`Unitxt`: Flexible, Shareable and Reusable Data Preparation and Evaluation for Generative AI

Elron Bandel **Yotam Perlitz** **Elad Venezian** **Roni Friedman-Melamed**
Ofir Arviv **Matan Orbach** **Shachar Don-Yehiya** **Dafna Sheinwald**
Ariel Gera **Leshem Choshen** **Michal Shmueli-Scheuer** **Yoav Katz**
IBM Research
elron.bandel@ibm.com

Abstract

In the dynamic landscape of generative NLP, traditional text processing pipelines limit research flexibility and reproducibility, as they are tailored to specific dataset, task, and model combinations. The escalating complexity, involving system prompts, model-specific formats, instructions, and more, calls for a shift to a structured, modular, and customizable solution. Addressing this need, we present `Unitxt`, an innovative library for customizable textual data preparation and evaluation tailored to generative language models. `Unitxt` natively integrates with common libraries like HuggingFace and LM-eval-harness and deconstructs processing flows into modular components, enabling easy customization and sharing between practitioners. These components encompass model-specific formats, task prompts, and many other comprehensive dataset processing definitions. The `Unitxt Catalog` centralizes these components, fostering collaboration and exploration in modern textual data workflows. Beyond being a tool, `Unitxt` is a community-driven platform, empowering users to build, share, and advance their pipelines collaboratively. Join the `Unitxt` community;
Project: `https://github.com/IBM/unitxt`.
UI: `https://bit.ly/unitxt-explore`
Video: `https://bit.ly/unitxt-video`

1 Introduction

Textual data processing has always been at the heart of NLP, but in the current landscape it has taken on new roles. A prominent one comes from LLMs' role as general interfaces, that receive an example, but also the task they should perform, general system instruction and other specifications, all in natural language. Thus, the inputs – or *prompts* – that a model receives now consist of many components, that can be combined in different ways: task instructions (Wei et al., 2022), in-context demonstrations (Brown et al., 2020), system prompts and more. At the same time, for text generation models, model outputs are themselves rich textual data, and thus can be processed and evaluated with a range of different approaches and paradigms. Therefore, textual data processing for LLMs is growing increasingly complex. It incorporates a large number of non-trivial design choices and parameters, which pose new challenges for maintaining flexibility and reproducibility in LLM research.

Broadly, research in computer science, and in particular within NLP, thrives on that combination of flexibility and reproducibility. On the one hand, it should be simple to try new ideas: to compare different approaches, choose parameters, and easily switch out one workflow or architecture with another. On the other hand, the results of these explorations must be shared in such a way that others are able to – and crucially, are likely to – reproduce and try them. To enable the above, code reuse, a well-defined API and ease of use are pivotal, ensuring reproducibility and applicability in practice. How such traits allow for widespread adoption is epitomized by the Hugging Face transformers library (Wolf et al., 2020). Today, a modest set of hyperparameters is sufficient to reproduce a training or inference workflow. This has had an undeniable and dramatic impact on the ability to make progress in the field.

Such is not the case, however, for textual data pipelines. Unfortunately, data-preparation for LLMs has no standards, Processing model inputs or outputs of the same data often comes with rewriting the code, leading to mismatches in reported values (Post, 2018), unanswerable examples and hidden bugs (Fourrier et al., 2023) and general time waste. Crucially, the additional components beyond traditional processing, such as in-context demonstrations, have no canonical API. This prevents fair comparisons between different studies, discourages exploring combinations, hinders integrating a particular approach (say, a new type of system prompt) into an existing NLP system, and prevents major

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations), pages 207–215
June 16-21, 2024 ©2024 Association for Computational Linguistics

scale-ups in terms of datasets, tasks and metrics.

To address these gaps, we introduce a new collaborative framework for unified textual data processing named Unitxt. This new Python library supports multilingual textual data processing through flexible pipelines called *recipes*. A recipe (see §4.1 and examples in §3) is a sequence of textual data processing operators, including, among others, operators that load data, pre-process it, handle the preparation of different parts of a prompt, or evaluate model predictions (see Figure 1).

