Multi-Task LLMs: What Can Pruning Tell Us?

Andrew Kettle (Lead)
kettlea@uci.edu

Abhinand Ganesh
aganesh5@uci.edu

Brianna Gutama
gautamb1@uci.edu

Diya Saha
sahad1@uci.edu

***Abstract*—This project explores multi-task pruning on BERT. In this report, we explore the prior work, methodology, and results that arose from pruning a fine-tuned version of BERT for different tasks.**

I. Introduction

Large Language Models (LLMs) have shown exceptional performance across various modalities such as text, vision, and audio. Like many breakthroughs in Machine Learning (ML), LLMs gained prominence through empirical findings. This naturally leads to a lack of explainability of why LLMs are able to outperform other models so comprehensively. Chief among these breakthroughs is LLM's capability to excel in multi-task learning, where the same pretrained architecture is fine tuned for different downstream tasks. Our goal is simple, we are exploring whether feature representation structure appears when pruning a specific LLM, BERT. We elected to prune because pruning helps reveal the most important weights for a particular layers feature representation by removing weights that don't contribute significantly. We expect to see the feature representations change significantly and be correlated with a drop off in accuracy on a per task basis. We are not sure if any structure will generalize to multiple tasks. We plan to use PyTorch for our experiments and evaluate the different feature representations using accuracy and Central Kernel Alignment (CKA) [1]. Code for the project can be found here: Github Repository.

II. Prior Work

A. BERT and Multi-Task Learning

The BERT (Bidirectional Encoder Representations from Transformers) model [4] is a pre-trained language representation model that is capable of understanding and representing contextual information in text, capturing intricate semantic relationships. The BERT model has been applied across various NLP tasks, including sentiment analysis, question answering, and many more, and has attained significant performance improvements across these tasks. [3] introduces a novel approach to compress multi-task LLMs in a task-specific manner. The approach tailors pruning to each specific task, by using importance scores to estimate the contributions of different model components to the task performance. Thus, this allows efficient pruning of less important components without affecting the overall model performance across multiple tasks.

B. Model Analysis with Centered Kernel Alignment (CKA)

Centered Kernel Alignment (CKA) is a statistical technique used to measure the similarity between different representations of data, such as those produced by neural networks or other machine learning models. It's useful for comparing the internal representations of different layers or different models.

In [6], Raghu. et al use Centered Kernel Alignment (CKA) to compare internal representations of Vision Transformers (ViTs) and Convolutional Neural Networks (CNNs). They observed that CKA reveals that ViTs exhibit more uniform representation structures across layers compared to CNNs, which show distinct stages. They observed that ViTs also integrate global information from early layers, unlike CNNs, which focus on local information initially.

This CKA analysis helped them understand how two networks differ in their approach to aggregating spatial information, influencing their effectiveness in various visual tasks.

C. Representations

Feature representations are essential to Machine Learning. At first, these features were hand selected, such as early edge detection algorithms in vision [7]. Lately, there has been a shift towards automatically learning important features, such as with neural networks. For a problem like image classification, we can view the earlier layers of the model as a feature extractor, with the final classification layer simply classifying over these features. Some network's representations are well known through empirical study [8]. For example, CNN's features tend to increase in complexity as the network gets deeper and the receptive field gets wider. An entire field of machine learning, representation learning, is dedicated to studying various feature representations. One of the most prominent examples of multi-task feature representations in modern literature is CLIP [9], which combines image and text features in the same embedding space to reason about images using language. Given LLMs trend towards performing many downstream tasks, this area of research seems to only be in its infancy.

III. Approach

A. Task Selection

For our study, we collected data for three distinct natural language processing tasks: SST-2 (Sentiment Analysis), QQP (Question Pairs), and QNLI (Question Natural Language Inference). Each task required a specific dataset tailored to its respective objective. The SST-2 dataset contains movie review sentences labeled with binary sentiment labels, which denote positive or negative sentiment. QQP involves pairs of questions along with a label indicating whether the questions are semantically equivalent or not. Lastly, QNLI comprises question-sentence pairs where the task is to determine whether the sentence contains the answer to the question. To prepare the data for model training, we tokenized the input sentences and questions using the BERT tokenizer, ensuring consistent padding and truncation for uniform input length across batches. Subsequently, the datasets were partitioned into training and validation sets, with appropriate batch sizes configured for efficient model training and evaluation. Our work is built upon the tasks and methodologies outlined in the GLUE benchmark, which provides a collection of tools for evaluating the performance of models across a diverse set of existing NLU tasks. By including tasks with limited training data, GLUE encourages the development of models that share general linguistic knowledge across tasks. The benchmark also includes a diagnostic test suite for detailed linguistic analysis of models [2].

