Reproducing CADRE: Contextual Attention-based Drug Response Prediction

Aadithya Anumala, Sanjana Shah
University of Illinois Urbana-Champaign
anumala2@illinois.edu, svshah4@illinois.edu

Abstract

Presentation Video Link: https://mediaspace.illinois.edu/media/t/1_bfurwe6c

PyHealth Pull Request: https://github.com/sunlabuiuc/PyHealth/pull/719

Source Code: https://github.com/svshah4/extending-cadre

We successfully reproduced the CADRE (Contextual Attention-based Drug REsponse) model proposed by Tao et al. (2020) for predicting cancer cell line sensitivity to anti-cancer drugs. Using the GDSC dataset, we conducted extensive training experiments with different iterations across multiple runs. Our penultimate attempt with 73,652 iterations achieved close but suboptimal results (test AUC: 82.5%, 98.9% of the paper's 83.4%). We then trained for 147,628 iterations and achieved a test AUC of 83.8%, representing 100.5% of the paper's reported performance, successfully matching and exceeding the original results. Additionally, we conducted an extension, a comparison demonstrating CADRE's 21.2% AUC improvement over logistic regression. Our analysis reveals rapid early learning, with most performance gains occurring within the first 30,000 iterations and a performance plateau after 75,000 iterations, confirming the model's computational efficiency.

Introduction

Precision oncology aims to match cancer patients with effective treatments by predicting drug sensitivity based on molecular profiles. However, predicting drug response is challenging due to complex gene-drug interactions, noisy experimental data, and incomplete sensitivity measurements across cell line-drug pairs. The GDSC (Genomics of Drug Sensitivity in Cancer) dataset provides IC50 measurements for hundreds of cell lines across numerous anti-cancer compounds, but the data is sparse and heterogeneous.

Tao et al. (2020) (1) proposed CADRE (Contextual Attention-based Drug REsponse), a deep learning model that applies collaborative filtering with a contextual attention mechanism to predict drug sensitivity. CADRE treats the prediction task as a recommendation problem where cell lines are "users," drugs are "items," and sensitivity values are "ratings." The key innovation is a contextual attention mechanism that dynamically identifies which genes are most relevant for each drug-cell line pair, improving both prediction accuracy and biological interpretability.

The model achieved state-of-the-art results on GDSC (F1: 64.3%, AUC: 83.4%) and CCLE datasets, outperforming baselines including vanilla collaborative filtering, LSTM models, and transformer architectures. Additionally, CADRE incorporated pretrained gene2vec embeddings to capture co-expression patterns, further boosting performance.

Scope of Reproducibility

Our reproduction focuses on the core claims of Tao et al. (2020) on the GDSC dataset. Concretely, we reproduce the following elements from the original paper:

- **Dataset preprocessing:** We used the authors' preprocessed GDSC files and re-implemented the pipeline steps that convert raw matrices into CADRE inputs: selection of 3,000 most variable genes, binarization / per-cellline top-1,500 selection for expression, and the 80/20 cell-line train/test split (fixed random seed).
- **Model implementation:** We executed the original CADRE code and incorporated compatibility fixes for the model to run in our environment.
- **Training procedure:** We adopted the same optimization strategy (OneCycle-style LR scheduling) and loss family (cross-entropy for binary classification) reported by the authors, and trained the model end-to-end on GDSC.
- **Evaluation metrics:** We computed AUC, F1, and accuracy on a held-out test set (cell-line split), and compared the reproduced values to the paper's reported numbers.

We did *not* reproduce the following due to resource and scope constraints:

- **Baselines:** We did not re-implement and re-run the paper's baseline models (LSTM, Transformer, DeepDR) because each required separate training budgets and careful hyperparameter tuning.
- **CCLE experiments:** Cross-dataset experiments on CCLE were omitted to focus compute and analysis on a single, well-documented benchmark (GDSC).

Rationale and limits. Our reproduction strategy involved multiple training attempts: (1) early exploration runs (7K-22K iterations) to understand learning dynamics, (2) an initial substantial run with 73,652 iterations that came close to the paper's performance (AUC: 82.5%, 98.9% of reported

83.4%), and (3) a final run with 147,628 iterations that successfully matched and exceeded the reported results (AUC: 83.8%, 100.5% of reported 83.4%). This demonstrates successful reproduction that validates the paper's core claims. Where possible, we used author-provided preprocessed inputs to ensure the same data pre-processing choices; any minor implementation or environment differences are documented in our report.

