Reliable, Reproducible, and Really Fast Leaderboards with Evalica

Dmitry Ustalov
JetBrains / Belgrade, Serbia
`dmitry.ustalov@jetbrains.com`

Abstract

The rapid advancement of natural language processing (NLP) technologies, such as instruction-tuned large language models (LLMs), urges the development of modern evaluation protocols with human and machine feedback. We introduce **Evalica**, an open-source toolkit that facilitates the creation of reliable and reproducible model leaderboards. This paper presents its design, evaluates its performance, and demonstrates its usability through its Web interface, command-line interface, and Python API.

1 Introduction

The emergent abilities, as exhibited by highly capable natural language processing (NLP) methods, such as instruction-tuned large language models (LLMs), urge the development of sound and reliable evaluation protocols. While the earlier methods could be reasonably evaluated on static datasets or individual benchmarks, modern methods require up-to-date benchmarks with live feedback from humans and machines (Faggioli et al., 2024). These benchmarks are often represented as pairwise comparison leaderboards (Figure 1), as popularized by LMSYS Arena (Chiang et al., 2024) and AlpacaEval (Dubois et al., 2024) projects.

As the NLP methodology evolves rapidly, today's evaluation methods are often implemented in computational notebooks and ad-hoc programs as an afterthought, which introduces errors, incompatibilities, and harms reproducibility and adoption. To improve the engineering aspect of benchmarking by reducing the number of methodological errors and simplifying the exchange and interpretation of the results, we present **Evalica**, an open-source evaluation toolkit that facilitates and speeds up the creation of reliable and reproducible NLP model benchmarks,[1] currently focused on the preference data. Based on our four-year experience in the development of production-grade tooling for quality control in crowdsourcing (Ustalov et al., 2024), we built Evalica with three practical goals in mind:

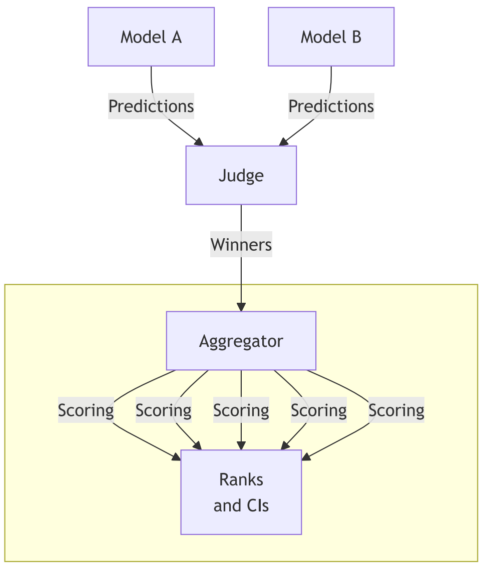


Figure 1: Evalica facilitates the highlighted aspects of leaderboard-making that involve aggregation of judgements, scoring the models with bootstrapped confidence intervals (CIs), and getting the final model ranks.

- *make the popular evaluation practices available* for a wide audience of users
- *ensure the performance and correctness* of the offered implementations
- *provide the best developer experience* possible

The remainder of this paper is organized as follows. Section 2 reviews the related work and its relationship with the declared goals. Section 3

[1]`https://github.com/dustalov/evalica`

shows Evalica's design and how it satisfies these goals. Section 4 describes the technical details of the Evalica implementation, including the means to ensure its correctness. Section 5 reports performance benchmarks against alternative implementations. Finally, Appendix A demonstrates a Web, a command-line , and a Python application programming interfaces (API) of Evalica.

2 Related Work

The research community has been developing various toolkits for ranking systems, such as Elo (1978) and TrueSkill (Herbrich et al., 2006). In our analysis, we distinguish several classes of them.

