The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models

Ian Tenney,[*] James Wexler,[*] Jasmijn Bastings, Tolga Bolukbasi,
Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna,
Carey Radebaugh, Emily Reif, Ann Yuan
Google Research
`{iftenney,jwexler}@google.com`

Abstract

We present the Language Interpretability Tool (LIT), an open-source platform for visualization and understanding of NLP models. We focus on core questions about model behavior: Why did my model make this prediction? When does it perform poorly? What happens under a controlled change in the input? LIT integrates local explanations, aggregate analysis, and counterfactual generation into a streamlined, browser-based interface to enable rapid exploration and error analysis. We include case studies for a diverse set of workflows, including exploring counterfactuals for sentiment analysis, measuring gender bias in coreference systems, and exploring local behavior in text generation. LIT supports a wide range of models—including classification, seq2seq, and structured prediction—and is highly extensible through a declarative, framework-agnostic API. LIT is under active development, with code and full documentation available at `https://github.com/pair-code/lit`.[1]

1 Introduction

Advances in modeling have brought unprecedented performance on many NLP tasks (e.g. Wang et al., 2019), but many questions remain about the behavior of these models under domain shift (Blitzer and Pereira, 2007) and adversarial settings (Jia and Liang, 2017), and for their tendencies to behave according to social biases (Bolukbasi et al., 2016; Caliskan et al., 2017) or shallow heuristics (e.g. McCoy et al., 2019; Poliak et al., 2018). For any new model, one might want to know: What kind of examples does my model perform poorly on? Why did my model make this prediction? And critically, does my model behave consistently if I change things like textual style, verb tense, or pronoun gender? Despite the recent explosion of work on model understanding and evaluation (e.g. Belinkov et al., 2020; Linzen et al., 2019; Ribeiro et al., 2020), there is no "silver bullet" for analysis. Practitioners must often experiment with many techniques, looking at local explanations, aggregate metrics, and counterfactual variations of the input to build a full understanding of model behavior.

Existing tools can assist with this process, but many come with limitations: offline tools such as TFMA (Mewald, 2019) can provide only aggregate metrics, interactive frontends (e.g. Wallace et al., 2019) may focus on single-datapoint explanation, and more integrated tools (e.g. Wexler et al., 2020; Mothilal et al., 2020; Strobelt et al., 2018) often work with only a narrow range of models. Switching between tools or adapting a new method from research code can take days of work, distracting from the real task of error analysis. An ideal workflow would be seamless and interactive: users should see the data, what the model does with it, and why, so they can quickly test hypotheses and build understanding.

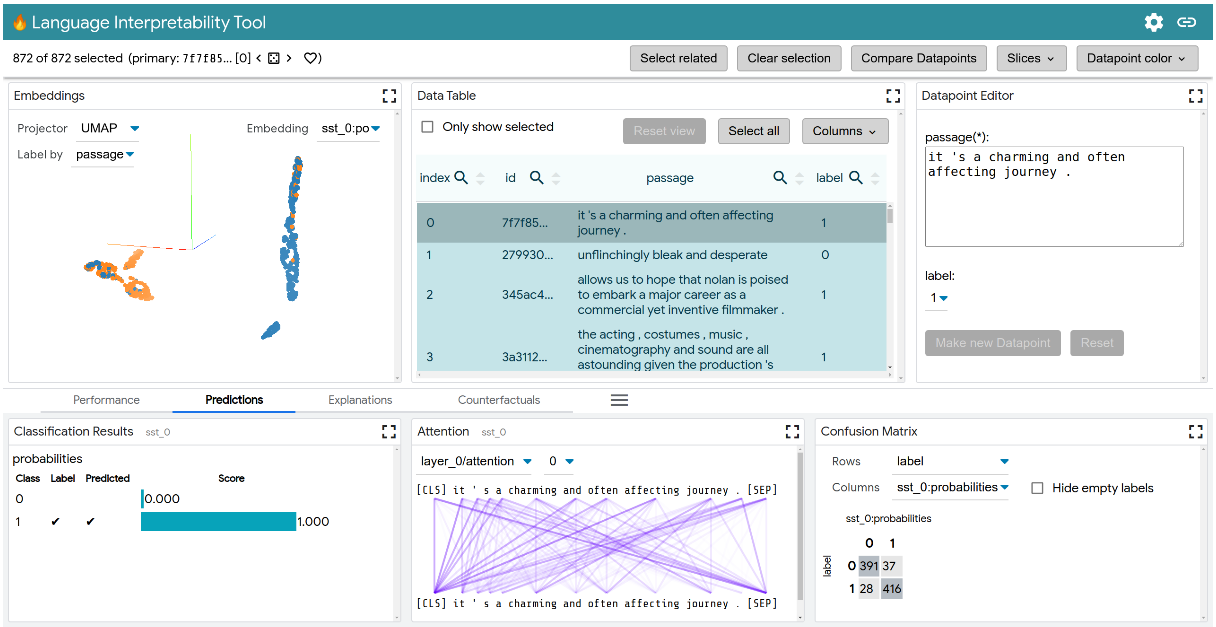
With this in mind, we introduce the Language Interpretability Tool (LIT), a toolkit and browser-based user interface (UI) for NLP model understanding. LIT supports local explanations—including salience maps, attention, and rich visualizations of model predictions—as well as aggregate analysis—including metrics, embedding spaces, and flexible slicing—and allows users to seamlessly hop between them to test local hypotheses and validate them over a dataset. LIT provides first-class support for counterfactual generation: new datapoints can be added on the fly, and their effect on the model visualized immediately. Side-by-side comparison allows for two models, or two datapoints, to be visualized simultaneously.

We recognize that research workflows are con-

[*] Equal contribution.

[1] A video walkthrough is available at `https://youtu.be/j0OfBWFUqIE`.

