Pipeline Inspection, Visualization, and Interoperability in PyTerrier

Emmanouil Georgios Lionis(✉), Craig Macdonald, and Sean MacAvaney

University of Glasgow, Glasgow, UK
e.lionis.1@research.gla.ac.uk,
{Craig.Macdonald,Sean.MacAvaney}@glasgow.ac.uk

Abstract. PyTerrier provides a declarative framework for building and experimenting with Information Retrieval (IR) pipelines. In this demonstration, we highlight several recent pipeline operations that improve their ability to be programmatically inspected, visualized, and integrated with other tools (via the Model Context Protocol, MCP). These capabilities aim to make it easier for researchers, students, and AI agents to understand and use a wide array of IR pipelines.

1 Introduction

PyTerrier [10] provides an interface between state-of-the-art retrieval components, including highly-efficient inverted indexes [8], generative re-rankers [3], and retrieval-augmented generation methods [9]. Users can combine these components into pipelines (Sect. 2) through a declarative Python syntax. For example, Fig. 1(a) shows a sophisticated pipeline that performs a variety of operations, including lexical retrieval, query reformulation, result fusion, neural re-ranking, and answer generation.

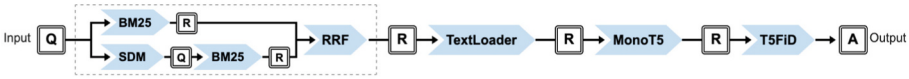
This demonstration presents three recent enhancements to PyTerrier pipelines designed to make them easier to understand and interact with. First, we introduce a core *inspection* mechanism (Sect. 3), which provides programmatic access to information about pipelines, including input/output specifications, constituent components, and other settings. Building on this foundation, we provide interactive pipeline visualizations that we call *schematics* (Sect. 4, Fig. 1(b)). Schematics are automatically rendered for pipelines in Python notebooks to help users understand the pipelines they construct. They are also integrated into the documentation to serve illustrative examples and can be rendered *ad hoc* for other purposes. Finally, we enable users to expose PyTerrier pipelines as tools via a Model Context Protocol (MCP) server [1], allowing other systems to interact with them through MCP's unified API (Sect. 5). For instance, the MCP feature enables pipelines to be easily integrated as components of agentic systems for research and deployment, or used as tools through systems such as Copilot[1] or Cursor[2].

[1] https://github.com/features/copilot.
[2] https://cursor.com/.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
R. Campos et al. (Eds.): ECIR 2026, LNCS 16486, pp. 159–165, 2026.
https://doi.org/10.1007/978-3-032-21321-1_23

```
index = pt.Artifact.from_hf('pyterrier/msmarco-passage.terrier')
pipeline = pta.RRFusion(
    index.bm25(),
    SequentialDependence() >> index.bm25(),
) >> index.text_loader() >> MonoT5() >> T5FiD('terrierteam/t5fid_base_nq')
```

(a) Python code to declare the pipeline (imports omitted for brevity).



(b) Generated schematic for the pipeline defined in (a).

Fig. 1. Example PyTerrier pipeline that performs retrieval, fusion, re-ranking, and answer generation stages. The pipeline is shown both as (a) declarative Python code and (b) the corresponding generated schematic.

2 Background: PyTerrier Pipelines

The core components of pipelines in PyTerrier are called transformers. A transformer provides a function that modifies the current *state* of one or more search requests. For example, a BM25 transformer accepts search queries and returns the search results from an index. The state is represented as a relation (i.e., a data table such as a Pandas DataFrame or a list of Python dictionaries). PyTerrier's data model defines a variety of typical column definitions that represent different stages of an engine's processing as shown in Table 1. These configurations can be extended with additional information as needed by transformers (e.g., a document's text or a query's vector). PyTerrier hosts components for a wide variety of IR pipeline operations, including lexical retrieval, dense retrieval, learned sparse retrieval, query reformulation, and answer generation.

Table 1. PyTerrier data model columns for typical stages of a search engine.