First, *dedicated leaderboard building tools*, such as IFEval (Zhou et al., 2023), LMSYS Arena (Chiang et al., 2024), Arena-Hard (Li et al., 2024), and AlpacaEval (Dubois et al., 2024). These toolkits were created by teams of researchers to implement a specific novel evaluation methodology. The code was generally written strictly tailored to the particular benchmark, requiring extra effort from the user to apply it to their own dataset and domain. Due to the high pace of today's scientific research, certain software engineering best practices were often omitted, such as test coverage, code documentation, continuous integration, and data format compatibility. At the same time, some implementations suffer from suboptimal computational performance on larger realistic datasets, which were out of scope of the original benchmarks.

Second, *ranking system implementations*, including Rust packages Propagon[2] and skillrating,[3] a Python package OpenSkill.py (Joshi, 2024), and others. As these packages are often written by skilled programmers in the best effort to bring correct implementations, these methods do not always match the ones used in current best practices in NLP evaluation. Also, the non-Python packages require an additional non-trivial effort to integrate with the existing Python code and notebooks.

Finally, *application-specific toolkits* like Elovation,[4] ArtistAssistApp,[5] and Crowd-Kit (Ustalov et al., 2024). These toolkits were built to accommodate user-generated content, usually in the form of crowdsourcing annotation, and often do not follow the methodology used in NLP evaluation.

[2]https://github.com/Refefer/propagon
[3]https://github.com/atomflunder/skillratings
[4]https://github.com/elovation/elovation
[5]https://github.com/eugene-khyst/pairwise-comparison

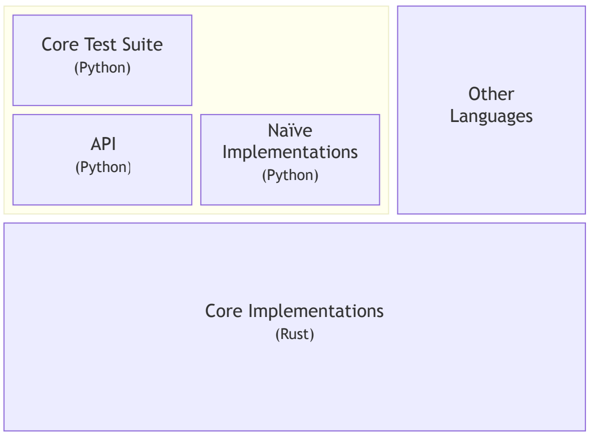


Figure 2: Evalica has a core in Rust that is covered by a comprehensive suite of tests in Python. We simplify prototyping and increase test reliability by keeping an independent implementation of each method in Python.

3 Design of Evalica

Evalica facilitates three tasks shown in Figure 1: it provides optimized single-threaded implementations of rating systems, simplifies the computation of confidence intervals for model scores, and offers convenient routines to prepare visualizations.

Figure 2 outlines the architecture of Evalica. In its *core*, there are performance-critical routines in Rust that process the raw data. These core routines are wrapped in convenient APIs for application developers in other languages. These APIs were responsible for transforming the representation into the indexed format as used by the core routines.[6] Examples of core routines are all the ranking algorithm implementations and helper routines for constructing win matrices.

We currently only support Python due to its popularity in machine learning. For the sake of reliability and ease of prototyping, we naïvely and implemented all the methods additionally in Python and built a comprehensive test suite that compares the Python implementations with the Rust ones. Other languages can be supported relatively easily (as long as there exists a bridge between Rust and that language), and improvements to the core implementations and tests will improve the state of all the derivative code.

We believe that these measures allowed satisfying the three goals mentioned in Section 1 ad-

[6]Models usually have names like `llama-3.1-405b-instruct` and `claude-3-5-sonnet-20240620`. Computers do not operate with strings *per se*, so we need to transform such names into the corresponding indices, e.g., `0` and `1`.

equately. Evalica accelerates popular evaluation practices by shipping the corresponding implementations in a high-performing compiled programming language, building on lessons learned from previously developed software to increase the developer's productivity.

