CS598 Project Write-Up: Predicting Drug Sensitivity of Cancer Cell Lines via Collaborative Filtering with Contextual Attention

Arjun Nair[1*] **Jimmy Shong**[1*],

[1]University of Illinois Urbana-Champaign

Abstract

Finding the right anti-cancer drugs and key biomarkers is essential for precision oncology. Testing how cancer cell lines respond to different drugs helps us understand how these drugs interact with cancer cells. The original paper ”Predicting Drug Sensitivity of Cancer Cell Lines via Collaborative Filtering with Contextual Attention” introduces the CADRE model, an architecture for assessing the response of cancer cell lines based on various omic profiles. In this project, we reproduce and evaluate the CADRE model on the GDSC dataset. We also perform several ablations and extensions to understand the model's intricacies better. Using insights from these experiments, we finally train and present a final version of the model that outperforms the original CADRE model. Our code is in this repository: `https://github.com/Jiminator/CADRE`. We present our findings in this short video: `https://shorturl.at/tlyhl`.

1 Introduction

1.1 Problem Statement

One target in precision oncology is using Omics data of cell lines to predict their sensitivities to a panel of potential molecules. However, this is challenging for three reasons.

First, drug sensitivity data are noisy and often contain many missing entries. Consequently, it is essential that the model can generalize well and not overfit the training data.

Second, the relationship between the molecular profiles of cell lines and drugs is complex. Not all genes contribute equally to the response; different genes can influence drug response to varying degrees. Moreover, the interaction of particular genes can create contextual effects, where a gene's influence on drug response depends on the presence or activity of different genes.

Third, it is essential to predict drug sensitivities, and models must also be explainable to help researchers better understand what is going on. While many modern deep-learning methodologies can achieve impressive results in predicting drug sensitivities, they behave like black boxes where it is difficult to explain the outputs.

*These authors contributed equally.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

1.2 CADRE

Researchers at Carnegie Mellon and the University of Pittsburgh introduced CADRE (Tao et al. 2020). CADRE's high-level approach was to build upon a collaborative filtering mechanism (Schafer et al. 2007). Collaborative filtering predicts a user's preferences based on the preferences of other users with similar behaviors. Within the medical domain, it can be used to predict how a patient will respond to a drug based on similar patients' responses. The researchers of the paper developed two attention-based collaborative filtering models in this paper: SADRE and CADRE.

Given a cell line and a drug, the CADRE approach uses a self-attention (Yang et al. 2016) mechanism to create an embedding vector to model the cell line. The attention mechanism allows the model to assign appropriate importance to genes composing the cell line to generate the cell line's embedding vector. Finally, the model passes the inner product through a logistic function to predict the probability that the cell line will be sensitive to the drug.

Figure 1 provides a high-level overview of the CADRE architecture. For a more detailed discussion of the CADRE model, please refer to section 2.3.

1.3 Scope of Reproducibility

The authors have already provided us with the code for the experiments and the preprocessed GDSC dataset. The other dataset (CCLE) was not available. We could reproduce the CADRE model and produce results similar to those in the papers. However, we noticed minor discrepancies between the numbers we found during our training runs and the reported numbers. These differences can be attributed to the lack of random seeding in the original codebase and the accumulation of inaccuracies due to floating point precision.

1.4 Contributions

In addition to reproducing the model, we performed an ablation study on it. Specifically, we examined the effects of CADRE's contextual attention and how it compares with standard self-attention. We also looked at more fine-grained model details, such as the dropout layer, the OneCycle scheduler (Smith 2018), the pre-trained gene embeddings, and the ReLU activation function. We then implemented a few extensions to improve the model's performance. We