B. Customizing the BERT Model

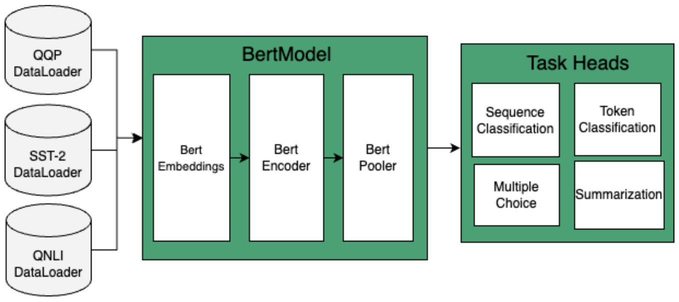

Fig. 1: Architecture of custom BERT

For our tasks, we implemented a custom BERT-based model tailored to handle various natural language processing tasks. The model architecture, referred to as BertCustomHead, extends the pre-trained BERT model with task-specific heads for different tasks. Specifically, it supports tasks such as sequence classification, token classification, multiple choice, and summarization.

By using BERT's strong contextual embeddings, the model can adapt to different tasks with minimal adjustments. Each task-specific head can be tailored to the requirements of its respective task without altering the core BERT structure. This is described in Fig. 1.

The core BERT model serves as a feature extractor, processing input sequences and producing contextualized representations. These representations are then fed into task-specific linear layers, which are responsible for generating task-specific outputs. For instance, in the case of sequence classification tasks like SST-2 sentiment analysis, the final hidden state of BERT is passed through a linear layer to produce class probabilities. This custom approach allows us to seamlessly adapt BERT to diverse NLP tasks while maintaining the benefits of pre-trained contextualized embeddings.

The "Sequence Classification" task assigns a single label to an entire sequence. For implementing the task head, the final hidden state of the BERT model is passed through a linear layer, which produces class probabilities using a softmax function. We use the cross-entropy loss as the loss function for optimization.

C. Pruning Techniques

1) Unstructured Pruning: Unstructured pruning is a technique for removing the least influential weights from a neural network. The name unstructured comes from the fact that only a single global rule is followed for selecting weights, such as the L1 or L2 norm. In our implementation of global unstructured pruning, we prune out encoder weights using the global L1 norm. Our choice of pruning only the encoder weights ties back to our introduction. If we view the BERT [4] encoder as a highly complex feature extractor, we can then view removing less important weights as trimming the unnecessary information used to produce the feature representations. It is important to note that existing hardware is incapable of taking advantage of this sparsity from a computational perspective.

2) Attribution-based Pruning: Task-specific attribution-based pruning [3] is a structured training-free pruning approach that uses an attribution method to calculate the importance of neurons for a specific task and prunes the less important neurons in a task-specific manner. Since it is a training-free approach, fine-tuning is not required, thereby reducing the need for computing resources and a large training dataset, while preserving pre-trained knowledge in language models.

Attribution calculation is done by performing a forward pass to obtain the network's outputs and a backward pass to calculate the gradients of the loss with respect to each neuron's activation. The attribution scores of each neuron are aggregated over multiple inputs, i.e. the entire training dataset, to indicate the importance of each neuron for the given task. Neurons with higher attribution scores are considered more important for the task.

Attribution formula:

$$A_i^{(x,y_j)}(h) = h_i \times \frac{\partial P(y_j \mid x, y_{1:j-1})}{\partial h_i}$$

where $A_i^{(x,y_j)}(h)$ is the attribution score of neuron i in layer h, and $\frac{\partial P(y_j|x,y_{1:j-1})}{\partial h_i}$ is the gradient of $P(y_j \mid x, y_{1:j-1})$ with respect to the i-th neuron. [3]

After the attribution scores are calculated over the dataset, the pruning process is carried out in a layer-wise manner, for all layers of the encoder. For each layer, neurons are ranked based on their importance scores, and a predefined pruning rate determines the proportion of neurons to be pruned. Neurons with the lowest importance scores are removed. This task-specific pruning ensures that the resulting model is optimized for the specific task it was pruned for, maintaining high performance while reducing the number of parameters. By focusing on the neurons that are most critical to the task, this method achieves significant compression without compromising the model's effectiveness.