Proceedings of the 2020 EMNLP (Systems Demonstrations), pages 107–118
November 16-20, 2020.



index	id	passage	label
0	7f7f85...	it 's a charming and often affecting journey .	1
1	279930...	unflinchingly bleak and desperate	0
2	345ac4...	allows us to hope that nolan is poised to embark a major career as a commercial yet inventive filmmaker .	1
3	3a3112...	the acting , costumes , music , cinematography and sound are all astounding given the production 's	1

```
it 's a charming and often
affecting journey .
```

Class	Label	Predicted	Score
0			0.000
1	✓	✓	1.000

	0	1
0	391	37
1	28	416

Figure 1: The LIT UI, showing a fine-tuned BERT (Devlin et al., 2019) model on the Stanford Sentiment Treebank (Socher et al., 2013) development set. The top half shows a selection toolbar, and, left-to-right: the embedding projector, the data table, and the datapoint editor. Tabs present different modules in the bottom half; the view above shows classifier predictions, an attention visualization, and a confusion matrix.

stantly evolving, and designed LIT along the following principles:

- **Flexible:** Support a wide range of NLP tasks, including classification, seq2seq, language modeling, and structured prediction.
- **Extensible:** Designed for experimentation, and can be reconfigured and extended for novel workflows.
- **Modular:** Components are self-contained, portable, and simple to implement.
- **Framework agnostic:** Works with any model that can run from Python —including TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2019), or remote models on a server.
- **Easy to use:** Low barrier to entry, with only a small amount of code needed to add models and data (Section 4.3), and an easy path to access sophisticated functionality.

2 User Interface and Functionality

LIT has a browser-based UI comprised of modules (Figure 1) which contain controls and visualizations for specific tasks (Table 1). At the most basic level, LIT works as a simple demo server: one can enter text, press a button, and see the model's predictions. But by loading an evaluation set, allowing dynamic datapoint generation, and an array of interactive visualizations, metrics, and modules that respond to user input, LIT supports a much richer set of user journeys:

J1 - Explore the dataset. Users can interactively explore datasets using different criteria across modules like the data table and the embeddings module (similar to Smilkov et al. (2016)), in which a PCA or UMAP (McInnes et al., 2018) projection can be rotated, zoomed, and panned to explore clusters and global structures (Figure 1-top left).

J2 - Find interesting datapoints. Users can identify interesting datapoints for analysis, cycle through them, and save selections for future use. For example, users can select off-diagonal groups from a confusion matrix, examine outlying clusters in embedding space, or select a range based on scalar values (Figure 4 (a)).

J3 - Explain local behavior. Users can deep-dive into model behavior on selected individual datapoints using a variety of modules depending on the model task and type. For instance, users can compare explanations from salience maps, including local gradients (Li et al., 2016) and LIME (Ribeiro et al., 2016), or look for patterns in attention heads (Figure 1-bottom).

Module	Description
Attention	Displays an attention visualization for each layer and head.
Confusion Matrix	A customizable confusion matrix for single model or multi-model comparison.
Counterfactual Generator	Creates counterfactuals for selected datapoint(s) using a variety of techniques.
Data Table	A tabular view of the data, with sorting, searching, and filtering support.
Datapoint Editor	Editable details of a selected datapoint.
Embeddings	Visualizes dataset by layer-wise embeddings, projected down to 3 dimensions.
Metrics Table	Displays metrics such as accuracy or BLEU score, on the whole dataset and slices.
Predictions	Displays model predictions, including classification, text generation, language model probabilities, and a graph visualization for structured prediction tasks.
Salience Maps	Shows heatmaps for token-based feature attribution for a selected datapoint using techniques like local gradients and LIME.
Scalar Plot	Displays a jitter plot organizing datapoints by model output scores, metrics or other scalar values.

Table 1: Built-in modules in the Language Interpretability Tool.

J4 - Generate new datapoints. Users can create new datapoints based on datapoints of interest either manually through edits, or with a range of automatic counterfactual generators, such as backtranslation (Bannard and Callison-Burch, 2005), nearest-neighbor retrieval (Andoni and Indyk, 2006), word substitutions ("great → terrible"), or adversarial attacks like HotFlip (Ebrahimi et al., 2018) (Figure A.1). Datapoint provenance is tracked to facilitate easy comparison.

J5 - Compare side-by-side. Users can interactively compare two or more models on the same data, or a single model on two datapoints simultaneously. Visualizations automatically "replicate" for a side-by-side view.

J6 - Compute metrics. LIT calculates and displays metrics for the whole dataset, the current selection, as well as on manual or automatically-generated slices (Figure 3 (c)) to easily find patterns in model performance.

LIT's interface allows these user journeys to be explored interactively. Selecting a dataset and model(s) will automatically show compatible modules in a multi-pane layout (Figure 1). A tabbed bottom panel groups modules by workflow and functionality, while the top panel shows persistent modules for dataset exploration.

These modules respond dynamically to user interactions. If a selection is made in the embedding projector, for example, the metrics table will respond automatically and compute scores on the selected datapoints. Global controls make it easy to page through examples, enter a comparison mode, or save the selection as a named "slice". In this way, the user can quickly explore multiple workflows using different combinations of modules.

A brief video demonstration of the LIT UI is available at `https://youtu.be/j0OfBWFUqIE`.

3 Case Studies

Sentiment analysis. How well does a sentiment classifier handle negation? We load the development set of the Stanford Sentiment Treebank (SST; Socher et al., 2013), and use the search function in LIT's data table **(J1, J2)** to find the 56 datapoints containing the word "not". Looking at the Metrics Table **(J6)**, we find that surprisingly, our BERT model (Devlin et al., 2019) gets 100% of these correct! But we might want to know if this is truly robust. With LIT, we can select individual datapoints and look for explanations **(J3)**. For example, take the negative review, "*It's not the ultimate depression-era gangster movie.*". As shown in Figure 2, salience maps suggest that "not" and "ultimate" are important to the prediction.

