Towards a Catalog of Energy Patterns in Deep Learning Development

Shriram Shanbhag[1], Sridhar Chimalakonda[1], Vibhu Saujanya Sharma[2], and Vikrant Kaulgud[2]
[1]*Research in Intelligent Software & Human Analytics (RISHA) Lab*
Department of Computer Science & Engineering
Indian Institute of Technology Tirupati, India
[2]Accenture Labs, India
{cs20s503,ch}@iittp.ac.in,{vibhu.sharma,vikrant.kaulgud}@accenture.com

ABSTRACT

The exponential rise of deep learning, aided by the availability of several frameworks and specialized hardware, has led to its application in a wide variety of domains. The availability of GPUs has made it easier to train networks with a huge number of parameters. However, this rise has come at the expense of ever-increasing energy requirements and carbon footprint. While the existing work tries to combat this issue by proposing optimizations in the hardware and the neural network architectures, there is an absence of general energy efficiency guidelines for deep learning developers. In this paper, we propose an initial catalog of 8 energy patterns for developing deep learning applications by analyzing 1361 posts from Stack Overflow. Our hope is that these energy patterns may help the developers adopt energy efficient practices in their deep learning projects. A survey with 14 deep learning developers showed us that the developers are largely in agreement with the usefulness of the catalog from an energy efficiency perspective. A detailed description of the catalog, along with the posts related to each energy pattern, is available at the following link: ***https://dl-energy-pattern.netlify.app***

CCS CONCEPTS

• **Software and its engineering → Software creation and management**.

KEYWORDS

deep learning, neural networks, energy efficiency

ACM Reference Format:
Shriram Shanbhag[1], Sridhar Chimalakonda[1], Vibhu Saujanya Sharma[2], and Vikrant Kaulgud[2]. 2018. Towards a Catalog of Energy Patterns in Deep Learning Development. In *Proceedings of Evaluation and Assessment in Software Engineering (EASE '22).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

EASE '22, June 13–15, 2022, Gothenburg, Sweden

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In recent years, deep learning has become a very active area of research. Deep learning techniques are being extensively used to solve a wide variety of machine learning problems ranging from computer vision [24], speech recognition [17], natural language processing [6], software engineering [27] and machine translation [50]. Deep learning has been used to improve the performance compared to the existing methods significantly and to develop the state of the art techniques. These improvements have been aided by the availability of deep learning libraries such as *Tensorflow*, *Pytorch*, *Theano*, *Caffe* etc. and also by the availability of faster specialized hardware like GPUs and TPUs. However, it has also led the deep learning models to have an increasingly high number of parameters making them energy intensive. For instance, the GPT-3 model has 175 billion parameters [5] and training it requires 28000 GPU-days without considering the training spent on research and development [1].

Few deep learning models have surpassed humans in performance. For example, a deep learning model has achieved a lower top-5 classification error rate of 3.57% [18] than the human top-5 classification error rate of 5.1% [45] on the large scale ImageNet dataset. However, obtaining these state-of-the-art techniques require training the network for thousands of hours on specialized hardware. The increase in the computational demand translates to the increase in demand for energy production, which also constitutes a significant portion of greenhouse gas emissions [14]. This makes the energy efficiency of deep learning a very important factor to consider from the perspective of both the environment and the economy. While the state of the art deep learning models are generally evaluated by their performance, they are not evaluated in terms of their energy efficiency.

The potential environmental and economic implications of deep learning have led the researchers to focus on the energy efficiency of the deep learning models. Methods that have been proposed to improve energy efficiency of the deep learning models include techniques like pruning [52], quantization [26, 35], use of energy efficient hardware accelerators [7, 23, 41], modification of methods that the hardware uses to perform computations [21, 48] and developing energy efficient models specific to a particular application [4].

In domains such as mobile application development, where energy considerations are critical, researchers have used the data from online platforms like GitHub and Stack Overflow to address the issue of energy efficiency in the development of mobile applications [10, 37, 43]. The discussions on popular platforms like Stack

Overflow and GitHub can provide insights into the developer community's practices. They have led to a greater understanding of the solutions used by the developers to save energy [37, 43]. They have also led to the development of energy efficient programming guidelines that the developers can refer to [10], also called *energy patterns* for developing their applications. An advantage of using data from these forums is that, since these platforms are used frequently by the developers, the set of energy patterns obtained are the ones that are already widely in use.

Given the energy-hungry nature of deep learning and its possible implications, there is a need for a similar catalog of energy patterns in deep learning development. So, we propose an initial catalog of 8 energy patterns in deep learning development as a first step. Our hope is that the developers would refer to these patterns to make their deep learning application development energy efficient.

Our study aims to answer the following research questions.

- **RQ1:** ***What are the energy patterns that can be found from the Stack Overflow discussions on deep learning?***
- **RQ2:** ***Do the practitioners think that the patterns obtained are useful in making the deep learning development energy efficient?***

We perform an analysis of 1361 posts from Stack Overflow to arrive at the energy patterns. We use keyword based filtering of posts as followed in other studies on energy efficiency [10, 43] followed by a thematic analysis of the posts to obtain these patterns. While the questions may not be asked with the intention of improving energy efficiency, we try to look at these posts from the lens of energy during our thematic analysis. We then perform a survey with 14 deep learning developers to understand their perception of whether the use of each of these patterns makes the deep learning application development energy efficient.

The contribution of the paper is as follows:

- A catalog of 8 energy patterns in deep learning software development. The detailed description of the catalog, source posts is available online at *https://dl-energy-pattern.netlify.app*
- A developer survey on the usefulness of these patterns from energy efficiency perspective.

We organize the rest of the paper as follows. We discuss the workflow of the study involving the data collection, data preprocessing, thematic analysis and the developer survey in the Section 2. The energy patterns are presented in the Section 3. The results of the survey are presented in the Section 4. The sections 5, 6 and 7 present threats to validity, discussion and related work. We finally present the conclusion and future directions in Section 8.

