Maintaining MTEB: Towards Long Term Usability and Reproducibility of Embedding Benchmarks

Isaac Chung [*1] **Imene Kerboua** [*2 3] **Márton Kardos** [4] **Roman Solomatin** [5] **Kenneth Enevoldsen** [4]

Abstract

The Massive Text Embedding Benchmark (MTEB) has become a standard evaluation platform for text embedding models. While previous work has established the core benchmark methodology, this paper focuses on the engineering aspects that ensure MTEB's continued reproducibility and extensibility. We present our approach to maintaining robust continuous integration pipelines that validate dataset integrity, automate test execution, and assess benchmark results' generalizability. We detail the design choices that collectively enhance reproducibility and usability. Furthermore, we discuss our strategies for handling community contributions and extending the benchmark with new tasks and datasets. These engineering practices have been instrumental in scaling MTEB to become more comprehensive while maintaining quality and, ultimately, relevance to the field. Our experiences offer valuable insights for benchmark maintainers facing similar challenges in ensuring reproducibility and usability in machine learning evaluation frameworks.

The MTEB repository is available at: https://github.com/embeddings-benchmark/mteb

1. Introduction

Benchmarks are critical in guiding machine learning progress, particularly in the rapidly evolving field of embeddings and representation learning. MTEB (Muennighoff et al., 2022) has emerged as a comprehensive evaluation framework for embedding models across diverse tasks and languages. While initially designed as a standalone benchmark, MTEB has evolved into a versatile evaluation ecosystem. This natural evolution has been driven by sustained interest and engagement from the research community, integrating numerous extensions and specialized variants: language-specific benchmarks like C-MTEB (Xiao et al., 2024), MTEB-French (Ciancone et al., 2024), and German Text Embedding Clustering Benchmark (Wehrli et al., 2024); regional benchmarks such as SEB (Enevoldsen et al., 2024); multilingual expansions through MMTEB (Enevoldsen et al., 2025) covering over 250 languages; domain-specific adaptations like ChemTEB (Kasmaee et al., 2024); and cross-modal evaluations like MIEB (Xiao et al., 2025).

However, as the benchmark grows in scope and adoption, maintaining its sustainability and effectiveness presents significant engineering challenges that have received limited attention in the literature. These challenges span several dimensions: **reproducibility** — ensuring results can be consistently replicated given a model; **usability** — evaluating and presenting comparisons effectively and efficiently; **extensibility** — accommodating new tasks, languages, and modalities without disrupting existing functionality; **integrity** — ensuring results reflect genuine generalization, free from training data contamination, and **relevance** — addressing current research challenges as the field evolves. The literature has emphasized benchmark relevance, but often neglects the practical infrastructure required to maintain a benchmark's utility over time.

This paper explores the technical design choices underpinning MTEB's sustainability and extensibility. We focus on the software engineering practices implemented to **1) ensure dataset quality**, **2) assess benchmark zero-shot levels**, and **3) facilitate community engagement and expansion**. Through concrete case studies, we demonstrate how our approach addresses real-world challenges in benchmark maintenance, from multilingual expansion to contamination detection and community contribution scaling.

By sharing our experiences in maintaining MTEB, we aim to highlight often overlooked aspects of benchmark development that are essential for long-term usability. The engineering approaches described here address common challenges in benchmark maintenance and can inform similar efforts across the machine learning community, ultimately con-

[*]Equal contribution [1]Zendesk [2]Esker [3]INSA Lyon, LIRIS [4]Aarhus University [5]ITMO University. Correspondence to: Isaac Chung <chungisaac1217@gmail.com>, Imene Kerboua <imene.kerboua@insa-lyon.fr>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

tributing to more reproducible and trustworthy evaluation standards.

2. The MTEB Framework

Maintaining a large-scale benchmark repository like MTEB requires robust infrastructure that balances reproducibility and flexibility to accommodate diverse and novel models and evaluation scenarios. Rather than focusing solely on implementation details, we highlight the design principles that enable MTEB's sustainable growth and innovation potential.

2.1. System Architecture

MTEB consists of a set of code repositories and a leaderboard. Figure 1 shows at a high level the relationship between models, tasks, and result aggregation: **Standardized Model interfaces** for embedding generation, allowing evaluation of diverse architectures from sentence transformers to instruction-tuned LLMs with minimal adaptation; **Task definitions** encapsulate evaluation logic for different paradigms (classification, clustering, retrieval, etc.) while remaining agnostic to specific datasets or models; **Dataset handlers** manage data loading, preprocessing, and validation while maintaining consistent interfaces across languages and domains; **Result processors** standardize output formats and compute metrics uniformly across tasks.