Name	Abbr.	Columns
Query Frame	Q	`qid` (PK, query identifier), `query`
Document Frame	D	`docno` (PK, document identifier), `text`
Result Frame	R	`qid` (PK), `docno` (PK), `score`, `rank`
Answer Frame	A	`qid` (PK), `qanswer`

PyTerrier's data model enables transformers to be combined together into larger pipelines. In a typical sequential pipeline (represented as `A >> B`), the output of one transformer (`A`) can be fed as input to the next transformer (`B`). Other types of pipelines include linear score combination (e.g., `A + B`) and reciprocal rank fusion [2] (e.g., `RRFusion(A, B)`). In all cases, it is important that the pipelines are combined in a way where the input/output specifications are compatible with one another. For instance, if a pipeline is constructed that does

not provide the text of a document to a neural re-ranking method that relies on it, the pipeline will raise an error when executed. This demonstration highlights new features that overcome shortcomings of these pipelines, namely the ability to programmatically inspect and validate the inputs/outputs of transformers, the ability to visualize them, and the ability to use them as tools for an LLM.

3 Inspecting Pipelines

PyTerrier transformers can be defined with as little as a function that maps one relation to another. This flexility makes implementing transformers simple and allows arbitrary pipelines to be constructed, but can result in incompatible pipelines. Without knowing the input and output specifications of individual transformers, it can be challenging to know whether a pipeline is compatible. Previously the only way to check these would be to check the documentation or run an example through and check the outputs (or interpret the resulting error).

To overcome this inconvenience, we introduce a new `pt.inspect` module that allows for the programmatic inspection of transformers and pipelines. The module provides functions that return which columns are required as inputs (e.g., a BM25 retriever expects `qid` and `query`, while a FAISS [4] dense retriever expects `qid` and `query_vec`) and which columns are returned for the provided inputs (e.g., BM25 returns `qid`, `query`, `docno`, `rank`, and `score`). Other functions in the module provide additional information about transformers, such as its attributes and any other transformers that it invokes (subtransformers). In most cases, `pt.inspect` identifies these automatically[3] by leveraging best practices in transformer implementation (e.g., performing input validation). To provide additional clarity on the implementation, a notebook is included in PyTerrier that demonstrates the inspection functionality.[4]

Transformer inspection has a number of applications. It enables pipeline *validation* to ensure that a pipeline is fully compatible (which is now applied automatically before experiments are conducted). Inspection also helps in visualization and interoperability, which are covered in the following two sections.

4 Visualizing Pipelines with Schematics

As shown in Fig. 1(a), pipeline definitions can get complex, especially when they involve multiple components and stages. Therefore, we added a new pipeline visualization feature that renders pipelines as interactive HTML snippets. We call these visitations schematics. Schematics render the pipeline as a sequence of transformers and the data between then, allowing users to better understand the transformers in a pipeline and how data flows between them. Hovering over elements provide additional information, like specific information about columns

[3] The details are too intricate to describe here, but are covered in the documentation.

[4] https://github.com/terrier-org/pyterrier/blob/master/examples/notebooks/inspect-demo.ipynb.

or transformer settings. Schematics leverage the inspection features covered in Sect. 3. A non-interactive screenshot of a schematic is shown in Fig. 1(b).

Although schematics can be displayed anywhere that can render HTML, we have applied them in a couple of convenient places. First, they are automatically rendered in Python notebooks when the output of a cell is a pipeline, aiding in the construction of pipelines in interactive sessions. We also render them in several places throughout PyTerrier's documentation to better demonstrate core concepts of the platform. PyTerrier contains an interactive notebook containing schematics for a range of pipelines.[5]

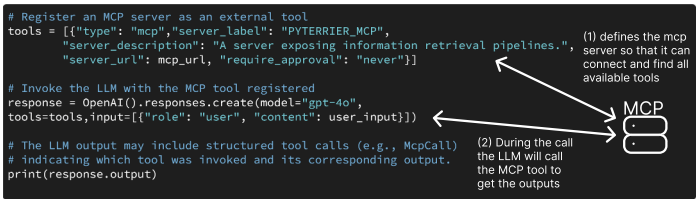


Fig. 2. Example usage of the MCP server through the OpenAI API.

5 Pipeline Interoperation with MCP

Another limitation of PyTerrier pipelines is the need for a Python runtime to execute them. Increasingly, external "tools" are invoked directly by Large Language Models (LLMs) [14]. The open-source Model Context Protocol (MCP) [1] is a popular protocol that standardizes LLM access to these tools by exposing them via HTTP endpoints. Despite MCP's original design for LLMs, its scope is much broader and has been referred to as a "universal plugin ecosystem" because it allows for simplified interoperation between virtually any Internet-connected system [15]. This quality makes MCP a natural choice to improve PyTerrier's interoperability with systems outside of Python.

