Leveraging Code Embeddings to Identify and Address Blind Spots in Benchmark Creation

Charles Moloney

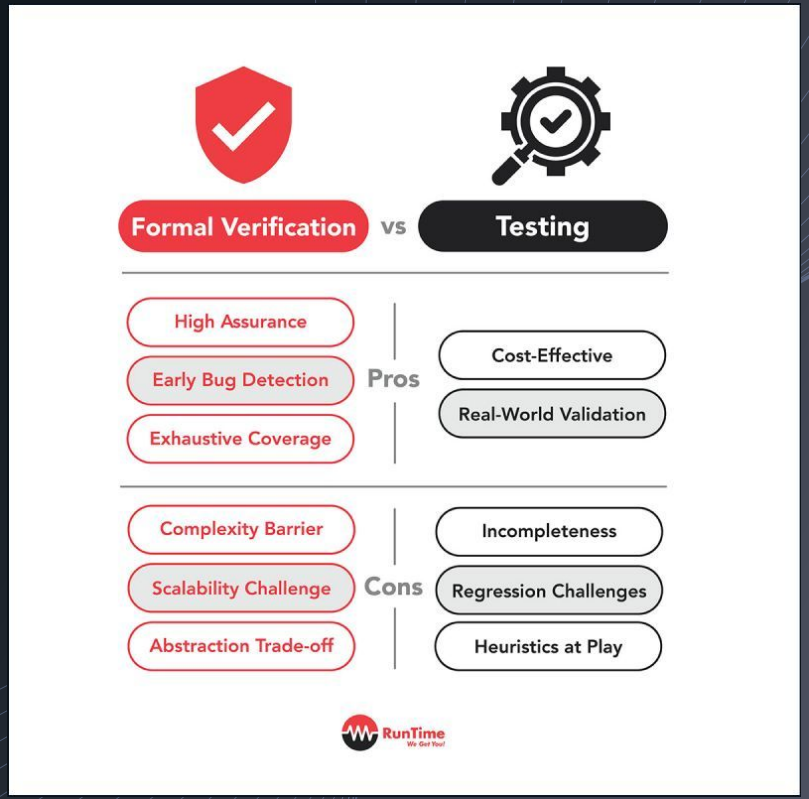
Advisor: Robert Dyer

University of Nebraska–Lincoln, Nebraska, USA

Motivation

Formal Verification is immensely useful.

- Formal Verification: proving or disproving a property of a system using formal mathematical methods
- Identifying bugs and proving program correctness can prevent disastrous consequences
  - Essential tool for mission-critical systems
  - Mathematically-assured soundness > incomplete testing
- **Developing a formal verifier is expensive and time-consuming**



SV-COMP (COMPetition on Software Verification)

- Community-led competition for comparing the state-of-the-art in Java and C formal verifiers
- Publicly available repository of formal verification benchmarks
  - 36,402 C tasks, 1,731 Java tasks (across 1,031 files)
- Unified platform (BenchExec) for executing and evaluating verifiers

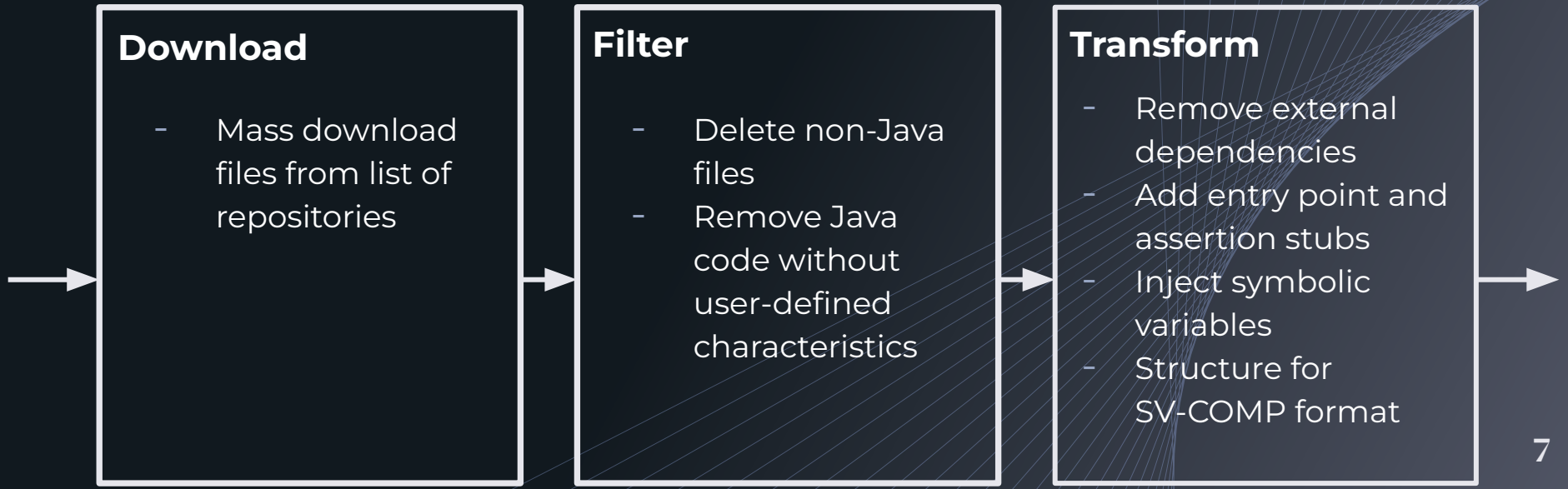
- **Many SV-COMP benchmarks are sourced from verifier regression suites - may not represent the bulk of software projects**
- **Need for scalable, robust method for creating benchmarks that will spur verifier development**