To automate validations and releases, we adopt standard CI/CD practice in the MTEB repository, which is detailed in Appendix A.

2.2. Leaderboard

In addition to the evaluation package, MTEB features an open leaderboard that displays model performance across the benchmark, enabling businesses or users to select suitable models for their needs and supporting researchers in driving progress in the field. For a visualization of the leaderboard, see Figure 3 in Appendix B.

Submissions of results to the leaderboard are often made by the broader community. These can be made by submitting a pull request to the results repository.

3. Ensuring Benchmark Reproducibility

While MTEB's modular architecture provides the foundation for its functionality, specific reproducibility measures are essential to ensure the benchmark's validity over time. The primary challenges we identified revolve around ambiguity in the evaluation workflow: which version of a dataset or model is being used, which version of the code is executing the evaluation, and whether saved results can be reliably loaded and trusted.

3.1. Multi-level Versioning System

A key challenge in benchmark maintenance is managing changes to various components while preserving result comparability. Our versioning system addresses this through: **Task versioning**: Tasks undergo updates when evaluation protocols are changed, e.g. using stratified sampling. Each version maintains its own evaluation protocol. **Dataset versioning**: Each dataset references a specific revision from its source (typically Hugging Face), ensuring identical data across evaluation runs. Versions are updated when issues like mislabeled examples, or duplicates are discovered. **Model versioning**: Models reference specific checkpoints or API versions to maintain consistency, as model behavior can change significantly between releases. **Code versioning**: The MTEB package itself follows semantic versioning, with clear compatibility boundaries.

3.2. Reproducible Evaluation Environments

Certain evaluation methods introduce potential variability that must be controlled. For example, clustering tasks using K-means and classification tasks using linear probing require hyperparameter search procedures that can produce different results based on initialization. To address this, we provide consistent seeds for deterministic execution of evaluation tasks. Additionally, recognizing the substantial computational resources required for benchmark evaluation, we added CO_2 emissions logging with the `codecarbon` library [1]. This helps users quantify and report the environmental impact of their evaluation runs, promoting transparency in the computational costs of benchmarking.

3.3. Community Verifiable Results

The MTEB leaderboard is designed to maximize transparency and verifiability. Submissions require: **Reference implementation**: Each model must have a well-documented reference implementation. For open-source models, this includes direct links to code repositories and model weights. For proprietary models, implementations may rely on API calls, though this introduces a trust assumption that the API consistently serves the same model version. **Peer review process**: Submitted results undergo community review via pull requests to the results repository, where maintainers and community members can verify methodology and question unusual patterns. **Version-specific tracking**: Results explicitly record which versions of MTEB, datasets, and models were used, enabling precise reproduction of evaluation conditions.

This comprehensive approach ensures that anyone can reproduce a benchmark result by running the specified dataset version against the specified model version using a specific

