IrEne: Interpretable Energy Prediction for Transformers

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

Abstract

Existing software-based energy measurements of NLP models are not accurate because they do not consider the complex interactions between energy consumption and model execution. We present IrEne, an interpretable and extensible energy prediction system that accurately predicts the inference energy consumption of a wide range of Transformer-based NLP models. IrEne constructs a model tree graph that breaks down the NLP model into modules that are further broken down into low-level machine learning (ML) primitives. IrEne predicts the inference energy consumption of the ML primitives as a function of generalizable features and fine-grained runtime resource usage. IrEne then aggregates these low-level predictions recursively to predict the energy of each module and finally of the entire model. Experiments across multiple Transformer models show IrEne predicts inference energy consumption of transformer models with an error of under 7% compared to the ground truth. In contrast, existing energy models see an error of over 50%. We also show how IrEne can be used to conduct energy bottleneck analysis and to easily evaluate the energy impact of different architectural choices. We release the code and data at https://github.com/StonyBrookNLP/irene.

1 Introduction

Accurately measuring the energy consumption of NLP models is becoming ever more important. Models are growing exponentially, with billions, even approaching trillions, of parameters with correspondingly large resource consumption (e.g. GPT-3 (Brown et al., 2020) has 175 billion parameters and Switch Transformers can have 1.6 trillion parameters (Fedus et al., 2021)). Recent works have sought to estimate energy consumption and suggest ways to reduce the resulting costs and carbon impacts (Strubell et al., 2019; Schwartz et al., 2019; Henderson et al., 2020; Anthony et al., 2020)

Unfortunately, there are no easy-to-use and accurate solutions for measuring or predicting the energy consumption. On the one hand, measuring energy consumption directly through hardware power monitors is not feasible as it requires exclusive access to the hardware and detailed instrumentation. On the other hand, there are software models that predict energy as a function of resource utilization (Strubell et al., 2019; Henderson et al., 2020) but these energy prediction models are inaccurate (Cao et al., 2020). The inaccuracy stems from the prediction models not accounting for the complex interactions between energy consumption and resource utilization.

In this work, we focus on inference energy which can incur substantial costs especially for models that support high-volume web services. We ask how we can build an energy prediction method that is accurate, interpretable, and extensible. We make three contributions in answering this question.

First, we frame the problem of interpretable energy prediction over a *model tree* abstraction. This abstraction represents the model as the root node that is composed from model-specific modules, which themselves are recursively composed from lower-level machine learning (ML) primitives, ones that are not model-specific. Given a model, the energy prediction problem is framed as the task of predicting the energy of all the nodes in its model tree abstraction. The result is that IrEne can predict not only the inference energy consumption of the entire model, but also of its components, making the energy prediction highly interpretable.

Second, we develop IrEne, that includes a multi-level prediction method that predicts energy in all nodes of the abstraction tree in a bottom-up fashion using resource utilization and model description features. For each of the leaf-nodes that are re-used

in different models, the ML primitives, IrEne uses a separate regressor trained on ground-truth energy measurements. One simple way to get energy for all other higher-level nodes is to recursively sum-up the values. While this works reasonably well (even better than a prior prediction model), direct summing of the raw predictions is sub-optimal because the error can propagate through the model tree thus making upper-level nodes estimation more erroneous. Instead, we learn a single regressor for all intermediate nodes, one that essentially adjusts the sum of children's predicted energy values based on features of the children. Since IrEne is built on top of energy predictions of ML primitives that are not model specific, it is generalizable and can be used to predict the energy for previously unseen (Transformer-based) models.

Third, to evaluate IrEne, we create an evaluation dataset with ground-truth energy measurements for multiple Transformer-based models at all levels in the model tree abstraction. Evaluations show that IrEne is more accurate – with an average model-level energy error of $5 \sim 7\%$ compared against the ground-truth, while existing software-based method (Strubell et al., 2019) has over 55% error. The module-level energy errors are also substantially small showing that IrEne is both accurate and interpretable. Last, we also conduct multiple analyses that show the utility of IrEne for interpretable energy predictions.

2 Related work

Over the last couple of years, there has been increased interest in the energy consumption of NLP models, starting with the work by Strubell et al. (Strubell et al., 2019). This work, and a follow up software framework called *experiment-impact-tracker* (Henderson et al., 2020) tracks the resource (i.e., CPU, GPU, memory) utilization of an NLP model and predicts energy consumption as a function of resources. However, our previous study shows that this type of resource utilization only modeling can be highly inaccurate (Cao et al., 2020). This is in part due to the complex relationship between resource utilization and energy consumption. Further, there are other activities that are not accounted via resource utilization such as data movement in GPU memory which can also cause significant energy footprint (Chen et al., 2016; Boroumand et al., 2018).

Other works (Zhou et al., 2020; Schwartz et al., 2019) report the energy numbers through alternate metrics including dollar cost or in terms of floating point operations. However, these do not directly map to the 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 these work require fine-grained understanding of the application. None of the existing systems predict energy for NLP applications.

3 Interpretable Energy Prediction

In this section we first state our design goals, motivate the abstraction, and problem formulation for interpretable energy prediction.