To this end, we built a new package that exposes PyTerrier pipelines via MCP, thereby allowing a growing ecosystem of MCP-compatible tools to use them. The MCP server exposes HTTP endpoints for PyTerrier pipelines along with their associated metadata, such as natural-language descriptions and input/output specifications (using the new inspection functionality described in Sect. 3). The code for the MCP server is publicly available[6].

The MCP server is useful in a variety of settings. First, it can be used programmatically by researchers or practitioners as components in agentic systems.

[5] https://github.com/terrier-org/pyterrier/blob/master/examples/notebooks/schematics-demo.ipynb.

[6] https://github.com/terrierteam/pyterrier-server.

For instance, one can enable pipeline access for LLMs using the OpenAI[7] package, as shown in Fig. 2. Alternatively, MCP-exposed PyTerrier pipelines can be leveraged by users directly through systems like the Copilot extension in Visual Studio Code. This allows users to leverage PyTerrier pipelines in their day-to-day activities.

To demonstrate the flexibility of the MCP server across different retrieval and reasoning paradigms, we showcase its use with three representative pipelines: (i) a traditional BM25 [13] retrieval pipeline implemented using PISA [8,12]; (ii) a retrieval-augmented generation (RAG) system employing a lightweight Fusion-in-Decoder model for direct question answering [6,7,9]; and (iii) a Doc2Query-based model designed to support question decomposition [5,11]. Given these pipelines, we provide a demo video[8] illustrating the programmatic implementation via a web-based interface, as well as a demonstration of day-to-day usage through Copilot (Fig. 3).

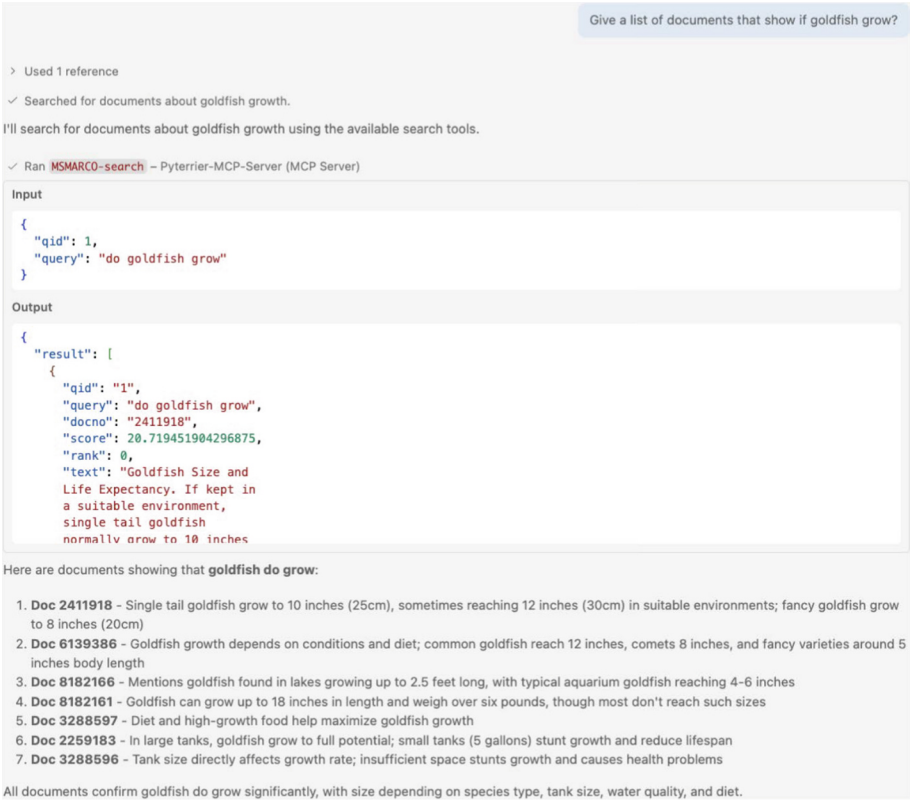


Fig. 3. A PyTerrier MCP server interaction example through Copilot.