D. Centered Kernel Alignment (CKA)

Centered Kernel Alignment [1] is used to measure the feature similarity of two networks to one another. Unlike validation accuracy, it provides rich information about how different layers' representations compare to each other. CKA is based on the Hilbert-Schmidt Interdependence Criterion (HSIC) [10] which compares two distributions to one another. We used the PyTorch_CKA package [11] modified to work with BERT encoder inputs and encoder layers. We track this specific version of PyTorch_CKA as PyTorch-Model-Compare in our github repository.

IV. Analysis

A. Unstructured Pruning

The SST heatmaps can be found in Figure 4. From the heatmaps, we can see that SST is remarkably robust to pruning. The feature representation only changes minimally even at 50% encoder sparsity. Additionally, we can see that earlier encoders are pruned more heavily, suggesting that the mid-depth encoders (5-8) are essential for performing the SST task. At about 80% sparsity, the feature representation is lost and the accuracy degrades significantly, evidenced by the dark puple and black regions. One interesting thing to note is that all of the pruned network's layers become similar to the baseline model's first layer after 80% sparsity. This suggests that the earliest encoder in the baseline is already sparse.

For unstructured pruning on QQP, in Fig. 6, we observe that with increase in pruning strength, the deeper layers start becoming more dissimilar with the baseline model. This could suggest that these layers are more unimportant and don't preserve weights easily with pruning. On the contrary, the initial layers have consistent high similarity with the baseline model, indicating their importance and the fact that preserving the integrity of lower layers becomes crucial for maintaining functional similarity to the baseline model.

On the other hand for unstructured pruning on QNLI, in Fig. 8, we observe that the deeper and lower layers become increasingly dissimilar with the baseline model as the pruning strength increases. However, the mid-level encoder layers still preserve their weights to some extent (at least until the pruning strength reaches 0.6). This is because mid-level encoders in neural networks often capture semantically rich representations that balance between low-level features and high-level abstractions. The increased similarity among mid-level encoders indicates that these layers might contain more robust and generalizable information compared to other layers.

B. Attribution-based Pruning

For attribution-based pruning on the SST-2 dataset, we can observe from Fig. 3 that the lower layers was able to retain its similarity to its equivalent layers baseline unpruned model up to a sparsity of 0.7. On the other hand, the deeper layers started showing low similarity at a sparsity of 0.3, as shown by the darker color in the heatmap for deeper layers. This suggests that the lower layers are more resistant to structured pruning, and deeper layers are sensitive to attribution-based pruning. Combining this observation to the accuracy results in Fig. 2, we can deduce that the deeper layers are more essential in model performance for the SST-2 task, and lower layers are less important, since the accuracy starts to decline rapidly at low sparsity levels (approximately 0.2).

From Fig. 5, we can observe that the heatmaps show darker color with increasing pruning strength, which suggests increasing dissimilarity to the baseline model. For lower sparsity levels (up to approximately 0.5), the similarity between corresponding layers of the pruned and unpruned model degrades in a relatively consistent rate, as shown by the similar color across the diagonal of the heatmap. However, starting from about 0.7 sparsity, the similarity of lower layers becomes very low. Referencing the accuracy plot in Fig. 2, the accuracy remained relatively constant for lower pruning strengths and starts declining rapidly from sparsity levels of 0.6. These results suggest that for the QQP task, the model is relatively resistant to structured pruning since it retained its high accuracy at lower sparsity levels. Additionally, lower layers are more important in QQP task performance since accuracy degradation begins when dissimilarity becomes significant in lower layers.