4 Implementation Details

Evalica implements scoring approaches from popular benchmarks, such as Chatbot Arena and Arena-Hard: Elo (1978) and Bradley and Terry (1952), and average win rate. We ensured that they provided the same results as in these benchmarks. The package also contains implementations of the eigenvalue method (Bonacich, 1987), PageRank (Brin and Page, 1998), tie-aware method of Newman (2023), and trivial vote counting.

To invoke the implementation in Evalica (Listing 1), one needs to supply a vector of left objects (`xs`), a vector of right objects (`ys`), a vector of winner labels (`winners`), and, optionally, a vector of example weights (`weights`) for style control, as proposed in Li et al. (2024). Possible values of the winner labels are "X won," "Y won," and "tie." All methods are available in a lightweight and uniform functional API. We intentionally decided to avoid making assumptions about the tabular form of data as our experience in running Crowd-Kit (Ustalov et al., 2024) in production showed that it required an error-prone data transformation step that could have been avoided.

Internally, Evalica does not operate with model names, and core implementations require an index to compare the model name to the unique numerical identifier (as described in Section 3). Since this operation takes short yet non-negligible time, we provided the possibility to pass the already built index to save time during bootstrapping the confidence intervals and other routines that require resampling and recomputing the scores (Listing 2).

Besides the API, Evalica offers a built-in Web interface and a command-line interface, see Appendix A for illustrative examples. More specifically, the built-in Web interface follows a well-known input-output separation paradigm from Abid et al. (2019) and was created using the Gradio toolkit (Figure 4).[7] The command-line interface was developed using the pandas library for data manipulation (McKinney, 2010) and the tools available from the Python standard library (Figure 5).

After computing the scores and ranks, it is often useful to visualize the pairwise win rates for the compared models. Following Chiang et al. (2024), we applied the Bradley and Terry (1952) definition of such a quantity for all pairs of models i and j:

$$p_{ij} = \frac{s_i}{s_i + s_j},$$

where p_{ij} is the probability of model i winning against the model j, s_i is the score of model i, and s_j is the score of model j.

4.1 Correctness and Reliability

We applied a set of reasonable means to ensure correctness and reliability of the method implementations in Evalica. First, we implemented all the methods independently in two different programming languages, Rust and Python. We ensured that the outputs for the same inputs are the same between these implementations. Second, we employed property-based tests with the Hypothesis library (MacIver et al., 2019) for Python, which enumerated corner cases including empty or illegal inputs to break the program.[8] We covered all such cases and provide reasonable numerical fallbacks, where possible. Third, we compared the outputs against the canonical scores from external benchmarks. Fourth, we ensured that the test coverage is no less than 100%, and the test suite was executed on every revision in the repository.

4.2 Governance and Availability

We built Evalica using the trusted open-source ecosystem. The source code of Evalica was available under the Apache License 2.0 on GitHub.[9] Feature requests and code contributions were processed using the Issues and Pull Requests features on GitHub, correspondingly. We used continuous integration on GitHub Actions to invoke per-revision checks, including unit tests, linting, type checking, test coverage measurement, and computational performance testing. Public dashboards with test coverage and performance tests were available on Codecov[10] and Codspeed,[11] correspondingly. We used the trusted publishing approach to release Python packages to PyPI for the Linux, Windows, and macOS platforms.[12] Our compiled

[7] `https://www.gradio.app/`

[8] `https://github.com/HypothesisWorks/hypothesis`

[9] `https://github.com/dustalov/evalica`

[10] `https://codecov.io/gh/dustalov/evalica`

[11] `https://codspeed.io/dustalov/evalica`

[12] `https://pypi.python.org/pypi/evalica`

Setup	Time ↑
BT in Evalica	1.174 ± 0.009
Elo in Evalica	1.256 ± 0.019
Elo from Arena-Hard	3.778 ± 0.322
BT from Chatbot Arena	51.949 ± 1.797

Table 1: Performance of Evalica, Chatbot Arena, and Arena-Hard on the Chatbot Arena dataset. Time is in seconds; a 95% confidence interval is shown for ten runs. Smaller is better. BT means Bradley and Terry (1952), Elo means Elo (1978).