How can we automate and accelerate the production of software verification benchmarks derived from real-world code?

Background

Our tool: *ARG-V* (Adaptable Realistic benchmark Generator for Verification)

- Open-Source research tool for automating the downloading, filtering, and transformation of real-world code for SV-COMP
- Modifies code based off its Abstract Syntax Tree (AST) using a series of Visitors



Demo class (untransformed)

```
1  import java.util.Map;
2  import util.ExternalClass;

3  public class DemoClass {

4      public static boolean importMapValuesIfKeyEmpty(Map<String, Integer> map
         , String key) {
5          if (!map.containsKey(key)) {
6              map.put(key, ExternalClass.defaultInt());
7        return true;
8          }
9      return false;
10     }

11 }
12
```

Demo class After *ARG-V*

```
1  /** [ARG-V](https://arg-v.dev) was used to collect, filter, and transform
       these benchmarks automatically */
2  import org.sosy_lab.sv_benchmarks.Verifier;
3  import java.util.Map;

4  /** filtered by ARG-V */
5   public class Main {

6      /** ARG-V: suitable */
7     public static boolean importMapValuesIfKeyEmpty(Map<String, Integer> map,
          String key) {
8           if (!map.containsKey(key)) {
9               map.put(key, Verifier.nondetInt());
10        assert true; //inline assert generated by ARG-V
11        return true;
12          }
13      assert true; //inline assert generated by ARG-V
14      return false;
15      }

16    /** This main was generated by ARG-V */
17     public static void main(String[] args) throws Exception{
18      importMapValuesIfKeyEmpty((java.util.Map<java.lang.String, java.lang.
          Integer>)null, Verifier.nondetString());
19    }

20 }
21
```

Demo class after manual transformation

```java
/** [ARG-V](https://arg-v.dev) was used to collect, filter, and transform
    these benchmarks automatically */
import org.sosy_lab.sv_benchmarks.Verifier;
import java.util.HashMap;
import java.util.Map;

/** filtered by ARG-V */
 public class Main {

    /** ARG-V: suitable */
   public static boolean importMapValuesIfKeyEmpty(Map<String, Integer> map,
       String key) {
         if (!map.containsKey(key)) {
             map.put(key, Verifier.nondetInt());
       assert map.containsKey(key); //inline assert generated by ARG-V
       return true;
         }
    assert map.containsKey(key); //inline assert generated by ARG-V
    return false;
    }

  /** This main was generated by ARG-V */
   public static void main(String[] args) throws Exception{
    Map<String, Integer> map = new HashMap<>();
    map.put(Verifier.nondetString(), Verifier.nondetInt());
    String key = Verifier.nondetString();
    importMapValuesIfKeyEmpty(map, key);
    assert map.values().size() == 1;
  }

}
```

Property Key file (yml)

```yaml
format_version: "2.0"
input_files:
  - ../common/
  - ArithmeticException1/
properties:
  - property_file: ../properties/valid-assert.prp
    expected_verdict: false
  - property_file: ../properties/no-runtime-exception.prp
    expected_verdict: true
options:
  language: Java
```

Initial evaluation: *Demonstrating ARG-V's Generation of Realistic Java Benchmarks for SV-COMP*

- Generated and ran 68 *ARG-V* benchmarks on the SV-COMP ‘25 platform (BenchExec)
  - Compared verifier performance to the existing set of non-*ARG-V* benchmarks

- **Key takeaways:**
  - *ARG-V* benchmarks provided a notable increase in challenge for verifiers
  - **68 new benchmarks were accepted for use in SV-COMP 2026**

- **Open Questions:**
  - How can we avoid targeting similar/duplicate properties for future runs?
  - What makes the *ARG-V* benchmarks more difficult than prior ones?