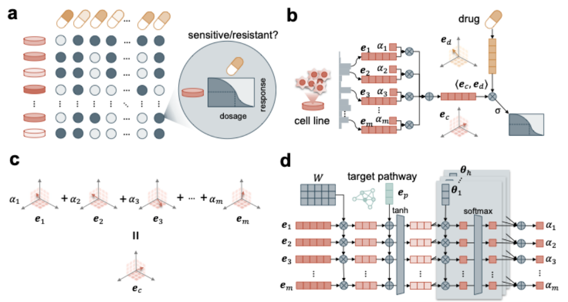

Figure 1: Overview of the CADRE model. (a) CADRE predicts cancer cell line responses to anti-cancer drugs based on gene expression. (b) For a cell line c and drug d, CADRE extracts gene embeddings $e_1, \ldots, e_m$ and a drug embedding e_d, computes a cell line embedding e_c via an attention-weighted sum, and predicts sensitivity as $\sigma(e_c^\top e_d)$. (c) The cell line embedding e_c is formed by aggregating gene embeddings using learned attention weights. (d) Attention weights $\alpha_1, \ldots, \alpha_m$ are computed via a contextual attention network conditioned on gene embeddings and the drug pathway embedding e_p.

first added focal loss (Lin et al. 2018) to the binary cross-entropy loss used in the original model, and we noticed some marginal improvements when we tuned the alpha parameters. We also tried training with AdamW optimizer (Loshchilov and Hutter 2019) alongside the cosine annealing learning scheduler (Loshchilov and Hutter 2017), as it is currently a popular option in training modern ML models compared to the SGD optimizer used in CADRE paper. While we did not observe improvements with AdamW, we did see that the model performed better when the SGD optimizer was paired with a cosine scheduler instead of the original OneCycle scheduler. We tried other model architecture adjustments, such as adding an MLP to the decoder to add more expressiveness and LayerNorms to the encoder with residual connections (He et al. 2015) to mimic the PreLN transformers (Xiong et al. 2020) being used in large foundational models in production today (Brown et al. 2020). Unfortunately, none of these experiments yielded performance gains except for the residual connections. We then trained a final version of the CADRE model with model architecture changes informed by the extensions and ablation experiment results. This led to a slight 3% improvement in F1 score over the author's model.

2 Methodology

2.1 Environment

To create this paper, we used Python version 3.11.12. This project involved various dependencies. We used Numpy, Pandas, scikit-learn, PyTorch, Matplotlib, and other standard libraries that come preinstalled with Python.

2.2 Data

Please visit the following repository: `https://github.com/Jiminator/CADRE`. Then, download the CADRE.ipynb file. Upload this file to Google Colab or your environment of choice, and you can run the file from there. The preprocessed dataset is already stored in the Github repository. The GDSC dataset includes response data between hundreds of cell lines and anti-cancer drugs. The researchers used the transcriptome data (profile of approximately 20,000 genes) as part of the research. Table 1 provides more details about the dataset.

Dataset	# cell lines	# drugs ($\lvert\mathcal{D}\rvert$)	# pathways ($\lvert\mathcal{P}\rvert$)	% missing	% positive
GDSC	846	260	25	18.2%	32.2%

Table 1: Statistics of the GDSC drug response dataset used in the work.

2.3 Model

The authors presented the CADRE (Tao et al. 2020) paper, along with a code repository that can be found at the following link: `https://github.com/yifengtao/CADRE`.

2.3.1 Collaborative Filtering We will first discuss the collaborative filtering mechanism that forms the architecture of CADRE. Given a cell line *c* and a drug *d*, CADRE will map c and d to respective characteristic vectors $e_c, e_d \in \mathbb{R}^s$. The inner product of these two vectors will be passed as input to the logistic function, which computes the probability that cell line c will be sensitive to drug d.

$$\hat{y}_{c,d} = \sigma\left(\langle e_c, e_d \rangle\right) = \frac{1}{1 + \exp(-e_c^\top e_d)} \tag{1}$$

Given that $\mathcal{W}$ represents all the trainable parameters, we want to optimize the loss function given by (2). We use the cross-entropy loss between the predicted sensitivity $\hat{y}_{c,d}$ and the true sensitivity $y_{c,d}$.

