



Delft University of Technology

Document Version

Final published version

Licence

CC BY

Citation (APA)

Bartlett, A., Liem, C. C. S., & Panichella, A. (2026). FSE-Alware Tool Competition 2026 - Agentic Based Python Dependency Resolution. In S. H. Tan, & F. Khomh (Eds.), *FSE Companion 2026 - Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering* (pp. 1430-1433). Association for Computing Machinery (ACM). <https://doi.org/10.1145/3803437.3808243>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

In case the licence states "Dutch Copyright Act (Article 25fa)", this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

FSE-AIware Tool Competition 2026 – Agentic Based Python Dependency Resolution

Antony Bartlett
Delft University of Technology
Delft, Netherlands
a.j.bartlett@tudelft.nl

Cynthia Liem
Delft University of Technology
Delft, Netherlands
c.c.s.liem@tudelft.nl

Annibale Panichella
Delft University of Technology
Delft, Netherlands
a.panichella@tudelft.nl

Abstract

This is the inaugural edition of the FSE-AIware Tool Competition for Agentic Based Python Dependency Resolution. The aim of this competition is to provide a platform for researchers and practitioners to build, test and validate their agentic tools for Python dependency resolution. The platform provides an existing state-of-the-art agentic tool and a validation framework to evaluate the performance of the tools. In this first edition, the competition includes two tools from different teams. All tools were validated on a newly generated dataset of Python projects with various dependency complexity. We compare all tools against the baseline tool and provide insights into the performance of the tools and the challenges of Python Dependency Resolution. This paper provides an overview of the competition, detailing its context, framework, participating tools, evaluation methodology, and key findings.

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; **Error handling and recovery**; • **Computing methodologies** → **Machine learning**.

Keywords

Agentic Engineering, Software Engineering, Software Testing, Dependency Resolution, Python

ACM Reference Format:

Antony Bartlett, Cynthia Liem, and Annibale Panichella. 2026. FSE-AIware Tool Competition 2026 – Agentic Based Python Dependency Resolution. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26)*, July 5–9, 2026, Montreal, QC, Canada. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3803437.3808243>

1 Introduction

Python Dependency Resolution is a critical aspect of software development [4, 7], ensuring that all required libraries and packages are correctly identified and installed for successful project execution. Despite numerous package managers and tools available, developers often face challenges when attempting to run Python projects from third parties, especially when dealing with complex dependencies and version conflicts; a conflict arises when two or more dependencies require different versions of the same package.



This work is licensed under a Creative Commons Attribution 4.0 International License. FSE Companion '26, Montreal, QC, Canada
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2636-1/2026/07
<https://doi.org/10.1145/3803437.3808243>

This can lead to a situation where manually attempting to resolve these conflicts becomes a daunting and time consuming task, often requiring deep knowledge of the dependencies.

Various traditional methods have attempted to address the challenge of dependency resolution [3, 8]. These methods typically involve the use of knowledge bases, to compile a list of dependencies and their versions, applying algorithms to solve conflicts. However, these traditional approaches often struggle with complex dependencies and projects containing a large number of dependencies [1]. The limitations of traditional tools highlight the need for more advanced and intelligent solutions to effectively manage Python dependencies.

Therefore, the FSE-AIware Tool Competition for Agentic Based Python Dependency Resolution was established to encourage the development of innovative tools that leverage agentic engineering principles to enhance the dependency resolution process. This competition provides a platform for researchers and practitioners to build, test, and validate their agentic tools for Python dependency resolution, fostering collaboration and innovation in this critical area of software engineering.

With this competition, we aim to explore the potential of agentic tools in resolving Python dependencies, and to identify the key challenges and opportunities in this domain. We ask the following research questions to guide our investigation:

RQ1: *To what extent can agentic tools effectively resolve Python dependencies?*

RQ2: *What is the leading cause of failure for agentic tools in Python dependency resolution?*

2 Methodology

To facilitate the competition, we provided on GitHub¹ a repository containing the baseline tool, the testing dataset, and the instructions for participating. To ensure fair comparison, we collected together the submitted tools and ran them on a newly generated dataset of Python projects not previously shared with participants.