*Accepted as tool paper at TACAS ‘26 (International Conference on Tools and Algorithms for the Construction and Analysis of Systems)

Non-systematically sampling small batches of code is not scalable or easily repeatable.

What can we use to identify and address blind spots in benchmark creation?

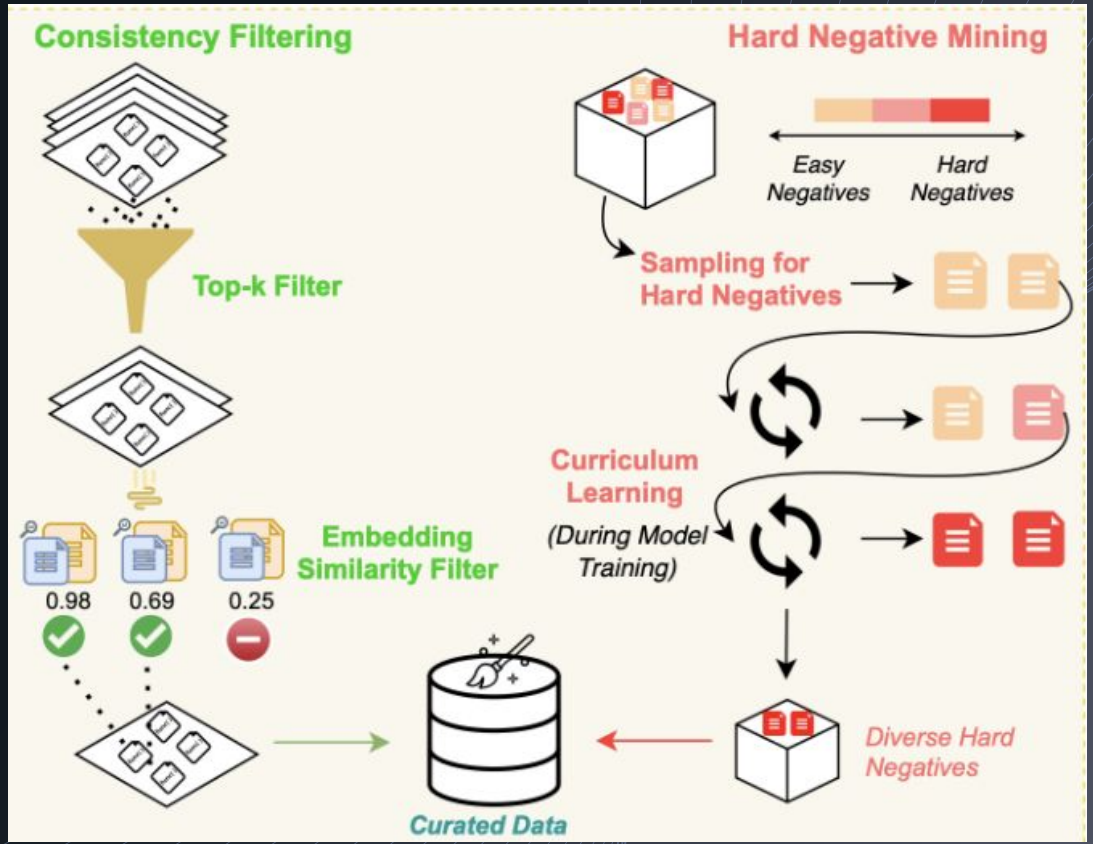
Approach

Enter: Code Embeddings

- Specialized embedding models that can convert code snippets or files into high-dimensional vectors
  - Capable of capturing detailed **semantic** information for code comparison, discerning even one author from another
- *For our experiments, we select Nomic Embed Code, in part due to its training on the CoRNStack dataset*

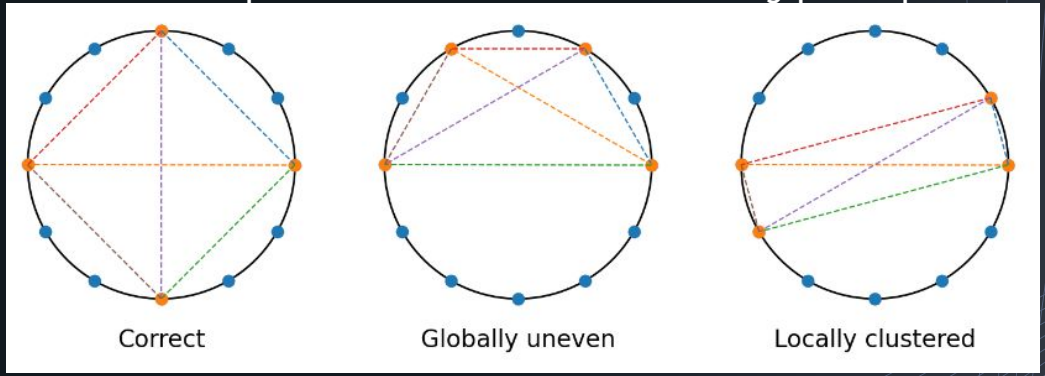
Nomic

Nomic Embed Code



Minimum Hyperspherical Energy (MHE)