$$\begin{aligned} \ell(\hat{y}_{c,d}, y_{c,d}; \mathcal{W}) = &-y_{c,d} \cdot \log \hat{y}_{c,d} \\ &- (1 - y_{c,d}) \cdot \log(1 - \hat{y}_{c,d}) \\ &+ \frac{\lambda_2}{2} \cdot \ell_2(\mathcal{W}) \end{aligned} \tag{2}$$

To obtain the drug embedding, the researchers used a lookup table that assigned each drug to its corresponding embedding. To determine the cell line's embedding, the researchers considered the $m = 1,500$ genes that were the most expressed. These genes were then mapped to their embedding with a gene-to-gene embedding lookup table and used to calculate the cell line embedding by naively taking the mean. The dropout layer in (4) zeroes out weights randomly to prevent the model from overfitting to the training set.

$$e_c = \sum_{i=1}^{m} 1 \cdot e_i = 1 \cdot e_1 + 1 \cdot e_2 + \ldots + 1 \cdot e_m. \tag{3}$$

$$e_c = \text{dropout}(e_c; \rho), \tag{4}$$

2.3.2 SADRE The issue with the naive collaborative filtering mechanism is that it assigns the same weight to each gene. The researchers wanted the model to be able to assign different levels of importance to different genes when aggregating the values for a cell line embedding. The formulation is given in (5).

$$e_c = \sum_{i=1}^{m} \alpha_i \cdot e_i = \alpha_1 \cdot e_1 + \alpha_2 \cdot e_2 + \ldots + \alpha_m \cdot e_m. \quad (5)$$

To calculate these α values, the authors first tested a standard multiheaded self-attention computation, incorporating the contextual impact of other expressed genes with a sub-neural network. The self-attention mechanism first calculates unnormalized attention weights from training parameters $W \in \mathbb{R}^{q \times s}, \theta_j \in \mathbb{R}^q$, and then normalizes them with softmax, shown in (6).

$$\begin{aligned} \alpha_{i,j} &= \frac{\exp\left(\boldsymbol{\theta}_j^\top \tanh(We_i)\right)}{\sum_{k=1}^{m} \exp\left(\boldsymbol{\theta}_j^\top \tanh(We_k)\right)} \\ i &= 1, \ldots, m \\ j &= 1, \ldots, h \end{aligned} \quad (6)$$

Finally, the model sums these values across the head dimension to get the final attention weights.

$$\alpha_i = \sum_{j=1}^{h} \alpha_{i,j} = \alpha_{i,1} + \ldots + \alpha_{i,h} \quad i = 1, \ldots, m. \quad (7)$$

2.3.3 CADRE The final CADRE model further builds upon SADRE, which uses gene embeddings to calculate attention weights. CADRE's contextual attention mechanism introduces the drug targets into the attention computation. Specifically, the CADRE model will take an additional parameter $e_p \in \mathbb{R}^q$ that models the target pathway embeddings to produce the attention weights α_i. Like e_c, e_p is mapped using a lookup table and represents the functional similarities of drugs, with drugs sharing the same target having similar target embeddings. Computationally, the attention mechanism is quite similar to SADRE, with just the introduction of the pathway embeddings e_p. We show the formulation in (8).

$$\begin{aligned} \alpha_{i,j} &= \frac{\exp\left(\boldsymbol{\theta}_j^\top \tanh(We_i + e_p)\right)}{\sum_{k=1}^{m} \exp\left(\boldsymbol{\theta}_j^\top \tanh(We_k + e_p)\right)} \\ i &= 1, \ldots, m \\ j &= 1, \ldots, h \end{aligned} \quad (8)$$

2.4 Training

2.4.1 Hyperparameters There were various hyperparameters, such as training batch size, training steps, maximum learning rate, etc, that the authors tuned. When recreating the work, we followed the authors' experiments as closely as possible. We used a learning rate $\eta = 0.3$, a batch size 8, and a dropout probability $\rho = 0.6$.