3.1 Design Goals

We design the energy prediction model with three design goals: (i) *accurate* prediction while incurring low profiling overheads; high overheads when measuring runtime resource utilization can hide the true energy costs of the NLP model, (ii) *provide interpretable energy analysis* of the components inside the NLP model, especially for analyzing energy bottlenecks; (iii) *extensible and generalizable*, in the sense that, they are trained once but can work on unseen NLP models to remain useful as new models emerge.

3.2 Model Tree Abstraction

To achieve the above goals, we first need a representation of the NLP model that is at a suitable abstraction both from interpretability and generalization standpoints.

On the one hand, using only low-level abstractions such as the math operations can help with easy generalization to new models as their units are basic math (or other compute) operations that are building blocks of any model. However, they lack interpretability since they don't directly convey the model architecture semantics. For example, a BERT (Devlin et al., 2019) model has matrix multiplications in both the self-attention and feed forward layers. Only having the energy of each matrix multiplication alone, without knowing which higher level logic units (i.e., either self-attention or feed forward layer) they belong to, does not help analyze if they are the bottlenecks for that particular unit. On the other hand, high-level abstractions preserve the architecture semantics and are interpretable for practitioners, but they don't

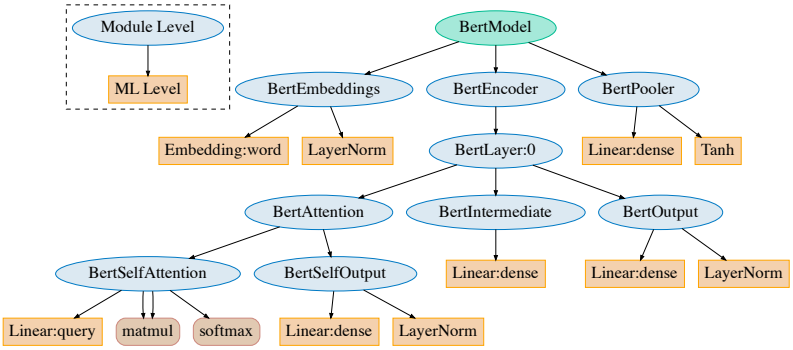

Figure 1: A tree view of a 1-layer BERT model. The yellow rectangle nodes stand for basic machine learning (ML) level operations. The brown rectangle nodes are also ML level which are non-parametric (i.e., has no trainable parameters). The ML level operations are model-agnostic and provided by machine learning software framework. The light blue oval nodes denote model-specific operations that reflect the architectural semantics given by the model developer, for example BertSelfAttention was designed to transform input sequence representations by 'attending" (weighted combination) to each position of the input sequence.

easily generalize to unseen models that may not have the same modules used for training.

Instead, we use 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 of a one-layer BERT (Devlin et al., 2019) model (omitted math level nodes). The execution of the model tree nodes can be in parallel, but current systems have a fixed sequential order for executing the sibling nodes. In this work, we only focus on sequential execution. Note that the model tree doesn't capture the order of execution. E.g., `BertOutput` appears right after `BertIntermediate` in BERT's computation graph, but here they'll be represented as siblings of the same parent `BertLayer:0`, and their energy will be treated separately. The parent node `BertLayer:0` encapsulates the energy and computation of its children node `BertIntermediate`, `BertOutput`, and `BertAttention`, in no particular order.

3.3 Problem Definition

With this new model tree abstraction, we formally state the problem of interpretable energy estimation of a NLP model. Given a model tree abstraction of a NLP model $\mathcal{M}$ consisting of a set of nodes $\mathcal{N} = \{n | n_{ml} \cup n_{mod}\}$ (n_{ml} is the set of ML level nodes, n_{mod} is the set of module level nodes), for an input size $\mathcal{I}$ (a pair of batch size b and sequence length s) [1], we can predict the energy E_n for every node n in the model tree. The energy of root node is the energy for the entire model.

4 Interpretable Prediction with IrEne

Figure 2 shows the IrEne architecture. IrEne takes the user-specified model and builds an energy predictor for a target hardware device. The model is run once on the target hardware and the run-time resource utilization is logged. During this run, IrEne uses code instrumentation and just-in-time (JIT) run-time tracing to break down the model into sub-components, and extracts a model tree representation (see details in §A).

IrEne then provides *interpretable energy analysis* by predicting 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. For both types of regressors, they use features that are derived from resource utilization (e.g. cpu utilization) and generalized node features

[1] 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.