We can verify this by creating modified inputs, using LIT's datapoint editor **(J4)**. Removing "not" gets a strongly positive prediction from "*It's the ultimate depression-era gangster movie.*", while replacing "ultimate" to get "*It's not the worst depression-era gangster movie.*" elicits a mildly positive score from our model.

Gender bias in coreference. Does a system encode gendered associations, which might lead to incorrect predictions? We load a coreference model

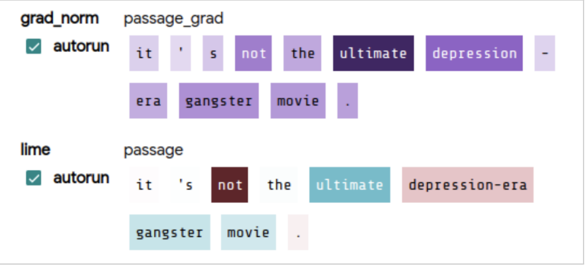
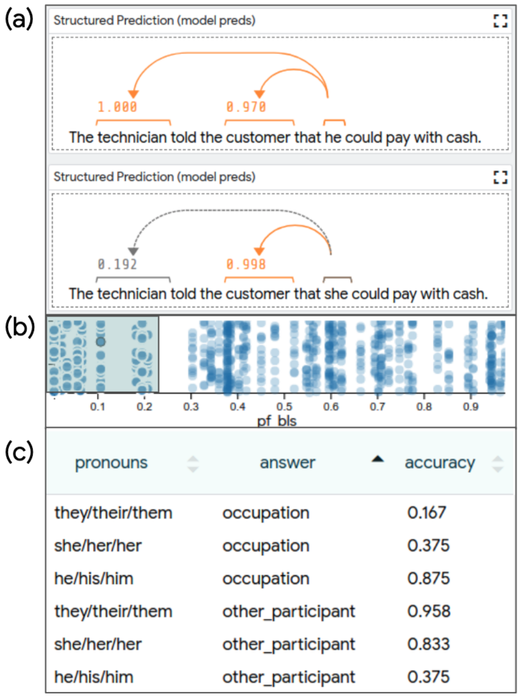


Figure 2: Salience maps on “*It’s not the ultimate depression-era gangster movie.*”, suggesting that “not” and “ultimate” are important to the model’s prediction.


(a)
Structured Prediction (model preds)
1.000
0.970
Structured Prediction (model preds)
0.192
0.998
(b)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
pf_bls
(c)

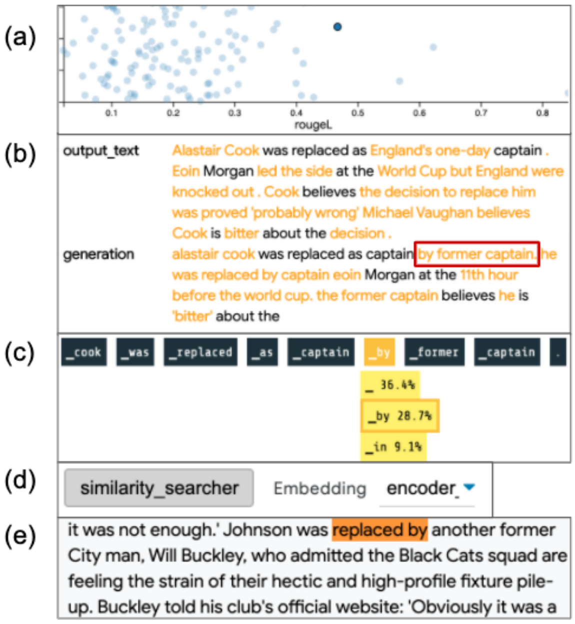
The technician told the customer that he could pay with cash.

The technician told the customer that she could pay with cash.

pronouns	answer	accuracy
they/their/them	occupation	0.167
she/her/her	occupation	0.375
he/his/him	occupation	0.875
they/their/them	other_participant	0.958
she/her/her	other_participant	0.833
he/his/him	other_participant	0.375

Figure 3: Exploring a coreference model on the Winogender dataset.

trained on OntoNotes (Hovy et al., 2006), and load the Winogender (Rudinger et al., 2018) dataset into LIT for evaluation. Each Winogender example has a pronoun and two candidate referents, one a occupation term like (“technician”) and one an “other participant” (like “customer”). Our model predicts coreference probabilities for each candidate. We can explore the model’s sensitivity to pronouns by comparing two examples side-by-side (see Figure 3 (a).) We can see how commonly the model makes similar errors by paging through the dataset (**J1**), or by selecting specific slices of interest. For example, we can use the scalar plot module (**J2**) (Figure 3 (b)) to select datapoints where the occupation term is associated with a high proportion of male or female workers, according to the U.S. Bureau of Labor Statistics (BLS; Caliskan et al., 2017).



output_text Alastair Cook was replaced as England's one-day captain . Eoin Morgan led the side at the World Cup but England were knocked out . Cook believes the decision to replace him was proved 'probably wrong' Michael Vaughan believes Cook is bitter about the decision .

generation alastair cook was replaced as captain by former captain. he was replaced by captain eoin Morgan at the 11th hour before the world cup. the former captain believes he is 'bitter' about the

it was not enough.' Johnson was replaced by another former City man, Will Buckley, who admitted the Black Cats squad are feeling the strain of their hectic and high-profile fixture pile-up. Buckley told his club's official website: 'Obviously it was a

Figure 4: Investigating a local generation error, from selection of an interesting example to finding relevant training datapoints that led to an error.

In the Metrics Table (**J6**), we can slice this selection by pronoun type and by the true referent. On the set of male-dominated occupations (< 25% female by BLS), we see the model performs well when the ground-truth agrees with the stereotype - e.g. when the answer is the occupation term, male pronouns are correctly resolved 83% of the time, compared to female pronouns only 37.5% of the time (Figure 3 (c)).