2 WORKFLOW

The workflow of our study is shown in the Figure 1. It involves collection of data, preprocessing, thematic analysis and a developer survey. This section presents the details of the entire process.

2.1 Data Collection

We used data from a popular platform, Stack Overflow[1] for our study. Stack Overflow provides the developers with convenient access to the knowledge of their peers. Developers post questions on Stack Overflow when they face issues. The usefulness of the questions and the answers on the platform can be gauged by the score based on the number of upvotes and downvotes.

Library	Tag	No. of posts
Tensorflow	*tensorflow*	72,654
Keras	*keras*	36,614
Caffe	*caffe*	2,842
Theano	*theano*	2,449
PyTorch	*pytorch*	13,712

Table 1: Summary of data collected from Stack Overflow

Stack Overflow provides users access to their data through their data explorer.[2] The explorer lets the users compose queries that can be executed to access the data required for the study. The data obtained by executing the query can be downloaded in the form of a CSV file.

Stack Overflow allows users to add tags or labels to their posts that would provide easier identification of the category that the posts belong to. We use the labels associated with the deep learning libraries to gather posts relevant to our study. We use the Stack Exchange data explorer to collect posts that are tagged with *tensorflow*, *keras*, *caffe*, *theano* and *pytorch*. We obtained 72,654 posts for *tensorflow*, 36,614 posts for *keras*, 2,842 posts for *caffe*, 2449 posts for *theano* and 13,712 posts for *pytorch*. The data obtained in this step is summarized in the Table 1. The total number of posts obtained during this process was 128,271.

2.2 Data Preprocessing

We apply the following preprocessing steps to filter out the noise in the text data extracted from Stack Overflow.

2.2.1 Removal of posts without accepted answer. When the users post a question on Stack Overflow, the platform gives them an option to *accept* the answer that helped them solve their problem. A post having an accepted answer indicates that the problem was resolved. To maintain the quality of the data in our study, we consider only the posts with an accepted answer and discard the ones that do not have an accepted answer.

2.2.2 Filtering of Posts based on Keywords. Many studies related to energy efficient software development that analyze discussions from Stack Overflow and GitHub have used keyword based filtering to gather relevant posts and discussions [10, 37, 43]. We looked at the list of keywords used in other energy related studies [9, 10, 28, 32, 37, 42, 43] and created a combined set of keywords. One of the authors and a volunteer collectively selected the list of keywords from the set that is relevant in the context of deep learning software development. Disagreements in the keywords selection process were resolved through discussions. In addition, we also added a few keywords that may be related to the existing work on energy efficient deep learning [2, 41, 52].

The final set of keywords were as follows.

[1] https://stackoverflow.com/

[2] https://data.stackexchange.com/stackoverflow/

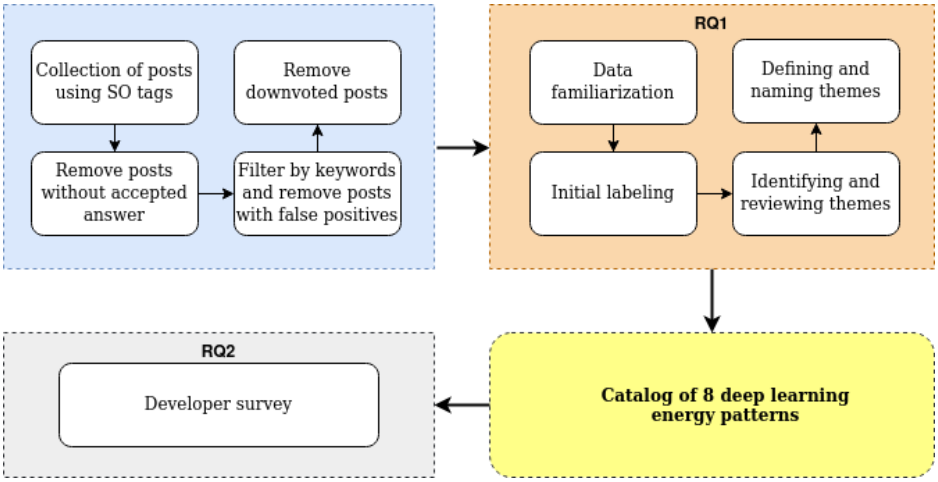

Figure 1: Workflow of the study

"*leak*", "*drain*", "*energy*", "*power*", "*efficien*", "*sustainab*", "*prun*", "*distil*", "*quantiz*"

The rationale behind the keywords are shown in the Table 2.

Upon applying keyword based filtering to the collected data, we arrived at a list of 2072 posts.

2.2.3 Removal of posts with false positives. After an initial inspection of the posts, we found that many of them had false positives. For example, the search term *power* leads to matches like *powershell* and *power of 2* that are clearly not related to energy. We took a random sample of 100 posts and compiled a list of such terms that may lead to false positives from a random sample. The posts containing these terms were then filtered out from the dataset. The number of posts after the removal of false positives was 1384.

2.2.4 Removal of downvoted posts. Stack Overflow lets the users vote on the questions posted by other users. These votes are used as an indicator of the usefulness of the question. The score assigned to each question is the difference between the number of upvotes and downvotes. The posts with negative scores have more downvotes and may not be very useful. So we filtered them out from our dataset.

After performing the above filtering steps, we end up with a dataset of 1361 posts[3]. We went on to proceed with the thematic analysis of these posts as described in Section 2.3.

2.3 Thematic Analysis

The goal of this step is to answer the research question *RQ1*. We used the method of thematic analysis [15] on the dataset of Stack Overflow posts to identify energy patterns among the posts. This method of thematic analysis has been used in other energy related studies such as [10, 37]. Our method of thematic analysis contained four stages.