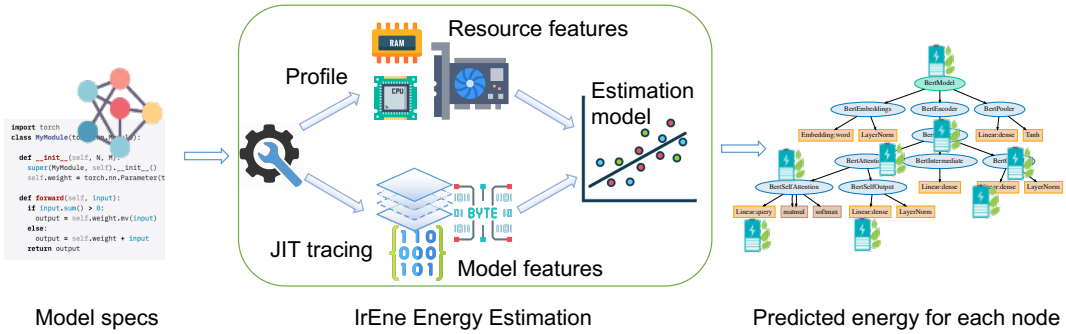

Figure 2: IrEne works by taking model specifications (for example, model code) as inputs and extracting a model tree representation using code instrumentation and run-time tracing. IrEne then runs the model once on a given hardware and feeds resource profiles combined with the model computation features 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.

(e.g. size of inputs) enabling accurate multi-level energy prediction.

IrEne represents higher-level modules via *generalizable features* and the ML primitives. Even if the intermediate modules are model-specific (e.g. BertSelfAttention), the features are general, allowing IrEne to predict energy of unseen models.

The IrEne model is trained using ground-truth energy measurements of ML primitives and a handful of NLP models; we use a highly accurate hardware power monitor to measure ground truth energy (§A). Of course, one can use the power monitor to measure energy directly at runtime. However, this is cumbersome and requires physical access to the device which is not always feasible with cloud-based deployments. Further, the hardware meter only measures the total energy, which is not interpretable in terms of its components.

4.1 Multilevel energy prediction

At the leaf-level, the energy prediction problem requires predicting the energy of ML primitives. As an offline step, IrEne first enumerates all relevant ML primitives and builds a specialized regressor for each primitive by training over ground truth data. In some cases, model developers can define their own ML primitives. We extract information about such custom primitives from the JIT trace.

Formally, for a leaf node n with ML primitive i, we predict the energy of the node as:

$$P_e^{ML_i}(n) = \mathbf{W}_i * \text{feat}(n) + b_i \qquad (1)$$

using primitive specific parameters $\mathbf{W}_i$ the weight vector and b_i the bias. We learn these parameters using a mean squared error loss between predicted $P_e(n)$ and ground-truth energy $G_e(n)$.

Our hierarchical tree representation gives a naturally interpretable way of propagating this prediction through the tree. Since each node represents total computation of its children nodes, the total energy from children nodes should also roughly correspond to that of the parent node. Formally,

$$\begin{aligned} P_e(n) &= \sum_{c \in child(n)} P_e(c) \quad \text{if n is non-leaf} \\ &= P_e^{ML_i}(n) \qquad \text{if n is leaf} \end{aligned} \qquad (2)$$

We call this baseline prediction model **PredictedSum**. This model is interpretable but naively summing up the energy values accumulates error going up the tree and results in noisy module-level predictions. To account for this, we use a *weighted* sum of child node energy, where the weights are learnt using node features. Formally,

$$\begin{aligned} P_e(n) &= \sum_{c \in child(n)} \alpha(c) * P_e(c) \quad \text{if n is non-leaf} \\ &= P_e^{ML_i}(n) \qquad \text{if n is leaf} \\ \alpha(c) &= 1 + \tanh(\mathbf{W} * \text{feat}(c) + b)/\tau \end{aligned} \qquad (3)$$

where $\mathbf{W}$ and b are parameters and τ is a hyperparameter. Unlike ML primitives, here we have a single regressor with one set of weight vector ($\mathbf{W}$) and bias scalar (b) parameters across all non-leaf nodes of any type. Note that this single regressor doesn't predict node's energy directly, but determines how much the predicted energy from its child node should be scaled before summing the children node energy. It does this recursively starting from the root, and hence encodes tree structure in its computation. We do not learn node-specific regressors because that does not allow generalizing to new models that may have different modules

than the ones during training.

Since the method is essentially calibrating the sum of the energy values, regularizing the model so that the computed weights on the energy values to be around 1 helps the learning. We do this by equation 3, which makes the range of computed weights, $\alpha(c)$ to be within $1 \pm \tau$. To supervise this model, we use the ground-truth energy from all the non-leaf nodes, and we train it in an end-to-end fashion. Formally,

$$loss(n) = \sum_{s \in subtree(n)} \frac{\left(P_e(s) - G_e(s)\right)^2}{G_e(s)^2} \quad (4)$$

We scale the mean squared error with ground-truth energy, since scales of energy at different levels of the tree are vastly different. We refer to this model as the **End2End** regressor, since the error signal in energy prediction of any node back-propagates through the whole subtree. We use this training scheme in IrEne. In our evaluation (section 5), we perform an ablation study to show why the tree structure and the end-to-end regressor is crucial for accuracy.

4.2 Featurization

We design two categories of energy-relevant features in IrEne : (i) the model features that reflect hardware-independent compute and memory information, and (ii) the resource features that capture how the models use hardware resources and cause energy activities. Table 1 shows the features used in IrEne. For the model description related information, we use features that characterize the compute, memory, and size of input etc. These are features that are independent of the underlying hardware. For resource features, we use utilization, usage and clock speed of hardware components including CPU, memory and GPU. Note that these two sets of features are extensible, meaning that one can add more either hardware-specific features or new model features. See Appendix sections A.2 and A.3 for details on how we obtain these features.