- Metric based on the Thomson problem in physics for how to minimize potential energy of N electrons across a unit sphere
  - Generalizes concept to N-dimensional hypersphere



- Promotes both local and global heterogeneity
  - Local: no two points individually very close
  - Global: considerable overall spread
- Allows us to evaluate each candidate benchmark's "difference" to existing corpus

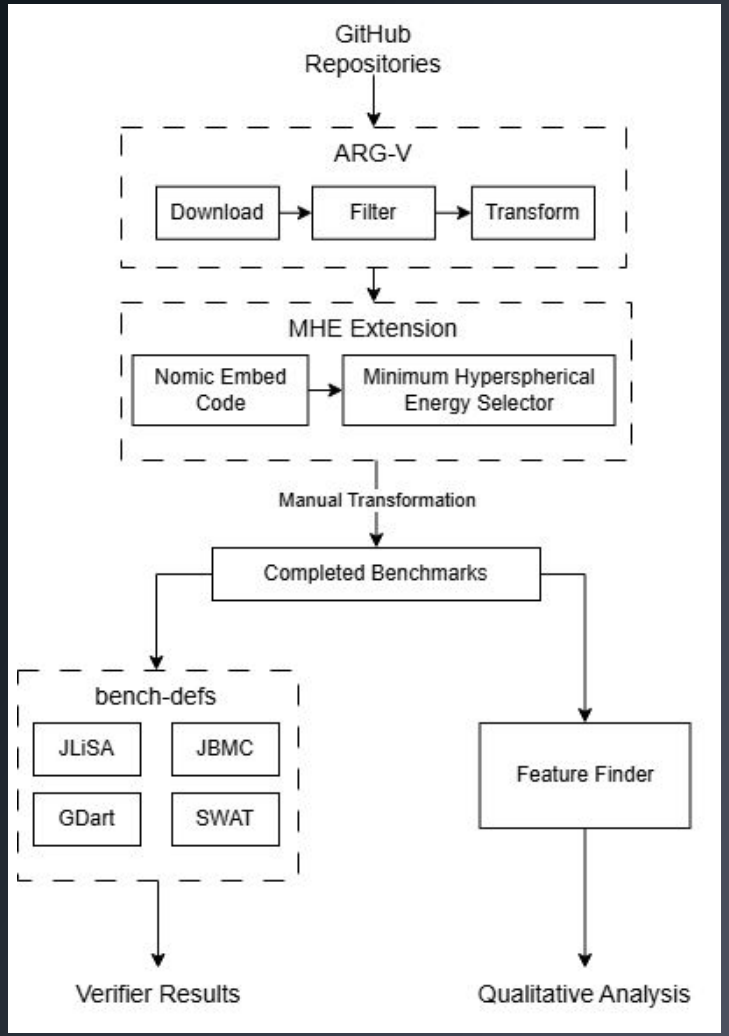
Minimum Hyperspherical Energy (equations)

- We derive ΔE to find the change in Hyperspherical Energy to a corpus to add one new vector **v**
- Using angular version because it's more stable at high dimensions
- Lower ΔE = better candidate for increasing embedding diversity