The baseline tool, held-out dataset, and competition results are available in the competition repository, and on Figshare [2].

2.1 Competition Framework

Constraints: To ensure a fair comparison between tools, we established the following constraints for the competition:

- Tools must run within a Docker container.
- Agents must operate withing a 10GB VRAM limit.
- Tools must accept either a folder of Python 'snippets' or a single snippet as input.

¹<https://github.com/checkdgt/fse-aiware-python-dependencies>

Docker: We provide a sample Dockerfile and Docker Compose configuration for the baseline tool, which participants can use as a template for their own tools. This ensures all tools are evaluated in a consistent environment, and allows for easy deployment.

Sample Tool and Data: We provide a sample tool that serves as a baseline for the competition. This tool is designed to demonstrate the basic functionality required for Python dependency resolution, and participants are encouraged to build upon and improve this baseline in their own submissions. Additionally, we provide the HG2.9K dataset [4], which is a collection of Python snippets with varying levels of dependency complexity.

2.2 Competition Tools

We received two submissions for the competition; MEMRES and SMT-LLM. Each tool and its corresponding code is explained and made available in the dedicated tool papers.

2.3 Baseline Tool

The baseline tool provided for the competition is the PLLM tool [1], which is an agentic tool designed to resolve Python dependencies. This tool serves as an example of the functionality required for the competition, and participants are encouraged to build upon and improve this baseline in their own submissions. The PLLM tool is designed to analyze Python snippets, identify dependencies, and attempt to resolve them using a combination of static analysis and language model capabilities.

2.4 Benchmark Test Set

To evaluate the performance of the submitted tools, we created a new set of Python projects from the GitHub Gist repository².

To facilitate the curation of the benchmark test set, we developed a Python project named ‘Gist Peeler’³, which is designed to scrape and filter Python snippets from the GitHub Gist repository. Using Gist Peeler, we collected all single file Gist snippets with more than 5 stars, from 2026 to 2018—avoiding duplicate Gists that may have been included in the HG2.9K dataset. Through this process, we curated a new Gist dataset containing ≈ 4669 Python projects. We followed the same naming process as the HG2.9K dataset, using the Gist UUID as folder name and the Gist filename as ‘snippet.py’.

For the competition, we randomly sampled 250 gists from the curated dataset to create the benchmark test set. The distribution of import dependencies in the datasets is shown in Table 1.

2.5 Evaluation Metrics

We assessed the performance of the tools based on several key metrics, that we considered to be the most relevant for evaluating the effectiveness of the tools.

- **Success Rate:** Number of snippets where all dependencies were successfully resolved.
- **Resolution Time:** Median time (in seconds) to resolve dependencies per snippet.
- **P90 Resolution Time:** 90th percentile resolution time (in seconds).

²<https://gist.github.com/>

³<https://github.com/checkdgt/gist-peeler>

Table 1: Distribution of import dependencies: full set vs test set

Dependencies	Full Set	Test Set
0	310	18
1	535	28
2	761	39
3	728	39
4	606	33
5	466	25
6	342	14
7	267	18
8	166	7
9	114	6
10	106	2
11	60	4
12	44	5
13	43	5
14	27	1
15	17	1
16	17	1
17	12	0
18	6	0
19	9	3
20	6	0
21	5	0
22	3	0
23	2	0
24	2	0
25	4	0
26	1	0
27	3	0
28	1	0
29	1	0
30	1	0
37	1	0
38	2	1
50	1	0
Total	4669	250

- **Unique Failures:** Number of snippets resolved by a single tool but failed by others.