Debugging text generation. Does the training data explain a particular error in text generation? We analyze a T5 (Raffel et al., 2019) model on the CNN-DM summarization task (Hermann et al., 2015), and loosely follow the steps of Strobelt et al. (2018). LIT’s scalar plot module (**J2**) allows us to look at per-example ROUGE scores, and quickly select an example with middling performance (Figure 4 (a)). We find the generated text (Figure 4 (b)) contains an erroneous constituent: “*alastair cook was replaced as captain by former captain* ...”. We can dig deeper, using LIT’s language modeling module (Figure 4 (c)) to see that the token “by” is predicted with high probability (28.7%).

To find out how T5 arrived at this prediction, we utilize the “similarity searcher” component through the counterfactual generator tab (Figure 4 (d)). This performs a fast approximate nearest-neighbor lookup (Andoni and Indyk, 2006) from a pre-built index over the training corpus, using embeddings from the T5 decoder. With one click, we can retrieve 25 nearest neighbors and add them to the LIT UI for inspection (as in Figure A.1). We see that the words “captain” and “former” appear 34 and 16 times in these examples–along with 3 occurrences of “replaced by” (Figure 4 (e))–suggesting a strong prior toward our erroneous phrase.

4 System design and components

The LIT UI is written in TypeScript, and communicates with a Python backend that hosts models, datasets, counterfactual generators, and other interpretation components. LIT is agnostic to modeling frameworks; data is exchanged using NumPy arrays and JSON, and components are integrated through a declarative “spec” system (Section 4.4) that minimizes cross-dependencies and encourages modularity. A more detailed design schematic is given in the Appendix, Figure A.2.

4.1 Frontend

The browser-based UI is a single-page web app, built with lit-element[2] and MobX[3]. A shared framework of “service” objects tracks interaction state, such as the active model, dataset, and selection, and coordinates a set of otherwise-independent modules which provide controls and visualizations.

4.2 Backend

The Python backend serves models, data, and interpretation components. The server is stateless, but includes a caching layer for model predictions, which frees components from needing to store intermediate results and allows interactive use of large models like BERT (Devlin et al., 2019) and GPT-2 (Radford et al., 2019). Component types include:

- **Models** which implement a `predict()` function, `input_spec()`, and `output_spec()`.
- **Datasets** which load data from any source and expose an `.examples` field and a `spec()`.
- **Interpreters** are called on a model and a set of datapoints, and return output—such as a salience map—that may also depend on the model’s predictions.
- **Generators** are interpreters that return new input datapoints from source datapoints.
- **Metrics** are interpreters which return aggregate scores for a list of inputs.

These components are designed to be self-contained and interact through minimalist APIs, with most exposing only one or two methods plus spec information. They communicate through standard Python and NumPy types, making LIT compatible with most common modeling frameworks, including TensorFlow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019). Components are also portable, and can easily be used in a notebook or standalone script. For example:

```
dataset = SSTData(...)
model = SentimentModel(...)
lime = lime_explainer.LIME()
lime.run([dataset.examples[0]],
         model, dataset)
```

will run the LIME (Ribeiro et al., 2016) component and return a list of tokens and their importance to the model prediction.

4.3 Running with your own model

LIT is built as a Python library, and its typical use is to create a short `demo.py` script that loads models and data and passes them to the `lit.Server` class:

```
models = {'foo': FooModel(...),
          'bar': BarModel(...)}
datasets = {'baz': BazDataset(...)}
server = lit.Server(models, datasets)
server.serve()
```

A full example script is included in the Appendix (Figure A.3). The same server can host several models and datasets for side-by-side comparison, and can also interact with remotely-hosted models.

4.4 Extensibility: the `spec()` system

NLP models come in many shapes, with inputs that may involve multiple text segments, additional categorical features, scalars, and more, and output modalities that include classification, regression, text generation, and span labeling. Models may have multiple heads of different types, and may also return additional values like gradients, embeddings, or attention maps. Rather than enumerate all variations, LIT describes each model and dataset with an extensible system of semantic types.

For example, a dataset class for textual entailment (Dagan et al., 2006; Bowman et al., 2015) might have `spec()`, describing available fields:

[2] https://lit-element.polymer-project.org/. Naming is coincidental; the Language Interpretability Tool is not related to the lit-html or lit-element projects.

[3] https://mobx.js.org/

- **premise:** `TextSegment()`
- **hypothesis:** `TextSegment()`
- **label:** `MulticlassLabel(vocab=...)`

A model for the same task would have an `input_spec()` to describe required inputs:

- **premise:** `TextSegment()`
- **hypothesis:** `TextSegment()`

As well as an `output_spec()` to describe its predictions:

- **probas:** `MulticlassPreds(`
  `vocab=..., parent="label")`

Other LIT components can read this spec, and infer how to operate on the data. For example, the `MulticlassMetrics` component searches for `MulticlassPreds` fields (which contain probabilities), uses the `vocab` annotation to decode to string labels, and evaluates these against the input field described by `parent`. Frontend modules can detect these fields, and automatically display: for example, the embedding projector will appear if `Embeddings` are available.

New types can be easily defined: a `SpanLabels` class might represent the output of a named entity recognition model, and custom components can be added to interpret it.

5 Related Work

A number of tools exist for interactive analysis of trained ML models. Many are general-purpose, such as the What-If Tool (Wexler et al., 2020), Captum (Kokhlikyan et al., 2019), Manifold (Zhang et al., 2018), or InterpretML (Nori et al., 2019), while others focus on specific applications like fairness, including FairVis (Cabrera et al., 2019) and FairSight (Ahn and Lin, 2019). And some provide rich support for counterfactual analysis, either within-dataset (What-If Tool) or dynamically generated as in DiCE (Mothilal et al., 2020).