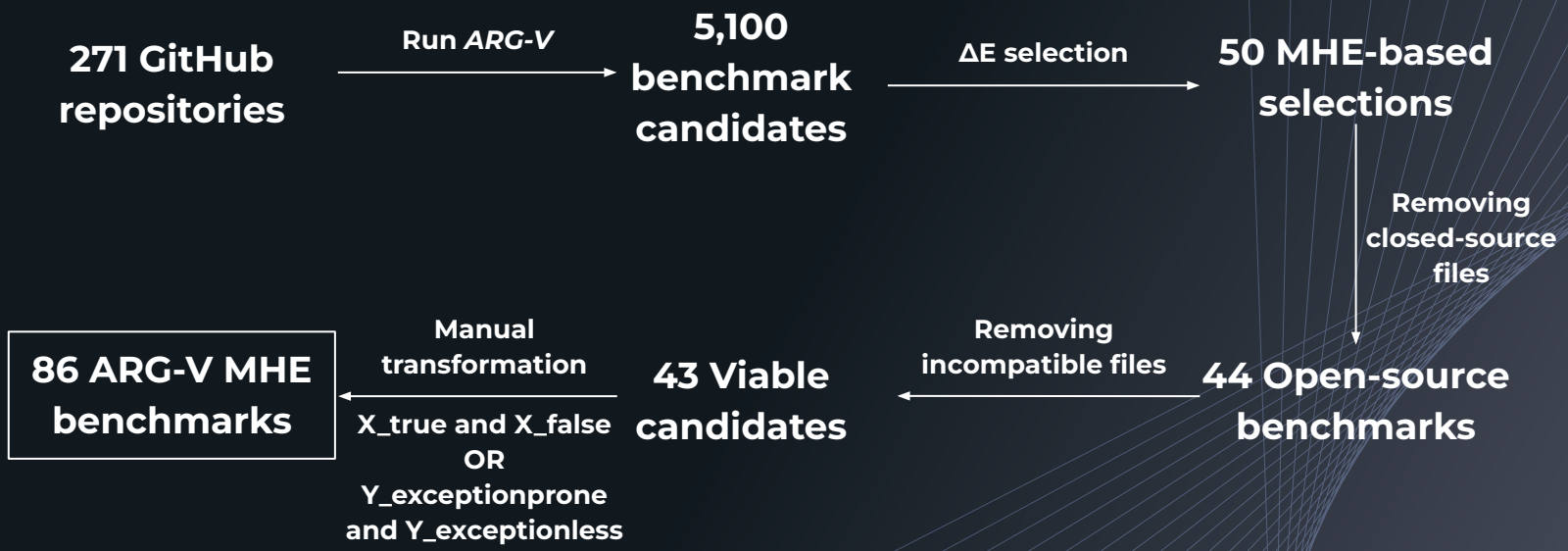
$$\mathbf{E}_{s,d}(\hat{\mathbf{w}}_i|_{i=1}^N) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} f_s(arccos(\hat{\mathbf{w}}_i{}^T \hat{\mathbf{w}}_j)) = \begin{cases} \sum_{i\neq j} arccos(\hat{\mathbf{w}}_i{}^T \hat{\mathbf{w}}_j)^{-s}, s > 0 \\ \sum_{i\neq j} log(arccos(\hat{\mathbf{w}}_i{}^T \hat{\mathbf{w}}_j)^{-1}), s = 0 \end{cases}$$

$$\Delta E_{s=2}^{\Lambda} = \sum_{i=1}^{N} (arccos(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}))^{-2}$$

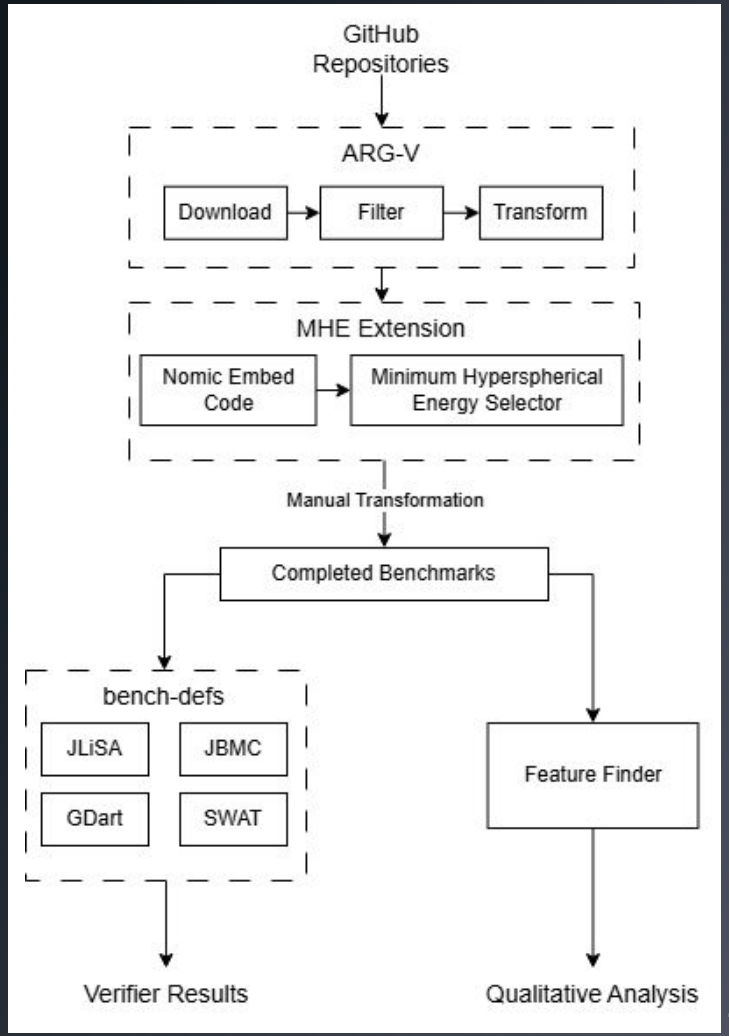
Approach Pipeline



Benchmark creation by the numbers



Experiments



Research Questions

RQ1 How closely do SV-COMP benchmarks match real-world code when comparing them in an embedding space?