2.3.1 Familiarization with the data. In this stage, we read through all the posts with the comments and the answers. This let us familiarize ourselves with the terminologies used in the posts. We also used internet forums to understand the terminologies we were unaware of.

2.3.2 Initial Labeling. For each post, we assigned a short sentence that describes the core idea expressed in the post. This step was performed in several iterations to refine the codes.

2.3.3 Identifying and Reviewing Themes. Once we already had the initial labels, we tried to combine various related labels under a theme or a pattern. Some labels had less than 3 occurrences and could not be merged with other labels to form a theme. We discarded such labels as they had a very low number of occurrences to form a theme of their own. Some of the labels could not be linked to energy efficiency and were discarded. After one round of labeling, the themes were reviewed, and appropriate reassignment of themes to the labels was done in cases where it was necessary.

2.3.4 Defining and Naming Themes. At this point, we had an initial set of themes with their initial names. These names were refined to describe the patterns found in the previous step more effectively. We also tried to assign the final themes to the labels that were not assigned any themes in the previous step. The posts that would still not fit under any of the themes were discarded. We ended up with 8 energy patterns across 217 related posts. The final outcome is summarized in the Table 3.

It must be noted that even though the total number of posts taken up for thematic analysis was 1361, only 217 could be linked to the factors that influence energy efficiency. This high discard rate is expected and is in line with the observations from other energy related studies [10, 37] that have employed keyword based filtering.

2.4 Developer Survey

In this section, we describe the developer survey we conducted to answer *RQ2*.

We deployed an online survey to the deep learning developers to find out their perception of the impact of using the energy patterns on the energy consumption of the development process. The

[3]https://anonymous.4open.science/r/ease_2022-835F/all_posts.csv

Original Terms	Stemmed Keywords	Rationale
Leak, Drain	*leak*, *drain*	To capture posts referring to leak and drain of power
Efficiency	*efficien*	To capture posts related to improving efficiency
Power Consumption	*power*	To capture posts refering to power consumption
Energy Consumption	*energy*	To capture posts related to energy consumption
Sustainability	*sustainab*	To capture posts related to sustainable development
Model Compression	*prun*, *distil*, *quantiz*	To capture posts related to model compression techniques which is a common method used in energy efficient models from the literature

Table 2: Rationale behind keywords chosen for data collection

survey form contained a section with a few questions about the demographics of the participants. The section also contained questions that seek to know the deep learning frameworks used by the developers, the type of data they work with, and whether or not they consider energy factors while developing deep learning applications. Each of the subsequent sections had an explanation of one of the energy patterns through a *context*, the *problem* that causes energy inefficiency, and the energy efficient *solution* to the problem. At the end of the explanation, we had the question *"To what extent do you agree that the use of the above energy pattern can reduce the overall energy consumption of the deep learning development?"*. The respondents were required to rate their agreement on a 5 point Likert scale with *Strongly Agree* corresponding to the highest score of 5 and *Strongly Disagree* corresponding d the lowest score of 1. In addition, the respondents were provided an optional "additional comments" section under every energy pattern to let them explain the reason for their agreement/disagreement and provide relevant feedback.

The survey form was sent to 26 developers, out of which only 14 responded. The results of the survey are presented in Section 4.

3 ENERGY PATTERNS

This section discusses the results of the thematic analysis. We arrived at a list of 8 energy patterns as summarized in the Table 3. Each of these energy patterns are explained in this section using the relevant *context*, the *problem*, the suggested *solution* and an *example*.

The catalog is also available online: *https://dl-energy-pattern.netlify.app*. The website also provides links to the all the related posts corresponding to each energy pattern.

3.1 Apply transfer learning with pre-trained networks whenever feasible

Context: Training the deep learning models typically requires labeled data points. At times, the training data can be huge, and networks with a large number of parameters may be required to achieve the desired performance.
Problem: The training process typically involves multiple rounds of training by trying out different permutations of hyperparameters to arrive at the best model. In the case of large networks with a huge amount of training data, the process can take a very long time due to the number of computations involved. This can make the training process expensive in terms of energy consumed.
Solution: The energy requirements of training the network could be cut down using transfer learning if pre-trained models exist for the given task. Transfer learning [49] is the approach where a machine learning model trained on one task is reused on a different task. Transfer learning can be used when the data available for training is limited or collecting them is too expensive. Sometimes, transfer learning may involve fine tuning the pre-trained model with a smaller dataset. Due to the absence of or reduction in the training involved, the corresponding computational energy spent on the process can be saved.
Example: Consider a scenario where the user is required to build a model that gives a vector representation of words from a natural language text. It would require training the neural network using the data from a large corpus of text over several iterations to arrive at an adequate vector representation. Instead, the user can use a pre-trained model like *Sentence-BERT* [44] and save the energy required to train a model from scratch.

3.2 Use checkpointing at regular intervals for networks with long training times.

Context: The training duration of some deep learning models can be very long, ranging from several hours to several months [5] even with the use of powerful hardware. This would require the training hardware to run for a long duration without any breakdown. Failure in hardware, interruption in the power supply, or errors in the software may cause termination of the training process before it completes.
Problem: Incomplete training due to termination of the process leads to the loss of knowledge gained during the training. This would render the energy spent on the training process useless. Subsequent retraining would require the process to start from scratch.
Solution: Checkpointing is a fault tolerance technique used in long processes. In neural networks, checkpointing lets the users save the intermediate model state generated during the training process. In case of a failure, the users can resume the training process from the latest checkpoint instead of restarting from scratch. While checkpointing at regular intervals may come with a few overheads of its own [38], it can help save energy wastage caused by the termination of a long training process due to faults.
Example: Consider a model that requires several hours of training on a preemptible instance to get decent results. Since the preemption may occur in the middle of the training, not having a checkpoint would lead to the loss of knowledge gained during the process since the training cannot be resumed from the point of preemption.