[1] https://codecarbon.io/

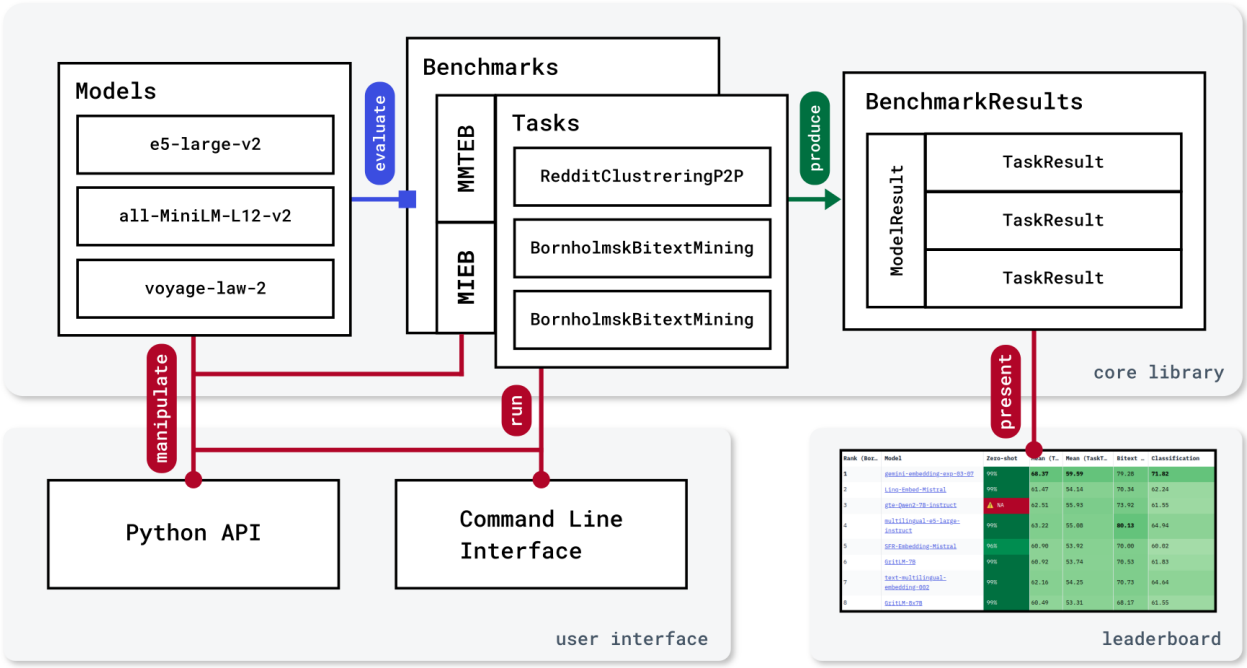

Figure 1. MTEB is designed with a modular architecture that separates concerns into distinct components. Users can select which model, task, or benchmark to run via the Python API or command line interface. The produced results can then be submitted to the results repository, which is read by the leaderboard.

MTEB version, yielding identical results across different machines and environments.

These reproducibility measures reflect lessons learned from past benchmark failures (See subsection 5.2) and help MTEB serve as a robust foundation for fair and consistent evaluation of embedding models.

4. Design for Extensibility

MTEB is an evaluation framework and should first and foremost be reproducible and thus stable. To achieve this, MTEB utilizes a modular design intended to allow contributors to extend any component without modifying others, enabling growth while preserving backward compatibility and result comparability.

Building MTEB to be extensible also encourages innovation. Any newly added custom models, tasks, and benchmarks can also be reproduced using the same underlying framework.

Unlike many existing benchmarks that enforce an often restrictive interface for model evaluation (Nielsen et al., 2024; Li et al., 2024), MTEB provides models with rich contextual information likely to be available at inference time, including task type, targeted language, or task-specific prompts, enabling researchers to explore diverse model architectures and prompting strategies. This flexibility allows models to be evaluated more realistically and creatively, promoting progress in representation learning. By incorporating recent models, tasks, and benchmarks, MTEB remains relevant to the field organically.

5. Case Studies

The sustainability of a benchmark depends not only on infrastructure and reproducibility, but also on community engagement and demonstrated utility. Community involvement not only strengthens the reliability of the benchmark, but also helps ensure its ongoing maintenance, relevance, and representativeness, particularly for low-resource communities (Singh et al., 2024). Overall, over 170 contributors participated in the development of the project. Their collective efforts have been instrumental in shaping MTEB into a multilingual, multi-domain, multimodal, community-driven benchmark. This section presents key challenges we encountered while maintaining MTEB and the solutions we implemented, offering practical insights for other benchmark maintainers.

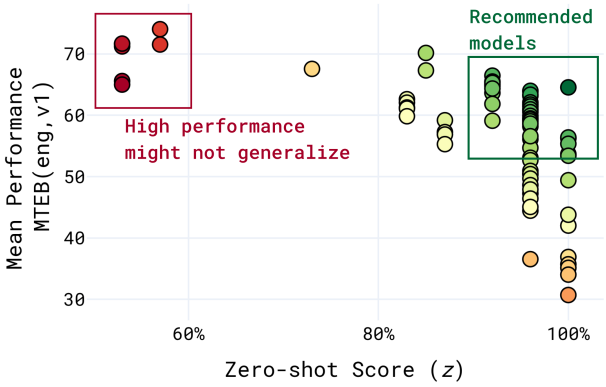

Figure 2. Models' mean performance against their zero-shot score on the legacy English MTEB. The highest ranking models achieve their scores by training on benchmark tasks, even though models with lower scores might generalize better to out-of-distribution environments.