packages were forward compatible with any version of Python newer than 3.8 due to the use of the stable CPython ABI. We also released Evalica on conda-forge for the users of Anaconda, a popular distribution of scientific computing tools.[13] Last but not least, we published the developer documentation on Read the Docs.[14]

5 Performance Tests

We performed two series of computational experiments to study the running time of algorithm implementations in Evalica after ensuring their correctness. First, we evaluated the difference in computational performance between the current implementations in a popular benchmark and the ones provided by Evalica. Second, we compared the performance of core and naïve implementations of all the methods inside Evalica. All the experiments were run using CPython 3.13.1, NumPy 2.2.0, and Evalica 0.3.2 on macOS 15.2 (Intel® Core™ i5-8500 CPU, 32 GB RAM). All confidence intervals were built using bootstrap with 10K samples and 95% significance level.

5.1 Chatbot Arena Experiment

We evaluated the performance of four setups in processing the August 14, 2024 version of the Chatbot Arena dataset (Chiang et al., 2024) that contained 1.7M pairwise comparisons of 129 models, ties were not excluded.[15] We compared four different setups: an implementation of the Elo (1978) ranking system in pure Python, as used in Chatbot Arena, an implementation of Bradley and Terry (1952) in Python with scikit-learn (Pedregosa et al., 2011), as used in Arena-Hard, and Rust implementations of these two methods in Evalica. For that, we ran every setup ten times to simulate the realistic problem of confidence interval estimation that does often appear in model leaderboards. As the results in Table 1 indicate, Evalica's implementations of ranking methods outperformed the current ones as used in the benchmarks by up to 46 times without any involvement of multi-threading processing. Although this was expected since Python is an interpreted language and Rust is a compiled language, we believe that the Evalica's combination of performance and ergonomics would allow running more experiments within the same time budget. At the same time, performing computation in multiple threads, e.g., processing one sampling round per thread, would allow one to better use of the modern multi-core CPUs and reduce the computation time by multiple times.

[13]https://anaconda.org/conda-forge/evalica
[14]https://evalica.readthedocs.io/
[15]https://storage.googleapis.com/arena_external_data/public/clean_battle_20240814_public.json

Algorithm	Rust	Python
Average Win Rate	0.005 ± 0.000	0.006 ± 0.000
Bradley–Terry	0.005 ± 0.000	0.012 ± 0.000
Counting	0.005 ± 0.000	0.009 ± 0.000
Eigenvalue	0.005 ± 0.000	0.006 ± 0.000
Elo	0.005 ± 0.000	0.484 ± 0.004
Newman	0.006 ± 0.000	0.010 ± 0.000
PageRank	0.005 ± 0.000	0.006 ± 0.000

Table 2: Running time comparison of core Rust and naïve Python implementations of methods in Evalica on the LLMFAO dataset. Time is in seconds; a 95% confidence interval for ten runs is shown for each implementation. Smaller is better.

5.2 Rust vs. Python in Evalica Experiment

We evaluated the performance of all the methods implemented in Evalica's core in Rust against their naïve implementations in Python. Despite the name, these Python implementations were written using NumPy (Harris et al., 2020), a highly optimized library built on decades of successful performance engineering work for numerical computation in C and Fortran. We used the dataset from a smaller benchmark called LLMFAO (Ustalov, 2023), which had 9K pairwise comparisons for 59 LLMs, gathered in October 2023 using crowdsourcing. As the results in Table 2 show, the differences between core and naïve implementation were statistically significant, according to the permutation test ($p < 0.01$), but the effect size was not noticeable on that scale due to the efficient NumPy