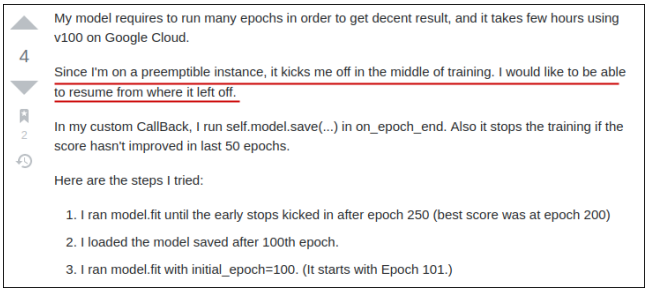

Figure 2: Example of a post involving *Checkpointing* pattern

Having checkpoints would preserve the state of the model at regular intervals, and the training could be resumed from the latest checkpoint in the case of preemption. This would save the energy wastage caused by the loss of knowledge due to the interruption in training. The example is taken from the Stack Overflow post shown in Figure 2.

3.3 Prune large networks to reduce the impact of over-parameterization

Context: *Over-parameterization* or the presence of redundant parameters that do not contribute much to the output of the model is a common problem in deep neural networks, especially in architectures like CNN [11]. It is a widely believed reason why a neural network may fit all training labels [12].
Problem: Over-parameterization can cause an increase in the size of the networks. More parameters can also increase the number of computation and memory accesses. The increased computational load can cause increased power consumption.
Solution: Network pruning is a strategy involving the removal of non-critical or redundant neurons from the neural network to reduce its size with minimal effect on the performance. The use of pruning on deep learning models can be an effective strategy to improve energy efficiency, as seen in some of the existing work [46, 52]. The pruned model improves energy efficiency during inferencing by reducing the bandwidth and power requirements due to the reduced number of FLOPs (floating point operations). Due to reduced size, pruned models also reduce the memory requirements of the network.
Example: Consider an object detection model using an R-CNN model that has a high accuracy. But due to its large size that result in large number of computations, inferencing a single image takes a long time. The process also consumes higher energy due to the large number of computations. Pruning the model appropriately would cut down the redundant computations resulting in both faster inferencing and lesser energy consumption. Figure 3 shows the post 63687033[4], the example is based on.

[4]https://stackoverflow.com/questions/63687033

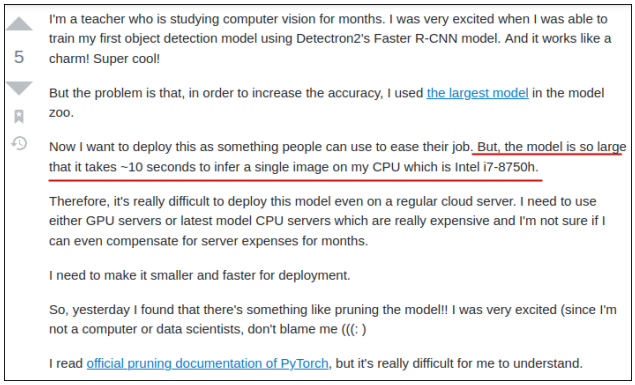

Figure 3: Example of a post involving *Pruning* pattern

3.4 Use network quantization in applications with memory constraints where a minor loss in performance is acceptable

Context: With the rising use of deep learning in different domains, the models are being used on battery powered devices such as smartphones [3, 25, 33, 34]. Doing so is also beneficial from the point of view of bandwidth and latency. However, the large size of these models with their energy consumption requirements can pose a significant challenge in battery powered devices.
Problem: Running the deep learning models can involve millions of multiplication and addition operations. Having a high precision representation of the parameters causes these operations to become expensive in terms of energy requirements on battery powered devices.
Solution: Network quantization involves reducing the number of bits to represent the parameters of the neural network. Quantization has been used in the existing work to improve the energy efficiency of the deep learning models [26, 41]. Quantizing the network can make the multiplication and addition operations less expensive computationally due to the reduction in the bit-width of the operands. This causes a reduction in power consumption. It also cuts down the memory requirements due to the reduction in the model size. If done properly, quantization only causes a minor loss in performance and does not affect the output significantly [30]
Example: Consider a MobileNet V2 model that needs to be deployed on a smartphone. Quantization of the model to use parameters using 4-bit precision can lead to a smaller model size with lesser energy consumption per computation.

3.5 If pre-trained models are too large for a given task, apply knowledge distillation or use distilled versions of pre-trained models

Context: The use of pre-trained models can save the energy spent on training a new network from scratch. However, these models can sometimes be too large for the given task and given power consumption and resource constraints.
Problem: Using pre-trained models that are too large for a given task may cause excessive computations and memory access due to

Pattern Number	Energy Pattern	Brief Description	Number of Occurrences
P1	Pre-trained Model	Apply transfer learning with pre-trained networks whenever feasible	36
P2	Checkpoint	Use checkpointing at regular intervals for networks with long training times.	13
P3	Pruning	Prune large networks to reduce the impact of over-parameterization	18
P4	Quantization	Use network quantization in applications with memory constraints where a minor loss in performance is acceptable	31
P5	Distillation	If pre-trained models are too large for a given task, apply knowledge distillation or use distilled versions of pre-trained models	11
P6	Efficient Read-Write	Minimize the memory footprint while performing read/write operations	19
P7	Memory Leaks	Take care of memory leaks and OOM errors before starting the training process	67
P8	Tensor Operations	Look for built in library functions for tensor operations before writing custom implementations	22

Table 3: Energy patterns in deep learning development

the number and size of model parameters. This leads to increased power consumption during fine tuning and inferencing.
Solution: Knowledge distillation [19] refers to the process of transferring the knowledge from a larger model to a smaller one. It involves training a smaller model called *student* to mimic a larger pre-trained model called *teacher* using an appropriate loss function. It solves the problem of expensive inferencing caused by larger pre-trained models.
Example: Consider a classification task that needs to classify movie reviews. A pre-trained model like BERT with some fine-tuning can be used without having to train the model from scratch. However, if a minor compromise in the accuracy can be tolerated, a distilled version of the BERT model called DistilBERT can be used with some fine-tuning. Due to its smaller size [47], DistilBERT can be much more energy-efficient for fine-tuning and inferencing. [47].

