrected code:
**<wrong invariant>**

# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

**Concrete Counterexamples**
Via **bounded model checking**

The provided invariant **does not hold for the following instantiations of variables**:
k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.
...
Here's the corrected code:
**<wrong invariant>**

**Feedback from Proof State**

- Pass on parser errors
- Pass on information on open proof branches

# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

## Concrete Counterexamples

Via **bounded model checking**

The provided invariant **does not hold for the following instantiations of variables**:
k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.
...
Here's the corrected code:
**<wrong invariant>**

## Feedback from Proof State

- Pass on parser errors
- Pass on information on open proof branches

**Focus:** Auxilliary Specifications

**Sampling** (pass@N) **vs. Feedback**

10 rounds with GPT 4o

# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

**Concrete Counterexamples**

Via **bounded model checking**

The provided invariant **does not hold for the following instantiations of variables**:
k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.
...
Here's the corrected code:
**<wrong invariant>**

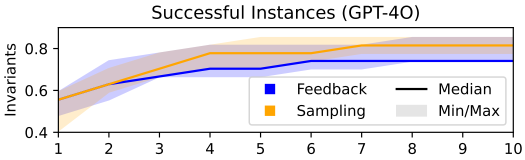
**Feedback from Proof State**

- Pass on parser errors
- Pass on information on open proof branches

**Focus:** Auxilliary Specifications

**Sampling** (pass@N) **vs. Feedback**

10 rounds with GPT 4o



# Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

**Concrete Counterexamples**

Via **bounded model checking**

The provided invariant **does not hold for the following instantiations of variables**:
k = 0
a = [0, 0
i = 1

You're
riant tc
**length**
should remain unchanged.
...
Here's the corrected code:
**<wrong invariant>**

**Feedback from Proof State**

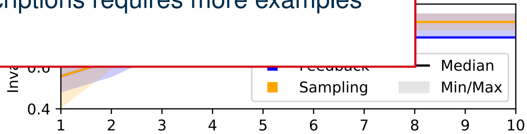
- Pass on parser errors
- Pass on information on open proof branches

**Focus:** Auxilliary Specifications

**back**

**We need more benchmarks for conclusive results**

Prompt Engineering for proof state descriptions requires more examples



# Feedback vs. Sampling
## What is the right metric?

### Classic Verification

Two verification techniques:

(A) 5 iterations, 2 seconds CPU time/iteration

(B) 3 iterations, 4 seconds CPU time/iteration

$\Rightarrow$ **(A) is better**

# Feedback vs. Sampling
## What is the right metric?

**Classic Verification**

Two verification techniques:
- (A) 5 iterations, 2 seconds CPU time/iteration
- (B) 3 iterations, 4 seconds CPU time/iteration

$\Rightarrow$ **(A) is better**

API usage **hides** computational cost!

# Feedback vs. Sampling

## What is the right metric?

### Classic Verification

Two verification techniques:

(A) 5 iterations, 2 seconds CPU time/iteration

(B) 3 iterations, 4 seconds CPU time/iteration

$\Rightarrow$ **(A) is better**

API usage **hides** computational cost!

**What we know:**

Computational cost increases with token count

Initial Query: $I$ tokens

LLM Output: $O$ tokens

Feedback: $F$ tokens

**Sampling:** $n\left(I+O\right) \in \mathcal{O}\left(n\right)$ tokens

**Feedback:** $n\left(I+O\right) + \frac{n(n-1)}{2}\left(O+F\right) \in \mathcal{O}\left(n^2\right)$ tokens

# Feedback vs. Sampling
## What is the right metric?

### Classic Verification

Two verification techniques:
- (A) 5 iterations, 2 seconds CPU time/iteration
- (B) 3 iterations, 4 seconds CPU time/iteration

$\Rightarrow$ **(A) is better**

API usage **hides** computational cost!

**What we know:**
Computational cost increases with token count

Initial Query: $I$ tokens

LLM Output: $O$ tokens

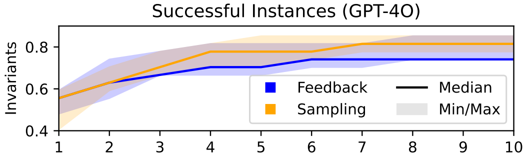
Feedback: $F$ tokens

**Sampling:** $n\,(I + O) \in \mathcal{O}\,(n)$ tokens

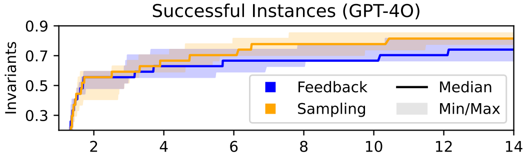
**Feedback:** $n\,(I + O) + \frac{n(n-1)}{2}\,(O + F) \in \mathcal{O}\left(n^2\right)$ tokens

