IrEne-viz: Visualizing Energy Consumption of Transformer Models

Yash Kumar Lal, Qingqing Cao, Harsh Trivedi, Reetu Singh,
Aruna Balasubramanian, Niranjan Balasubramanian
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794, USA
{ylal,qicao,hjtrivedi,reesingh,arunab,niranjan}@cs.stonybrook.edu

Abstract

IrEne (Cao et al., 2021) is an energy prediction system that accurately predicts the interpretable inference energy consumption of a wide range of Transformer-based NLP models. We present the IrEne-viz tool, an online platform for visualizing and exploring energy consumption of various Transformer-based models easily. Additionally, we release a public API that can be used to access granular information about energy consumption of transformer models and their components. The live demo is available at http://stonybrooknlp.github.io/irene/demo/.

1 Introduction

Pretrained transformers have shown strong results on downstream NLP tasks, resulting in wide-spread adoption. With their deployment in large-scale public-facing systems serving hundreds of millions of requests per day, it has become important to study their energy footprint at inference time. Inference energy can incur substantial costs especially for models that are critical for high-volume web services.

Designing energy efficient and cost-effective models requires both accurate and interpretable energy modeling. Current approaches to energy modeling treat the model as a monolithic entity. In our previous work (Cao et al., 2021), we introduced a tree-like abstraction to decompose a model into its components. We designed a multi-level prediction method that predicts energy in all the components of the abstraction tree in a bottom-up fashion using resource utilization and model description features. This system called IrEne is used as the base of this work. IrEne provides more accurate energy prediction than other methods and is designed to be interpretable. However, it is non-trivial to retrieve data from that system, making it difficult to perform analysis or visualization for the same.

In this work, we present IrEne-viz, a user-friendly dashboard that allows visualization of inference energy consumption of a transformer-based model and its various components. Users will be able to interact with the different operations present in a model. Our interface allows people to easily understand the energy bottlenecks during inference. Additionally, we make our pipeline public by exposing it as an API endpoint. Having such data readily available will further research in the area and allow the community to use it for their own purposes, such as analyzing accuracy or latency trade-offs against energy. For instance, Cao et al. (2021) compared accuracy of BERT on a specific task while varying the number of layers and made observations about the energy-accuracy tradeoff. We design IrEne-viz to be:

- **Easy to use** - Our browser interface is intuitive and allows for thorough exploration of a model, its operations, and their energy usage.
- **Easy to access** - The model tree and its features are readily available through a public API in an easy-to-use JSON format.
- **Easy to extend** - New models to be tracked can be included easily.

2 Related Work

There has been increased interest in the energy consumption of NLP models in recent years. Despite some progress in modeling, there is a lack of visualisation and analysis tools for the same.

2.1 Energy Estimation

Schwartz et al. (2019) suggest using metrics like floating point operations (FPO) to measure energy efficiency. However, Henderson (2020) argues such

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 251–258
August 1–6, 2021.

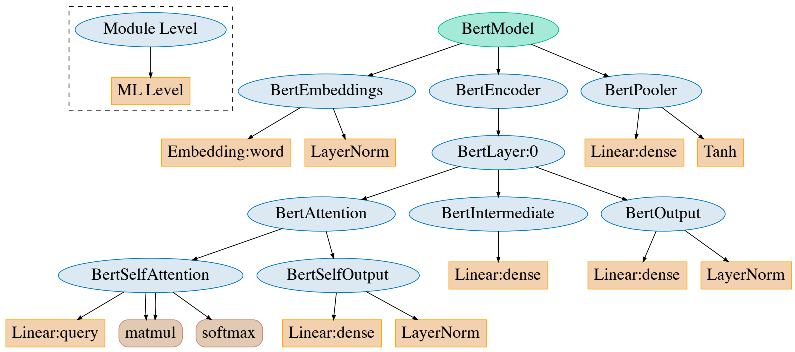


Figure 1: A tree view of a 1-layer BERT model. The yellow rectangle nodes stand for basic machine learning (ML) level operations. The brown rectangle nodes are also ML level which are non-parametric (i.e., has no trainable parameters). The ML level operations are model-agnostic and provided by machine learning software framework. The light blue oval nodes denote model-specific operations that reflect the architectural semantics given by the model developer .

metrics alone cannot accurately reflect energy consumption. Energy prediction of applications on mobile devices is a well-studied topic in the systems community (Pathak et al., 2011, 2012; Yoon et al., 2012; Cao et al., 2017) but they require fine-grained understanding of the application. None of these systems predict energy for NLP models.