3.6 Minimize the memory footprint while performing read/write operations

Context: Working with deep learning models requires reading and writing enormous amounts of data during *data cleaning*, *data preparation*, and *inferencing* stages of the deep learning workflow.
Problem: Read and write operations influence the energy consumption of a processor [22, 51]. Due to the size of data involved in deep learning, not performing them efficiently may lead to increased machine cycles, unnecessary data movements, and increased memory footprint leading to increased energy consumption.
Solution: While reading and writing the data, take care to minimize the number of operations using efficient implementation methods. Avoid non-essential referencing of data to reduce the memory footprint.
Example: Suppose a user wants to load a large number of images in batches to train a model in *tensorflow*. Loading these images in the RAM before the training requires a lot of memory to hold the images increasing the memory footprint. Instead, the user could use *map* function from *tf.data.Dataset* to have only the path to the images and load the batch of images only during the training step. Loading the data this way minimizes the memory usage required to hold all the data in the RAM. Figure 4 shows the answer from the corresponding post [5] the example is based on.

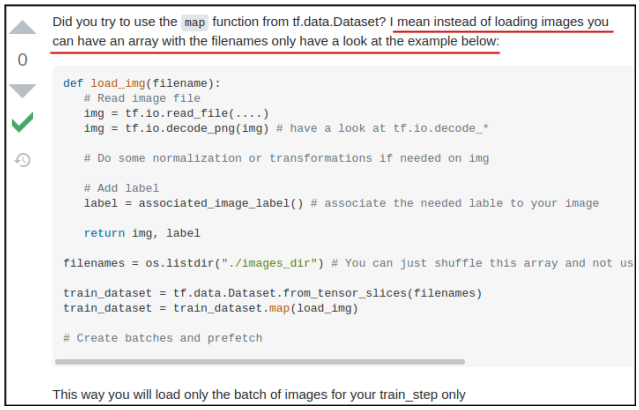

Figure 4: Example of a post involving *Efficient read-write* pattern

3.7 Take care of memory leaks and OOM errors before starting the training process

Context: Memory management related exceptions constitute a major portion of fault triggers in deep learning [13]. Memory leaks and OOM (out of memory) errors may occur during the training, loading of data, writing/saving of data, and also during the inferencing.
Problem: When a program terminates due to OOM errors during the process of training the network, in the absence of a checkpoint,

[5] https://stackoverflow.com/questions/61368378

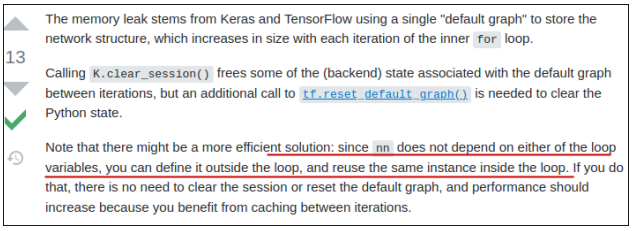

Figure 5: Example of a post involving *Memory leaks* pattern

the knowledge gained during the training process is lost. Due to this, the computational energy spent on the training is wasted.

Solution: Factor in the memory availability constraints and possible OOM exceptions while designing the program to train the network and take appropriate steps to avoid them. This may reduce the chances of OOM errors and prevent the corresponding waste of energy.

Example: An example of this pattern can be seen in the Stack Overflow post with ID 42886049[6]. The user runs into an out of memory exception while training a Keras sequential model due to the inefficient management of the memory in a program that uses a *for* loop, causing a waste of energy spent on the training. This could be avoided by defining the network outside the *for* loop and reusing the same instance every iteration, as suggested in the answer shown in Figure 5.

3.8 Look for built in library functions for tensor operations before writing custom implementations

Context: A tensor is a generalization of vectors and matrices in higher dimensions. Tensors are a fundamental data structure used in deep learning and are used to represent the input, output, and transformations. Thus tensor operations are performed very frequently in deep learning programs.

Problem: Given the large number of operations that can be performed on tensors, the users may lack awareness about appropriate library functions that can help them implement the operation and go for custom implementations. These custom implementations may not be the most efficient or optimized ones.

Solution: Look for existing library functions or ways to combine multiple library functions that can perform the given tensor operation. Library functions are often designed to perform the operations more efficiently with optimized use of resources. Due to this, they may be more energy efficient.

Example: Consider a case based on the post 67452064[7] where the user wants to multiply every element in a batch of tensors with every other element except for itself. The user could write a custom implementation as shown in (a) of the figure 6. But this method may not be optimized in terms of usage of memory and the number of computations. A more energy efficient way instead would be use combination of built-in *Pytorch* functions as shown in Figure 6 (b).

[6]https://stackoverflow.com/questions/42886049
[7]https://stackoverflow.com/questions/67452064

```
a
import torch

tensor = torch.randn(50, 512, 512)

batch_size = tensor.size(0)
list1 = []
for i in range(batch_size):
    list2 = []
    for j in range(batch_size):
        if j != i:
            x_out = (tensor[i] * tensor[j]).sum()
            list2.append(x_out )
    list1.append(sum(list2))

out = sum(list1)
```

```
b
import torch
tensor = torch.randn(50, 512, 512)

batch_size = tensor.size(0)
tensor = tensor.reshape(batch_size, -1)
prod = torch.matmul(tensor, tensor.transpose(0,1))
out = torch.sum(prod) - torch.trace(prod)
```

Figure 6: Example of a post involving *Tensor operations* pattern

4 SURVEY RESULTS

This section presents the results of the online survey on the energy patterns obtained in our study. We first present the demographics and other details about the respondents, followed by the answer to our research question *RQ2*. The detailed survey results are available here[8].