2.4.2 Computational Requirements For most of our experiments, we used an A40 GPU on NCSA Delta, but we also created a notebook compatible with Google Colab. Each training run used three trials with fixed seeds to maximize reproducibility. We trained the model with 48,000 training iterations, reaching 71 epochs.

2.4.3 Training Details We used the same training loop created by the authors, and the loss function they used was the Binary Cross-Entropy loss function followed by a sigmoid layer.

2.5 Evaluation

We adopted the evaluation metrics and protocols used in the original CADRE paper and its accompanying repository to assess model performance. The primary metrics reported include the F1 Score and Accuracy, which measure the balance between precision and recall and the overall proportion of correct predictions. These metrics are particularly relevant for drug response prediction tasks, where class imbalance and the cost of false positives or negatives can be significant considerations. In addition to predictive performance, we recorded each model's total training time to provide insight into computational efficiency and scalability. This consideration is critical in practical settings, where accuracy and computational efficiency matter. It is important to note that we decided to prioritize the F1 score over accuracy as the dataset is significantly imbalanced. The original authors of CADRE also shared this sentiment.

3 Results

3.1 Reproducing CADRE

Model	F1 Score	Accuracy
Paper Implementation	64.3 ± 0.22	78.6 ± 0.32
Our Implementation	63.23 ± 0.15	78.57 ± 0.15

Table 2: Comparison Between Reported and Reproduced Results

We attempted to reproduce the researchers' results and trained a model with similar, albeit slightly inferior, performance on the GDSC dataset. To ensure consistency and reproducibility, we fixed the random seed across all three trials and then reported the average and standard deviation of the results obtained. We followed the researchers' instructions to the best of our ability. We believe that the differences between the figures presented in the paper and our results are to be expected. One reason that could explain these differences could be the inherent randomness present in many machine learning algorithms and deep learning frameworks. The authors never set a random seed in their codebase, so their training run is not fully reproducible. Even though we used a fixed seed in our experiments, there is still inherent randomness present in the model, such as the limitations of floating point precision and CUDA kernel decisions the PyTorch backend makes. We believe our results are consistent with those reported in the original paper.

3.2 Ablation Study

Disabled	Accuracy	F1 Score	Training Time (s)
Baseline	78.57 ± 0.45	63.23 ± 0.15	315.74 ± 0.74
Attention	78.03 ± 0.35	62.17 ± 0.15	20.13 ± 0.24
Contextual Attention	78.53 ± 0.40	63.20 ± 0.26	273.12 ± 0.18
Dropout	$\mathbf{78.90 \pm 0.40}$	$\mathbf{64.27 \pm 0.15}$	$\mathbf{315.09 \pm 0.09}$
OneCycle	78.17 ± 0.42	63.03 ± 1.68	315.90 ± 0.43
Pretrain Embeds	78.63 ± 0.40	63.53 ± 0.15	436.97 ± 0.35
ReLU	$\mathbf{78.63 \pm 0.55}$	$\mathbf{63.40 \pm 0.20}$	$\mathbf{315.27 \pm 0.42}$

Table 3: Ablation Study of CADRE

We also performed an ablation study of the CADRE model, turning off specific components and observing how they would impact performance. Surprisingly, the model performed better when we turned off the Dropout layer and ReLU activation function. It also performed better when we did not use the pre-trained embeddings but at the cost of a significant increase in required training time. These results suggest that the original model could have been improved if they had done a better job at hyperparameter tuning. Specifically, the dropout rate was very high, which can excessively weaken the signal, especially on small datasets. Removing it allowed the model to learn more stable representations. The ReLU activation function might have been unnecessary after attention-weighted embeddings and introduced dead units. Since most information is preserved in the weighted average, skipping ReLU keeps the signal intact.