2.6 Experimental Setup

To ensure a fair and consistent evaluation of the tools, we ran all tools on the same hardware—AMD Ryzen Threadripper 3970X 32-Core Processor, 62 GB RAM, NVIDIA GeForce RTX 3080—, and software environment—Ubuntu 22.04.5 LTS, Docker version 29.3.0. Each tool was executed within its respective Docker container, with a VRAM limit of 10GB. We used the same benchmark test set for all tools, and recorded the performance metrics for each snippet in the test set.

Parameter Settings: Each tool was built with a similar set of parameters, outlined below:

- **model:** Gemma 2 - All tools used Gemma 2 [6] as the underlying language model with Ollama [5] as the local LLM server.
- **loops:** 10 - The maximum number of iterations the agent can perform to resolve dependencies.
- **workers:** 4, SMT-LLM: 1 - The number of parallel workers to use for resolving dependencies.
- **range:** PLLM: 1, MEMRES: 0, SMT-LLM: N/A - The range defines the number of simultaneous Python versions to consider for each snippet. Each tool has a different approach and thus, each tool was run with their default value for this parameter.

Evaluation Criteria. : To determine the efficacy of the tools, we executed each tool against the test dataset (250 gists) a total of 2 times each, to account for the stochastic nature of LLMs.

For each snippet, we recorded whether the tool succeeded or not in resolving the dependencies, recording the failure type in cases where it failed. We also recorded the time taken to resolve dependencies for each snippet, and calculated the median and 90th percentile resolution times across all snippets in the test set.

3 Experiments and Results

Table 2: Performance metrics of all tools.

Metric	PLLM	MEMRES	SMT-LLM
Success Rate	142	226	214
Pass Rate	56.80%	90.40%	85.60%
Run Time (IQR)	204.03 (288.97)	7.76 (10.74)	35.25 (131.88)
90th Percentile	463.73	26.87	284.16

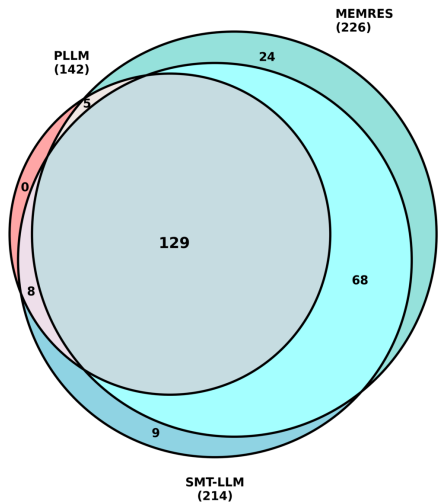


Figure 1: Venn diagram showing the overlap of successfully resolved snippets between the three tools.

From Table 2, we can see that both MEMRES and SMT-LLM significantly outperform the baseline PLLM tool in terms of success rate, with MEMRES achieving the highest overall success rate of 90.40%,

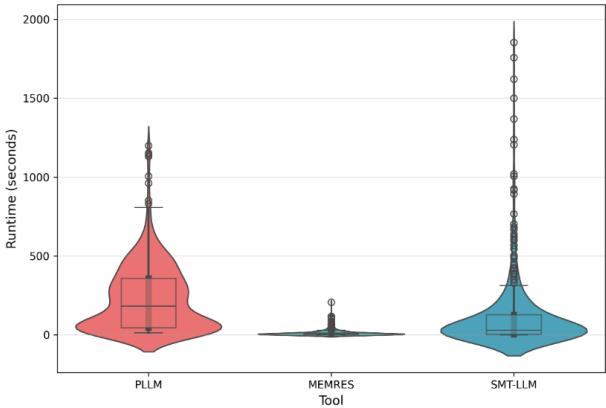


Figure 2: Violin plot showing the distribution of resolution times for the three tools.

closely followed by SMT-LLM with a success rate of 85.60%. The baseline PLLM tool, on the other hand, achieved a success rate of only 56.80%, indicating that the agentic tools developed by the participants were able to significantly improve the dependency resolution process compared to the baseline. This is also reflected in Figure 1, which shows the overlap of successfully resolved snippets between the three tools. Interestingly, we find that PLLM had zero, while SMT-LLM had 9 and MEMRES had 24 unique successes.