The survey was answered by 14 respondents in the age group of 20 to 33 years. The respondents included 8 industry practitioners and 6 researchers. They had a general programming experience ranging from a year to 13 years. With respect to the experience of the respondents in deep learning development, six of them had an experience of 1-2 years, two of them had an experience of 2-3 years, one had an experience of 3-4 years and one had an experience of more than 5 years. The type of data that the respondents worked with included text, images, audio and biomedical signals. The frameworks used by the respondents included *Tensorflow*, *Pytorch*, *Theano*, *Caffe* and *Keras*. Half of the respondents admitted that they never think of energy efficiency when developing deep learning applications. Only 2 of the 14 respondents claimed to worry about energy efficiency every time they develop deep learning applications. This shows that there is a greater need for energy awareness in the developer community.

The agreement of the developers on whether the use of each of the patterns contribute to energy efficiency or not, is summarized in the Figure 7. The energy patterns in the graph are represented using their pattern numbers (P1 to P8) mentioned in Table 3. It can be seen that all the energy patterns other than P4 (*Quantization*) and P7 (*Memory leaks*) had over 78% of the respondents either "agree" or "strongly agree" that the use of the corresponding energy pattern can help reduce energy consumption of the deep learning development.

The patterns P4 and P7 had 71.4% and 64.3% of the developers respectively either "agree" or "strongly agree" on their usefulness. For pattern P4, one respondent was of the opinion that the *quantization* of the model is not preferred in most cases despite the energy benefits as it could cause a loss of accuracy. The use of a quantized model would first require its performance comparison with the non-quantized version to find out the quantization loss,

[8]https://anonymous.4open.science/r/ease_2022-835F/developer_survey.csv

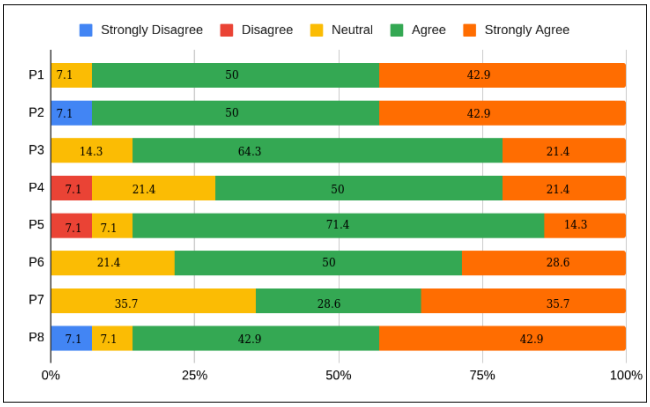

Figure 7: Results of the developer survey

which constitutes a significant overhead. For pattern P7, 35.7% of the developers had a neutral response. One of them commented that handling memory leaks before training is very much dependent on the developer's experience. Another respondent pointed out that there is a lesser risk of OOM errors when the input data is sent in batches for the training.

5 THREATS TO VALIDITY

In this section, we discuss potential systematic errors in our work that may pose threats to the outcome of our study.

5.1 Internal Validity

We performed a keyword based search of Stack Overflow posts as shown in the Table 2 to filter posts that may be related to energy. However, there is a possibility that we may have missed out on some relevant data that do not contain these keywords. We also used a keyword based search to find out posts that may be false positives, as described in the Section 2.2.3. In this method of false positive removal, there is a possibility that a relevant post containing both the valid keyword and false positive keyword would be lost from the final dataset. However, the number of such posts may be very small as it is not common for both the phrases to occur together. The dataset obtained after false positive removal may still have posts with false positives or posts that may not be relevant from an energy perspective. We argue that such posts are easily discarded during thematic analysis.

We considered only the Stack Overflow posts that have an accepted answer for our study. This was done to ensure that we only looked at the posts where the developers got an answer to their questions. It is possible that in many cases, the users may not mark the answers as accepted despite getting their queries resolved. While the relevant posts under this category will be missing from our analysis, we cannot guarantee their quality due to a lack of validity.

We used a method of manual thematic analysis as described in Section 2.3. Many of the questions were not asked with the intention of improving energy efficiency. But the authors looked at them from the lens of energy. Despite the best of our efforts, human errors are possible during this process. The bias of the authors may influence the energy patterns obtained. Some posts may have gotten misclassified or discarded due to the misinterpretation by the authors.

5.2 External Validity

The data analyzed in our study are Stack Overflow posts that are tagged with popular deep learning libraries and frameworks. These posts typically come from users that use these frameworks to build their deep learning applications. However, the methods for handling energy issues in deep learning projects that use other less popular frameworks may be different. The dataset contains posts until November 2021. Considering the fast paced nature of the field, our study may have missed out on any popular theme or practice that came after November 2021. Since Stack Overflow is only a Q&A platform, the posts may lack the context of the projects they originated from. This lack of context may also influence our results.

6 DISCUSSION

This paper is an attempt towards creating a catalog of energy patterns for deep learning development. We believe that this catalog might help the deep learning developers in making their development more energy efficient. The current version of the catalog is based on the analysis of Stack Overflow posts. This ensures that the patterns obtained are currently in use and being discussed by the developers. Few of the patterns such as *Pruning*, *Distillation* and *Quantization* are common themes present in the literature on energy efficient deep learning [2, 41, 52].