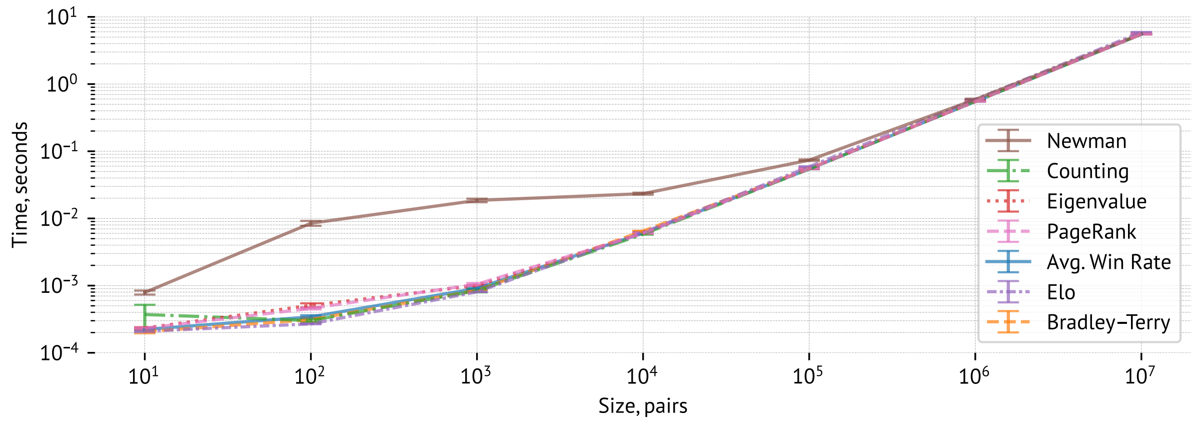


Figure 3: Performance scaling analysis of the Rust implementations in Evalica on the synthetic version of the Chatbot Arena dataset. Both scales are logarithmic. Time is in seconds, dataset size is the number of pairs; a 95% confidence interval is shown for ten runs. Lower is better.

routines used in the pure Python implementations. One important exception was Elo, whose equivalent implementation in Rust appeared to be more than 96 times faster than in Python due to the efficient compiler optimizations. At the same time, the Rust implementations had a smaller runtime variance and more predictable performance, which should be useful on larger-scale datasets.

5.3 Scaling on Synthetic Data Experiment

We analyzed the relationship between dataset size and computation time using Evalica on a synthetic dataset derived from Chatbot Arena as the original dataset was already larger than most existing preference-based NLP datasets. We selected seven dataset sizes, ranging from 10^1 to 10^7 pairs, with each size increasing by a factor of ten. For each size, we sampled the required number of pairs with replacement from Chatbot Arena ten times to study the time variance. Computation times were measured using Rust implementations of the methods available in Evalica, and we constructed 95% confidence intervals using bootstrapping. Figure 3 shows that the relationship between dataset size and computation time scales linearly for all methods, indicating good scalability. However, there are clear performance differences for small input sizes, with methods like Newman (2023) being slower initially but converging to similar trends as input size increases. Note that our analysis was limited by the number of models in the version of Chatbot Arena used in our experiments.

6 Conclusion

We believe that Evalica will foster the creation of reliable and reproducible benchmarks for future NLP systems. We define several potential directions of further work: (1) implementing a larger set of use cases widely used in practice, including confidence interval construction out of the box and additional ranking algorithms, (2) bringing additional performance and memory optimizations, and (3) supporting other popular programming languages with good interoperability with Rust, including JavaScript and Ruby. To the best of our knowledge, Evalica is the first attempt to offer drop-in accelerated preference-based benchmarks, which affects their computational performance and numerical reliability. We expect that a broader adoption of Evalica will result in faster iteration times, more useful experiments, and fewer model-selection mistakes.

Acknowledgements

We would like to thank three anonymous reviewers for their useful feedback. We are grateful to the early adopters of Evalica from the Vikhr team (Nikolich et al., 2024) and JetBrains AI team, whose kind feedback allowed improving the library. Note that the correct pronunciation of the library name is *eh-vah-lee-tsah*, which resembles a typical village name in the Balkans.

References

Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Y. Zou. 2019. Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild. *Preprint*, arXiv:1906.02569.

