

# SV-COMP’25 Reproduction Report (Competition Contribution)

Levente Bajczi<sup>✉</sup>, Zsófia Ádám<sup>✉</sup>, and Zoltán Micskei<sup>✉</sup>

Department of Artificial Intelligence and Systems Engineering  
Budapest University of Technology and Economics, Budapest, Hungary  
{bajczi,adamzsofi,micskeiz}@mit.bme.hu

**Abstract.** The International Competition on Software Verification (SV-COMP) has been an important driver of progress in the formal verification community, fostering tool development, benchmarking, and reproducibility. As the competition grows in scale and complexity, a reproducibility study is essential to evaluate its robustness across environments, uncover hidden dependencies, and ensure long-term sustainability. This work aims to reaffirm the reliability of SV-COMP’s results, provide insights for similar competitions, and facilitate the adoption of its infrastructure beyond the competition. We reproduced the verification and validation results of active participants, including score and ranking calculations for the verification track. We found several problems prohibiting reusability and reproducibility of some participating tools, but we did not find serious issues with the competition infrastructure itself.

## 1 Introduction and Goals

The International Competition on Software Verification (SV-COMP) [2] is a long-running driving force in the software verification community, with a continuously expanding portfolio of benchmarks and tools.

Reproducibility, reusability, and availability were always a key priority. However, the size and complexity of SV-COMP mean a complex technological stack for benchmarking. Compared to just running a verifier on a program, instead, it is multiple tens of tools, each run on a subset of thousands of programs, multiplied by the number of validators that have to be executed on the results of these verifiers. This necessitates scaling up to a large cluster of machines. With each step, technological complexity grows.

There has been a partial reproduction of SV-COMP’23 [4], but we believe that a full reproducibility study of SV-COMP is due, to:

- evaluate reproducibility across different environments,
- provide an overview of the tasks and solutions involved, offering insights valuable to other competition organizers,
- facilitate the adoption of this technological stack for applications beyond SV-COMP, e.g., benchmarking by researchers and tool developers,
- identify hidden issues, e.g., hidden dependencies or missing documentation,
- ensure that the competition can endure changes in organization,
- enhance trust in SV-COMP’s results by reaffirming their reliability.

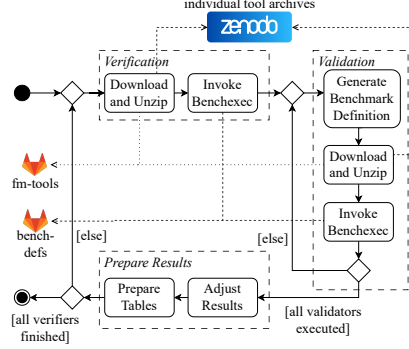


Fig. 1: Reproduction Workflow

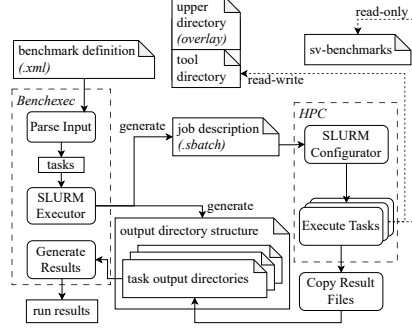


Fig. 2: Benchmarking Workflow

## 2 Reproduction Workflow

We did not use the official reproduction packages for SV-COMP’25, as at the time of writing, they are not yet finalized. Instead, we executed our experiments in parallel with the competition. This should not influence the validity of our findings, but some highlighted problems might have been already solved.

We relied on the official execution/reproduction guide for SV-COMP<sup>1</sup>, which provides scripts to run the verification tasks, run the validation tasks, and prepare the tables for the results. It is referenced by the benchmark definition repository<sup>2</sup>, which contains the `.xml` files used for running BenchExec [3] for both the verifiers and the validators; and the `category-structure.yml` file, which defines the participation and role of each tool in the competition. This repository also references the Formal-Methods Tools repository<sup>3</sup>, which contains further metadata (such as necessary packages) as well as a DOI referencing each version of each tool. The execution scripts use this DOI to download the tool binaries and execute them against the benchmarks<sup>4</sup>. The reproduction workflow can be seen (summarized) in Figure 1.

We used the Hungarian *Komondor* cluster<sup>5</sup>, which consists of computation nodes with two 64-core AMD EPYC 7763 CPUs and 256 GB of system memory each, running Red Hat Enterprise Linux 8.6. We requested array jobs via the SLURM Workload Manager with resource allocations outlined in SV-COMP’s rules (i.e., 15 GB memory, 4 CPU cores, and 15 CPU-minutes for each verification task), with multiple tasks aggregated in the same job both sequentially and in parallel, in a Singularity<sup>6</sup> container based on Ubuntu 24.04. RUNEXEC was used to sequester the tasks to non-overlapping CPU cores.