To understand the efficiency of the tools, we analyzed the resolution times. The median resolution time for MEMRES was faster than that of SMT-LLM, with MEMRES taking a median time of 7.76s compared to 35.25s for SMT-LLM. The 90th percentile resolution time also shows a large difference, with MEMRES taking 26.87s compared to 284.16s for SMT-LLM. The baseline PLLM tool had a longer median resolution time of 204.03s, further highlighting the improvements made by the agentic tools developed by the participants.

We further analyzed the distribution of resolution times using a violin plot (Figure 2). The plot reveals that MEMRES demonstrates superior distribution consistency, with 50% of executions completing within 6.03 seconds. In contrast, SMT-LLM exhibits significant outlier burden—statistical outliers—with extreme cases reaching 1,854 seconds, indicating unreliable performance predictability. PLLM shows both slow baseline performance and scattered distribution, making its runtime unpredictable.

MEMRES displays a higher success rate, faster resolution times, and a larger number of unique successes. Both MEMRES and SMT-LLM show increased performance over the baseline, revealing the potential of new agentic methods for Python dependency resolution.

Although the tools performed well overall, there were still several snippets that were unsolvable for all tools. Table 3 details the 7 universally failing snippets (2.8% of test set), revealing hard limits in automated code correction: dominated by ImportError (42.9%) and SyntaxError (35.7%), they stem from fundamental code structure issues, unavailable packages like pymol and teletthon, and version incompatibilities that no tool can resolve. These failures

Table 3: Analysis of Universally Failing Snippets (7 Total)

#	Primary Issue	Key Modules	Error Types	Avg Runtime (PLLM / MEMRES / SMT)	Category
1	Visualization libs	matplotlib, seaborn, missingno	SyntaxError (all 3)	273.9s / 1.5s / 886s	Code Structure
2	Google Vertex AI	google-cloud-vertexai, vertexai	NoMatchingDist. / SyntaxError	270.3s / 14.9s / 189s	Version Conflict
3	Windows specific		ImportError (all 3)	404.4s / 2.6s / 60.6s	Windows Dependency
4	Telethon library	telethon	SyntaxError / ImportError	262.1s / 12.9s / 188.6s	Unavailable Module
5	Data visualization	matplotlib, pandas	SyntaxError / ImportError	224.2s / 67.0s / 203.8s	Lib Compatibility
6	Molecular science	pymol, pymol2	ModuleNotFound (all 3)	308.4s / 5.1s / 197.2s	Unavailable Module
7	Async framework	fastapi, nats, starlette	NoMatchingDist. / ImportError / NonZeroCode	429.7s / 17.3s / 242.7s	Complex Dependency

span four distinct root cause categories—from syntax errors that persist across environments to complex multi-package dependency cascades—indicating these are legitimate problem boundaries rather than tool limitations.

Critically, we observe that both PLLM and SMT-LLM struggle to isolate failures early, which results in prolonged runtime on unsolvable snippets. In comparison, MEMRES demonstrates a remarkable ability to quickly identify unsolvable cases.

Unavailable packages and version incompatibilities appear as the leading contributors to failure across all tools, which are near impossible to resolve without possible human intervention. However, the key insight for our tools would be the importance of early detection of unsolvable snippets.

4 Discussion

When comparing the performance of the tools, it is evident that both MEMRES and SMT-LLM outperform the baseline PLLM tool in terms of success rate. However, while MEMRES and SMT-LLM have similar success rates, MEMRES demonstrates faster resolution times compared to SMT-LLM, indicating that MEMRES is not only more effective but also more efficient in resolving dependencies compared to SMT-LLM. This suggests that the design and implementation of MEMRES may be more optimized for efficiency, allowing it to resolve dependencies quicker than SMT-LLM. The baseline PLLM tool, while having a significantly lower success rate, also had a much longer median resolution time (204.03s), further highlighting the improvements made by the agentic tools developed by the participants.