Although we believe that the catalog may be useful for the developers, the choice of using some of these patterns has to be exercised carefully based on their feasibility under the given circumstances. Using the patterns would require the developers to have a good understanding of the trade-offs involved. For instance, the use of *Checkpointing* pattern requires the parameters to be saved. The process of saving the parameters at regular intervals also contains energy overhead. Hence, the frequency of checkpointing has to be chosen such that the energy consumed on saving the model should not exceed the energy consumed on training. The pattern should be used only if it is feasible. Even though the model compression techniques like *pruning*, *distillation* and *quantization* may lead to efficient inferencing, they may at times also cause a minor loss in accuracy [36, 47]. In cases where high accuracy is very critical, it may not be feasible to use these techniques. Performing these compression techniques may themselves consume energy, after which it has to be compared with the original model. Only in the case where the compressed model meets the required accuracy criteria can it be used for inferencing. So exercising these patterns may require the developers to take a call on whether the energy saved during inferencing offsets the energy spent on compression. In the case of patterns like *Memory leaks* and *Efficient read-write* a significant level of experience may be required to apply them.

In some cases, the developers may have to decide on whether or not a pattern needs a strong focus. For example, regular checkpointing may prevent the loss of knowledge caused by OOM errors to some extent. Therefore, when the developers use *Checkpointing*, they need not focus too much on *Memory leaks*. However, if checkpointing comes with significant overheads to make it less

feasible, extra attention may be required towards handling memory leaks and OOM errors. In the *Tensor operations* pattern, we go by a general assumption that the libraries tend to have an optimized implementation of functions. But this may not always be the case as the efficiency of library functions is dependent on the team that develops and maintains them. However, chances of bad implementations may be rare in the case of popular deep learning libraries as they are backed by large organizations.

7 RELATED WORK

Given the energy intensive nature of deep learning models, researchers have been working on ways to make deep learning more energy efficient. Several hardware accelerators have been proposed to enable energy efficient implementation of deep learning models [7, 23, 29, 41]. Park et al. [41] proposed an accelerator that performs low precision computations for majority of the weights and activations using outlier aware quantization. Chen et al. [7] proposed *Eyeriss*, an accelerator for CNN which optimizes for the energy efficiency of the system by reconfiguring the architecture using row-stationary dataflow. Ko et al. [23] proposed an accelerator that trains CNNs with frequency domain computations. It lets the convolution operations get replaced with simple pointwise multiplication operations which reduces the training time and energy. Liang et al. [29] presented a specialized accelerator, *EnGN*, that enables high-throughput and energy-efficient processing of large-scale GNNs (Graph Neural Networks) using *ring-edge-reduce* update dataflow.

Some other methods include modification in the hardware to reduce the size of the model parameters used or modify the method of computations to make the networks more energy efficient [21, 26, 35, 48]. Lee et al. [26] proposed the use of logarithmic encoding of non uniform weights and activations to reduce the bit-width of the parameters. Jiao et al. [21] proposed a method for energy efficient neural network implementation that exploits computation reuse opportunities using bloom filters. Sarwar et al. [48] proposed use of multiplier-less neurons by replacing conventional multipliers with shift and add operations to achieve reduction in power consumption. Moons et al. [35] used approximate computing by exploiting the fault tolerance of CNNs to achieve better energy efficiency.

But the above solutions primarily focus on individual networks or modification of the hardware. While use of accelerators help in energy efficiency, they tend to be beyond the control of a software developer implementing deep learning applications using libraries and frameworks.

In domains such as mobile development, researchers have used the data from platforms like Stack Overflow and GitHub to find out energy efficient practices [37, 43], the energy awareness among the developers [40] in the community and come up with a catalog of energy efficient practices [10]. The surge in the popularity of deep learning has led the researchers to use these platforms to gain insights about the practices and the challenges faced by the developers. Zhang et al. [53] analyzed 715 posts from Stack overflow to understand the challenges faced by the developers while building deep learning applications. Islam et al. [20] analyzed 415 Stack Overflow posts and 555 GitHub repairs to understand the bug fix patterns and challenges in deep learning. Pan [39] analyzed 321 Stack Overflow posts to understand the impact of bug fixes on the robustness of the deep learning models. Zhang et al. [54] analyzed 175 bugs to study the characteristics of program bugs in deep learning applications built in *Tensorflow*. Chen et al. [8] analyzed a set of 304 posts from Stack overflow and GitHub to understand the deployment faults in mobile based deep learning applications. The type of discussions taking place on deep learning frameworks [16] and the study of technical debts [31] have also been performed using the data from these forums.

However, there has not been an attempt to leverage the data from online platforms to look at the deep learning development from energy perspective. To this end, our work study serves as a first step towards that direction.

8 CONCLUSION AND FUTURE WORK

In this paper, we presented an initial catalog of 8 energy patterns for deep learning development. We manually analyzed 1361 posts from Stack Overflow to come up with the energy patterns. This is the first catalog of energy patterns in deep learning. Our hope is that this catalog would serve as guidelines for deep learning developers to make their development energy efficient. A survey of 14 developers on their perception of these patterns and their usefulness is also included in our study. The survey results showed that the developers largely agree that the energy patterns in the catalog can help in making the deep learning development energy efficient. Application of some of the energy patterns obtained would require a strong understanding of the trade-offs, as discussed in Section 6. In subsequent studies, it would be interesting to work on including more concrete quantified recommendations for applying these patterns in the projects. We also believe that the energy patterns from the initial catalog can be further classified into more granular sub-patterns, and we plan to work in that direction in the future. The current version of the catalog is obtained by analyzing Stack Overflow posts. The *issues*, *commits* and *pull requests* from GitHub have also been used to form various catalogs in other domains. In the future, we plan to analyze GitHub artifacts from deep learning projects to extend our catalog.

REFERENCES

[1] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. 2020. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051* (2020).