Aiming for reuse, Unitxt ships with a catalog containing a wide variety of pre-defined recipes for various tasks. These are all based on a diverse set of built-in operators that are also shared in the catalog. Having a centralized location for these components, where anyone can add new ingredients (such as recipes or operators), or share existing ones, fosters collaboration, transparency and reproducibility.

As fitting a Recipe, the modularity of Unitxt enables mixing and matching of ingredients to create new recipes. This ability to mix and match ingredients enables Unitxt to support 100K+ recipe configurations, allowing users to experiment with a large set of such recipes by to obtain multiple configurations of tasks, datasets and new formatting (see §3 for example).

Changing libraries is always a nuisance; therefore, Unitxt is designed to seamlessly integrate with preexisting code, offering a hassle-free experience without even needing a pip install. For instance, Unitxt can load HuggingFace datasets and produce outputs that adhere to the same format, allowing it to integrate seamlessly with other parts of your codebase (§4.4.1). Demonstrating this, incorporating Unitxt, with all its tasks, datasets, templates and metrics into LM-eval-harness (Gao et al., 2023) required only 30 lines of code, while preserving the current API and ensuring a smooth transition and compatibility with existing workflows (App. A).

Unitxt, an open-source library, is under active development by IBM and the community. The code and documentation are available on GitHub at: https://github.com/IBM/unitxt, the UI, at https://bit.ly/unitxt-explore while the demo video is at https://bit.ly/unitxt-video.

2 Use cases

Unitxt for evaluation: The increasing capabilities of LLMs require evaluation frameworks that test models over an unprecedented number of datasets, tasks and configurations (Liang et al., 2022; Gao et al., 2023; Contributors, 2023). Unitxt can serve as the backbone of such evaluation efforts, by supporting easy changes across multiple important axes, including tasks, languages, prompt structure (e.g. instructions, verbalizations, etc.), augmentation robustness and more. Moreover, with the Unitxt Catalog, different distinct projects can share their full evaluation pipelines, making their data-preparation and evaluation metrics reproducible.

Unitxt for training: Modern LLM training frameworks have extensive data requirements to attain state-of-the-art performance. Multiple datasets across diverse domains and languages need to be leveraged to impart broad capabilities; Various prompt formulations and *verbalizations* are necessary to enable instruction-following, where verbalizations are the final text form. However, combining heterogeneous data sources and textual representations poses significant engineering challenges. Without a common underlying framework, data augmentation, multitask learning and few-shot tuning become prohibitively complex. This is where Unitxt steps in, as an indispensable data backend.

Unitxt enables seamless fusion of diverse datasets. Moreover, the standardized format also facilitates changes to the datasets, dynamic prompt generation, data augmentations and model-specific format, to name just a few. By handling the data wrangling complexity, Unitxt empowers researchers to focus on creating performant, robust and safe LLMs.

For both evaluation and training, Unitxt *has already been adopted* as a core utility for LLMs in IBM by multiple teams working on various NLP tasks, including classification, extraction, summarization, generation, question answering, code, biases and more. In total, the open source catalog contains more than 100K possible pipeline configurations.

3 Unitxt: Library Tour

To introduce unitxt, we begin with a tour of the library, and specifically, with the creation of a recipe. A recipe contains all the data-processing and metric configurations needed, including the data, task,