For NLP, a number of tools exist for specific model classes, such as RNNs (Strobelt et al., 2017), Transformers (Hoover et al., 2020; Vig and Belinkov, 2019), or text generation (Strobelt et al., 2018). More generally, AllenNLP Interpret (Wallace et al., 2019) introduces a modular framework for interpretability components, focused on single-datapoint explanations and integrated tightly with the AllenNLP (Gardner et al., 2017) framework.

While many components exist in other tools, LIT aims to integrate local explanations, aggregate analysis, and counterfactual generation into a single tool. In this, it is most similar to Errudite (Wu et al., 2019), which provides an integrated UI for NLP error analysis, including a custom DSL for text transformations and the ability to evaluate over a corpus. However, LIT is explicitly designed for flexibility: we support a broad range of workflows and provide a modular design for extension with new tasks, visualizations, and generation techniques.

Limitations LIT is an evaluation tool, and as such is not directly useful for training-time monitoring. As LIT is built to be interactive, it does not scale to large datasets as well as offline tools such as TFMA (Mewald, 2019). (Currently, the LIT UI can handle about 10,000 examples at once.) Because LIT is framework-agnostic, it does not have the deep model integration of tools such as AllenNLP Interpret (Wallace et al., 2019) or Captum (Kokhlikyan et al., 2019). This makes many things simpler and more portable, but also requires more code for techniques like integrated gradients (Sundararajan et al., 2017) that need to directly manipulate parts of the model.

6 Conclusion and Roadmap

LIT provides an integrated UI and a suite of components for visualizing and exploring the behavior of NLP models. It enables interactive analysis both at the single-datapoint level and over a whole dataset, with first-class support for counterfactual generation and evaluation. LIT supports a diverse range of workflows, from explaining individual predictions to disaggregated analysis to probing for bias through counterfactuals. LIT also supports a range of model types and techniques out of the box, and is designed for extensibility through simple, framework-agnostic APIs.

LIT is under active development by a small team. Planned upcoming additions include new counterfactual generation plug-ins, additional metrics and visualizations for sequence and structured output types, and a greater ability to customize the UI for different applications.

LIT is open-source under an Apache 2.0 license, and we welcome contributions from the community at `https://github.com/pair-code/lit`.

Acknowledgments

We thank Slav Petrov, Martin Wattenberg, Fernanda Viegas, Kellie Webster, Emily Pitler, Dipanjan Das, Leslie Lai, Kristen Olson, and other members of PAIR and the Language team at Google Research for many productive discussions during development. We also thank our anonymous reviewers for their helpful feedback, and Pere Lluis, Luke Gessler, and Kevin Robinson for their contributions to the open-source code.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Yongsu Ahn and Yu-Ru Lin. 2019. Fairsight: Visual analytics for fairness in decision making. *IEEE Transactions on Visualization and Computer Graphics*, page 1–1.

Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 459–468. IEEE.

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 597–604, Ann Arbor, Michigan. Association for Computational Linguistics.

Yonatan Belinkov, Sebastian Gehrmann, and Ellie Pavlick. 2020. Interpretability and analysis in neural NLP. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 1–5, Online. Association for Computational Linguistics.

John Blitzer and Fernando Pereira. 2007. Domain adaptation of natural language processing systems. *University of Pennsylvania*, pages 1–106.

Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. 2016. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in Neural Information Processing Systems 29*, pages 4349–4357.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. 2019. Fairvis: Visual analytics for discovering intersectional bias in machine learning. In *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 46–56. IEEE.

Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. AllenNLP: A deep semantic natural language processing platform.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28*, pages 1693–1701.

Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. 2020. exBERT: A visual analysis tool to explore learned representations in Transformer models. In *Proceedings of the 58th Annual Meeting of* *the Association for Computational Linguistics: System Demonstrations*, pages 187–196, Online. Association for Computational Linguistics.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: The 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 57–60, Stroudsburg, PA, USA. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark. Association for Computational Linguistics.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Jonathan Reynolds, Alexander Melnikov, Natalia Lunova, and Orion Reblitz-Richardson. 2019. Pytorch captum. `https://github.com/pytorch/captum`.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 681–691, San Diego, California. Association for Computational Linguistics.

Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors. 2019. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Florence, Italy.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Clemens Mewald. 2019. Introducing tensorflow model analysis: Scaleable, sliced, and full-pass metrics. *`https://blog.tensorflow.org/2018/03/introducing-tensorflow-model-analysis.html`*.

Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 607–617.

Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. 2019. InterpretML: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis only baselines in natural language inference. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 180–191, New Orleans, Louisiana. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *`https://blog.openai.com/better-language-models`*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

Marco Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California. Association for Computational Linguistics.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. 2018. Gender bias in coreference resolution. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 8–14, New Orleans, Louisiana. Association for Computational Linguistics.

Daniel Smilkov, Nikhil Thorat, Charles Nicholson, Emily Reif, Fernanda B Viégas, and Martin Wattenberg. 2016. Embedding projector: Interactive visualization and interpretation of embeddings. In *NIPS*

2016 Workshop on Interpretable Machine Learning in Complex Systems.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M Rush. 2018. Seq2seq-vis: A visual debugging tool for sequence-to-sequence models. *IEEE transactions on visualization and computer graphics*, 25(1):353–363.

Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. 2017. LSTMvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3319–3328. PMLR.