Henderson (2020) use the *experiment-impact-tracker* software framework to report the aggregated energy of benchmark programs, built on Strubell et al. (2019). However, Cao et al. (2020) show that this type of resource utilization only modeling can be highly inaccurate. Zhou et al. (2020) presents an energy efficient benchmark for NLP models. However, they only report the time (hours) and cost (dollars) for training and testing NLP models, the actual energy numbers remain unknown.

2.2 Transformer Model Visualization

For NLP, a number of tools exist for investigating specific model classes, such as RNNs (Strobelt et al., 2018), 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. Lal et al. (2021) present a tool to visualize token embeddings through each layer of a Transformer and highlight distances between certain token embeddings. No such visualization work exists for energy consumption of NLP models.

3 IrEne - Prediction Engine

We briefly review the IrEne system which we use as the energy prediction engine. Please refer to (Cao et al., 2021) for more details. IrEne is an interpretable energy prediction system. It represents transformer models in a tree-based abstraction, and generates energy prediction for each node of the tree, thus directly supporting interpretability. IrEne also comes with data it was trained on – for each tree node, it has associated resource utilization and model-related features, and ground-truth energy measured with a hardware power monitor.

Tree Abstraction

IrEne uses a model tree abstraction that represents the model nodes in three-levels: math level, machine learning (ML) level and module level. Math level nodes are a finite set of mathematical operations (like addition, subtraction, matrix multiplication etc); they form model-agnostic ML level nodes (such as Linear, LayerNorm etc.), which further can be used to construct complex module level nodes. Module level nodes are groups of lower ML level node operations that reflect the logic units of the NLP algorithms defined by model authors. The model tree abstraction is such that each parent node captures computation of all of its children nodes. Figure 1 shows an example tree representation for a 1-layer BERT transformer. This abstraction makes energy calibration more interpretable by allowing us to understand and analyze how the components of a model contribute to its energy usage.

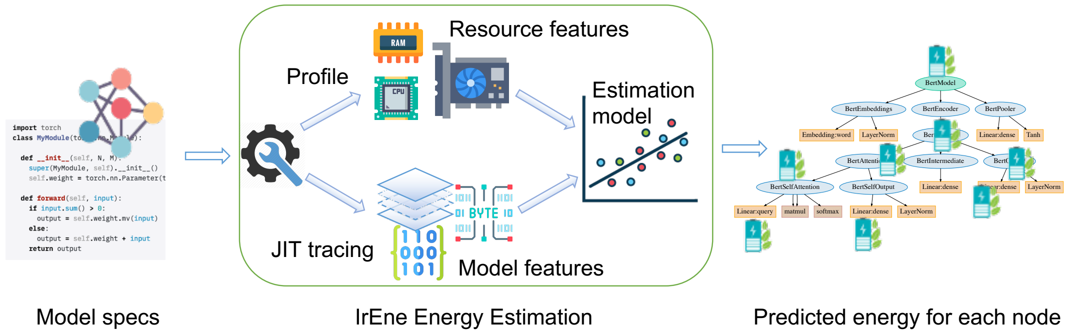


Figure 2: IrEne works by taking model specifications (for example, model code) as inputs and extracting a model tree representation using code instrumentation and run-time tracing. IrEne then runs the model once on a given hardware and feeds resource profiles combined with the model computation features into a regressor to predict the energy of the entire model tree representation. The root of the tree represents the energy of the entire NLP model and each child node represents the energy of different modules/ML operators that make up the model.