5.1. Case Study 1: Assessing zero-shot levels.

A significant challenge in embedding benchmarks is determining whether models genuinely demonstrate out-of-distribution generalization. Unlike traditional data leakage (where test examples appear in training data) (Magar & Schwartz, 2022; Choi et al., 2025), MTEB faces a more nuanced challenge: models being trained on datasets with similar distributions to benchmark tasks, particularly when using training splits from the same source as benchmark evaluation tasks. To address this, we implemented a transparency approach that: 1) encourages model contributors to disclose training datasets, 2) computes a *zero-shot score* quantifying distributional overlap:

$$z = 1 - \frac{n_{\text{train}}}{n_{\text{total}}}$$

where z is the zero-shot score, n_{train} is the number of benchmark datasets the model was trained on, and n_{total} is the total number of benchmark datasets.

This metric offers a coarse estimate of how "in-domain" or "out-of-domain" a model is with respect to the benchmark. For example, the `e5-mistral-7b-instruct` model (Wang et al., 2024a) has a 95% zero-shot score on MTEB (English, v2), indicating that it was trained on only ∼5% of the benchmark's training splits, i.e. a 5% *leak*. Despite this, the model performs strongly across the board, suggesting strong generalization to unseen tasks.

Our initial implementation filtered the leaderboard to show only models with 100% zero-shot scores or unknown training data. This approach proved too restrictive, as community members noted it penalized transparency by hiding otherwise strong models that honestly disclosed their training data. Following this feedback, we developed a more granular approach that preserves transparency while providing users with critical context for interpreting benchmark scores. This case study demonstrates how benchmarks must balance methodological rigor with practical usability, and how community feedback can drive improvements in benchmark design. Detailed key exchanges on this topic can be found in Appendix C.

5.2. Case Study 2: Reproducing Reported Results.

The MTEB leaderboard recently transitioned from relying on self-reported results in Hugging Face model cards to a centralized repository of verified results. This transition revealed several challenges in reproducing reported model performance: 1) embedding models can use prefixes (E5 (Wang et al., 2024b) using `query:` and `passage:` as query and passage prompts for retrieval), 2) prompts can differ per task-type (Nomic models), 3) prompts can be added to both the query and the passage during retrieval (E5-mistral(Wang et al., 2023)) or only to queries (NV Embed), 4) some models do not normalize embeddings (Nomic ModernBERT), 5) models can have custom parameters during encoding (jina-v3 (Sturua et al., 2024) `task_type` argument that loads LoRAs during inference), and 6) model can have additional stages (CDE (Morris & Rush, 2024) have different stages for encoding to store embeddings).

To ensure accurate evaluation across this diverse landscape of embedding model architectures, we implemented additional features that accommodate each of these requirements. For instance, implementing prefix support allowed us to reproduce the reported performance of BGE models, which had previously shown significant performance degradation when evaluated. The improved reproducibility increased community confidence in the leaderboard rankings. The full details of the related pull requests are in Appendix D.

6. Limitations

The framework does not currently address issues in bias (Rakivnenko et al., 2024; Cao, 2025). Coverage in other diverse domains like arts, culture, health (Cao, 2024) is also limiting.

MTEB has also received criticism from model makers for its loose adherence to semantic versioning and backwards compatibility. These issues mainly came about as a result of the large refactoring effort that was required for multilingual expansion. While these refactoring effort did benefit the community and the package at large, breaking changes were at times introduced in minor/patch releases, which frequent users of the package found confusing and frustrating. We have since then put more emphasis on maintaining backward

compatibility with older versions of the library to avoid further inconvenience.

Since the initial launch of MTEB, the package has maintained a simple documentation structure, initially in the README and as it expanded across multiple hyperlinked markdown files. Although this was initially simple to maintain, the code base has grown in size and complexity and is now more prone to code-documentation inconsistencies. We plan to incorporate documentation within the docstrings and generate docs automatically in future releases (v2.0.0 or above).

7. Conclusion

MTEB's transition from a standalone benchmark to a scalable evaluation ecosystem has hinged on robust design choices. We present our approach to maintaining MTEB towards better reproducibility, maintainability, and seamless integration of new tasks and modalities. These infrastructure choices have enabled broad community engagement and sustained growth without compromising quality. Our experience offers practical guidance for designing and maintaining benchmarks, contributing to more reproducible, reliable, and globally representative machine learning research.

Impact Statement

MTEB promotes equitable progress in language technology by enabling rigorous, scalable evaluation across high- and low-resource languages.