5 IrEne Evaluation

Our evaluation is aimed at measuring the accuracy of IrEne relative to ground truth and the state-of-the-art. We show the IrEne only causes 5-7% error for the model energy prediction. We also show that for a given Transformer model, IrEne can be used to find the energy bottlenecks and analyze the energy versus task performance trade-offs.

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.

Specification	**PC1**	**PC2**
CPU	Intel i9-7900X	Intel i7-6800K
Memory	32 GiB	32 GiB
GPU	2× GTX 1080 Ti	2× GTX 1070
GPU Memory	11.2 GiB per GPU	8 GiB per GPU
Storage	1 TiB SSD	1 TiB SSD

Table 2: Target hardware specifications.

5.1 Setup

Target Hardware: we use 2 GPU-equipped desktop PCs as the target hardware for running our models. See Table 2 for details.

Software and models: We perform inference in Transformer models using PyTorch (Paszke et al., 2019) v1.7 through the HuggingFace Transformers (Wolf et al., 2020) library. The six models we study are — BERT-base (Devlin et al., 2019), RoBERTa-base (Liu et al., 2019), DistillBERT (Sanh et al., 2020), DistilGPT2 (Sanh et al., 2020; Radford et al., 2019), OpenAI GPT (Radford et al., 2018) and GPT2 (Radford et al., 2019).

Software-based Measurement Baseline: For comparisons, we use the software-based energy measurements provided by the *experiment-impact-tracker* (Henderson et al., 2020) which estimates energy as a function of the GPU, CPU, and memory utilization. The method computes energy by aggregating resource usage as follows: $e_{total} = PUE \sum_p (p_{dram} e_{dram} + p_{cpu} e_{cpu} + p_{gpu} e_{gpu})$, where $p_{resource}$ [2] are the percentages of each system resource used by the attributable processes relative to the total in-use resources and $e_{resource}$ is the energy usage of that resource. The constant

[2] $resources$ can be $dram$, cpu, gpu

for power usage effectiveness (PUE) compensates for extra energy used to cool or heat data centers.

5.2 Dataset and Evaluation Methodology

For each model, we obtain the model tree and for each node in it, we associate ground-truth energy measurements using the power monitor and its resource features using low-overhead logging (Section A). For each node we run it repetitively for 20 seconds, since it often takes a very short time for one run (e.g. from 0.1 to 100 millisecond). We repeat this process for five rounds (the variations are within <1%) and record the average energy as the ground-truth for the node. We use 1 GPU to run all experiments. We record the start and end timestamp of the model program, and extract the energy values by comparing and aligning the timestamps from the resource profiler logs and power monitor logs.

Ground Truth Energy: We measure ground truth energy using a emonPi power monitor (Hudson, 2021) which is open source. The emonPi uses a clip-on CT sensor to monitor the energy consumed by the computer which records the passthrough current and voltage every 170 ms. This allows us to accurately measure the power draw at a sub second granularity. We obtain current, voltage, and timestamp values from the power meter's built-in serial port. The energy (e) consumed during a time period is then calculated using the sampled current (I_t) and voltage (V_t) values in that period: $e = \sum_t V_t I_t$.

To guarantee the consistency and reliability of the hardware energy measurements, we cool down the PCs after each experiment finishes to avoid potential overheating issue that can cause subsequent energy distortions. We measure the standby power consumption (when the CPU load is $< 0.1\%$) and ensure before running the experiments that the PC does not draw more than the standby power. Further, no other application is running during our experiments.

To understand the scale of energy usage, Table 3 shows the estimated energy consumption (in kWh) using our ground truth measurement. We also show the cost of answering one million queries (in USD) when using a BERT-base model in a reading comprehension (over one passage), and in an end-to-end setting (over 150 passages) ignoring retrieval compute. For reference, Google search handles millions of queries every minute (Kenshoo, 2019).

Use Case	Energy/1M Qns (kWh)	Cost/1M Qns (USD)
QA over a single passage	161	21.24
QA over 150 passages (ignore search/retrieval)	24,000	3,165

Table 3: Example energy for BERT-base QA models using batch size 16 and sequence length 256 on PC1 using one GPU. The cost is estimated at 13.19 cents per kWh. [3]

Quantity	BERT-base	DistilBERT	GPT2
# ML Nodes	3864	1932	2997
# Module Nodes	2100	560	972
# Model Nodes	28	28	28
# Tree Depth	6	5	4

Table 4: Energy dataset statistics for BERT-base, DistilBERT and GPT2 model. For each model, we construct 28 trees (model nodes) with batch sizes from 8 to 32 with a step of 8, and input sequence lengths from 32 to 256 with a step of 32. We associate features and ground-truth energy for each node in these trees.