For the QNLI task, when we conducted the attribution-based pruning, we observed some distinct patterns at different pruning strengths. During the initial strengths (0.1-0.4) as seen in Fig. 7, the layers were similar to that of the baseline model. This observation suggests that the pruned weights did not hold much significance for the QNLI task. As the pruning strength increased beyond 0.4, the layers were pruned at a higher rate. At a pruning strength of 0.7-0.9, the graph appeared almost completely dark, indicating that most of the neurons were heavily pruned, and the model becomes highly dissimilar to the baseline model, especially the lower and middle layers. This trend highlights that as we increased the pruning strength, the model's capacity to retain important neurons diminished. Referencing the accuracy plot in Fig. 2, the accuracy starts to decline rapidly at approximately 40% sparsity level, which was in line with the beginning of decline in similarity for the lower layers, as shown in the heatmap, underscoring the importance of the initial layers in the model performance for the QNLI task.

C. Accuracies

As seen in Fig.2, attribution pruning performs better for QNLI and QQP, while unstructured pruning performs better for SST-2. QNLI (Question Natural Language Inference) and QQP

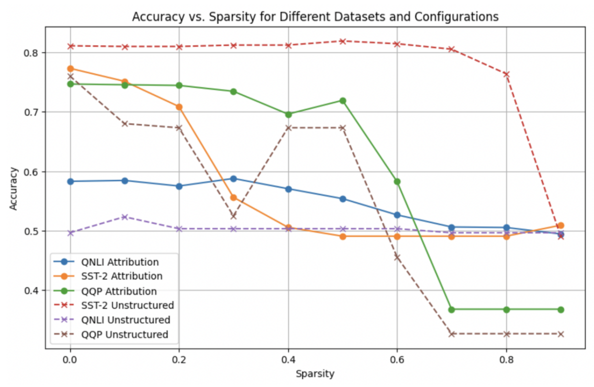

Fig. 2: Accuracies of Different Models

(Quora Question Pairs) are both tasks that require understanding relationships between pairs of sentences. They involve tasks like determining semantic similarity or entailment. This could mean that for these tasks, the model's performance is more sensitive to specific important features, making attribution pruning, which focuses on preserving key influential weights, more effective. SST-2 involves sentiment classification, which might depend on broader, less sparse patterns (such as overall sentence structure or common sentiment-indicative words) that are less disrupted by unstructured pruning.

V. Methodology

The methodology for this project involved several key steps to ensure a thorough understanding and effective implementation of multi-task learning models. Below is a detailed breakdown of our process:

- **Literature Review:** We began with an extensive review of existing literature to understand the current state of multi-task models and their applications.
- **Designing Task Heads:** Based on our literature review, we designed custom task heads suitable for different types of tasks, such as sequence classification, token classification, multiple-choice, and summarization.
- **Selecting Tasks:** We carefully selected three tasks: SST-2 (Sentiment Analysis), QQP (Question Pairs), and QNLI (Question Natural Language Inference).
- **Data Pre-processing and Cleaning:** For each task, we pre-processed and cleaned the datasets to ensure high-quality input data. This involved tokenizing the text, handling missing values, and standardizing the format of the datasets.
- **Creating DataLoaders:** We created DataLoaders for each task, ensuring that the shapes of the DataLoaders matched the input requirements of the BERT model.
- **Fine-tuning BERT on Selected Tasks:** We fine-tuned the BERT model on the selected tasks, adjusting the model's weights based on the specific characteristics and requirements of each task. This fine-tuning process was essential to optimize the model's performance on diverse tasks.
- **Implementing Pruning Techniques:** After fine-tuning, we applied unstructured pruning and attribution pruning to the model. Pruning involves removing less important weights from the model to reduce its size and improve computational efficiency without significantly affecting performance.
- **Analyzing Pruning Using CKA:** To evaluate the impact of pruning, we used Centered Kernel Alignment (CKA) to analyze the similarity between the original and pruned models. CKA provided insights into how pruning affected the internal representations and performance of the model.

VI. Conclusion

Our goal was to explore structure in BERT's encoders through pruning. To that end, we were successfully able to observe structure using CKA. Different tasks showed different feature representations. This is not surprising given that each model performs an entirely different task on a different dataset. In fact, it shows how critical fine tuning is for a model to extract relevant features for its task. Unfortunately, generalizing this structure across tasks was not trivial, and no clear patterns emerged for all tasks. We leave it as future work to explore how one might generalize feature representation findings across different tasks and possibly even different architectures.