Acknowledgements

We thank the reviewers for their insightful feedback and all contributors to the MTEB library.

References

Cao, H. Recent advances in text embedding: A comprehensive review of top-performing methods on the mteb benchmark, 2024. URL https://arxiv.org/abs/2406.01607.

Cao, H. Writing style matters: An examination of bias and fairness in information retrieval systems. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, WSDM '25, pp. 336–344. ACM, March 2025. doi: 10.1145/3701551.3703514. URL http://dx.doi.org/10.1145/3701551.3703514.

Choi, H. K., Khanov, M., Wei, H., and Li, Y. How contaminated is your benchmark? quantifying dataset leakage in large language models with kernel divergence, 2025. URL https://arxiv.org/abs/2502.00678.

Ciancone, M., Kerboua, I., Schaeffer, M., and Siblini, W. Mteb-french: Resources for french sentence embedding evaluation and analysis, 2024. URL https://arxiv.org/abs/2405.20468.

Enevoldsen, K., Kardos, M., Muennighoff, N., and Nielbo, K. The scandinavian embedding benchmarks: Comprehensive assessment of multilingual and monolingual text embedding. In *Advances in Neural Information Processing Systems*, 2024. URL https://nips.cc/virtual/2024/poster/97869.

Enevoldsen, K., Chung, I., Kerboua, I., Kardos, M., Mathur, A., Stap, D., Gala, J., Siblini, W., Krzemiński, D., Winata, G. I., Sturua, S., Utpala, S., Ciancone, M., Schaeffer, M., Misra, D., Dhakal, S., Rystrøm, J., Solomatin, R., Çağatan, Ö. V., Kundu, A., Bernstorff, M., Xiao, S., Sukhlecha, A., Pahwa, B., Poświata, R., GV, K. K., Ashraf, S., Auras, D., Plüster, B., Harries, J. P., Magne, L., Mohr, I., Zhu, D., Gisserot-Boukhlef, H., Aarsen, T., Kostkan, J., Wojtasik, K., Lee, T., Suppa, M., Zhang, C., Rocca, R., Hamdy, M., Michail, A., Yang, J., Faysse, M., Vatolin, A., Thakur, N., Dey, M., Vasani, D., Chitale, P. A., Tedeschi, S., Tai, N., Snegirev, A., Hendriksen, M., Günther, M., Xia, M., Shi, W., Lù, X. H., Clive, J., K, G., Anna, M., Wehrli, S., Tikhonova, M., Panchal, H. S., Abramov, A., Ostendorff, M., Liu, Z., Clematide, S., Miranda, L. J. V., Fenogenova, A., Song, G., Safi, R. B., Li, W.-D., Borghini, A., Cassano, F., Hansen, L., Hooker, S., Xiao, C., Adlakha, V., Weller, O., Reddy, S., and Muennighoff, N. MMTEB: Massive multilingual text embedding benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zl3pfz4VCV.

Kasmaee, A. S., Khodadad, M., Saloot, M. A., Sherck, N., Dokas, S., Mahyar, H., and Samiee, S. Chemteb: Chemical text embedding benchmark, an overview of embedding models performance & efficiency on a specific domain. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop, PMLR 262:512-531*, 2024. URL https://doi.org/10.48550/arXiv.2412.00532.

LAION-AI. CLIP_benchmark: CLIP-like model evaluation. https://github.com/LAION-AI/CLIP_benchmark, 2025. Accessed: 2025-05-03.

Li, X., Dong, K., Lee, Y. Q., Xia, W., Yin, Y., Zhang, H., Liu, Y., Wang, Y., and Tang, R. Coir: A comprehensive benchmark for code information retrieval models, 2024. URL https://arxiv.org/abs/2407.02883.

Magar, I. and Schwartz, R. Data contamination: From memorization to exploitation. In Muresan, S., Nakov, P., and Villavicencio, A. (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 157–165, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.18. URL https://aclanthology.org/2022.acl-short.18/.

Morris, J. X. and Rush, A. M. Contextual document embeddings, 2024. URL https://arxiv.org/abs/2410.02525.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022. doi: 10.48550/ARXIV.2210.07316. URL https://arxiv.org/abs/2210.07316.

Nielsen, D. S., Enevoldsen, K., and Schneider-Kamp, P. Encoder vs decoder: Comparative analysis of encoder and decoder language models on multilingual nlu tasks. *arXiv preprint arXiv:2406.13469*, 2024.