RQ2 How much does the manual transformation step of *ARG-V* change the vector representation of benchmark candidates and their degree of difference from the SV-COMP corpus?

RQ3 How do the benchmarks selected based on their code embeddings compare to the existing SV-COMP benchmarks and past *ARG-V* benchmarks in terms of performance on state-of-the-art verifiers?

RQ4 How do the syntactic features of the new benchmarks selected using code embeddings differ from what is currently present in the SV-COMP repository?

Key Groups

Group	Quantity	Description
Candidate files	5,100	*ARG-V* output files pre-manual transformation
***ARG-V* MHE benchmarks**	86	New benchmarks produced by *ARG-V* and selected with MHE objective
TACAS benchmarks	68	Old *ARG-V* benchmarks created from randomly selected code – first appear in TACAS paper
SV-COMP benchmarks[1]	963	Benchmarks in SV-COMP **not** created by *ARG-V*
SV-COMP helper files[2]	465	Not benchmarks but referenced by SV-COMP benchmarks as objects and helper methods

[1] - Excludes the 68 TACAS benchmarks

[2] - Helper files not used by any *ARG-V* benchmarks

RQ1: SV-COMP in the embedding space

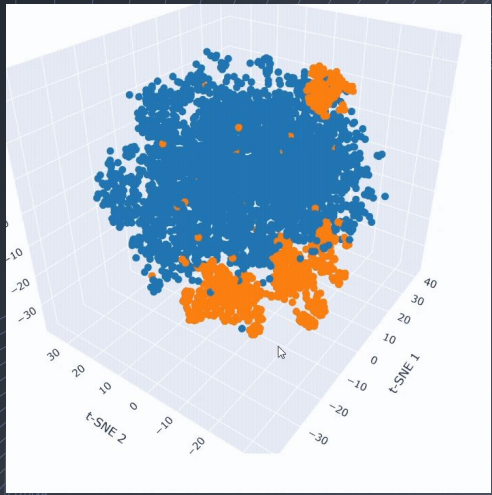
- Compare 1,031 SV-COMP benchmarks (including TACAS) to 5,100 candidate files from public repositories

- Analyze “embedding diversity” for each corpus
  - Also compare across corpus

- SV-COMP benchmarks **do not** closely match real-world code compared in an embedding space

Table 5.1: Embedding diversity for SV-COMP and *ARG-V* candidate files

SV-COMP corpus average cosine similarity:	**0.481**
***ARG-V* candidate file cosine similarity:**	**0.425**
Cross-class average cosine similarity:	**0.395**



Blue = candidate files
Orange = SV-COMP

RQ3: Verifier analysis

- Run *ARG-V* MHE benchmarks on BenchExec platform
  - Use top 4 SV-COMP'26 verifiers: jLiSA, GDart, SWAT, and JBMC
- Repeat on TACAS benchmarks and SV-COMP benchmarks

- Analyze including and excluding **Undecidable**
  - Undecidable: "Unknown", "error", "timeout", or "out of memory"
- Benchmarking setup:
  - 4 GB of memory
  - 120 seconds of cumulative CPU time, 2 processing units.
  - Machine: Intel Core i9-12900HK CPU at 2.5GHz, 32 GB of RAM
  - OS: Ubuntu 24.04.1 LTS
  - Software: BenchExec

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

RQ3: Verifier analysis - SV-COMP vs. ARG-V MHE

- **Significant** increase in verifier difficulty
  - Persists across all **metrics**, for all **categories**, for all **verifiers**

- Recall especially bad

- Shorter runtimes
  - Fast errors
  - Terminating early as “unknown”

Table 5.3: Classification metrics and runtimes between benchmark sets - SV-COMP vs. ARG-V MHE

	valid-assert		no-runtime-exception		cumulative	
	existing	new	existing	new	existing	new
Accuracy	0.99	0.69	1.00	0.77	1.00	0.74
Precision	0.98	0.76	1.00	0.92	0.99	0.87
Recall	1.00	0.62	1.00	0.76	1.00	0.71
Specificity	0.99	0.77	0.96	0.79	0.99	0.78
F1-Score	0.99	0.68	1.00	0.83	1.00	0.79
Results including Undecidable (Unknown, Error, Timeout, or Out of Memory)						
UI-Accuracy	0.53	0.29	0.60	0.40	0.56	0.35
UI-Recall	0.58	0.23	0.60	0.37	0.59	0.32
UI-Specificity	0.51	0.38	0.49	0.55	0.51	0.44
UI-F1-Score	0.73	0.35	0.75	0.53	0.74	0.46
% Undecidable	47%	58%	40%	47%	44%	52%
Verifier Runtimes						
runtime	16.59	11.80	25.64	11.92	20.46	11.87
runtime (excluding timeouts)	9.58	8.05	11.32	7.76	10.32	7.90