We used the `fuse-overlayfs` package to create an overlay file system in the containers. For performance and stability reasons, we found that only mapping

<sup>1</sup> [benchmarking/competition-scripts](#)

<sup>2</sup> [sv-comp/bench-defs](#)

<sup>3</sup> [benchmarking/fm-tools](#)

<sup>4</sup> [benchmarking/sv-benchmarks](#)

<sup>5</sup> [hpc.kifu.hu/komondor](#)

<sup>6</sup> [sylabs/singularity](#)

	2ls	aise	approve	brick	bubaak	bubaak-split	cpachecker	cpv	dartagnan	deagle	emergenttheta	esbmc-incr	esbmc-kind	gdart	goblint	hornix	java-ranger	jbmc	korn	mlb	mopsa	proton	racerf	svf-svc	sv-sanitizers	swat	symbiotic	theta	thorn	uautomizer	ugencutter	ukojak	utaipan
Conc.Safety	<sup>14</sup>				?	?	?		<sup>11</sup>	?		✓									<sup>113</sup>						?	✓	?	✓	✓	?	✓
MemSafety	<sup>14</sup>	?	?		✓	✓	✓					✓									<sup>113</sup>			<sup>13</sup>	<sup>14</sup>		✓		?	✓	✓	?	✓
NoOverflows	<sup>14</sup>				✓	✓	✓				<sup>15</sup>	✓									<sup>113</sup>			<sup>13</sup>	<sup>14</sup>		✓	<sup>15</sup>	<sup>15</sup>		✓	✓	✓
ReachSafety	<sup>14</sup>	?		<sup>112</sup>	✓	✓	✓	<sup>17</sup>			✓	✓				✓		<sup>11</sup>			<sup>113</sup>			<sup>13</sup>	<sup>14</sup>		✓	<sup>15</sup>	<sup>15</sup>		✓	✓	✓
Termination	<sup>14</sup>		<sup>19</sup>		✓	✓	✓				<sup>15</sup>	✓									<sup>113</sup>	✓		<sup>13</sup>			✓	<sup>15</sup>	<sup>15</sup>		?	✓	?
JavaOverall														✓			<sup>110</sup>	✓		<sup>110</sup>					<sup>111</sup>								
																				<sup>113</sup>													
Missing packages:																	Other:																
<sup>1</sup> java	<sup>2</sup> libmpfr	<sup>3</sup> clang	<sup>4</sup> gcc	<sup>5</sup> clang-format	<sup>6</sup> pandas	<sup>7</sup> ply.pycparser	<sup>13</sup> read-only sv-benchmarks																										
<sup>8</sup> libfile-copy-link-perl	<sup>9</sup> libbsd	<sup>10</sup> javac	<sup>11</sup> python3	<sup>12</sup> libgfortran5																													

Missing packages:

<sup>1</sup>java <sup>2</sup>libmpfr <sup>3</sup>clang <sup>4</sup>gcc <sup>5</sup>clang-format <sup>6</sup>pandas <sup>7</sup>ply,pycparser <sup>13</sup>read-only sv-benchmarks  
<sup>8</sup>libfile-copy-link-perl <sup>9</sup>libbsd <sup>10</sup>javac <sup>11</sup>python3 <sup>12</sup>libgfortran5


Other:

Fig. 3: Possible causes of failure. “?” denotes uninterpretable logs. Tools with “!” or “?” failed to run in the category, tools with ✓ were at least partly successful.

the current working directory as read-write is the best option, as we expect to find a witness there. Mapping the entire directory hierarchy (including sv-benchmarks with almost 200 000 files) as read-write caused instability problems when run on multiple hundreds of nodes from the same network filesystem, often causing **fuse** to become stuck in uninterruptible sleep. Therefore, all directories above the tool directory (the working directory in SV-COMP) are mapped read-only. The benchmarking workflow can be seen summarized in Figure 2.

We only ran non-*hors-concours* and non-inactive tools participating in SV-COMP’25. This resulted in running 33 verifiers (for a list, see Figure 3) and 33 validators. The results are published as a Zenodo artifact [1]. The tables are hosted at [home.mit.bme.hu/~bajczi/sv-comp-repro-25](https://home.mit.bme.hu/~bajczi/sv-comp-repro-25). The ranking summary is available at [results-verified/scoretable.html](https://results-verified/scoretable.html). The rankings correspond to the official rankings of verifiers (excluding tools with problems, see Sect. 3). Together, 38 358 tasks have mismatching verdict categories out of the more than 495 828 tasks (19 222 due to errors detailed in Sect. 3, but 19 136 due to other factors, mostly by reaching resource limits at different times). 1 officially correct result flipped to wrong (**mlb**), and 24 officially wrong results flipped to correct (**mlb**, **swat**). Compared to the official results, tasks correctly solved in both environments used 24% less walltime, 32% less cputime, and 11% more memory in the reproduction environment. Detailed statistics and differences are published on Zenodo [1].