VII. Individual Contributions

A. *Andrew Kettle (Lead)*

My contribution to this project centered around guiding the team through the research and implementation process. I identified the problem statement and created a plan to divide the work up amongst each member. My personal contribution was developing code for transitioning between HuggingFace and PyTorch as well as the unstructured pruning in PyTorch. Additionally, I performed research to identify CKA as an evaluation metric and then modified the torch_cka package to work with BERT inputs and encoder layers to create the heatmaps. I then stacked those heatmaps to look at the pruning performance over time. Finally, I ran the SST-2 unstructured pruning task to create the heatmap gif for SST2 unstructured.

B. *Abhinand Ganesh*

My contributions first included researching about how different BERT models cater to different tasks. I used that information to code the 'BertCustomHead' class and adding task heads for different tasks. I obtained the QQP and SST-2 data loaders from Diya, fine-tuned the BERT models on these datasets, and uploaded them to Google Drive for easy access by the team. I also performed unstructured pruning with various pruning strengths on the BERT models fine-tuned on the QQP and QNLI datasets. After pruning, I used torch_cka to generate heatmaps for comparing the pruned models with the baseline. Additionally, I collected accuracy values from my teammates and created Figure 2, which presents the accuracy plot for all models. I assisted in analyzing the heatmaps

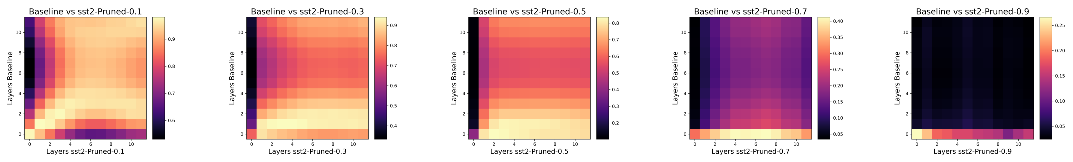

Fig. 3: Attribution Pruning with SST (p=0.1, 0.3, 0.5, 0.7, 0.9)

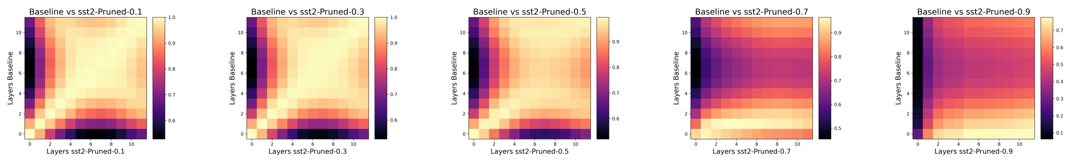

Fig. 4: Unstructured Pruning with SST (p=0.1, 0.3, 0.5, 0.7, 0.9)

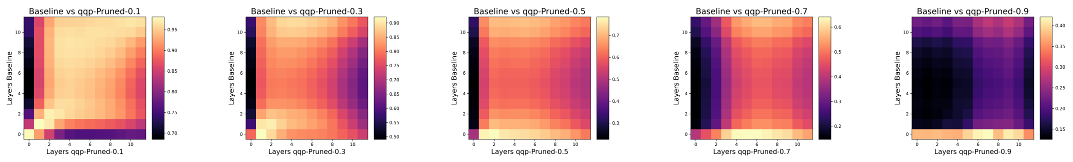

Fig. 5: Attribution Pruning with QQP (p=0.1, 0.3, 0.5, 0.7, 0.9)

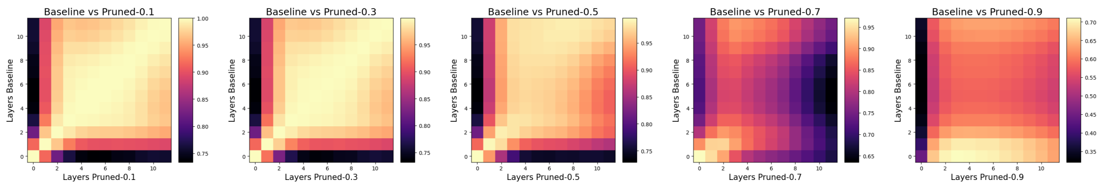

Fig. 6: Unstructured Pruning with QQP (p=0.1, 0.3, 0.5, 0.7, 0.9)