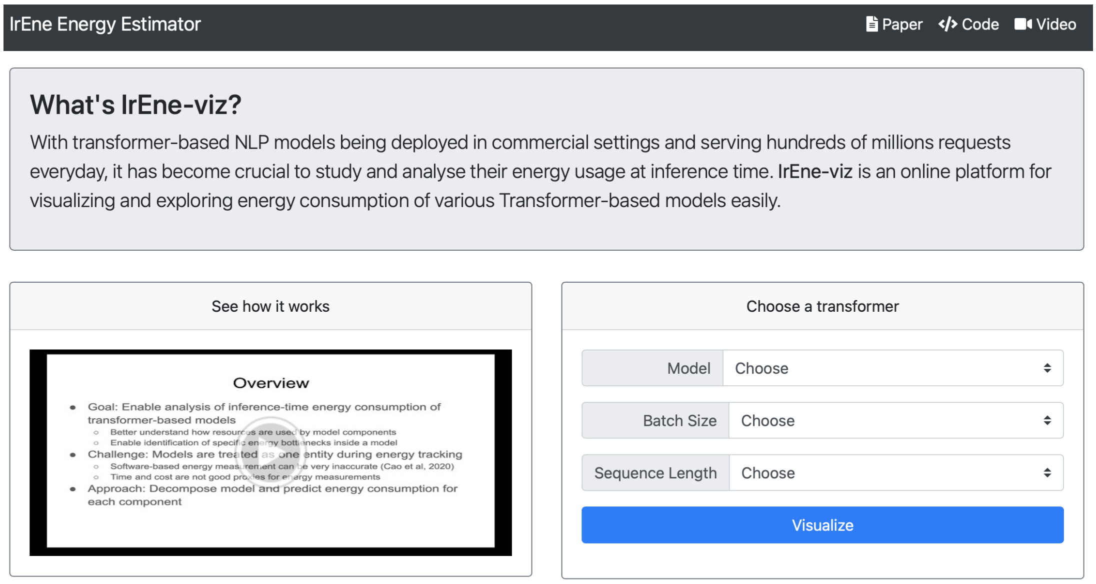


Figure 3: IrEne-viz has a simple input screen where a user can select which Transformer model they want to analyze, and specify the input sequence length and batch size for the model.

Resource Usage Collection

For a given transformer model, IrEne generates a tree representation in the aforementioned abstraction and populates each node with relevant features and ground-truth energy measurement.

To construct the tree, the transformer model[1] is run on the target hardware on randomly generated input for given batch size and input sequence length[2]. This provides execution graph and the JIT trace containing runtime information, which is combined as to form the final tree representation.

Irene uses resource utilization and model-based features. Resource features capture how the models use hardware resources and cause energy activities. Model features like input size and number of parameters are obtained from PyTorch model directly. A list of features, as described in Cao et al. (2021), is shown in Table 1.

Irene collects ground-truth energy for each node using a highly accurate power monitor, and runs it several times to get a reliable estimate. One can use the power monitor to measure energy directly at runtime for visualization. However, this is cumbersome and requires physical access to the device which is not always feasible with cloud-based deployments.

[1]We used HuggingFace Transformers library v4.2.2

[2]The batch size and input sequence length together decide the amount of input data to the model, therefore, they both affect the model energy consumption.