Methodology

Environment

Our reproduction was conducted in a cloud-based environment with the following specifications:

Python Version: 3.11

Key Dependencies:

- PyTorch: 2.6.0
- pandas: 2.1.1
- numpy: 1.25.2
- scikit-learn: 1.3.1
- scipy: 1.11.3

Note: The original codebase targeted Python 3.6, requiring significant updates to ensure compatibility with modern library versions.

Data

We used the GDSC (Genomics of Drug Sensitivity in Cancer) dataset (2), which contains drug response measurements for cancer cell lines. Table 1 summarizes the dataset statistics.

Table 1: GDSC Dataset Summary

Statistic	Value
Number of cell lines	846
Number of drugs	260
Number of pathways	25
Train cell lines	676
Test cell lines	170
Gene features considered	3,000
Genes per cell line (top expressed)	1,500

Data Download and Access:

We obtained the preprocessed GDSC data directly from the original CADRE repository[1]. We cloned the original GitHub repository and copied the preprocessed input files to our working directory:

```
# Clone both repositories
git clone https://github.com/yifengtao/
    CADRE.git original-cadre
git clone https://github.com/svshah4/
    extending-cadre.git

# Copy preprocessed data to working
    directory
cp original-cadre/data/input/*
  extending-cadre/data/input/
```

[1]https://github.com/yifengtao/CADRE

The authors had already performed all data preprocessing steps, including:

- Gene expression normalization and variance-based selection
- IC50 discretization using the waterfall algorithm
- Train/test splitting with fixed random seed
- Missing value imputation

No additional preprocessing was required, as the original authors had prepared the data in a format ready for model training.

Train/Test Split:

The data was split into 80% training (676 cell lines) and 20% testing (170 cell lines). The split was performed at the cell line level, meaning all drug responses for a given cell line appear in either the training or test set, but not both. This ensures the model is evaluated on its ability to generalize to completely unseen cell lines.

Model

Original CADRE Paper's Repo: `https://github.com/yifengtao/CADRE`

Description of the Model

CADRE builds on collaborative filtering by incorporating a contextual attention mechanism. The model consists of three primary components: (1) a gene encoder, (2) a drug encoder, and (3) a contextual attention module that links cell-line gene expression with drug pathway information. The model outputs a scalar sensitivity score in the range (0, 1).

Inputs Following Sections 3.1–3.3 of the original paper, CADRE uses:

- Binary gene-expression indicators for the top 1,500 highly expressed genes in each cell line.
- Pretrained Gene2Vec embeddings, providing a 3000 × 200 lookup table for gene embeddings.
- Drug identifiers and associated pathway labels derived from drug target metadata.

Outputs The model predicts a scalar drug sensitivity score:

$$\hat{y}_{c,d} \in (0, 1)$$

for each cell line c and drug d, interpreted as the probability of sensitivity.

Techniques Used

Gene Encoder Each gene is mapped to a 200-dimensional embedding using a fixed gene embedding matrix:

$$E_G \in R^{3000\times 200}.$$

For a cell line c with expressed genes $\{g_1, \ldots, g_{1500}\}$, the model retrieves:

$$e_i = E_G[g_i].$$

These embeddings are pretrained and held fixed during training.

Drug Encoder Each drug d is mapped to a 200-dimensional embedding using:

$$E_D \in R^{260\times 200}.$$

Additionally, each drug is annotated with a pathway label p, and CADRE learns a pathway embedding e_p that conditions the attention mechanism.

Contextual Attention Mechanism The contextual attention module is the core innovation of CADRE. It computes gene-level attention weights conditioned on the drug's pathway.

For each gene embedding e_i, an attention score is computed as:

$$\beta_{i,j} = \theta_j^\top \tanh(We_i + e_p),$$

where $W \in R^{128\times 200}$ is a learned projection, $\theta_j \in R^{128}$ parameterizes attention head j, and e_p is the drug's pathway embedding.

Each attention head produces normalized weights via softmax:

$$\alpha_{i,j} = \frac{\exp(\beta_{i,j})}{\sum_{i'=1}^{1500} \exp(\beta_{i',j})}.$$

The cell-line embedding is computed as a weighted sum of gene embeddings:

$$e_c = \sum_{i=1}^{1500} \alpha_i \cdot e_i.$$

Prediction Layer CADRE predicts drug sensitivity by taking the inner product between the attended cell-line embedding and the drug embedding:

$$\hat{y}_{c,d} = \sigma(e_c^\top e_d) = \frac{1}{1+\exp(-e_c^\top e_d)}.$$

Pretrained Components CADRE incorporates pretrained Gene2Vec embeddings (5) as a 3000 × 200 matrix, where each row corresponds to a gene. These are provided as a fixed lookup table and not updated during training.