Furthermore, the analysis of universally failing snippets reveals that unavailable packages and version incompatibilities are the leading contributors to failure across all tools. This suggests that a tool's ability to quickly identify unsolvable cases is a critical factor in improving the efficiency of dependency resolution tools, as it allows the tool to avoid spending time on snippets that cannot be resolved. The significant difference in runtime for unsolvable snippets between MEMRES and the other tools highlights the importance of this capability, and suggests that future development in this area should focus on improving the ability of tools to quickly identify unsolvable cases.

5 Conclusion

The ability to effectively resolve Python dependencies is crucial for the successful execution of Python projects, and the 'FSE-AIware

Competition for Agentic Based Python Dependency Resolution' has provided a valuable platform for researchers and practitioners to develop and evaluate innovative tools in this area. The results of the competition demonstrate that agentic tools can significantly improve the success rate of dependency resolution compared to traditional approaches, with MEMRES achieving the highest success rate and demonstrating superior efficiency. These findings highlight the potential of agentic tools in addressing the challenges of Python dependency resolution, and suggest that future research should focus on further improving the efficiency and effectiveness of these tools, particularly in terms of their ability to quickly identify unsolvable cases. The insights gained from this competition can inform the development of more advanced and intelligent solutions for managing Python dependencies, ultimately enhancing the software development process and enabling developers to more easily run Python projects from third parties.

6 Acknowledgments

We thank the participants of the competition for their time and effort in developing their tools and contributing to the advancement of Python dependency resolution. We also thank the organizers of the FSE-AIware conference for providing a platform for this competition and supporting research in this area. A special thanks to Ali Al-Kaswan and Ziyu Li of Delft University of Technology for their assistance in reviewing the submitted tool papers.

References

- [1] Antony Bartlett, Cynthia Liem, and Annibale Panichella. 2025. The Last Dependency Crusade: Solving Python Dependency Conflicts with LLMs. *2025 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)* (2025), 66–73. <https://api.semanticscholar.org/CorpusID:275921543>
- [2] Antony Bartlett, Cynthia Liem, and A. (Annibale) Panichella. 2026. FSE-AIware Tool Competition 2026 – Agentic Based Python Dependency Resolution. [doi:10.6084/m9.figshare.31975203](https://doi.org/10.6084/m9.figshare.31975203)
- [3] Wei Cheng et al. 2024. Revisiting Knowledge-Based Inference of Python Runtime Environments: A Realistic and Adaptive Approach. *IEEE Transactions on Software Engineering* 50, 2 (2024), 258–279. [doi:10.1109/TSE.2023.3346474](https://doi.org/10.1109/TSE.2023.3346474)
- [4] Eric Horton and Chris Parnin. 2018. Gistable: Evaluating the Executability of Python Code Snippets on GitHub. 217–227. [doi:10.1109/ICSME.2018.00031](https://doi.org/10.1109/ICSME.2018.00031)
- [5] Ollama. 2026. *Ollama*. <https://github.com/ollama/ollama>
- [6] Gemma Team et al. 2024. Gemma 2: Improving Open Language Models at a Practical Size. [arXiv:2408.00118](https://arxiv.org/abs/2408.00118) [cs.CL] <https://arxiv.org/abs/2408.00118>
- [7] Di Yang et al. 2016. From query to usable code: an analysis of stack overflow code snippets (*ICSE '16*). *ACM*. [doi:10.1145/2901739.2901767](https://doi.org/10.1145/2901739.2901767)
- [8] Hongjie Ye et al. 2022. Knowledge-based environment dependency inference for python programs (*ICSE '22*). Association for Computing Machinery, New York, NY, USA, 1245–1256. [doi:10.1145/3510003.3510127](https://doi.org/10.1145/3510003.3510127)