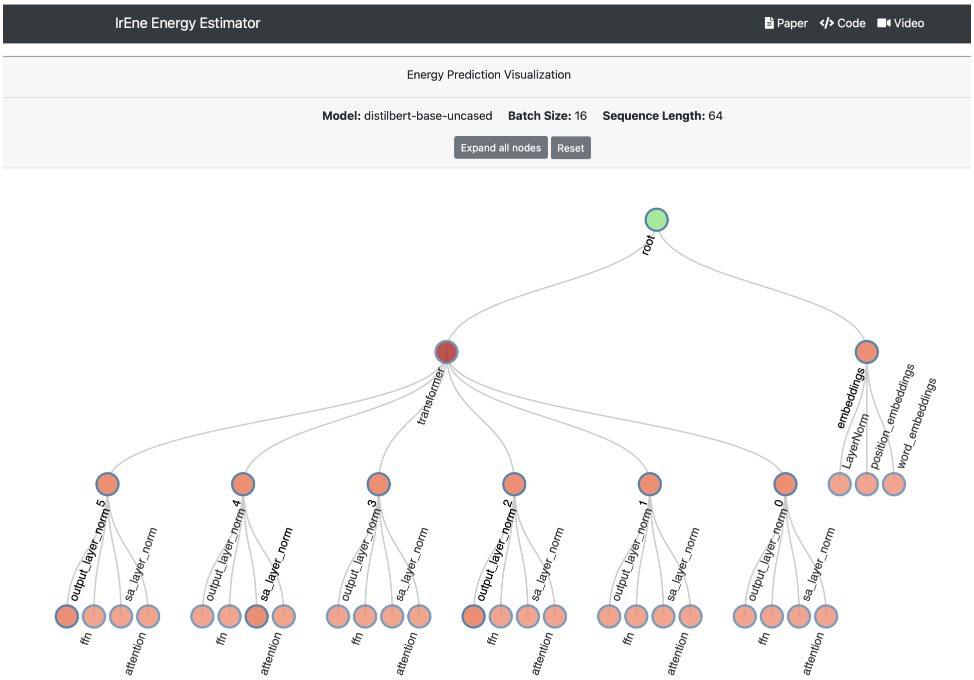


Figure 4: The user will be able to see an interactive visualization of the model components in a tree format. They will be able to expand and collapse it as per their need for granularity in energy analysis. Additionally, to the right, a list of model operations, in order of energy consumption, is provided for easy browsing.

Training and Prediction

IrEne predicts the energy for every node in the model tree in a bottom-up fashion. At the leaves, where the nodes correspond to the ML primitives, IrEne uses separate regression models for each type of ML primitive (e.g., one regressor for Linear Layer, another for LayerNorm etc.). For the intermediate nodes, their energy is predicted recursively using a single regressor that makes a weighted combination of the predicted energy values from its children, and mean squared loss between predicted and ground-truth energy for all tree nodes is jointly minimized. For both types of regressors, IrEne uses features that are derived from resource utilization (e.g. cpu utilization) and generalized node features (e.g. size of inputs) enabling accurate multi-level energy prediction. Using the model tree abstraction and multi-level prediction model makes IrEne generalizable, in the sense that once trained, it can work on unseen NLP models with similar components.

4 User Interface and Functionality

The goal of IrEne-viz is to provide an easy way for users to analyze the energy of a given Transformer model (for a specified input size). To do so, we design a browser-based user interface (UI) in IrEne-viz that controls the input size and selects the model, as shown in Figure 3. We then estimate the energy consumption of the model and visualize the energy for each part in the Transformer model. Specifically, an user selects a predefined Transformer model[3] via the dropdown menu and enters the batch size and input sequence length. After pressing the visualize button, IrEne-viz backend server will run the energy estimation and send the energy result back to the browser for visualization.

batch_size : batch size
seq_len : # of input tokens
flops : floating point operations (unit: million)
mem_bytes : memory read and write (unit: MiB)
cpu_util : CPU utilization (unit: %)
mem_usg : memory usage (unit: %)
gpu_util : GPU processor utilization (unit: %)
gm_usg : GPU memory usage (unit: %)
g_clk : GPU processor clock speed (unit: MHz)
gm_clk : GPU memory clock speed (unit: MHz)
latency : inference latency (unit: s)
gpu_energy : GPU driver energy (unit: joule)

Table 1: Features used for energy estimation in IrEne.

[3]We are adding functionality to support customized models

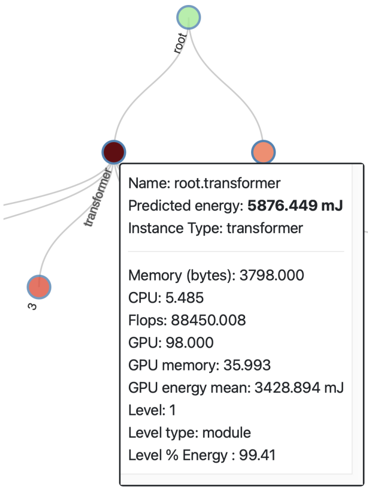
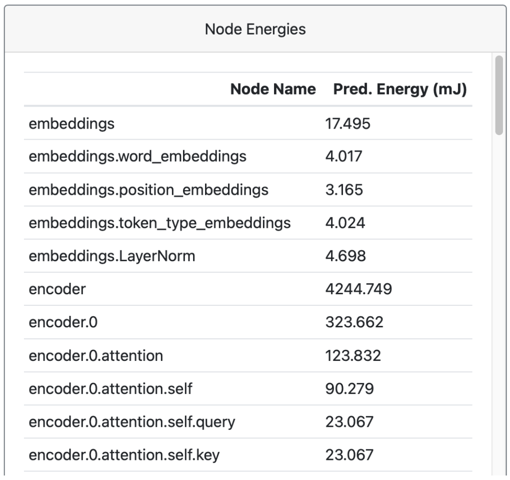


Figure 5: Hovering over any node provides the user with additional information about that node. This includes measurements of memory usage, flops and CPU cycles. Users can select models optimal for their hardware requirements.