The other components of the model proved to provide value for training, as performance fell by varying degrees when they were deactivated. When self-attention is switched off, the model skips the attention mechanism over omic features and defaults to simple mean pooling. The model then loses the ability to emphasize the most relevant gene features per drug selectively. Consequently, the model fails to learn nuanced omic importance and underperforms. When contextual attention is switched off, the attention weights are global instead of personalized to each drug, reducing drug specificity in the attention computation. The OneCycle learning rate scheduling helps convergence by starting low, ramping up, and then annealing. Removing it leads to a less aggressive exploration of learning dynamics, which hurts the model convergence.

3.3 Focal Loss

One of the first extensions we decided to work on was implementing focal loss with binary cross entropy (BCE) loss. We decided to try this extension because the GDSC dataset is quite imbalanced (Section 2.2), and focal loss is designed to address the class imbalance problem. Unlike BCE loss, which treats all examples equally, focal loss adds a modulating factor to down-weight the contribution of easy examples and focus learning on hard, misclassified ones. Formally, focal loss is defined as follows:

$$\mathcal{L}_{\text{focal}} = -\alpha_t(1-p_t)^\gamma \log(p_t) \tag{9}$$

p_t represents the model's estimated probability for the true class, γ represents the focusing parameter that emphasizes harder-to-classify examples, and α_t represents the class weighting factor in balancing classes. After implementing the focal loss function, we had to tune two hyperparameters. Since the search space became a bit large and we had limited computing at the time with NCSA Delta, we decided to fix $\gamma = 0.2$ as it is commonly used in the literature and to tune α as α directly addressed class imbalance. After doing so, we discovered that $\alpha = 0.7$ resulted in a minimal 0.07 increase in the F1 score from 63.23 to 63.30. This high α makes complete sense as the positive labels in the GDSC dataset only comprise 32.2% of the total number of labels, and a higher alpha value puts more emphasis on learning the positive class. We also tried to use an automatic per-class alpha tuning strategy, which assigned an alpha for each class. Unfortunately, we can see from Table 4 that it significantly degraded the model performance.

α	Accuracy	F1 Score	Training Time (s)
Baseline	78.57 ± 0.45	63.23 ± 0.15	315.74 ± 0.74
0.0	69.10 ± 1.73	0.13 ± 0.06	318.98 ± 0.79
0.1	71.40 ± 1.50	17.13 ± 0.31	318.99 ± 0.12
0.2	73.93 ± 1.43	35.17 ± 0.91	318.66 ± 0.24
0.3	75.90 ± 1.18	48.23 ± 1.12	318.38 ± 0.14
0.4	77.20 ± 0.72	56.83 ± 0.32	319.15 ± 0.89
0.5	77.47 ± 0.29	61.20 ± 0.36	321.47 ± 1.81
0.6	76.30 ± 0.20	62.93 ± 0.80	323.46 ± 2.99
0.7	$\mathbf{73.57 \pm 0.78}$	$\mathbf{63.30 \pm 1.54}$	$\mathbf{319.68 \pm 0.39}$
0.8	68.00 ± 1.40	61.70 ± 2.12	319.37 ± 1.68
0.9	57.47 ± 1.70	57.37 ± 2.32	318.58 ± 0.35
1.0	31.47 ± 1.79	47.40 ± 2.00	318.62 ± 0.32
Auto	31.60 ± 1.73	47.43 ± 2.04	315.99 ± 0.22