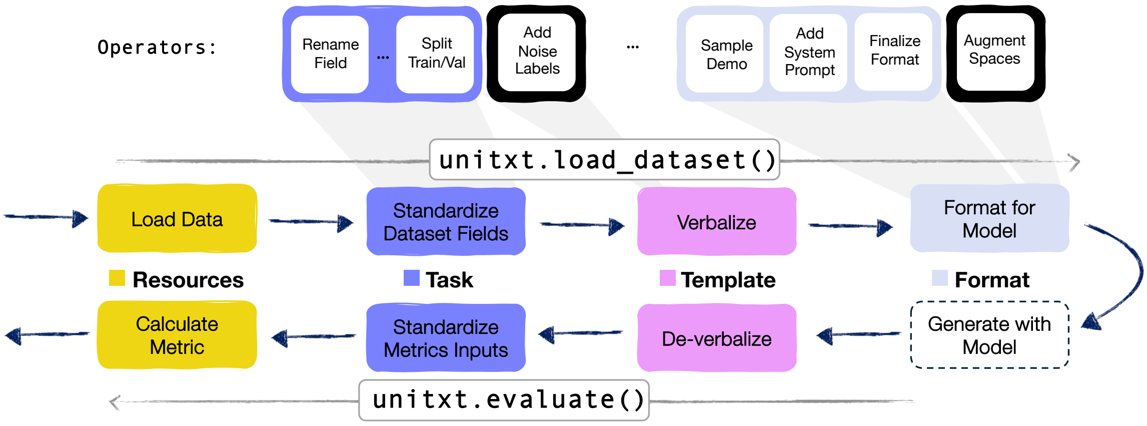


Figure 1: Unitxt flow: The upper section illustrates the data-preparation pipeline §4.4.1, encompassing raw dataset loading, standardization according to the task interface, verbalization using templates, and application of formatting. The lower section showcases the evaluation pipeline §4.4.2, involving de-verbalization operations and output standardization before performance evaluation with task-defined metrics. All components are described in §4.2.

template and formatting (see details in §4). Here we define a recipe that loads the STS-B dataset for a sentence similarity task:

```
recipe = """
    card=cards.stsb, # dataset info card
    template=templates.text_similarity,
    sys_prompt=prompts.helpful,
    format=formats.user_agent,
    num_demos=1
"""
```

With a recipe, a concrete dataset can be loaded:

```
dataset = unitxt.load_dataset(recipe)
```

Importantly, every data instance in a dataset loaded with a unitxt recipe contains a fully prepared source text, which can be directly passed as input to the model. For example, here is such source text for one sentence-similarity data instance, integrated with three formatting decisions, a "helpful model" system-prompt, a user-agent response schema and one demonstration:

```
[System] you are helpful model [/System]
[User]: for the following texts rank the
        similarity between 1 to 5.
        Text 1: "i love ice cream"
        Text 2: "i like ice cream"
[Agent]: 4.8
[User]: Text 1: "i hate pizza"
        Text 2: "i like pizza"
[Agent]:
```

Loading a dataset with a unitxt recipe also adds a metric-ready target text (created from the original target) to each data instance. To Evaluate the model's textual predictions, we call:

```
results = unitxt.evaluate(
    dataset,
    predictions=predictions,
)
```

The evaluation results are a dictionary of task defined metric names and the values computed for them.

4 Design

In this section we outline the design of Unitxt. Unitxt processes data by applying a modular sequence of operators, which are segmented into 5 key ingredients (§4.2) color-coded as in Fig. 1: 🟨 Resources, 🟦 Task, 🟪 Template, ⬜ Format and ⬛ Extensions. These ingredients are then used to build the **data preparation** (§4.4.1) and **evaluation** (§4.4.2) pipelines.

4.1 Unitxt Building Blocks

When loading a dataset (as demonstrated in §3), the Unitxt ingredients are retrieved based on a *Data-Task Card* and a *Recipe*.

🟨🟦 **Data-Task Card** Defines how raw data (inputs and targets) are standardized for a certain task. Typically, this includes data wrangling actions, e.g. renaming fields, filtering data instances, modifying values, train/test/val splitting etc. It also describes the resource from which the data is loaded.

🟨🟦🟪⬜⬛ **Recipe** A *Recipe* holds a complete specification of a Unitxt pipeline: including the Resources, Task, Template, Format and Extensions.

4.2 Unitxt Ingredients

🟨 **Resources** Raw data and metrics are external resources utilized by Unitxt. Unitxt implements several APIs for raw-data and metric loading (e.g., from Huggingface Hub, local files, and cloud storage).