Rakivnenko, V., Maslej, N., Cervi, J., and Zhukov, V. Bias in text embedding models, 2024. URL https://arxiv.org/abs/2406.12138.

Singh, S., Vargus, F., Dsouza, D., Karlsson, B. F., Mahendiran, A., Ko, W.-Y., Shandilya, H., Patel, J., Mataciunas, D., OMahony, L., Zhang, M., Hettiarachchi, R., Wilson, J., Machado, M., Moura, L. S., Krzemiński, D., Fadaei, H., Ergün, I., Okoh, I., Alaagib, A., Mudannayake, O., Alyafeai, Z., Chien, V. M., Ruder, S., Guthikonda, S., Alghamdi, E. A., Gehrmann, S., Muennighoff, N., Bartolo, M., Kreutzer, J., Üstün, A., Fadaee, M., and Hooker, S. Aya dataset: An open-access collection for multilingual instruction tuning, 2024. URL https://arxiv.org/abs/2402.06619.

Sturua, S., Mohr, I., Akram, M. K., Günther, M., Wang, B., Krimmel, M., Wang, F., Mastrapas, G., Koukounas, A., Koukounas, A., Wang, N., and Xiao, H. jina-embeddings-v3: Multilingual embeddings with task lora, 2024. URL https://arxiv.org/abs/2409.10173.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=wCu6T5xFjeJ.

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*, 2023.

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. Improving text embeddings with large language models. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11897–11916, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.642. URL https://aclanthology.org/2024.acl-long.642/.

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. Multilingual e5 text embeddings: A technical report, 2024b. URL https://arxiv.org/abs/2402.05672.

Wehrli, S., Arnrich, B., and Irrgang, C. German text embedding clustering benchmark. *arXiv preprint arXiv:2401.02709*, 2024. URL https://arxiv.org/abs/2401.02709.

Xiao, C., Chung, I., Kerboua, I., Stirling, J., Zhang, X., Kardos, M., Solomatin, R., Moubayed, N. A., Enevoldsen, K., and Muennighoff, N. Mieb: Massive image embedding benchmark. *arXiv preprint arXiv:2504.10471*, 2025. doi: 10.48550/ARXIV.2504.10471. URL https://arxiv.org/abs/2504.10471.

Xiao, S., Liu, Z., Zhang, P., Muennighoff, N., Lian, D., and Nie, J.-Y. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 641–649, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657878. URL https://doi.org/10.1145/3626772.3657878.

A. Full Descriptions of Technical Infrastructure

A.1. Continuous Integration

Our continuous integration (CI) pipeline validates both code and datasets, a step often neglected in other benchmarks (Thakur et al., 2021; LAION-AI, 2025).

A.2. Dataset and Model Validation

Any pull request that adds or modifies datasets triggers automated checks, including: 1) format consistency, 2) metadata completeness, 3) metadata field validation, and 4) and dataset availability on Hugging Face. Similarly, any new or modified model metadata is automatically checked for completeness and model loading. Both types of validations heavily rely on Pydantic[2]. These help catch malformed examples and pre-processing inconsistencies early.

Each dataset and model has their respective Pydantic model that specifies the mandatory fields to capture crucial information for reproducibility. Specifically, each dataset has `TaskMetadata` and each model has `ModelMeta`. For example, Table 1 shows all of the `ModelMeta` parameters and their descriptions as of version 1.38.4.

A.3. Pull Request Checks

Contributions to the codebase are subject to an automated validation workflow comprising three sequential stages: 1) Linting and formatting, 2) Execution of unit and integration tests, 3) Verification of any required documentation updates.

By delegating routine checks to this pipeline, the burden on maintainers is reduced and the project remains receptive to contributions at scale.

A.3.1. Linting

All incoming changes are analyzed with ruff[3] according to a shared configuration that enforces coding conventions (e.g., import ordering, modernized Python syntax) and detects common errors.

A.3.2. Testing

To ensure reproducibility, the test suite is executed on both Linux and Windows platforms, with Linux jobs covering Python versions 3.9 through 3.12. This cross-platform strategy has been instrumental in uncovering environment-specific defects during development.

Mock infrastructure We employ mock tasks—each defined by two to three illustrative examples—and synthetic models that emit random embeddings. These mocks enable rapid feedback by exercising task evaluators and model-interaction code without incurring the overhead of full model instantiation.