RQ3: Verifier analysis - TACAS vs. ARG-V MHE

- *ARG-V* MHE benchmarks **remain a higher challenge** for verifiers even when **comparing to other *ARG-V* benchmarks**
  - All metrics
  - All categories
  - Most verifier categories*

Table 5.5: Classification metrics and runtimes between benchmark sets - TACAS Benchmarks vs. New ARG-V-MHE Benchmarks

	valid-assert		no-runtime-exception		cumulative	
	existing	new	existing	new	existing	new
Accuracy	0.93	0.69	1.00	0.77	0.97	0.74
Precision	0.97	0.76	1.00	0.92	0.99	0.87
Recall	0.86	0.62	1.00	0.76	0.95	0.71
Specificity	0.98	0.77	1.00	0.79	0.99	0.78
F1-Score	0.91	0.68	1.00	0.83	0.97	0.79
Results including Undecidable (Unknown, Error, Timeout, or Out of Memory)						
UI-Accuracy	0.41	0.29	0.46	0.40	0.43	0.35
UI-Recall	0.28	0.23	0.41	0.37	0.36	0.32
UI-Specificity	0.55	0.38	0.64	0.55	0.58	0.44
UI-F1-Score	0.43	0.35	0.59	0.53	0.53	0.46
% Undecidable	56%	58%	54%	47%	55%	52%
Verifier Runtimes						
runtime	28.05	11.80	32.63	11.92	30.36	11.87
runtime (excluding timeouts)	13.28	8.05	11.91	7.76	12.59	7.90

RQ3: Verifier analysis - TACAS vs. ARG-V MHE (continued)

- GDart performed better on *ARG-V* MHE for Undecidable-Inclusive metrics
  - Timeouts main culprit
  - Twice as likely to timeout on TACAS benchmarks
  - **Time-intensive vs. unsupported tasks**

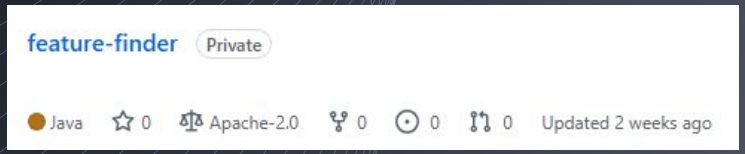
Table 5.6: Classification metrics and runtimes between individual verifiers - TACAS Benchmarks vs. New ARG-V-MHE Benchmarks

	jlisa		swat		gdart		jbmc	
	existing	new	existing	new	existing	new	existing	new
Accuracy	0.96	0.68	0.95	0.84	0.97	0.88	0.97	0.60
Precision	0.96	0.74	1.00	0.89	1.00	0.89	1.00	0.89
Recall	1.00	0.89	0.89	0.78	0.93	0.93	0.96	0.52
Specificity	0.50	0.00	1.00	0.90	1.00	0.77	1.00	0.82
F1-Score	0.98	0.81	0.94	0.83	0.96	0.91	0.98	0.65
Results including Undecidable (Unknown, Error, Timeout, or Out of Memory)								
UI-Accuracy	0.23	0.11	0.41	0.32	0.37	0.50	0.73	0.48
UI-Recall	0.34	0.15	0.25	0.21	0.19	0.49	0.66	0.42
UI-Specificity	0.03	0.00	0.71	0.59	0.71	0.52	0.86	0.64
UI-F1-Score	0.50	0.24	0.40	0.34	0.32	0.63	0.80	0.57
% Undecidable	76%	84%	25%	20%	62%	43%	25%	20%
Verifier Runtimes								
runtime	8.25	7.80	58.85	16.24	31.19	16.26	23.15	7.16
runtime (excluding timeouts)	8.25	7.80	26.97	10.02	14.73	7.85	5.18	5.73

RQ4: Qualitative Analysis - Feature Finder

Starting with codebook derived from symbolic execution, Java feature analysis, and C Verifier analysis:

1. Refine with relevant features from Java language and SV-COMP
   - Final codebook identifies 36 features such as loops, lambda expressions, recursion, or GUI libraries
2. Develop reusable tool for collecting features in benchmarks
   - Built on Eclipse toolkit for Java Abstract Syntax Trees
3. Run feature finder on *ARG-V* MHE, TACAS, SV-COMP, and SV-COMP helper files
   - Compare presence and ratio of features



RQ4: Qualitative Analysis - Results (selected subset)