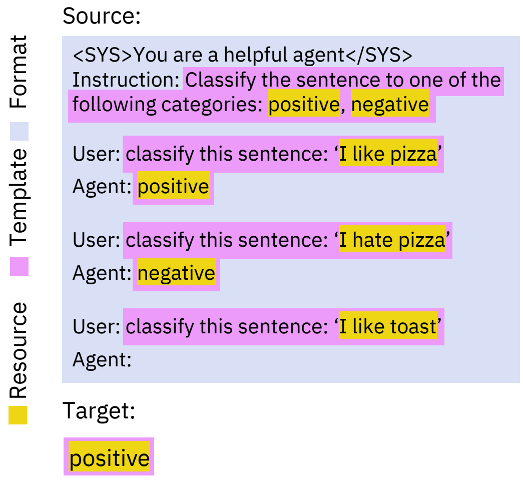
🟦 **Task** A Unitxt *Task* follows the formal definition of an NLP task, such as multi-label classification, named entity extraction, abstractive summarization or translation. A task is defined by its standard interface – namely, input and output fields – and by its evaluation metrics. Given a dataset, its contents are standardized into the fields defined by an appropriate task by a Data-Task Card (§4.1).

As an example of a defined task, consider sentence similarity: it has two input fields (named "sentence1" and, "sentence2"), one output field (named "label") and the conventional metric is Spearman correlation (Spearman, 1904).

🟪 **Template** A Unitxt *Template* defines the verbalizations to be applied to the inputs and targets, as well as the de-verbalization operations over the model predictions. For example, in Fig 2, applying the template to I like toast verbalizes it into classify the sentence: "I like toast".

In the other direction, template de-verbalization involves two steps. First, a general standardization of the output texts: taking only the first non-empty line of a model's predictions, lowercasing, stripping whitespaces, etc. The second step standardizes the output to the specific task at-hand. For example, in Sentence Similarity, a prediction may be a quantized float number outputted as a string (e.g "2.43"), or a verbally expressed numeric expression (e.g "two and a half"). This depends on the verbalization defined by the template and the in-context demonstrations it constructs. Both types of outputs should be standardized before evaluation begins – e.g. to a float for sentence similarity. Having the de-verbalization steps defined within the template enables templates reuse across different models and datasets.

Crucially, in contrast to existing solutions (e.g., Bach et al., 2022) the templates, datasets and tasks in Unitxt are not exclusively tied. Each task can harness multiple templates and a template can be used for different datasets. Thus, the modularity of Unitxt allows mixing and matching, significantly enhancing re-usability and flexibility.



Source:

<SYS>You are a helpful agent</SYS>
Instruction: Classify the sentence to one of the following categories: positive, negative

User: classify this sentence: 'I like pizza'
Agent: positive

User: classify this sentence: 'I hate pizza'
Agent: negative

User: classify this sentence: 'I like toast'
Agent:

Target:

positive

Figure 2: Illustration of the data preparation pipeline (§4.4.1), depicting the transformation from raw data and formatting specifications to the final text output. Components include Resources (raw data), Format (model-specific formatting requirements), and Template (verbalization).

⬜ **Format** A Unitxt *Format* defines a set of extra formatting requirements, unrelated to the underlying data or task, including those pertaining to system prompts, special tokens or user/agent prefixes, and in-context demonstrations. Continuing the example from Figure 2, the Unitxt format receives the text produced by the template classify the sentence: "I like toast", and adds the system prompt <SYS>You are a helpful agent</SYS>, the Instruction-User-Agent schema cues, and the two presented demonstrations.

■ **Extensions** Unitxt supports *Extensions* such as *input-augmentation* (for example, adding random whitespace, introducing spelling mistakes, or replacing words with their synonyms) or *label-noising* (replaces the labels in the demonstrations randomly from a list of options). Such extensions can be added anywhere in the data-preparation pipeline between any two operators, depending on the desired logic (see Fig. 1). Unitxt supports the addition of custom extensions to the Catalog. Each extension is an independent unit, reusable across different datasets and tasks, templates and formats.