Parameter validation Tests assert that all mandatory parameters are propagated correctly from evaluator modules to model classes under a variety of configurations, and that prompting routines apply inputs as intended.

Data integration Certain tests dynamically retrieve datasets from the Hugging Face Hub to validate end-to-end data loading and preprocessing.

Reliability enhancement We integrate the `pytest-rerunfailures` plugin to automatically retry tests that fail due to transient connection errors, thereby improving overall suite stability.

A.4. Versioning, Releases, and Automations

Versioning: We follow a SemVer-like versioning strategy adapted to benchmarks: 1) major versions indicate code-incompatible changes, 2) minor versions add tasks or datasets without affecting existing results, and 3) patch versions fix bugs or update documentation.

Changelog and Release Automation: Releases automatically generate changelogs with: 1) contributor lists and 2) change descriptions. Version-tagged releases trigger: 1) PyPI publishing, 2) documentation deployment, and 3) release note preparation. This ensures reproducible and traceable versions.

Automated Artifacts: To standardize result reporting, we support automated generation of: 1) Markdown tables of benchmark and tasks with descriptions and citations, 2) descriptive statistics of datasets, and 3) language coverage of the entire repository.These tools improve result clarity and reduce manual reporting errors. Latex tables can be generated at will for academic publications and are not automated.

Leaderboard: The MTEB leaderboard is refreshed and rebuilt daily. The previous workflow involved scraping the huggingface model cards for self-reported scores. The current workflow requires active submissions to the results repository.

B. Leaderboard

Figure 3 shows the MTEB leaderboard in light mode.

C. Model Zero-Shot Score

[2]https://github.com/pydantic/pydantic

[3]https://github.com/astral-sh/ruff

Parameter	Type	Description
loader	Callable or None	Function to load the model; defaults to SentenceTransformer if None.
name	str	Name of the model, ideally in the format "organization/model_name".
n_parameters	int or None	Number of parameters in the model, e.g., 7_000_000.
memory_usage_mb	float or None	Memory usage of the model in MB.
max_tokens	int or None	Maximum tokens the model can handle.
embed_dim	int	Dimension of embeddings produced.
revision	str or None	Model revision identifier.
release_date	str	Date the revision was released.
license	str	License under which the model is released.
open_weights	bool	Whether model weights are open source.
public_training_code	str or None	URL to training code if publicly available.
public_training_data	str or None	URL to training data if publicly available.
similarity_fn_name	str	Distance metric used by the model.
framework	list of str	List of frameworks used, e.g., ["Sentence Transformers", "PyTorch"].
reference	str	URL to the model's page (e.g., HuggingFace).
languages	list of str	Languages supported, e.g., ["eng-Latn"].
use_instructions	bool	Whether the model uses instruction-based formatting.
training_datasets	dict	Datasets used in training, e.g., {"ArguAna": ["test"]}.
adapted_from	str or None	Base model this one was adapted from.
superseded_by	str or None	Model that supersedes this one.
is_cross_encoder	bool	Whether the model functions as a cross-encoder.
modalities	list of str	Modalities supported, default is ["text"].

Table 1. Parameters of the `ModelMeta` class

Key discussions on zero-shot scores are in this section:

- Discussion on the definition of zero-shot: https://github.com/embeddings-benchmark/mteb/discussions/2351
- Community feedback on the initial version and filtering of zero-shot models on the leaderboard: https://github.com/embeddings-benchmark/mteb/discussions/2119

D. Reproducing Reported Results

We provide example pull requests on each of the takeaways during this period on benchmark reproducibility:

1. Embedding models can use prefixes (e5, bge models): https://github.com/embeddings-benchmark/mteb/issues/1912
2. Prompts can differ per task-type (Nomic models): https://github.com/embeddings-benchmark/mteb/pull/1685
3. Prompts can be added to both the query and the passage during retrieval (E5-mistral(Wang et al., 2023)) or only to queries (Nvembed): https://github.com/embeddings-benchmark/mteb/pull/1436
4. Some models do not normalize embeddings (Nomic ModernBERT): https://github.com/embeddings-benchmark/mteb/pull/1684
5. Models can have custom parameters during encoding (jina-v3(Sturua et al., 2024) `task_type` argument that loads LoRAs during inference): https://github.com/embeddings-benchmark/mteb/pull/1319
6. Models can have additional stages (CDE (Morris & Rush, 2024) have different stages for encoding to store embeddings): https://github.com/embeddings-benchmark/mteb/pull/2076