Phillip Bonacich. 1987. Power and Centrality: A Family of Measures. *American Journal of Sociology*, 92(5):1170–1182.

Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4):324–345.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117. Proceedings of the Seventh International World Wide Web Conference.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 8359–8388. PMLR.

Yann Dubois, Percy Liang, and Tatsunori Hashimoto. 2024. Length-Controlled AlpacaEval: A Simple Debiasing of Automatic Evaluators. In *First Conference on Language Modeling*.

Arpad E. Elo. 1978. *The Rating Of Chess Players, Past & Present*. Arco Publishing Inc., New York.

Guglielmo Faggioli, Laura Dietz, Charles L. A. Clarke, Gianluca Demartini, Matthias Hagen, Claudia Hauff, Noriko Kando, Evangelos Kanoulas, Martin Potthast, Benno Stein, and Henning Wachsmuth. 2024. Who Determines What Is Relevant? Humans or AI? Why Not Both? *Communications of the ACM*, 67(4):31–34.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, et al. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Ralf Herbrich, Tom Minka, and Thore Graepel. 2006. TrueSkill™: A Bayesian Skill Rating System. In *Advances in Neural Information Processing Systems 19*, pages 569–576. MIT Press.

Vivek Joshi. 2024. OpenSkill: A faster asymmetric multi-team, multiplayer rating system. *Journal of Open Source Software*, 9(93):5901.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. 2024. From Live Data to High-Quality Benchmarks: The Arena-Hard Pipeline.

David R. MacIver, Zac Hatfield-Dodds, et al. 2019. Hypothesis: A new approach to property-based testing. *Journal of Open Source Software*, 4(43):1891.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, SciPy 2010, pages 56–61.

Mark E. J. Newman. 2023. Efficient Computation of Rankings from Pairwise Comparisons. *Journal of Machine Learning Research*, 24(238):1–25.

Aleksandr Nikolich, Konstantin Korolev, Artem Shelmanov, and Igor Kiselev. 2024. Vikhr: The Family of Open-Source Instruction-Tuned Large Language Models for Russian. *Preprint*, arXiv:2405.13929.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830.

Dmitry Ustalov. 2023. Large Language Model Feedback Analysis and Optimization (LLMFAO). Dataset.

Dmitry Ustalov, Nikita Pavlichenko, and Boris Tseitlin. 2024. Learning from Crowds with Crowd-Kit. *Journal of Open Source Software*, 9(96):6227.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, et al. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-Following Evaluation for Large Language Models. *Preprint*, arXiv:2311.07911.