4.3 Unitxt Catalog

All Unitxt artifacts – recipes, data-task cards, templates, pre-processing operators, formats and metrics – are stored in the Unitxt Catalog. In addition to the open-source catalog, that can be found in the documentation, users can choose to define a private catalog. This enables teams and organizations to harness the open `Unitxt Catalog` while upholding organizational requirements for additional proprietary artifacts.

4.4 `Unitxt` Pipelines

4.4.1 Data Preparation Pipeline

The data preparation pipeline (top part ot Fig. 1) begins with standardizing the raw data into the task interface, as defined in the data-task card (§4.1). The examples are then verbalized by the template, and the format operator applies system prompts, special tokens and in-context learning examples (§4.2), as illustrated in Figure 2. To maintain compatibility, the output of this pipeline is an HF dataset, that can be saved or pushed to the hub.

4.4.2 Evaluation Pipeline

The evaluation pipeline (bottom part of Fig. 1) is responsible for producing a list of evaluation scores that reflect model performance. It includes a deverbalization of the model outputs (as defined in the template, see §4.2), and a computation of performance by the metrics defined in the task. The standardization of the task interface, namely, having fixed names and types for its input and output fields, allows the use of any metric that accept such fields as input. In addition to the computed evaluation scores, `Unitxt` metrics supports a built in mechanism for confidence interval reporting, using statistical bootstrap (Perlitz et al., 2023).

5 `Unitxt` UI: Explore & Preview

The objective of the user interface is to guide users through the essential steps of recipe creation, illustrated with pertinent examples. Additionally, it allows for catalog exploration. The UI complements the experience with the option to execute the examples on some pre-set model (e.g., flan-t5-base), get the predictions and associated scores.

The interaction entry point is the tasks. Upon clicking, the tasks taxonomy is presented, and the users have the option to choose the applicable task type. Selecting a task results in showing only the relevant datasets and templates. Once the user selects a dataset, and a template, and presses “Generate Prompts” a random example enhanced with the template is loaded. If the user wants to augment the input with system prompt, or response-schema those will be instantly added when opted for. As in-context learning evaluations are supported, the user can select the preferred number of shots. Once satisfied with the example, the user has the option to proceed with executing it on a model, wherein the predictions and corresponding scores will be displayed for this specific example. Further, going to the code tab, the user can copy the associated code into a notebook and run. Users have the option to explore various examples, enhancing their comprehension and confidence in the chosen configuration.