Energy Dataset: To evaluate the energy prediction, we create a dataset that cover a wide range of input sizes for the six studied Transformer models and the 24 BERT model variants (Turc et al., 2019). Each instance in the dataset can be of type ML, Module or Model level and is associated with features shown in Table 1 and hardware measured energy. We show the statistics of the dataset for BERT-base, DistilBERT and GPT2 in Table 4.

Energy Error Metric: We measure the energy error percentage as $100 \times |PE - GE|/GE$, where PE is the predicted energy and GE is the ground truth energy.

5.3 Energy Prediction Results

We compare IrEne with the existing software measurement methods (Strubell et al., 2019; Henderson et al., 2020). We apply their method directly for all the models in our dataset. Note that their method is a fully-defined estimation model with a fixed set of parameters without any training. For IrEne experiments, we report cross-validated evaluation on the energy prediction dataset — leaving data from one model out of training set and evaluating on it, and then repeating the same for all the models.

[3] based on the US national average as of May 2021 according to https://www.electricchoice.com/electricity-prices-by-state.

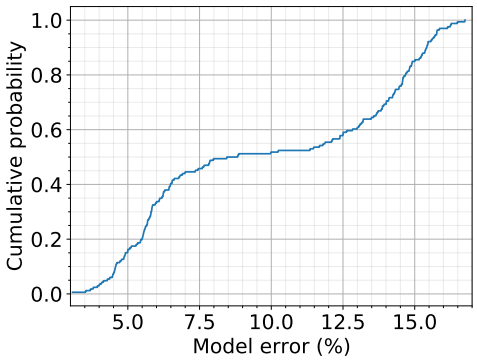

Figure 3: The CDF of model's predicted energy errors. We see that for 99% of the cases, the error is under 16%

IrEne is accurate Table 5 shows the energy prediction errors at the model-level for all the models on the two PCs. The existing software-based baseline method from Strubell et al. (2019) incurs large energy prediction errors of over 50%.

IrEne on the other hand incurs substantially lower errors, with at most 7.6% errors across the models, showing its value for reliable and accurate energy analysis. As seen from the cumulative distribution function for the model errors in Figure 3, all of IrEne's errors are below 17% and nearly half of its errors are below 10%. We note here that our leave-one-model-out cross validation specifically evaluates the generalizability of IrEne.

ML and Module Levels Errors are also low. Table 7, 6 show a break down of the IrEne errors at the ML and module levels respectively. Accurately predicting ML level energy is key to accurate predictions for at the module level and higher, as the errors will accumulate up the model tree in IrEne. It turns out that we can indeed predict ML level energy with high-levels of accuracy — errors are lower than 1%, providing reliable values for the module level predictions. Note that even unseen models (ie ones evaluated in the test partition) will be made up of the same set of ML primitives (perhaps with different input and batch sizes). The results here cannot be directly generalized to unseen ML-primitives. Module level errors are higher and vary in range (5.4% to 16.7%) across different models. Module level errors also turn out to be higher than the model level errors. This is mainly because the module level errors are averages across all intermediate module level nodes in the model tree; some modules might have bigger errors, but these get calibrated by our **End2End** energy regressor. We further characterize these effects in IrEne ablation and validation analysis.

5.4 Feature Ablations

Table 8 shows the contribution of model and resource features in IrEne energy prediction. We observe that resource features provide most of the benefits for energy estimation IrEne for all levels, confirming that resource information is important for energy prediction. Model features do not reduce ML level error because the error is already small, but they help further reduce the prediction errors for module and model levels and combining model and resource features together brings the average estimation errors further down to 8.5% and 5.5%.

5.5 Modeling Ablations

To understand the impact of learning and the architectural choices of aggregating ML level energy into module level energy in IrEne affect the model accuracy, we build three (ablated) models:

Is end-to-end learning necessary? To test this, we build a **StepWise** regressor that simply learns to predict the energy of parent node from the ground-truth energy of its child nodes at the training time. At the test time, it uses predicted energy generating predictions from ground up.

$$P_e(n) = \sum_{c \in child(n)} \alpha(c) * G_e(c) \quad \text{Training}$$
$$P_e(n) = \sum_{c \in child(n)} \alpha(c) * P_e(c) \quad \text{Testing} \quad (5)$$

Here, $\alpha(c)$ and loss are still as defined in equation 3 and 4 respectively. However, unlike the IrEne (**End2End**) regressor, the errors in the prediction of root node, do not backpropogate to its prediction of descendant nodes i.e. there is no end-to-end training.

Is tree-structure necessary? To test this, we build an **Unstructured** regressor that ignores the tree structure completely, and directly predicts the energy from the feature representation of nodes (Module and Model level) using linear regression as in equation (1). Unlike ML-level regressor though, here we need to use single set of parameters for common across the nodes.

Is learning necessary? To test this, we use the **PredictedSum** model (equation 2). Recall this model also aggregates energy predictions over the tree-structure but has no parameters to train.