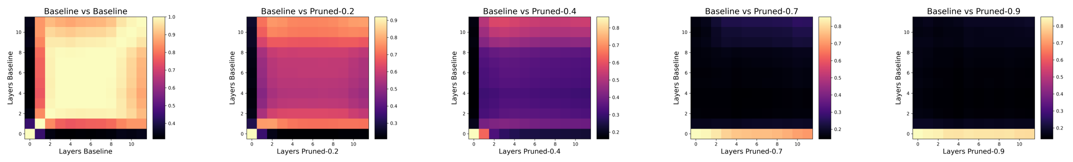

Fig. 7: Attribution Pruning with QNLI (p=0.1, 0.3, 0.5, 0.7, 0.9)

and accuracy plots and contributed to the project report and presentation.

C. *Brianna Gautama*

My contribution was primarily on implementing the attribution-based pruning method, and in generating structured pruning evaluation results. I researched on structured pruning methods for LLMs and implemented the attribution-based structured pruning method. Since there is no existing open-source implementation for attribution-based pruning, I wrote the code according to the algorithm described in the paper, and applied it on the models fine-tuned for the SST-2 and QQP tasks. Afterwards, I generated accuracy and CKA results to evaluate the effects of attribution-based pruning on the model's task performance.

D. *Diya Saha*

My contributions encompassed extensive research into the functionality of multitask models and the customization of task-specific heads for optimizing the BERT model's performance across various tasks. I played a pivotal role in identifying a diverse set of tasks suitable for our project's objectives. Subsequently, I undertook the responsibility of developing robust data loaders tailored to each selected task, which were subsequently subjected to pruning methodologies devised by Andrew and Brianna. I actively participated in fine-tuning the QNLI and SST tasks using the custom BERT model developed by Abhinand. Additionally, I contributed to conducting experiments focused on attribution pruning for the QNLI task. Furthermore, I took charge of creating the project presentation and provided substantial support in compiling the final report.

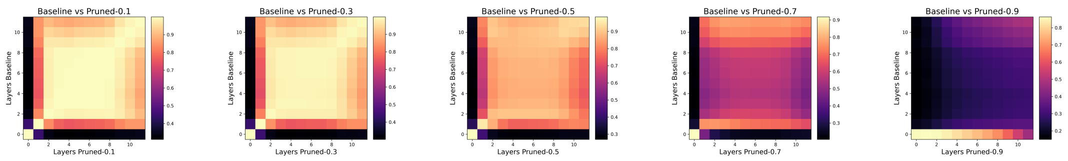

Fig. 8: Unstructured Pruning with QNLI (p=0.1, 0.3, 0.5, 0.7, 0.9)

References

[1] Similarity of neural network representations revisited. (n.d.). https://arxiv.org/pdf/1905.00414

[2] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding,” arXiv preprint arXiv:1804.07461, 2018. Available: https://openreview.net/pdf?id=rJ4km2R5t7.

[3] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[4] N. Yang, Y. Jang, H. Lee, S. Jung, and K. Jung,“Task-specific Compression for Multi-task Language Models using Attribution-based Pruning,” Findings of the Association for Computational Linguistics: EACL 2023, pages 582–592., Association for Computational Linguistics, 2023.

[5] J. Devlin, M. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” arXiv, 2019, arXiv:1810.04805.

[6] Raghu, Maithra, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. ”Do Vision Transformers See Like Convolutional Neural Networks?” arXiv, 2022, arXiv:2108.08810.

[7] J. Canny, ”A Computational Approach to Edge Detection,” *Pattern Recognition*, 1986, https://www.sciencedirect.com/science/article/abs/pii/S0031320300000236

[8] Google DeepDream, https://github.com/google/deepdream

[9] Radford et al., ”Learning Transferable Visual Models From Natural Language Supervision,” arXiv, 2021, https://arxiv.org/abs/2103.00020

[10] J. Johnson, ”HSIC: Hilbert-Schmidt Independence Criterion,” https://jejjohnson.github.io/research_journal/appendix/similarity/hsic/

[11] A. Subramanian, ”torch_cka,” 2021, https://github.com/AntixK/PyTorch-Model-Compare