[2] Amin Banitalebi-Dehkordi. 2021. Knowledge distillation for low-power object detection: A simple technique and its extensions for training compact models using unlabeled data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 769–778.

[3] Shabab Bazrafkan, Tudor Nedelcu, Pawel Filipczuk, and Peter Corcoran. 2017. Deep learning for facial expression recognition: A step closer to a smartphone that knows your moods. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 217–220.

[4] Nazanin Beheshti and Lennart Johnsson. 2020. Squeeze U-net: A memory and energy efficient image segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 364–365.

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[6] Ruichu Cai, Boyan Xu, Xiaoyan Yang, Zhenjie Zhang, Zijian Li, and Zhihao Liang. 2017. An encoder-decoder framework translating natural language to database queries. *arXiv preprint arXiv:1711.06061* (2017).

[7] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.

[8] Zhenpeng Chen, Huihan Yao, Yiling Lou, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, and Xuanzhe Liu. 2021. An empirical study on deployment faults of deep learning based mobile applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 674–685.

[9] Shaiful Alam Chowdhury and Abram Hindle. 2016. Characterizing energy-aware software projects: Are they different?. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 508–511.

[10] Luis Cruz and Rui Abreu. 2019. Catalog of energy patterns for mobile applications. *Empirical Software Engineering* 24, 4 (2019), 2209–2235.

[11] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. 2013. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*. 2148–2156.

[12] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. 2018. Gradient Descent Provably Optimizes Over-parameterized Neural Networks. In *International Conference on Learning Representations*.

[13] Xiaoting Du, Guanping Xiao, and Yulei Sui. 2020. Fault triggers in the tensorflow framework: An experience report. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 1–12.

[14] Ottmar Edenhofer. 2015. *Climate change 2014: mitigation of climate change*. Vol. 3. Cambridge University Press.

[15] Jennifer Fereday and Eimear Muir-Cochrane. 2006. Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International journal of qualitative methods* 5, 1 (2006), 80–92.

[16] Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. 2020. What do programmers discuss about deep learning frameworks. *Empirical Software Engineering* 25, 4 (2020), 2694–2747.

[17] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[20] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. 2020. Repairing deep neural networks: Fix patterns and challenges. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1135–1146.

[21] Xun Jiao, Vahideh Akhlaghi, Yu Jiang, and Rajesh K Gupta. 2018. Energy-efficient neural networks using approximate computation reuse. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1223–1228.

[22] Gokcen Kestor, Roberto Gioiosa, Darren J Kerbyson, and Adolfy Hoisie. 2013. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 56–65.

[23] Jong Hwan Ko, Burhan Mudassar, Taesik Na, and Saibal Mukhopadhyay. 2017. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[25] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. 2015. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*. 283–294.

[26] Edward H Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and S Simon Wong. 2017. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5900–5904.

[27] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on foundations of software engineering*. 926–931.

[28] Xueliang Li, Yuming Yang, Yepang Liu, John P Gallagher, and Kaishun Wu. 2020. Detecting and diagnosing energy issues for mobile applications. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 115–127.

[29] Shengwen Liang, Ying Wang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. 2020. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Trans. Comput.* (2020).

[30] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*. PMLR, 2849–2858.

[31] Jiakun Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping Li. 2021. An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks. *Empirical Software Engineering* 26, 2 (2021), 1–36.

[32] Haroon Malik, Peng Zhao, and Michael Godfrey. 2015. Going green: An exploratory analysis of energy-related questions. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 418–421.

[33] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Haşim Sak, Alexander Gruenstein, Françoise Beaufays, et al. 2016. Personalized speech recognition on mobile devices. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5955–5959.

[34] Gaurav Mittal, Kaushal B Yagnik, Mohit Garg, and Narayanan C Krishnan. 2016. Spotgarbage: smartphone app to detect garbage using deep learning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 940–945.

[35] Bert Moons, Bert De Brabandere, Luc Van Gool, and Marian Verhelst. 2016. Energy-efficient convnets through approximate computing. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 1–8.

[36] Bert Moons, Koen Goetschalckx, Nick Van Berckelaer, and Marian Verhelst. 2017. Minimum energy quantized neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 1921–1925.

[37] Irineu Moura, Gustavo Pinto, Felipe Ebert, and Fernando Castor. 2015. Mining energy-aware commits. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 56–67.

[38] Bogdan Nicolae, Jiali Li, Justin M Wozniak, George Bosilca, Matthieu Dorier, and Franck Cappello. 2020. Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 172–181.

[39] Rangeet Pan. 2020. Does fixing bug increase robustness in deep learning?. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 146–148.

[40] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E Hassan. 2015. What do programmers know about software energy consumption? *IEEE Software* 33, 3 (2015), 83–89.

[41] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. 2018. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 688–698.

[42] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. 2011. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. 1–6.

[43] Gustavo Pinto, Fernando Castor, and Yu David Liu. 2014. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. 22–31.

[44] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[46] Hojjat Salehinejad and Shahrokh Valaee. 2021. Edropout: Energy-based dropout and pruning of deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[47] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[48] Syed Shakib Sarwar, Swagath Venkataramani, Aayush Ankit, Anand Raghunathan, and Kaushik Roy. 2018. Energy-efficient neural computing with approximate multipliers. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 14, 2 (2018), 1–23.

[49] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A survey on deep transfer learning. In *International conference on artificial neural networks*. Springer, 270–279.

[50] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[51] Mingyuan Xia, Wenbo He, Xue Liu, and Jie Liu. 2013. Why application errors drain battery easily? A study of memory leaks in smartphone apps. In *Proceedings of the Workshop on Power-Aware Computing and Systems*. 1–5.

[52] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 6071–6079.

[53] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. 2019. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 104–115.

[54] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on TensorFlow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 129–140.