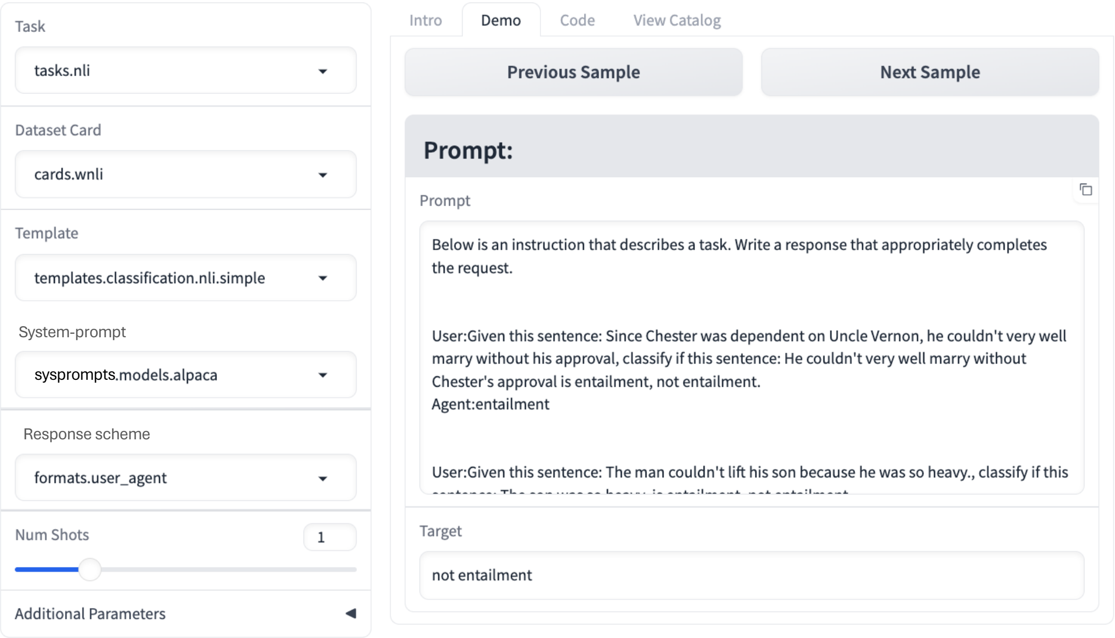
6 Related work

Standardized data processing for evaluation and training has been a longstanding need in the NLP community and has been repeatedly addressed in the past. Datasets (Lhoest et al., 2021) and Evaluate[1] are community-driven libraries, providing a standardized interface to diverse corpora and metrics, as well as supporting many data processing operations. These packages, however, fall short of providing a standardized, shareable and reproducible framework to cast the raw data into textual prompts and cast them back from text to a metric digestible format. The lack of such a framework hinders reproducibility, as often slight variations in ad-hoc text processing code may yield significantly different evaluation scores. Moreover, it also prevents users from easily scaling up their experiment, as each task and dataset often requires specific code for processing and evaluation. `Unitxt` builds on top of these frameworks, harnessing them as resources (§4.1) to produce a full data-preparation and evaluation framework.

While several existing frameworks have contributed to data pipeline management workflows, a common drawback, for those we are aware of, is the absence of a well-defined and flexible modularity in their design, such as the ability to define specific components for system prompts, task instructions and model-specific formats. This absence of clearly defined components makes it challenging to share and customize such pipelines effectively, across different datasets and tasks.

Like `Unitxt`, Tasksource (Sileo, 2023) supplies tools for consistent preprocessing over different datasets, simplifying their usage. However, it is primarily designed for discriminative tasks, uses fixed formats and lacks a modular design that enables sharing, mixing and matching, and overall

[1] `https://github.com/huggingface/evaluate/`



Task
tasks.nli

Dataset Card
cards.wnli

Template
templates.classification.nli.simple

System-prompt
sysprompts.models.alpaca

Response scheme
formats.user_agent

Num Shots
1

Prompt

Below is an instruction that describes a task. Write a response that appropriately completes the request.

User:Given this sentence: Since Chester was dependent on Uncle Vernon, he couldn't very well marry without his approval, classify if this sentence: He couldn't very well marry without Chester's approval is entailment, not entailment.
Agent:entailment

User:Given this sentence: The man couldn't lift his son because he was so heavy., classify if this sentence: The son was so heavy, is entailment, not entailment.

Target

not entailment

Figure 3: Exploration UI showcasing configuration options for model input creation on the left, including parameters such as task, dataset card, template, system-prompt, response-schema, number of examples, and optional augmentations. The resulting model input is displayed in the prompt window.

flexibility in processing steps. Promptsource (Bach et al., 2022) focuses on making and sharing natural language prompts but doesn't handle other types of data processing. Each prompt is tied to just one dataset, making it hard to reuse and share. Furthermore, prompts aren't split into system, instruction, and format parts, limiting options for flexibility and reuse. SeqIO (Roberts et al., 2022), offers task-based pipelines encompassing pre-processing, post-processing, and handling metrics. However, a structured breakdown of these processing steps is absent, limiting the creation of shareable catalogs within the community. In this framework, each process is a generic function and specialized steps are missing, like those designed for system prompts.

A different branch of solutions are language model evaluation frameworks such as OpenCompass (Contributors, 2023), HELM (Liang et al., 2022) and LM-eval-harness (Gao et al., 2023) also implement their own standardized data processing pipelines in order to obtain verbalized prompts for LMs. These, however, are highly coupled with the inference engine and cannot be used as standalone data-processing pipelines or integrated into other code bases.

7 Conclusion

In this paper, we have introduced `Unitxt`, an open-source Python library aimed at unifying textual data processing pipelines for large language models. `Unitxt` provides a modular, flexible framework that enables mixing and matching of various pipeline components like loaders, templates, formats and metrics. `Unitxt` key capabilities are, standardization, flexibility, collaboration and scale.