Table 9 shows the ablation of IrEne with respect to different algorithmic choices of the module level energy aggregation. First, we find that the regressor that ignores the tree structure (**Unstructured**)

Machine	System	BERT-base	DistilBERT	RoBERTa-base	GPT2	DistilGPT2	OpenaiGPT	Average
PC1	Strubell et al., 2019	57.9	56.3	62.5	62.6	55.9	61.8	57.8
	IrEne	**5.8**	**11.6**	**7.1**	**3.5**	**2.2**	**2.7**	**5.5**
PC2	Strubell et al., 2019	55.1	52.6	58.9	54.6	49.8	60.6	55.6
	IrEne	**10.0**	**9.4**	**7.1**	**6.1**	**4.9**	**5.9**	**7.2**

Table 5: Energy Prediction Errors at Model level: Comparing IrEne and a software measurement baseline for the two PCs. IrEne is significantly more accurate than Strubell et al., 2019.

Machine	BERT-base	DistilBERT	RoBERTa-base	GPT2	DistilGPT2	OpenaiGPT	Average
PC1	5.37	5.93	5.44	14.92	14.73	13.98	8.54
PC2	6.78	7.96	6.69	16.65	16.41	16.07	10.16

Table 6: Energy Prediction Errors at module levels using IrEne on two PCs. Note that in Table 11 at the appendix, we also show a subset of the module level energy errors using Strubell et al., 2019.

Machine	Embedding	LayerNorm	Linear	Tanh	MatMul	Softmax	Conv1D	Average
PC1	0.65	0.89	0.60	0.82	0.61	1.0	0.58	0.70
PC2	0.38	0.66	0.55	0.43	0.43	0.67	0.41	0.53

Table 7: Energy Prediction Errors at ML levels using IrEne on two PCs. Note that the evaluation for these operation-specific (eg. Embedding) regressors is done using the leave-one-model out setting as before.

	ML	Module	Model
IrEne	0.70	**8.54**	**5.52**
w/o resource features	5.76	11.54	7.08
w/o model features	**0.63**	8.87	7.32

Table 8: Energy Prediction Errors of IrEne with ablated features. Both model and resource features help the IrEne's performance at model and module levels, while resource features are sufficient for ML-level.

	Module	Model
IrEne (End2End)	**8.54**	**5.52**
StepWise	9.28	14.84
PredictSum	16.4	17.69
Unstructured	278.0	39.79

Table 9: Energy Prediction Errors of IrEne using different module/model level regressors on PC1. Tree structure of the regressor crucial, and end-to-end optimisation on tree helps IrEne to get lower errors.

performs significantly worse than all other regressors that do consider it. Interestingly, learning without structure even performs worse than **PredictedSum** regressor that naively adds child energy without any learning, highlighting the importance of tree-structure. Further, learnt weighted sum outperforms **PredictedSum** regressor. In particular, **End2End** regressor performs better than **StepWise** regressor showing the importance of optimizing on whole tree in an end-to-end fashion.

5.6 Interpretable Energy Analysis

In this section, we use the interpretable energy analysis from IrEne to show energy bottlenecks for given Transformer models, how energy varies for different model architectures, and how it can be used to effectively pick accuracy-energy trade-offs.

Finding energy bottlenecks: We use IrEne to analyze the energy bottlenecks in Transformer models. For simplicity of analysis, we predict the energy for modules that are immediate parents of the ML level nodes and use it calculate the percentage of energy it contributes to the model overall. Table 10 shows the energy breakdown of two models: RoBERTa-base and GPT2. We observe that self-attention layers in RoBERTa-base model consume 31% of the total energy while it is the feed forward layers in GPT2 that consume more than 59% of the energy. The module level energy breakdown of all models in Table 12 in Appendix C. We also present the full energy breakdown of the BERT-base model and annotate each node with predicted energy percentage in Figure 5 in the Appendix.

Task accuracy versus energy tradeoffs:

We fine-tune BERT-24 models (Turc et al., 2019) on the Stanford Sentiment Treebank V2 (SST2) (Socher et al., 2013) using the default examples in the HuggingFace Transformers (Wolf et al., 2020) without any hyperparameter tuning. We evaluate the accuracy on the dev set of SST2. These

Module	Energy %
RobertaSelfAttention	31.24
RobertaIntermediate	30.57
RobertaOutput	28.64
RobertaSelfOutput	09.11
RobertaEmbeddings	00.41
RobertaPooler	00.03

(a) RoBERTa-base

Module	Energy %
MLP	59.13
Attention	37.94
LayerNorm	2.84
Embedding	0.1

(b) GPT2

Table 10: Module level predicted energy breakdown of two Transformer models. We average the energy of these modules across all input sizes for each model architecture. Self-attention is the energy bottleneck in RoBERTa-base, but for GPT2, the bottleneck is feed forward layers (MLP module).

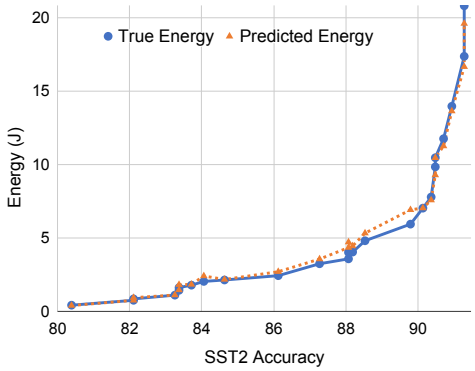

Figure 4: Ground-truth and predicted energy vs accuracy on SST2 task for BERT-24 models. Energy data is collected with batch size 16 and sequence length 128. Because our energy predictions are accurate, we can use energy consumption vs NLP model accuracy trade-offs to select a model.

models are not part of our energy prediction training data. We additionally exclude BERT-base from training data to show the extensibility of IrEne.