### 3 Conclusion and Recommendations

Besides some minor issues that we submitted solutions to via merge requests ( [benchmarking/competition-scripts!160](#), [!161](#)), we did not find any problems in the competition scripts that would impede the reproduction workflow. Our only recommendation is to extend the documentation with details on how a competition can be run – e.g., how to run ranking calculations or run pre-runs by restricting resource allocation.

However, multiple tools produced no correct results, misrepresenting their actual SV-COMP performance. This stems from differences between SV-COMP’s

execution environment and our reproduction attempt. Solving these issues is key to enhancing SV-COMP’s reliability and simplifying independent tool execution.

**Missing Packages.** Tool developers declare the packages they need alongside the tool archives, but many tools leave out necessary packages, or declare non-existent ones. In the reproduction environment, we installed all the packages explicitly declared by the tools. Consequently, some tools had access to missing dependencies if they were declared by others. To uncover how widespread this issue is, we ran a separate experiment, executing each tool in each major category (opt-ins were left out) with just the declared packages of the respective tools installed. The results can be seen in Figure 3, and in the Zenodo archive [1].

**Assumed Read-Write Access.** Some tools tried to overwrite input files. While this is not against the rules of SV-COMP, we feel this goes against the purpose of making verification tools accessible to people outside of SV-COMP: if a verification tool modifies the input files it verifies, users may (justifiably) avoid using it. Such problems are also denoted in the table in Figure 3.

**Result Files Pattern Mismatch.** The benchmark definition files specify which files to keep after verification. Some tools do not include their own witness filenames (e.g., DEAGLE produces an `error-witness.graphml` file, but the definition only keeps `**/witness.*` files). The competition environment presumably disregards this directive and keeps every file (otherwise, the tools in question could not get their witnesses validated). However, this means that the benchmark definition file could not be reused outside of SV-COMP.

**Working Directory Must Be Tool Directory.** Some tool developers presume that BenchExec will always be executed from the tool directory. Offending tools mostly rely on relative paths holding from the working directory rather than prepending the directory of the tool archive to these paths. This makes it hard to use the tool binaries outside of SV-COMP and benchmarking.

### 3.1 Recommendations

In order to address the uncovered problems, we recommend to make the following modifications to SV-COMP:

- Enforce package declaration by a CI-check for the Formal-Methods Tools repository (preferred) or part of the tool qualification review (otherwise);
- Encode in the rules which files the verification tools may modify/create (e.g., all files within the current working directory);
- Enforce the result files pattern in the competition environment
- Encode in the rules that tools must be able to execute when called from outside their respective directories, and enforce this rule by introducing a CI-check (preferred) or a manual review (otherwise).

While we understand that rules will never cover all corner cases that would make tools SV-COMP-specific, these modifications cover a large set of problems that make the reuse of participating tools difficult. We believe that making participating verifiers accessible to anyone (and not just as benchmarking baselines, but as actual verification tools) is one of the most significant purposes of SV-COMP, and these modifications would enhance its impact.

*Funding.* This research was partially funded by the EKÖP-24-3 New National Excellence Program under project numbers EKÖP-24-3-BME-288 and EKÖP-24-3-BME-213, and the Doctoral Excellence Fellowship Programme under project numbers 400434/ 2023 and 400433/2023; funded by the NRD Fund of Hungary. We also acknowledge KIFŰ (Governmental Agency for IT Development, Hungary, [ror.org/01s0v4q65](https://ror.org/01s0v4q65)) for awarding us access to the Komondor HPC facility based in Hungary.

## References

1. Bajczi, L.: SV-COMP25 Reproduction Data (Dec 2024). <https://doi.org/10.5281/zenodo.14913865>
2. Beyer, D., Strejček, J.: Improvements in Software Verification and Witness Validation: SV-COMP 2025. In: Proc. TACAS. LNCS, Springer (2025)
3. Beyer, D., Löwe, S., Wendler, P.: Benchmarking and Resource Measurement. In: Fischer, B., Geldenhuys, J. (eds.) Model Checking Software. pp. 160–178. Springer International Publishing, Cham (2015). [https://doi.org/10.1007/978-3-319-23404-5\\_12](https://doi.org/10.1007/978-3-319-23404-5_12)
4. Gerhold, M., Hartmanns, A.: Reproduction report for SV-COMP 2023. Tech. rep., University of Twente (2023). <https://doi.org/10.48550/arXiv.2303.06477>