Node Energies

Node Name	Pred. Energy (mJ)
embeddings	17.495
embeddings.word_embeddings	4.017
embeddings.position_embeddings	3.165
embeddings.token_type_embeddings	4.024
embeddings.LayerNorm	4.698
encoder	4244.749
encoder.0	323.662
encoder.0.attention	123.832
encoder.0.attention.self	90.279
encoder.0.attention.self.query	23.067
encoder.0.attention.self.key	23.067

Figure 6: The dashboard also provides a list of all model operations along with their predicted energy consumption for easy identification of bottlenecks.

In IrEne-viz, we support two core functionalities:
Functionality 1 - Explore the energy consumption of the model. Besides the entire model energy, users can interactively explore the energy consumed by any block inside the model, as shown in Figure 4. Additionally, we support inspecting the resource and model features used to estimate the energy, as described in Figure 5.

Functionality 2 - Find energy bottlenecks. At each level of the model, users can easily identify operations that can be improved (or pruned) in terms of their relative energy usage. The visualization dashboard also displays a list of model operations along with their predicted energy usage, as presented in Figure 6.

5 System Implementation

To make IrEne-viz modular and extensible, we design an energy analysis pipeline consisting of three components: a visualization panel that accepts user requests and presents energy results, a prediction engine (IrEne) that predicts energy consumption and a backend server that encapsulates IrEne and serves information through an API endpoint. The API and the prediction engine can be used as individual entities as well. They are also designed to be extensible, so adding new features is easy. The visualization panel is intuitive and informative, allowing easy exploration of data.

Figure 7 shows the full pipeline used for this application. The visualization panel queries the API with the user-desired model name, input sequence length and batch size. This information is passed on to the prediction engine. The engine performs resource collection for the corresponding model specifications and predicts the energy usage of each component. The API sends the visualization panel a full tree representation of the model containing all the model information.

5.1 Visualization Panel

The browser-based UI is built up of HTML webpages using a bootstrap template. The visualization widget is developed using D3.js (Bostock, 2012) embedded in a Flask (Grinberg, 2018) application. A user can decide which model they want to analyze, and provide desired values for batch size and input sequence length. Upon selection, a full tree with information about the model is presented. We also provide an option to display the entire tree at once and, since there are lot of components in a model, collapse it into one root component for easier analysis. Users are able to interact with different components to explore every component in the model. They can click on a component to expand and show all the components in that subtree. When the cursor hovers over it, all the resource information about that component is shown to the user. At any level, the color of the component indicates the percentage of energy consumption it is responsible for. Additionally, we present a list of model components with their predicted energy use on one part of the screen. This frontend applica-

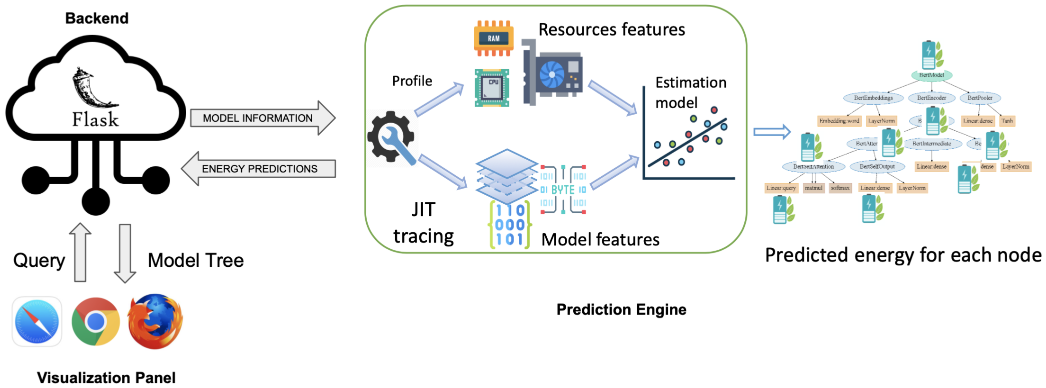


Figure 7: Full system pipeline. The **visualization panel** queries the **backend** with the model name, input sequence length and batch size. This information is passed on to the **prediction engine**, which performs resource collection and predicts the energy usage of each component. The **prediction engine** generates full tree with all the model information and prediction energy back to the **backend**, which in turn passes it to **visualization panel** .

tion is deployed on Heroku and will be available publicly soon.