Table 4: Hyperparameter Tuning of Alpha in Focal Loss

3.4 Adam Optimizer

The following extension we decided to implement was replacing SGD with a more modern optimizer like AdamW. AdamW is a variant of the Adam optimizer that decouples weight decay from the gradient-based update. Theoretically, AdamW should lead to improved performance through its adaptive learning rates and enhanced generalization with decoupled weight decay, and lower sensitivity to learning rate compared to momentum-based SGD. We also paired AdamW with a cosine annealing scheduler, as OneCycle was designed for momentum-based optimizers and AdamW benefits from the smooth decay that cosine annealing provides. Despite these advantages, AdamW underperformed in our experiments. Across all tested configurations, AdamW failed to match the F1 score and accuracy achieved by the baseline (SGD + OneCycle), with the best setting (LR=1e-3) reaching only 61.43 F1 versus 63.23 in the baseline. A potential reason for this underperformance is that adaptive optimizers can lead to overfitting or poor generalization due to their aggressive updates. Also, momentum in SGD can act as a form of smoothing, which is beneficial when data is noisy or the batch size is small. However, performance improved slightly to 68.87 when we paired the SGD optimizer with the cosine annealing scheduler we originally added for AdamW. We hypothesize this occurred because the OneCycle scheduler was ramping learning rate too aggressively, which could have destabilized early training. Furthermore, a more conservative scheduler was more suited to our model with its moderate size, thus having a limited overfitting risk.

Setting	Accuracy	F1 Score	Training Time (s)
Baseline	78.57 ± 0.45	63.23 ± 0.15	315.74 ± 0.74
LR=1e-4 + OC	63.77 ± 0.38	48.17 ± 0.87	316.54 ± 0.84
LR=1e-3 + OC	76.70 ± 0.30	60.07 ± 0.29	315.60 ± 0.53
LR=1e-4 + Cosine	64.67 ± 0.46	48.83 ± 0.91	315.88 ± 0.51
LR=1e-3 + Cosine	77.00 ± 0.36	60.47 ± 0.12	317.12 ± 0.43
LR=1e-4	71.37 ± 0.15	54.47 ± 0.42	315.89 ± 0.69
LR=1e-3	77.60 ± 0.40	61.43 ± 0.12	318.00 ± 2.11
SGD + Cosine	$\mathbf{78.83 \pm 0.50}$	$\mathbf{63.87 \pm 0.15}$	$\mathbf{319.07 \pm 1.23}$

Table 5: Tuning LR of AdamW with other Schedulers

3.5 Decoder MLP and LayerNorm with Residual Connections

The final set of extensions we tested involves multiple small model architecture changes. We added a nonlinear MLP layer at the end of the decoder before predicting drug response in the hopes of introducing more expressiveness to the model and better fitting complex gene-drug interactions. However, model performance degraded, suggesting that the encoder had already produced a rich contextual embedding from the contextual attention mechanism and that the MLP disrupted the learned attention-based representation. We also tried adding LayerNorms to normalize inputs across features to stabilize training and improve gradient flow. We paired the LayerNorms with residual connections to enable the model to learn identity mappings and preserve information across layers. LLMs heavily inspired us in designing this extension, and we wanted to use this opportunity to observe the benefits of PreLN vs PostLN that was discovered for the transformer. However, the LayerNorms did not improve performance, as LayerNorms were generally only needed if we stacked multiple encoder and decoder layers. Furthermore, the LayerNorms could have blurred meaningful variance between features and prevented the model from learning sharp activations. The residual connections did not hurt or help with the average F1 score but did reduce the variance in our observed results.

Modification	Accuracy	F1 Score	Training Time (s)
Baseline	78.57 ± 0.45	63.23 ± 0.15	315.74 ± 0.74
MLP	78.33 ± 0.57	62.73 ± 0.51	315.63 ± 0.23
Residual	$\mathbf{78.57 \pm 0.40}$	$\mathbf{63.23 \pm 0.06}$	$\mathbf{327.72 \pm 0.28}$
PreNorm	78.20 ± 0.26	62.53 ± 0.32	608.34 ± 0.25
PostNorm	77.87 ± 0.68	61.70 ± 0.50	315.66 ± 0.54
PreNorm + Residual	78.17 ± 0.31	62.53 ± 0.32	622.77 ± 0.74
PostNorm + Residual	77.80 ± 0.66	61.63 ± 0.46	327.97 ± 0.42