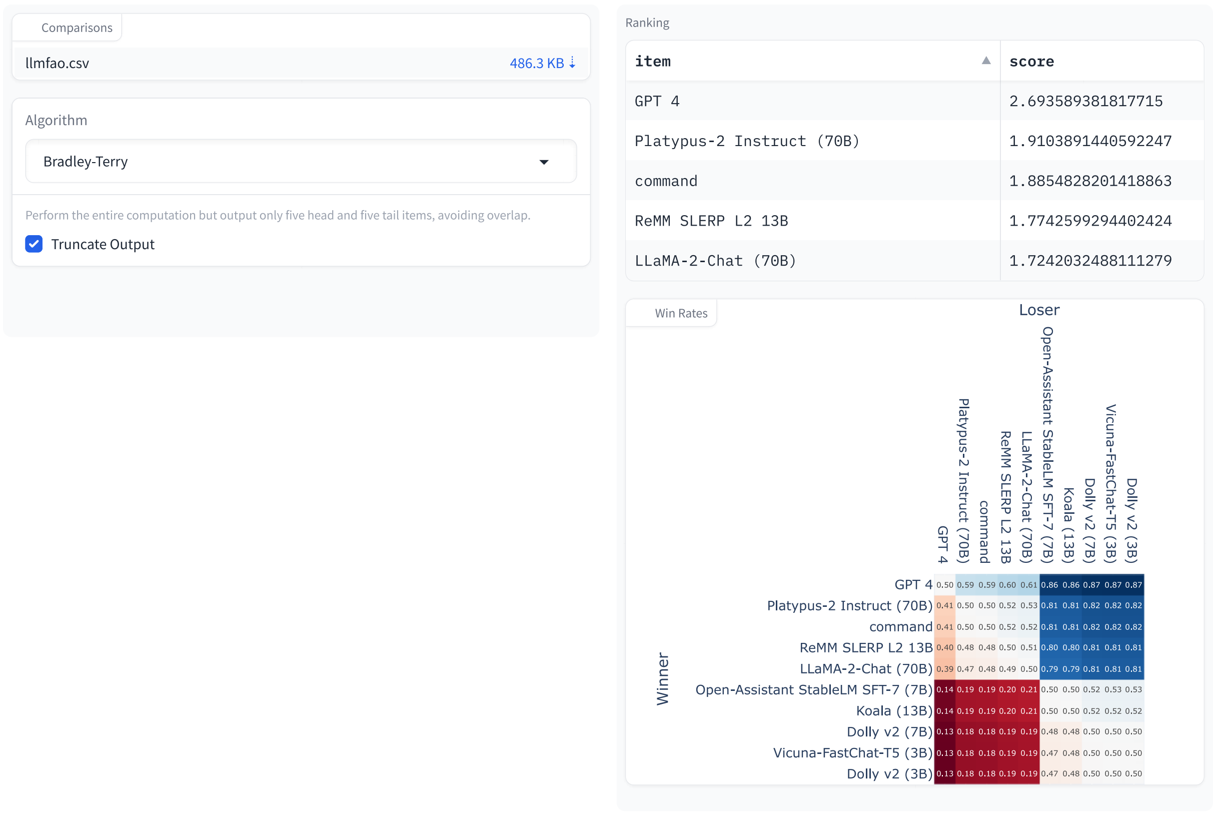
A Usage Examples

```
>>> from evalica import elo, pairwise_frame, Winner
>>> result = elo(
...     xs=["pizza", "burger", "pizza"],
...     ys=["burger", "sushi", "sushi"],
...     winners=[Winner.X, Winner.Y, Winner.Draw],
... )
>>> result.scores
pizza     1014.972058
burger     970.647200
sushi     1014.380742
Name: elo, dtype: float64
>>> df_scores = pairwise_frame(result.scores)
>>> df_scores  # can be used for plotting the pairwise win rate
           pizza     sushi    burger
pizza   0.500000  0.500003  0.501499
sushi   0.499997  0.500000  0.501496
burger  0.498501  0.498504  0.500000
```

Listing 1: An example of computing Elo ranking and the corresponding pairwise win rates with Evalica. Other methods can be applied similarly with a trivial modification: `bradley_terry`, `average_win_rate`, etc. See `https://github.com/dustalov/evalica/blob/master/Tutorial.ipynb` for an executable example.

```
# index the compared models to save time by not re-indexing them at each round
*_, index = evalica.indexing(
    xs=df["model_a"],  # series with model A identifiers
    ys=df["model_b"],  # series with model B identifiers
)

bootstrap: list["pd.Series[str]"] = []  # assuming model names are strings

for r in range(BOOTSTRAP_ROUNDS):
    # for reproducibility, set the random seed equal to the number
    # of the bootstrapping round
    df_sample = df_arena.sample(frac=1.0, replace=True, random_state=r)

    # estimate the Bradley-Terry scores for the given sample
    result_sample = evalica.bradley_terry(
        xs=df_sample["model_a"],
        ys=df_sample["model_b"],
        winners=df_sample["winner"],
        index=index  # use the index built above to speed up
    )

    bootstrap.append(result_sample.scores)

# this is a data frame with BOOTSTRAP_ROUNDS rows,
# each row represents the score of each model at the r-th round
df_bootstrap = pd.DataFrame(bootstrap)

# this is a data frame with confidence intervals of scores
# for each compared model
df_bootstrap_ci = pd.DataFrame({
    "lower": df_bootstrap.quantile(.025),
    "rating": df_bootstrap.quantile(.5),
    "upper": df_bootstrap.quantile(.975),
}).reset_index(names="model").sort_values("rating", ascending=False)
```