Jesse Vig and Yonatan Belinkov. 2019. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 63–76, Florence, Italy. Association for Computational Linguistics.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019. AllenNLP interpret: A framework for explaining predictions of NLP models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 7–12, Hong Kong, China. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

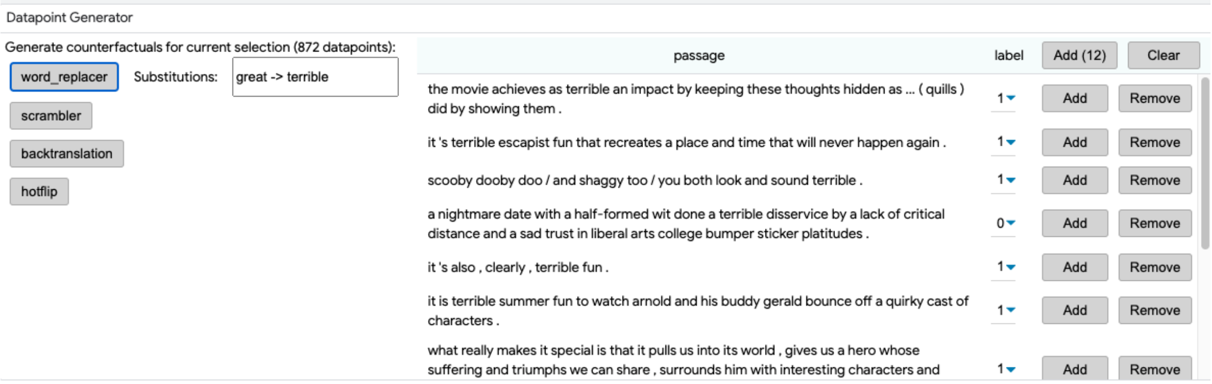
J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. 2020. The what-if tool: Interactive probing of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):56–65.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, Florence, Italy. Association for Computational Linguistics.

Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David Ebert. 2018. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, PP:1–1.

A Appendices



Datapoint Generator

Generate counterfactuals for current selection (872 datapoints):

word_replacer | Substitutions: great -> terrible

scrambler

backtranslation

hotflip

passage	label	Add (12)	Clear
the movie achieves as terrible an impact by keeping these thoughts hidden as ... (quills) did by showing them .	1	Add	Remove
it 's terrible escapist fun that recreates a place and time that will never happen again .	1	Add	Remove
scooby dooby doo / and shaggy too / you both look and sound terrible .	1	Add	Remove
a nightmare date with a half-formed wit done a terrible disservice by a lack of critical distance and a sad trust in liberal arts college bumper sticker platitudes .	0	Add	Remove
it 's also , clearly , terrible fun .	1	Add	Remove
it is terrible summer fun to watch arnold and his buddy gerald bounce off a quirky cast of characters .	1	Add	Remove
what really makes it special is that it pulls us into its world , gives us a hero whose suffering and triumphs we can share , surrounds him with interesting characters and	1	Add	Remove

Figure A.1: The counterfactual generator module, showing a set of generated datapoints in the staging area. The labels can be maually edited before adding these to the dataset. In this example, the counterfactuals were created using the word replacer, replacing the word "great" with "terrible" in each passage.

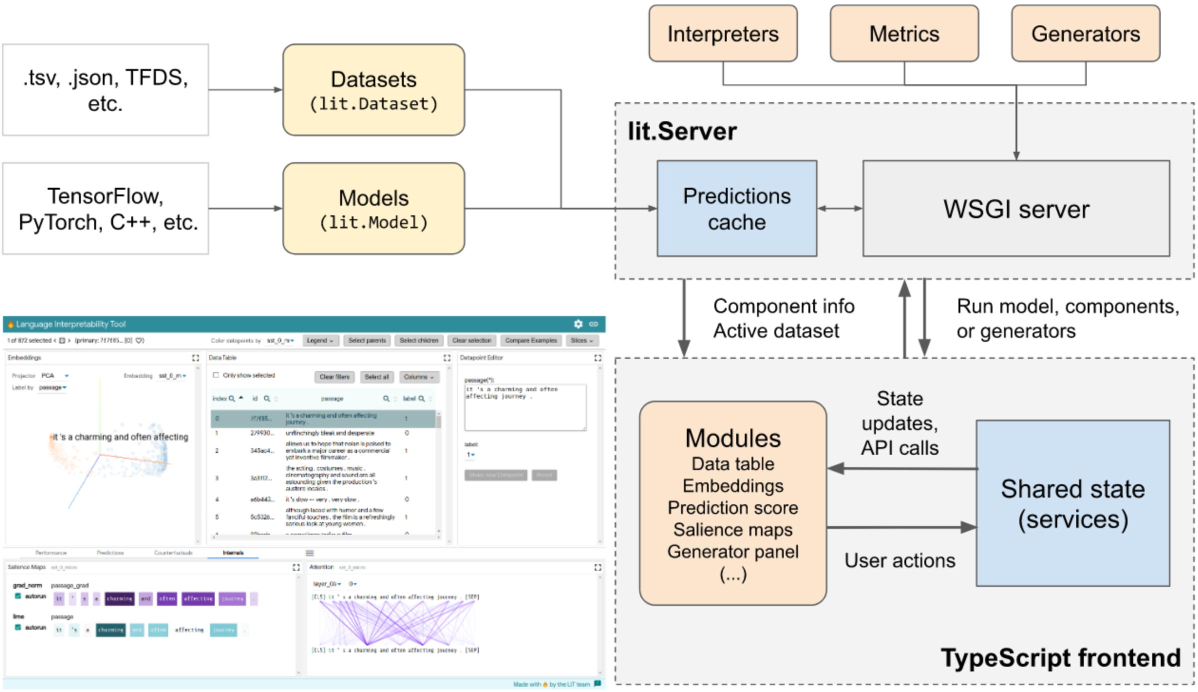
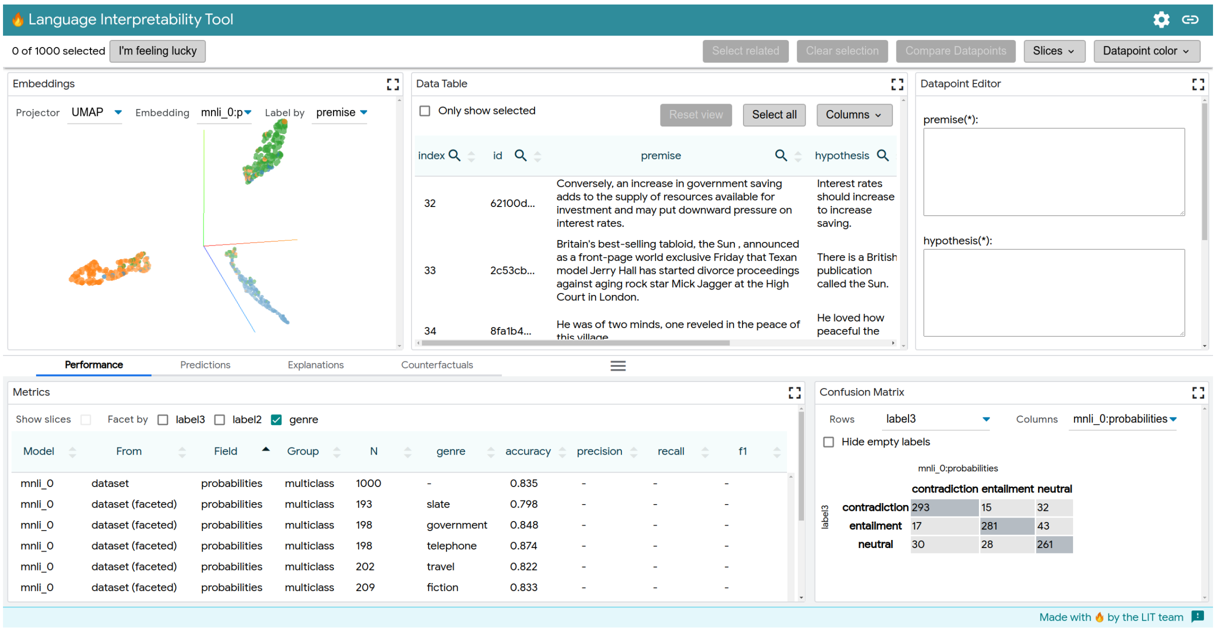