`Unitxt` has already been successfully deployed for large language model evaluation and training within IBM. As the library matures through open-source community involvement, we hope its adoption will grow to push the frontiers of textual data processing for LLMs. We believe `Unitxt` has the potential to significantly impact research and development of large language models by unifying textual data processing. Through its emphasis on flexibility, reproducibility and collaboration, unitxt can help drive progress towards more capable, safer and trustworthy LLMs.

8 Limitations

While unitxt makes significant progress towards unified textual data processing for LLMs, some limitations still remain:

- The `Unitxt Catalog`, while already substantial in coverage, needs expansion to encompass more datasets, languages, and niche tasks. Community contributions will be key to enhancing catalog diversity.

- Coverage of evaluation metrics, especially for generative tasks, needs improvement. We plan to incorporate more reference-free and LLM-based metrics going forward.

- Training data augmentation abilities, while flexible currently, can be expanded further with techniques like back-translation for multilinguality.

- While using Unitxt recipes is as simple as specifying the recipe ingrediants, adding new datasets or operators requires learning the Unitxt operator language. Additional documentation, examples and IDE support could help alleviate this.

Addressing these limitations through open-source community involvement is the major focus going forward. By tapping into collective expertise, we envision unitxt becoming an indispensable textual data processing backbone for the responsible development, evaluation and deployment of large language models.

References

Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Xiangru Tang, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. Promptsource: An integrated development environment and repository for natural language prompts.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Clémentine Fourrier, Alex Cabrera, Stella Biderman, Nathan Habib, and Thomas Wolf. 2023. Open llm leaderboard: Drop deep dive.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Yotam Perlitz, Elron Bandel, Ariel Gera, Ofir Arviv, Liat Ein-Dor, Eyal Shnarch, Noam Slonim, Michal Shmueli-Scheuer, and Leshem Choshen. 2023. Efficient benchmarking (of language models). *arXiv preprint arXiv:2308.11696*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James

Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. 2022. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*.

Damien Sileo. 2023. tasksource: Structured dataset preprocessing annotations for frictionless extreme multi-task learning and evaluation. *arXiv preprint arXiv:2301.05948*.

C. Spearman. 1904. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

A LM Eval Harness Integration

LM-eval-harness (Gao et al., 2023) is one of the most commonly used open source evaluation frameworks. It leverages a yaml-based declarative language which defines loading of the dataset, the dataset splits, the prompt used and the metrics in a single file for each task. Many tasks are supported, including multi-class classification, multiple choice question answering, and generation tasks. Unitxt was integrated into LM-eval-harness to extend LM-eval-harness to support new tasks and metrics that currently are not supported, including multi-label classification, named entity extraction, and target sentiment analysis.

Since `Unitxt` recipes can be loaded as standard HF datasets, no code changes were required to add the `Unitxt` data preparation pipeline to LM-eval-harness. Adding a `Unitxt` recipe requires only one line change in a LM-eval-harness yaml (see Figure 4 in Appendix). Adding the `Unitxt` metrics required about 30 lines of code, to register the `Unitxt` metrics to the LM-eval-harness metrics registry.

```
1  group: glue
2  task: unitxt_unfair_tos
3  dataset_path: unitxt/data
4  dataset_name: card=cards.unfair_tos,template_card_index=templates.classification.
       multi_label.default,format=formats.user_agent
5  output_type:  generate_until
6  training_split: train
7  validation_split: validation
8  doc_to_text: "{{source}}"
9  doc_to_target: target
10 generation_kwargs:
11   until:
12     - "</s>"
13 metric_list:
14   - metric: unitxt_f1_micro_multi_label
15 metadata:
16   version: 1.0
```

Figure 4: **Unitxt and LM-eval-harness integration**. A Unitxt recipe can be integrated as an LM-eval-harness task, by setting the *dataset_path* (line 3) to *unitxt/data* and the setting the recipe in the *dataset_name* (line 4). Unitxt metrics can be used like any other metric (line 14).