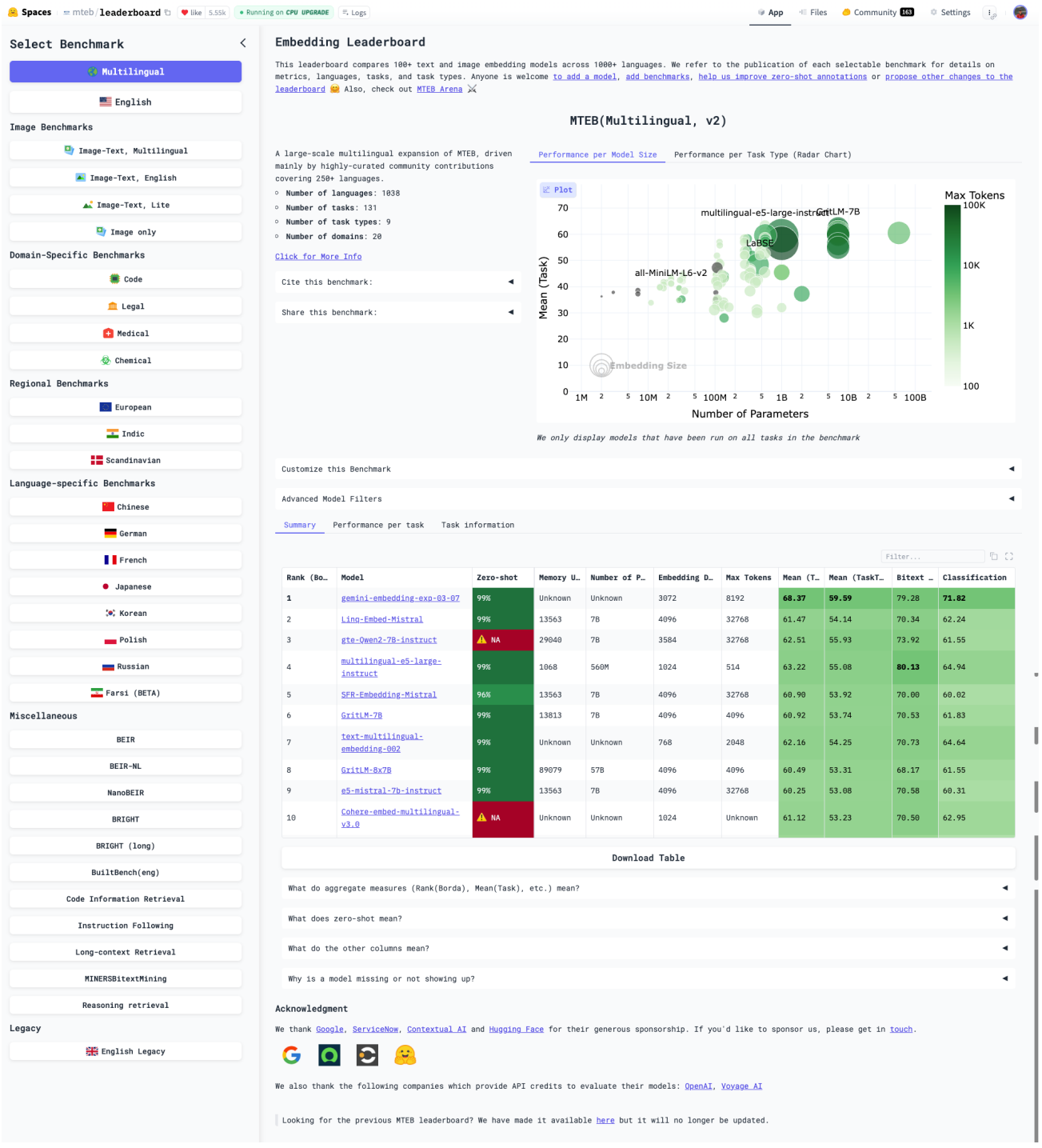

Figure 3. The MTEB Leaderboard offers an expandable collection of embedding benchmarks, with its default set as Multilingual (MMTEB, (Enevoldsen et al., 2025)). For every benchmark, a model performance per model size graph is shown, with an option to show model performance per task type in a separate tab. Models are ranked by Borda Count by default. The MTEB leaderboard also offers options to customize a benchmark by filtering on fields such as task types, languages, and domain. The left sidebar contains highlighted benchmarks by groups, while all benchmarks are available in the code.