**Another evaluation:**
Iterations $\longrightarrow$ *Normalized* Token Count ($\frac{\#\text{ Tokens}}{I}$)

**Iteration based:**




**Token based:**



# Related Work

This is a **rapidly** growing research field

# Related Work

This is a **rapidly** growing research field

A lot of work on **Dafny** (e.g. due to DafnyBench) Loughridge u. a. 2025

# Related Work

This is a **rapidly** growing research field

A lot of work on **Dafny** (e.g. due to DafnyBench) Loughridge u. a. 2025

AutoSpec for C and ACSL with Frama C Wen u. a. 2024

- Beyond *filling the gap*:
  Strategy for generating all method/invariant annotations
- No proof state feedback for individual annotations
- Also use ChatGPT API

# Related Work

This is a **rapidly** growing research field

A lot of work on **Dafny** (e.g. due to DafnyBench) — Loughridge u. a. 2025

AUTOSPEC for C and ACSL with Frama C — Wen u. a. 2024

- Beyond *filling the gap*:
  Strategy for generating all method/invariant annotations
- No proof state feedback for individual annotations
- Also use ChatGPT API

Joint generation of code and specification — Sun u. a. 2024

- No soundness guarantees
- Empirically: Consistency between code and spec helps

# Related Work

This is a **rapidly** growing research field

A lot of work on **Dafny** (e.g. due to DafnyBench) Loughridge u. a. 2025

AUTOSPEC for C and ACSL with Frama C Wen u. a. 2024

- Beyond *filling the gap*:
  Strategy for generating all method/invariant annotations
- No proof state feedback for individual annotations
- Also use ChatGPT API

Joint generation of code and specification Sun u. a. 2024

- No soundness guarantees
- Empirically: Consistency between code and spec helps

Dataset for JML via GitHub scraping Greiner u. a. 2024

# Conclusion

- OpenAI's models are **surprisingly good** at JML
- Iteration helps, but feedback not (yet)
- **Foundation Models:**
  - Make effective usage of ML for niche languages like JML possible
  - Cannot solve all data problems (we still need well-curated benchmark sets)

**Open Questions**

- Effective Feedback from the theorem prover
- Prompt Engineering in the presence of *"perfect checkers"*
  Sampling may not be that bad afterall?
- What is a fair comparison between an LLM approach and a "classic" approach?

# References I


[1] Wolfgang Ahrendt u. a., Hrsg. *Deductive Software Verification - The KeY Book - From Theory to Practice*. Bd. 10001. LNCS. Cham: Springer, 2016. ISBN: 978-3-319-49811-9. DOI: 10.1007/978-3-319-49812-6.

[2] Bernhard Beckert. “A dynamic logic for the formal verification of Java Card programs”. In: *International Java Card Workshop*. Springer, 2000, S. 6–24.

[3] Bernhard Beckert und André Platzer. “Dynamic Logic with Non-rigid Functions: A Basis for Object-oriented Program Verification.”. In: *IJCAR*. Hrsg. von Ulrich Furbach und Natarajan Shankar. Bd. 4130. LNCS. ISSN: 0302-9743. Springer, 2006, S. 266–280. ISBN: 3-540-37187-7. DOI: 10.1007/11814771_23.

[4] Sandra Greiner u. a. “Automated Generation of Code Contracts: Generative AI to the Rescue?” In: *Proceedings of the 23rd ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. GPCE ’24. Pasadena, CA, USA: Association for Computing Machinery, 2024, S. 1–14. ISBN: 9798400712111. DOI: 10.1145/3689484.3690738. URL: https://doi.org/10.1145/3689484.3690738.

[5] K Rustan M Leino und Michał Moskal. “Usable auto-active verification”. In: *Usable Verification Workshop*. Hrsg. von Tom Ball, Lenore Zuck und Natarajan Shankar. 2010. URL: https://fm.csl.sri.com/UV10.

[6] Chloe R Loughridge u. a. “DafnyBench: A Benchmark for Formal Software Verification”. In: *Transactions on Machine Learning Research* (2025). ISSN: 2835-8856. URL: https://openreview.net/forum?id=yBgTVWccIx.

[7] Chuyue Sun u. a. “Clover: Closed-Loop Verifiable Code Generation”. In: *AI Verification - First International Symposium, SAIV 2024, Montreal, QC, Canada, July 22-23, 2024, Proceedings*. Hrsg. von Guy Avni u. a. Bd. 14846. Lecture Notes in Computer Science. Springer, 2024, S. 134–155. DOI: 10.1007/978-3-031-65112-0_7. URL: https://doi.org/10.1007/978-3-031-65112-0%5C_7.

# References II

[8] Cheng Wen u. a. “Enchanting program specification synthesis by large language models using static analysis and program verification”. In: *International Conference on Computer Aided Verification*. Springer. 2024, S. 302–328.

[9] Haoze Wu, Clark Barrett und Nina Narodytska. “Lemur: Integrating Large Language Models in Automated Program Verification”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=Q3YaCghZNt.