Given an energy budget, IrEne allows for selection of an optimal architecture that gets the highest accuracy for a task. In Figure 4, we see that it is possible for models to use more energy but return lower accuracy than other models which might use less energy. Similarly, given an accuracy target, we can choose an architecture with the lowest energy use. For example, for a target of 88% accuracy or above, there are many such models ranging from 4J all the way to 12J. Last, we point out that the trade-off curve based on the predicted energy mirrors that of the ground-truth well enough to be used as an accurate proxy.

6 Discussion

This work focused on inference energy predictions of Transformers on a target hardware device. The model tree abstraction is general and not tied to Transformer architectures nor to specific deep learning frameworks, it is extensible to other neural networks like LSTM and frameworks like TensorFlow. The abstraction is built from the computational graph and knowledge about the model architecture and underlying software. As long as these are available we can apply our methodology to other architectures as well.

Predicting the training energy is an important and a more challenging problem. We believe our methodology can be extended. However, it will require tracking the energy of both forward and backward processes and even modeling other aspects training dynamics, for example, time to converge to specific accuracy.

Scaling to unseen hardware is an important and challenging area that needs further research. It requires both measuring the ground truth energy for a more diverse collection of hardware and designing proper hardware-specific features (i.e., L1 cache size, CPU cores, etc.). We believe IrEne's methodology can be extended to calibrate software reported energy as a way to scale how we collect ground truths (as weak-supervision). In the future, we plan to study workloads on more hardware to choose proper features that capture the hardware energy differences.

7 Conclusions

Energy consumption of NLP models is an important consideration from a cost perspective and increasingly, from an environmental impact perspective as well. Designing energy efficient and cost-effective models requires both accurate and interpretable energy modeling. In this work, we showed that by carefully combining resource utilization with model description based features, we can develop a multi-level energy prediction model that is not only highly accurate but is also able to provide a break-down of how its different components contribute to its overall energy.

8 Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No 2007362.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283.

Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. 2020. Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models. *ICML Workshop on "Challenges in Deploying and monitoring Machine Learning Systems"*.

Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 316–331, New York, NY, USA. Association for Computing Machinery.

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. *Advances in Neural Information Processing Systems*, 33.

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.

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.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH computer architecture news*, volume 44, pages 367–379, New York, NY, USA. IEEE.

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.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *arxiv*.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. *arXiv:2002.05651 [cs]*.

Glyn Hudson. 2021. emonPi - OpenEnergyMonitor.

Kenshoo. 2019. How Many Google Searches Per Day? SEM Pros Should Know This!

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*.

Nvidia. 2021. NVML API Reference Guide.

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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc.

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.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011a. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011b. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *OpenAI Blog*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs]*.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. *arXiv:1907.10597 [cs, stat]*.

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.

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.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. *arXiv:1908.08962 [cs]*. ArXiv: 1908.08962.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv:1910.03771 [cs]*.

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. *arXiv:2002.05829 [cs]*.

A IrEne Implementation Details

In this section, we provide the implementation details of IrEne. IrEne is implemented for PyTorch (Paszke et al., 2019), but can be extended to TensorFlow (Abadi et al., 2016) in future.

A.1 Constructing the model tree

The first step to extracting the model tree is to run the model on the target hardware. We run the version of the model on HuggingFace Transformers library v4.2.2 (Wolf et al., 2020) for random data of different input sizes. Once run, we have both the execution graph and the JIT trace that provides runtime information. We use existing PyTorch APIs to obtain module level nodes, ML primitives, and the relationships between them, from the execution graph. In some cases, the NLP model may use customized ML primitives. To extract information about these custom primitives, we combine information from the JIT trace and the execution graph. Once we obtain all the component, we can construct the model tree.

The following ML primitives are used in Transformers: Linear, LayerNorm, Embedding, BatchNorm1d, Conv1d, MaxPool1d, AvgPool1d, LSTM, Tanh, Conv1D, LogSigmoid, ReLU, Sigmoid, GELU, and LeakyReLU. Two custom primitives: matrix multiplications (including torch.matmul, torch.bmm and torch.einsum), softmax (torch.softmax).

Machine	PC1
BERT-base	32.54
DistilBERT	62.80
RoBERTa-base	13.36
GPT2	24.96
DistilGPT2	35.93
OpenaiGPT	42.37
Average	35.33

Table 11: Energy Prediction Errors at Module levels using Strubell et al., 2019 methodology on PC1.

.

A.2 Model features

The model features reflect hardware-independent compute and memory information for a given model. We use the model execution to extract model features used by IrEne for energy prediction. We add forward hooks to each node in the model to track the shape and input data of each module and ML primitive. PyTorch hooks only support tuple arguments, but we extend these to also support keyword based arguments. The JIT trace contains information about the number of FLOPs and memory bytes for each module and ML primitive. By combining JIT information and the information obtained from our hooks, we get the model features.