Feature	*ARG-V* MHE	TACAS	SV-COMP (main and helper files)
Lambdas	0.070	0.000	0.000
Streams	0.023	0.000	0.000
Regular Expressions	0.023	0.000	0.000
GUI Libraries	0.163	0.000	0.000
Object Casting	0.360	0.000	0.039
Generics	0.744	0.000	0.004
Maps	0.465	0.015	0.009
Collections	0.488	0.000	0.025
Recursion	0.767	0.471	0.056

RQ2: Impact of manual transformation

- **MHE-selection process is performed on 5,100 candidate files prior to manual transformation**

- Manual transformation step may involve:
  - Scaffolding out data structures for use in method parameters
  - Creating intermediate variables for use in assert statements
  - Fixing assert stubs
- Process is non-deterministic and may influence the effectiveness of selecting for ΔE pre-transformation

Table 5.2: Degree of difference to SV-COMP before and after manual transformation

	Average $\Delta E^{A}_{s=2}$ to SV-COMP	Average cosine similarity to SV-COMP
Untransformed benchmarks	605.0	0.258
Transformed benchmarks	691.8	0.334
Average differential by benchmark	+86.9	+0.076
% increase	**14.5%**	**30.4%**

RQ2: Manual Transformation – Results

- Impact on difference to SV-COMP:
  - 14.5% increase to ΔE
    - Still 10.8% lower (and 15.4% lower cosine similarity) than general corpus of candidates
  - Perform Welch's t-test over whole distribution
    - $p = 1.66 * 10^{-21} < 0.05$
    - Gap in scores is **not** random – some degree of difference is preserved
- Impact on self-similarity
  - Median cosine similarity of 0.89
    - Mean: 0.876
    - Range: 0.986 to 0.607

- Despite **substantial differences after manual transformation**, our approach remains **more effective than random selection** at maximizing diversity

Related Work

Improvements from *PAClab* (our prior work)

Feature	PAClab	ARG-V
Scope	Intra-procedural (Method-level only)	Inter-procedural (Class-level and JDK packages)
Member Variables	Static only	Static and Non-Static
Supported types	Only ints and booleans	All primitive types and primitive arrays
Type Resolution	Implicit resolution only	Implicit and Explicit resolution via method bindings in classpath and type table
Artifact Preparation	None	Creation of main method and benchmark config (yml) file, preservation of provenance information, and re-naming to formatting standard

Benchmark Generation Efforts

- **Java benchmarks**
  - Adaptations from other libraries
    - Concolic Walk, MinePump, Securibench Micro
  - Handmade for SV-COMP
    - java/objects, java/float-UnboundedLoop

- **C benchmarks**
  - Many smaller projects and libraries
  - Linux Driver Verification (LDV) project
    - Pipeline for C benchmarks from Linux kernel
    - Extract → Environment Modeling → Property Specification

Embedding-based Comparisons

- Cosine similarity is foundational metric in similarity detection
  - Especially stable at higher dimensions (Verleysen et al., 2005)
- Code-specific models used for clone detection and semantic search (Guo et al, 2021.)
- “Embedding Diversity” has been used to measure overall distribution in embedding space
  - Average pairwise cosine similarity across all points (Chaudhary, 2025)
- **Undersampling from embeddings**
  - *Clustering-based undersampling in class-imbalanced data* (Lin et al., 2017)
  - *Embedding Undersampling Rotation Forest for Imbalanced Problem* (Guo, Diao, and Liu, 2018)

Impacts + Wrap Up

Impacts

- Code embeddings coupled with MHE objective function promising
  - Procedure is **repeatable and scalable** is SV-COMP grows

- Approach may be useful outside field of formal verification
  - Diversity maximization is a useful tool for AI training
  - Value of off-the-shelf embedding models

- Tangible benefits:
  - **86 new benchmarks** for consideration at SV-COMP
  - Reusable and extensible feature finder for monitoring properties within SV-COMP
  - Adapted *ARG-V* MHE filter and pipeline for use by other developers

Threats to Validity

- **Subjectivity of manual transformation**
  - Degree of difference found in investigation concerning
  - *Untransformed benchmarks remain substantially more "diverse" from SV-COMP corpus relative to randomly-selected code*

- **Concerns with benchmark key accuracy**
  - Can substantially impact results of verifier evaluation
  - *Gap in performance substantial; established detailed procedure for assuring key accuracy*

- **Feature selection for qualitative analysis**
  - Could be biased by familiarity with *ARG-V* MHE benchmarks
  - *Does not detract from significance of feature's presence or absence*

Conclusion

- In summary, we contribute:
  - 86 new benchmarks for use at SV-COMP
  - An investigation into a novel approach for using code embeddings to identify blind spots in benchmark creation
  - An evaluation of qualitative features in SV-COMP and its relationship to real-world code

- Future work:
  - More procedural approach for manual transformation
  - Expansion of *ARG-V* to support multi-class and object-oriented features
  - **Creation of more benchmarks for SV-COMP**

Questions?