5.2 Backend

First, we download the configuration of the specified model from Huggingface Hub (Wolf et al., 2020) and use it convert it into a tree object. A model is composed of multiple module-level components, and a module-level component itself is made up of other module-level or ML-level components. Each parent component encapsulates the computation of all of its child components.

First, we run the model to extract the model tree structure. A profiler process is started in the background to monitor usage of various resources. For each type of abstraction described, we find every component in the model.[4] It is run with dummy inputs of the required input size for a fixed number of times so that the profiler can log energy usage reliably (low standard deviation in energy measurements). We reconcile resource usage logs with their respective components using the timestamp at which they were run. Next, we annotate the model tree objects with these features.

To generate energy predictions, we use the Cao et al. (2021) model. We load the saved weights, use the features we just collected to perform inference. The same model tree object is populated with the predicted energy numbers, and can now be used for visualization. The backend encapsulates the prediction engine, which is deployed as a Flask API hosted on a GPU desktop using nginx.

For currently supported models, it takes 15-25 minutes to gather resource usage and make predictions. So, to speed up visualization, we cache results for these models and serve them to the user.

We expose the full end-to-end-pipeline as a Flask API endpoint, and make it available for public use. Querying it for model energy usage information only requires a simple GET request to be made. In addition to this, we plan to expose the model tree abstraction as another API endpoint so that the community can use it for other purposes like runtime analysis.

6 Conclusion and Roadmap

IrEne-viz provides an integrated UI and components for visualizing and exploring the energy consumption of various Transformer models. It is under active development and is being constantly refined for release. We are adding support for live models immediately. For new models, users will be sent an email with a custom link to their requested visualization. As the community uses it, we will cache resource usage and predictions for more intermediate nodes found in various transformer-based models. This optimization will gradually result in lower times for newer models.

Our end-to-end pipeline, served as an API, can be used to build an energy leaderboard. This platform can be extended to compare the energy of architectural modifications (e.g. activation or normalization function) of different models for given input. By extending this work to other harware, we aim to

[4]For the profiler to collect correct energy statistics, we make sure no other significant process is running on the same machine.

provide energy optimization suggestions based on energy profiles of a model on the given hardware. In our previous work, (Cao et al., 2021) we also studied accuracy vs energy trade-offs, which will be integrated into the dashboard.

7 Acknowledgements

This work was supported in part by the National Science Foundation under grants IIS-1815358.

References

Mike Bostock. 2012. D3.js - data-driven documents.

Qingqing Cao, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. Towards accurate and reliable energy measurement of NLP models. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 141–148, Online. Association for Computational Linguistics.

Qingqing Cao, Yash Kumar Lal, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2021. IrEne: Interpretable energy prediction for transformers. In *Association for Computational Linguistics: ACL 2021*, Online. Association for Computational Linguistics.

Yi Cao, Javad Nejati, Muhammad Wajahat, Aruna Balasubramanian, and Anshul Gandhi. 2017. Deconstructing the Energy Consumption of the Mobile Page Load. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):6:1–6:25.

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.

Miguel Grinberg. 2018. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.".

James Henderson. 2020. The unstoppable rise of computational linguistics in deep learning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6294–6306, Online. Association for Computational Linguistics.

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.

Vasudev Lal, Arden Ma, Estelle Aflalo, Phillip Howard, Ana Simoes, Daniel Korat, Oren Pereg, Gadi Singer, and Moshe Wasserblat. 2021. InterpreT: An interactive visualization tool for interpreting transformers. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 135–142, Online. Association for Computational Linguistics.

Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. 2012. Where is the energy spent inside my app? fine grained energy accounting on smartphones with Eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA. Association for Computing Machinery.

Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 2011. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 153–168, New York, NY, USA. Association for Computing Machinery.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. *CoRR*, abs/1907.10597.

H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush. 2018. Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. *ArXiv e-prints*.

Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M. Rush. 2018. 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.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

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 *Empirical Methods in Natural Language Processing*.

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.

Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. 2012. AppScope: application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, page 36, USA. USENIX Association.

Xiyou Zhou, Zhiyu Chen, Xiaoyong Jin, and William Yang Wang. 2020. HULK: an energy efficiency benchmark platform for responsible natural language processing. *CoRR*, abs/2002.05829.