[7] https://platform.openai.com/docs/overview.
[8] https://github.com/terrierteam/pyterrier-server/blob/main/videos/demo.gif.

6 Conclusions

This demonstration covers new features in PyTerrier that improve the inspection, visualization, and interoperability of pipelines. ECIR attendees will be able to interact with each of these features through notebooks and a web interface in the demonstration.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Anthropic: Introducing the model context protocol (2024). https://www.anthropic.com/news/model-context-protocol
2. Cormack, G.V., Clarke, C.L.A., Büttcher, S.: Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In: Allan, J., Aslam, J.A., Sanderson, M., Zhai, C., Zobel, J. (eds.) Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 758–759. SIGIR 2009, Boston, MA, USA, July 19–23, 2009. ACM (2009). https://doi.org/10.1145/1571941.1572114
3. Dhole, K.D.: PyTerrier-genrank: the PyTerrier plugin for reranking with large language models. CoRR abs/2412.05339 (2024). https://doi.org/10.48550/ARXIV.2412.05339
4. Douze, M., et al.: The faiss library. CoRR abs/2401.08281 (2024). https://doi.org/10.48550/ARXIV.2401.08281
5. Gospodinov, M., MacAvaney, S., Macdonald, C.: Doc2query-: When less is more. In: Kamps, J., et al. (eds.) Advances in Information Retrieval - 45th European Conference on Information Retrieval, ECIR 2023, vol. 13981, pp. 414–422. Dublin, Ireland, April 2–6, 2023, Proceedings, Part II. Lecture Notes in Computer Science, Springer, Cham (2023). https://doi.org/10.1007/978-3-031-28238-6_31
6. Izacard, G., Grave, E.: Leveraging passage retrieval with generative models for open domain question answering. In: Merlo, P., Tiedemann, J., Tsarfaty, R. (eds.) Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pp. 874–880.. EACL 2021, Online, April 19–23, 2021. Association for Computational Linguistics (2021). https://doi.org/10.18653/V1/2021.EACL-MAIN.74
7. Lewis, P., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, virtual (2020). https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html
8. MacAvaney, S., Macdonald, C.: A python interface to pisa! In: Amigó, E., Castells, P., Gonzalo, J., Carterette, B., Culpepper, J.S., Kazai, G. (eds.) SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 3339–3344. Madrid, Spain, July 11–15, 2022. ACM (2022). https://doi.org/10.1145/3477495.3531656

9. Macdonald, C., Fang, J., Parry, A., Meng, Z.: Constructing and evaluating declarative RAG pipelines in PyTerrier. In: Ferro, N., Maistro, M., Pasi, G., Alonso, O., Trotman, A., Verberne, S. (eds.) Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 4035–4040. SIGIR 2025, Padua, Italy, July 13–18, 2025. ACM (2025). https://doi.org/10.1145/3726302.3730150
10. Macdonald, C., Tonellotto, N.: Declarative experimentation in information retrieval using PyTerrier. In: Balog, K., Setty, V., Lioma, C., Liu, Y., Zhang, M., Berberich, K. (eds.) ICTIR '20: The 2020 ACM SIGIR International Conference on the Theory of Information Retrieval, pp. 161–168. Virtual Event, Norway, September 14-17, 2020. ACM (2020). https://doi.org/10.1145/3409256.3409829
11. Macdonald, C., Tonellotto, N.: Declarative experimentation in information retrieval using PyTerrier. In: Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval. pp. 161–168 (2020)
12. Mallia, A., Siedlaczek, M., Mackenzie, J.M., Suel, T.: PISA: performant indexes and search for academia. In: Clancy, R., Ferro, N., Hauff, C., Lin, J., Sakai, T., Wu, Z.Z. (eds.) Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019, Paris, France, July 25, 2019. CEUR Workshop Proceedings, vol. 2409, pp. 50–56. CEUR-WS.org (2019). https://ceur-ws.org/Vol-2409/docker08.pdf
13. Robertson, S.E., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Found. Trends Inf. Retr. **3**(4), 333–389 (2009). https://doi.org/10.1561/1500000019
14. Schick, T., et al.: Toolformer: Language models can teach themselves to use tools. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10–16, 2023 (2023). http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html
15. Werner, S.: MCP: an (accidentally) universal plugin system (2025). https://worksonmymachine.ai/p/mcp-an-accidentally-universal-plugin