Figure A.2: Overview of LIT system architecture. The backend manages models, datasets, metrics, generators, and interpretation components, as well as a caching layer to speed up interactive use. The frontend is a TypeScript single-page app consisting of independent modules (webcomponents built with lit-element) which interact with shared "services" that manage interaction state. The backend can be extended by passing components to the `lit.Server` class in the demo script (Section 4.3 and Figure A.3), while the frontend can be extended by importing new components in a single file, `layout.ts`, which both lists available modules and specifies their position in the UI (Figure 1).

```
NLI_LABELS = ['entailment', 'neutral', 'contradiction']

class MultiNLIData(lit.Dataset):
  """Loader for MultiNLI dataset."""

  def __init__(self, path):
    # Read the eval set from a .tsv file
    df = pandas.read_csv(path, sep='\t')
    # Store as a list of dicts, conforming to self.spec()
    self._examples = [{
      'premise': row['sentence1'],
      'hypothesis': row['sentence2'],
      'label': row['gold_label'],
      'genre': row['genre'],
    } for _, row in df.iterrows()]

  def spec(self):
    return {
      'premise': lit_types.TextSegment(),
      'hypothesis': lit_types.TextSegment(),
      'label': lit_types.Label(vocab=NLI_LABELS),
      # We can include additional fields, which don't have to be used by the model.
      'genre': lit_types.Label(),
    }


class MyNLIModel(lit.Model):
  """Wrapper for a Natural Language Inference model."""

  def __init__(self, model_path, **kw):
    # Load the model into memory so we're ready for interactive use.
    self._model = _load_my_model(model_path, **kw)

  ##
  # LIT API implementations
  def predict(self, inputs: List[Input]) -> Iterable[Preds]:
    """Predict on a single minibatch of examples."""
    examples = [self._model.convert_dict_input(d) for d in inputs]  # any custom preprocessing
    return self._model.predict_examples(examples)  # returns a dict for each input

  def input_spec(self):
    """Describe the inputs to the model."""
    return {
        'premise': lit_types.TextSegment(),
        'hypothesis': lit_types.TextSegment(),
    }

  def output_spec(self):
    """Describe the model outputs."""
    return {
      # The 'parent' keyword tells LIT where to look for gold labels when computing metrics.
      'probas': lit_types.MulticlassPreds(vocab=NLI_LABELS, parent='label'),
      # This model returns two different embeddings, but you can easily add more.
      'output_embs': lit_types.Embeddings(),
      'mean_word_embs':  lit_types.Embeddings(),
      # In LIT, we treat tokens as another model output. There can be more than one,
      # and the 'align' field describes which input segment they correspond to.
      'premise_tokens': lit_types.Tokens(align='premise'),
      'hypothesis_tokens': lit_types.Tokens(align='hypothesis'),
      # Gradients are also returned by the model; 'align' here references a Tokens field.
      'premise_grad': lit_types.TokenGradients(align='premise_tokens'),
      'hypothesis_grad': lit_types.TokenGradients(align='hypothesis_tokens'),
      # Similarly, attention references a token field, but here we want the model's full "internal"
      # tokenization, which might be something like: [START] foo bar baz [SEP] spam eggs [END]
      'tokens': lit_types.Tokens(),
      'attention_layer0': lit_types.AttentionHeads(align=['tokens', 'tokens']),
      'attention_layer1': lit_types.AttentionHeads(align=['tokens', 'tokens']),
      'attention_layer2': lit_types.AttentionHeads(align=['tokens', 'tokens']),
      # ...and so on. Since the spec is just a dictionary of dataclasses, you can populate it
      # in a loop if you have many similar fields.
    }


def main(_):
  datasets = {
      'mnli_matched': MultiNLIData('/path/to/dev_matched.tsv'),
      'mnli_mismatched': MultiNLIData('/path/to/dev_mismatched.tsv'),
  }

  models = {
      'model_foo': MyNLIModel('/path/to/model/foo/files'),
      'model_bar': MyNLIModel('/path/to/model/bar/files'),
  }

  lit_demo = lit.Server(models, datasets, port=4321)
  lit_demo.serve()

if __name__ == '__main__':
  main()
```

Figure A.3: Example demo script to run LIT with two NLI models and the MultiNLI (Williams et al., 2018) development sets. The actual model can be implemented in TensorFlow, PyTorch, C++, a REST API, or anything that can be wrapped in a Python class: to work with LIT, users needs only to define the spec fields and implement a `predict()` function which returns a dict of NumPy arrays for each input datapoint. The dataset loader is even simpler; a complete implementation is given above to read from a TSV file, but libraries like TensorFlow Datasets can also be used.