Listing 2: An example of bootstrapping a 95% confidence interval of Bradley and Terry (1952) scores with Evalica and pandas (McKinney, 2010). Any other supported model can be applied after a trivial modification. For simplicity, we do not show an example with `scipy.stats.bootstrap` (Virtanen et al., 2020), yet it is possible. See `https://github.com/dustalov/evalica/blob/master/Chatbot-Arena.ipynb` for an executable example.



Comparisons

llmfao.csv 486.3 KB

Algorithm

Bradley-Terry

Perform the entire computation but output only five head and five tail items, avoiding overlap.

☑ Truncate Output

Ranking

item	score
GPT 4	2.693589381817715
Platypus-2 Instruct (70B)	1.9103891440592247
command	1.8854828201418863
ReMM SLERP L2 13B	1.7742599294402424
LLaMA-2-Chat (70B)	1.7242032488111279

Win Rates

Winner \ Loser	GPT 4	Platypus-2 Instruct (70B)	command	ReMM SLERP L2 13B	LLaMA-2-Chat (70B)	Open-Assistant StableLM SFT-7 (7B)	Koala (13B)	Dolly v2 (7B)	Vicuna-FastChat-T5 (3B)	Dolly v2 (3B)
GPT 4	0.50	0.59	0.59	0.60	0.61	0.86	0.86	0.87	0.87	0.87
Platypus-2 Instruct (70B)	0.41	0.50	0.50	0.52	0.53	0.81	0.81	0.82	0.82	0.82
command	0.41	0.50	0.50	0.52	0.52	0.81	0.81	0.82	0.82	0.82
ReMM SLERP L2 13B	0.40	0.48	0.48	0.50	0.51	0.80	0.80	0.81	0.81	0.81
LLaMA-2-Chat (70B)	0.39	0.47	0.48	0.49	0.50	0.79	0.79	0.81	0.81	0.81
Open-Assistant StableLM SFT-7 (7B)	0.14	0.19	0.19	0.20	0.21	0.50	0.50	0.52	0.53	0.53
Koala (13B)	0.14	0.19	0.19	0.20	0.21	0.50	0.50	0.52	0.52	0.52
Dolly v2 (7B)	0.13	0.18	0.18	0.19	0.19	0.48	0.48	0.50	0.50	0.50
Vicuna-FastChat-T5 (3B)	0.13	0.18	0.18	0.19	0.19	0.47	0.48	0.50	0.50	0.50
Dolly v2 (3B)	0.13	0.18	0.18	0.19	0.19	0.47	0.48	0.50	0.50	0.50

Figure 4: A screenshot of the Evalica's Web interface with the LLMFAO benchmark (Ustalov, 2023). On the left, there are the input file, algorithm choice, and additional parameters. On the right, there is a table with the ranking results and a win rate plot. For the sake of brevity, we showed only a truncated output, with no columns corresponding to the number of compared pairs and the current rank of the model. A live example can be accessed at https://huggingface.co/spaces/dustalov/pair2rank.

```
$ head -n6 food.csv | column -ts,
left    right  winner
Pizza   Sushi  left
Burger  Pasta  right
Tacos   Pizza  left
Sushi   Tacos  right
Burger  Pizza  left
$ evalica -i food.csv bradley-terry | column -ts,
item    score               rank
Tacos   2.509025136024378   1
Sushi   1.1011561298265815  2
Burger  0.8549063627182466  3
Pasta   0.7403814336665869  4
Pizza   0.5718366915548537  5
```

Figure 5: An example of using a command-line interface of Evalica to process a file in the comma-separated values format and print the item ranks and estimated scores.