Table 6: Training Results of Model Architecture Changes

3.6 Final Model

After all these experiments, we trained a new model with the best configurations we had previously discovered. We dropped the ReLU and Dropout layers and added focal loss with $\alpha = 0.7$, a cosine annealing learning rate scheduler, and residual connections. While our testing accuracy dropped by 2%, we saw a 3% increase in the F1 Score, which was the metric we were most interested in due to our imbalanced dataset. This improvement only required an extra 15 seconds of training time, which is minimal in the healthcare domain where model performance is crucial. This model could have possibly been improved with more comprehensive hyperparameter tuning, but we were reasonably satisfied with the final result.

Model	Accuracy	F1 Score	Training Time (s)
Baseline	78.57 ± 0.45	63.23 ± 0.15	315.74 ± 0.74
Ours	76.20 ± 0.56	$\mathbf{66.23 \pm 1.32}$	331.68 ± 0.71

Table 7: Final Model Comparison

4 Discussion

The paper's results are largely reproducible. The authors have a public GitHub repository with preprocessed data and the code to train the models. We achieved very similar model performance numbers compared to the reported numbers from the paper.

Since the preprocessed data was also included in the repository, it was not too difficult to train and iterate on the model, as we did not need to write any code to do data preprocessing.

However, we could not reproduce all of the paper's results, as the CADRE model was also evaluated on the CCLE dataset, which is not publicly accessible. Furthermore, the authors did not include random seeds, so we could never achieve the exact reported numbers in the paper. Another annoyance was that the repository was unfriendly for Google Colab use, requiring significant re-engineering to get the code working through notebooks. This limitation significantly hindered our ability to iterate quickly, as only one of the authors had access to computing that Google Colab did not provide, so experimentation was slow until these engineering efforts were carried out.

We believe the work could have been more easily reproduced if the authors had provided seed numbers for their training runs and a Colab-compatible notebook. This is now generally considered standard practice in machine learning research.

5 Author Contributions

Jimmy worked on most of the coding tasks, while Arjun mainly worked on documentation. Arjun spent more time working on the write-ups and video. Jimmy provided most of the reported numbers used in the documentation above. Both authors worked on the project proposal, ablation study, extensions, video presentation, and final write-up.

References

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2018. Focal Loss for Dense Object Detection. arXiv:1708.02002.

Loshchilov, I.; and Hutter, F. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv:1608.03983.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101.

Schafer, J. B.; Frankowski, D.; Herlocker, J.; and Sen, S. 2007. *Collaborative Filtering Recommender Systems*, 291–324. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-72079-9.

Smith, L. N. 2018. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. arXiv:1803.09820.

Tao, Y.; Ren, S.; Ding, M. Q.; Schwartz, R.; and Lu, X. 2020. Predicting Drug Sensitivity of Cancer Cell Lines via Collaborative Filtering with Contextual Attention. In Doshi-Velez, F.; Fackler, J.; Jung, K.; Kale, D.; Ranganath, R.; Wallace, B.; and Wiens, J., eds., *Proceedings of the 5th Machine Learning for Healthcare Conference*, volume 126 of *Proceedings of Machine Learning Research*, 660–684. PMLR.

Xiong, R.; Yang, Y.; He, D.; Zheng, K.; Zheng, S.; Xing, C.; Zhang, H.; Lan, Y.; Wang, L.; and Liu, T.-Y. 2020. On Layer Normalization in the Transformer Architecture. arXiv:2002.04745.

Yang, Z.; Yang, D.; Dyer, C.; He, X.; Smola, A.; and Hovy, E. 2016. Hierarchical Attention Networks for Document Classification. In Knight, K.; Nenkova, A.; and Rambow, O., eds., *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1480–1489. San Diego, California: Association for Computational Linguistics.