Projector UMAP Embedding mnli_0:p Label by premise

□ Only show selected

index	id	premise	hypothesis
32	62100d...	Conversely, an increase in government saving adds to the supply of resources available for investment and may put downward pressure on interest rates.	Interest rates should increase to increase saving.
33	2c53cb...	Britain's best-selling tabloid, the Sun , announced as a front-page world exclusive Friday that Texan model Jerry Hall has started divorce proceedings against aging rock star Mick Jagger at the High Court in London.	There is a British publication called the Sun.
34	8fa1b4...	He was of two minds, one reveled in the peace of this village	He loved how peaceful the

premise(*):

hypothesis(*):

Show slices □ Facet by □ label3 □ label2 ☑ genre

Model	From	Field	Group	N	genre	accuracy	precision	recall	f1
mnli_0	dataset	probabilities	multiclass	1000	-	0.835	-	-	-
mnli_0	dataset (faceted)	probabilities	multiclass	193	slate	0.798	-	-	-
mnli_0	dataset (faceted)	probabilities	multiclass	198	government	0.848	-	-	-
mnli_0	dataset (faceted)	probabilities	multiclass	198	telephone	0.874	-	-	-
mnli_0	dataset (faceted)	probabilities	multiclass	202	travel	0.822	-	-	-
mnli_0	dataset (faceted)	probabilities	multiclass	209	fiction	0.833	-	-	-

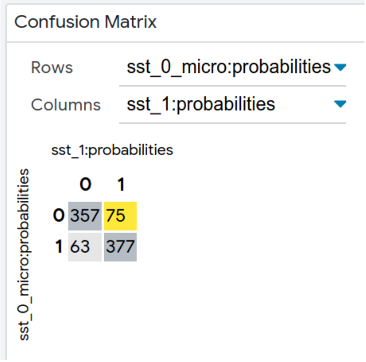
Rows label3 Columns mnli_0:probabilities

□ Hide empty labels

mnli_0:probabilities

label3	contradiction	entailment	neutral
contradiction	293	15	32
entailment	17	281	43
neutral	30	28	261

Figure A.4: Full UI screenshot, showing a BERT (Devlin et al., 2019) model on a sample from the "matched" split of the MultiNLI (Williams et al., 2018) development set. The embedding projector (top left) shows three clusters, corresponding to the output layer of the model, and colored by the true label. On the bottom, the metrics table shows accuracy scores faceted by genre, and a confusion matrix shows the model predictions against the gold labels.



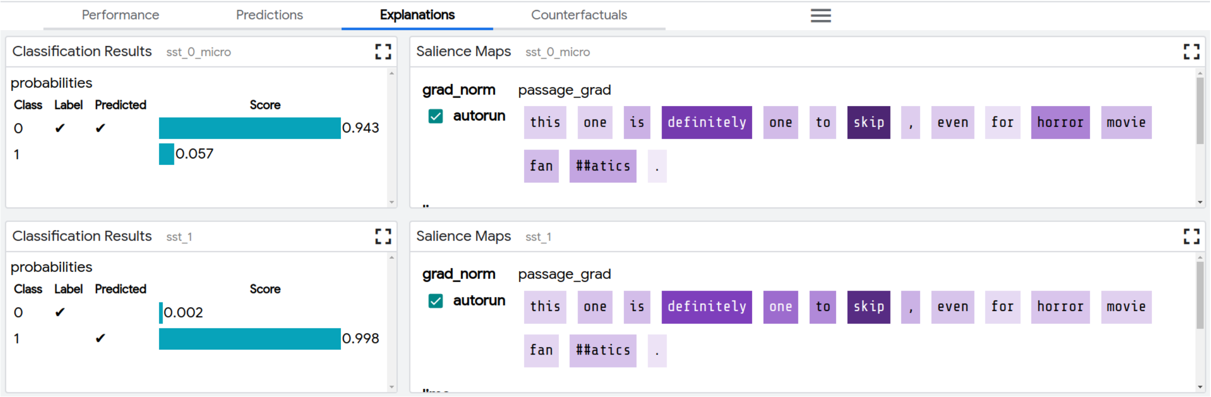
Rows sst_0_micro:probabilities

Columns sst_1:probabilities

sst_1:probabilities

sst_0_micro:probabilities	0	1
0	357	75
1	63	377

(a)


Performance
Predictions
Explanations
Counterfactuals
Classification Results sst_0_micro
Salience Maps sst_0_micro
Classification Results sst_1
Salience Maps sst_1

probabilities

Class	Label	Predicted	Score
0	✓	✓	0.943
1			0.057

grad_norm passage_grad

☑ autorun

this one is definitely one to skip , even for horror movie fan ##atics .

probabilities

Class	Label	Predicted	Score
0	✓		0.002
1		✓	0.998

grad_norm passage_grad

☑ autorun

this one is definitely one to skip , even for horror movie fan ##atics .

(b)

Figure A.5: Confusion matrix (a) and side-by-side comparison of predictions and salience maps (b) on two sentiment classifiers. In model comparison mode, the confusion matrix can compare two models, and clicking an off-diagonal cell with select examples where the two models make different predictions. In (b) we see one such example, where the model in the second row ("sst_1") predicts incorrectly, even though gradient-based salience show both models focusing on the same tokens.