# Demo (Backup)

Back to Slides

# Example: Find the Maximum

```
/*@ normal_behavior
  @ ensures (\forall int j;j >= 0 && j < a.length; \result >= a[j]);
  @ ensures a.length > 0 ==>
  @ (\exists int j;j >= 0 && j < a.length; \result == a[j]);
  @ */
public static /*@ pure */ int max(int[] a) {
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

# Example: Find the Maximum (1st try)

Given the following Java class:
[ ... ]
Please **provide a loop invariant** for the loop construct with the **comment ‘//Add invariant here‘** of the method ‘max‘. Beware this annotation has to be a loop invariant. A loop invariant **typically has the following structure**:
[Template: **loop_invariant, decreases, assignable** ]

Based on the method ‘max‘, the goal is to find the maximum element in the given array ‘a‘. [...]

```
/*@ loop_invariant 0 <= i && i <= a.length;
  @ loop_invariant (\forall int k; 0 <= k && k < i; max
        >= a[k]);
  @ decreases a.length - i;
  @ assignable i, max;
  @ */
```

**Problem:** `max` not required to be in the array

# Example: Find the Maximum (2nd try)

The provided JML does not solve the task. [...] **some proof goals were not closed**:
[...]
During verification, the following proof branches could not be closed:
**Normal Execution (_a != null)**
Please fix the JML loop invariant.

Let's refine the loop invariant [...]

```
/*@ loop_invariant a != null;
  @ loop_invariant 0 <= i && i <= a.length;
  @ loop_invariant (\forall int k; 0 <= k && k < i; max
        >= a[k]);
  @ decreases a.length - i;
  @ assignable i, max;
  @*/
```

**Problem:** same (only added `a != null`)

# Example: Find the Maximum (3rd try)

[Same message]

To refine the loop invariant further [...]

```
/*@ loop_invariant a != null;
  @ loop_invariant 0 <= i && i <= a.length;
  @ loop_invariant (\forall int k; 0 <= k && k < i; max
        >= a[k]);
  @ loop_invariant (i > 0) ==>
  @ (\exists int k; 0 <= k && k < i; max == a[k]);
  @ decreases a.length - i;
  @ assignable i, max;
  @*/
```

Invariant that is strong enough to prove the post-condition.

**Problem:** Invariant is not inductive (because of `i > 0`)

# Example: Find the Maximum (4th try)

[Same message]

Let's ensure that the JML loop invariant comprehensively captures [...]

```
/*@ loop_invariant a != null;
  @ loop_invariant 0 <= i && i <= a.length;
  @ loop_invariant (\forall int k; 0 <= k && k < i; max
      >= a[k]);
  @ loop_invariant
  @ \exists int m; m >= 0 && m < i && max == a[m];
  @ decreases a.length - i;
  @ assignable i, max;
  @ */
```

Correct invariant

**Syntax error:** ACSL `\exists ...` vs. JML `(\exists ...)`

# Example: Find the Maximum (5th try)

The provided code is not valid JML.
[...]
Error during JML parsing: Failed to parse JML fragment: **Encountered unexpected token: “\exists” “\exists” at line 4, column 20**.

Was expecting one of: [...]

[unfortunately still fails]

# Example: Find the Maximum (restart)

## Second Attempt...

[Same initial question]

In the given ‘max‘ method, the loop is responsible for [...]

```
/*@  loop_invariant 0 <= i && i <= a.length;
  @ loop_invariant (\forall int k; 0 <= k && k < i; max
      >= a[k]);
  @ decreases a.length - i;
  @ assignable max, i;
  @*/
```

**Problem:** `max` not required to be in array.

[Same feedback]

**[Correct answer]**

# More Difficult Example: Rotate an Array

```
public static void rotate(int[] a, int len) {
    int[] b = new int[a.length];
    int i = 0;
    /*@ [...] @*/
    for (i = 0; i < len; i++) {
        b[i] = a[a.length - len + i];
    }
    /*@ [...] @*/
    for (i = len; i < a.length; i++) {
        b[i] = a[i - len];
    }
    /*@ [...] @*/
    for (i = 0; i < a.length; i++) {
        a[i] = b[i];
    }
}
```

# More Difficult Example: Rotate an Array

## Correct specification generated by GPT 4o

```
/* @ normal_behavior
   @
   @ requires a != null && 0 <= len && len <= a.length;
   @
   @ assignable a[*];
   @
   @ ensures (\forall int i; 0 <= i && i < len;
   @                         a[i] == \old(a[a.length - len + i]));
   @ ensures (\forall int i; len <= i && i < a.length;
   @                         a[i] == \old(a[i - len]));
   @*/
```

# Demo (Backup)

Back to Slides

# Evaluation: Isolated Methods

| Category | # Benchmarks | $\mu \pm \sigma$ of success rate (%) | |
|---|---|---|---|
| | | GPT 3.5 ($n$ = 10) | GPT 4o ($n$ = 3) |
| Isolated Method | 36 | $52.2 \pm 4.3$ | **$62.0 \pm 1.6$** |

**Success criterion: KeY proves code satisfies generated spec**

**Manual Inspection:** Spec **adequately specifies** code behavior
Sometimes incomplete

**Repetition/Feedback helps**

- 75% of benchmarks successful (over 10 runs, GPT 3.5)
- Feedback from the verifier can help

Step distribution for verified contracts