A.3 Resource features

Resource features capture how the models use hardware resources and cause energy activities. Existing work (Henderson et al., 2020) uses the OS resource profiler to log the resource utilization of CPU, memory and GPU events. However, this incurs high profiling overhead, and profiling is only done at a low rate of once every second. Instead, to monitor resources, we obtain the CPU utilization by directly reading `/proc/stat` and memory usage by reading `/proc/meminfo` via a C program. We simultaneously log the GPU utilization, GPU memory usage, GPU Streaming processor (SM) clock frequency and GPU memory frequency using the Nvidia NVML API (Nvidia, 2021). To maintain low monitoring overhead, we log resources every 170 ms, resulting in less than 0.5% increase in CPU utilization and < 15 MB memory footprint.

Note that both model and resource features are extensible, meaning that one can add more either hardware-specific features or new model features for newer deep learning frameworks or emerging hardware like customized deep learning accelerators.

A.4 Regressor Training Procedures

We've implemented IrEne using SciKit Learn (Pedregosa et al., 2011a) and PyTorch (Paszke et al., 2019). We learn linear regressors for ML-level in SciKit Learn (Pedregosa et al., 2011b), and module and model level regressor in PyTorch, which allows easily optimizing on dynamic tree-structured computation graphs. We use Adam optimizer (Kingma and Ba, 2014) with 0.001 learning rate. In our experiments τ in equation 3 is fixed value of 10. We normalize all the features to have 0 mean and 1 standard deviation, learning mean and standard deviation from the training set and applying it on the test set.

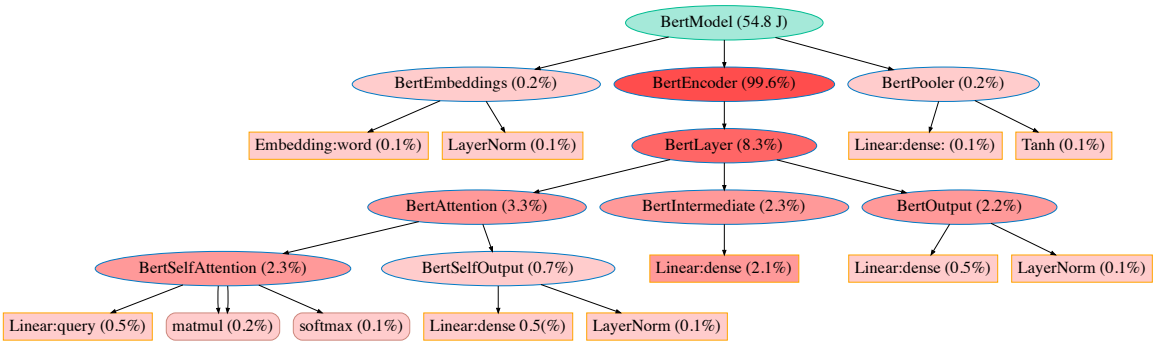

Figure 5: Abridged view of a BERT-base-uncased model annotated with predicted energy from our prediction method. The root contains the absolute energy of the model while every other node is annotated with its respective energy percentage share. Darker colors represent nodes that consume a higher percentage of energy. There are 12 BertLayer modules in the actual model. We show just one for brevity. The shown energy is an average of energy of the node across all (batch size, sequence length) models of BERT-base-uncased type.

B Software Measurements Results

We use *experiment-impact-tracker* (Henderson et al., 2020) to estimate software-based energy measurements for the models at a module level as well as ML level. Table 11 shows the percentage error in software based measurements for module level operations. We calculate a model's module level error as average percentage error over runs for batch sizes 24 and 38, and sequence length 32 and 128. Getting granular ML level software energy corresponding to Strubell et al. (2019) requires modifying the existing framework which is non-trivial. We leave this to future work.

C Energy Breakdowns

We show module level predicted energy breakdown of four Transformer models in Table 12, and show an abridged view of BERT-base-uncased tree annotated with predicted energy and distribution in Figure 5.

Module	Energy %
BertOutput	31.89
BertSelfAttention	29.26
BertIntermediate	27.97
BertSelfOutput	09.74
BertEmbeddings	00.34
BertPooler	00.11

(a) BERT-base

Module	Energy %
MLP	61.41
Attention	35.70
LayerNorm	2.79
Embedding	0.11

(b) OpenAI-GPT

Module Name	Energy %
FFN	57.23
MultiHeadSelfAttention	39.46
LayerNorm	2.69
Embeddings	0.62

(c) DistilBERT

Module Name	Energy %
FFN	57.50
MultiHeadSelfAttention	39.43
LayerNorm	2.86
Embeddings	0.21

(d) DistilGPT2

Table 12: Module level predicted energy breakdown of four Transformer models. We average the energy of these modules across all available input sizes for each model architecture. Interestingly, we find that even models with similar architecture have different types of energy bottlenecks. For example, BERT-base has similar architecture to DistilBERT but has different energy bottlenecks.