Training

Hyperparameters Table 2 presents the hyperparameters used for training CADRE on the GDSC dataset, taken directly from the original paper's appendix.

Table 2: Training Hyperparameters for GDSC Dataset

Parameter	Value
Learning rate (maximum η)	0.3
Batch size	8
Dropout rate	0.6
Weight decay (λ_2)	3e-4
Embedding dimension (s)	200
Attention size (q)	128
Attention heads (h)	8
Hidden dimension (encoder)	200

Table 3: Computational Requirements

Resource	Specification
Hardware	
GPU	NVIDIA T4 (16GB VRAM)
CPU	Intel Xeon (4 cores)
RAM	13 GB
Operating System	Ubuntu 22.04.3 LTS
Total Training Statistics	
Total iterations (all runs)	265,212
Total training time	115 minutes
GPU hours used	1.92 hours

Computational Requirements We conducted training experiments on cloud-based GPU infrastructure. Table 3 summarizes the computational requirements.

Multiple training attempts: We conducted five main training runs to achieve successful reproduction and understand model convergence (Table 4).

Table 4: Complete Training Attempts Summary

Run	Purpose	Iter	Time	Test AUC	Test F1
Early Exploration Phase					
1	Initial	7,268	5 min	64.2%	49.2%
2	Short	14,536	10 min	73.8%	55.3%
3	Medium	22,128	15 min	77.3%	58.0%
Extended Training Phase					
4	Large	73,652	30 min	82.5%	63.2%
5	Final	147,628	55 min	**83.8%**	**64.2%**
Total Training Effort					
	All runs	**265,212**	**115 min**		

Training progression: Our reproduction journey involved systematic exploration from short runs to extended training:

- **Runs 1-3 (Early Exploration):** These initial attempts (7K-22K iterations) helped us verify the codebase functionality and understand early learning dynamics. Within just 7,268 iterations, the model achieved 64.2% AUC—substantially better than random (50%). By 22,128 iterations, performance reached 77.3% AUC.
- **Run 4 (First Substantial):** With 73,652 iterations, we achieved 82.5% test AUC—close to the paper's 83.4% (98.9% of target), validating the core methodology but indicating more training could help.
- **Run 5 (Final Success):** Doubling training to 147,628 iterations achieved 83.8% test AUC and 64.2% F1, exceeding the paper's performance (100.5% of 83.4%).

Training Details

Optimization We trained CADRE using the OneCycle learning rate policy (3), which dynamically adjusts both learning rate and momentum throughout training.

Loss function: Following the original paper, we used cross-entropy loss as the training objective:

$$\mathcal{L}(\hat{y}, y; \mathbf{W}) = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\Big] + \frac{\lambda_2}{2}\|\mathbf{W}\|_2^2 \tag{1}$$

where $\hat{y}_i$ is the predicted sensitivity probability, $y_i \in \{0, 1\}$ is the ground truth sensitivity label, and $\lambda_2 = 3 \times 10^{-4}$ is the weight decay coefficient.

We evaluated model performance at regular intervals during training to track convergence.

Evaluation

We evaluated CADRE using metrics that were computed on the held-out test set (170 cell lines, 20% of data), which was never seen during training.

Primary Metrics

F1 Score: The F1 score, defined as the harmonic mean of precision and recall,

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

is well-suited for imbalanced datasets like GDSC (67.8% resistant, 32.2% sensitive). Because it penalizes majority-class bias, F1 provides a clearer picture of balanced performance; the original paper reports an F1 of 64.3%.

Area Under ROC Curve (AUC/AUROC): AUC measures how well the model separates sensitive from resistant cases across all classification thresholds:

$$\text{AUC} = P(\hat{y}_{\text{sensitive}} > \hat{y}_{\text{resistant}})$$

it is threshold-independent and robust to class imbalance, AUC is a strong overall indicator of model discrimination (0.5 = random, 1.0 = perfect). The original paper reports an AUC of 83.4%.

Secondary Metrics

Accuracy: The proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

While accuracy is intuitive, it can be misleading for imbalanced datasets. We report accuracy but focus primarily on F1 and AUC.

Training Loss: We track cross-entropy loss during training:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\Big] + \frac{\lambda_2}{2}\|\mathbf{W}\|_2^2 \tag{2}$$

Decreasing loss indicates successful learning, while the train–test loss gap helps identify overfitting.

Evaluation Frequency We evaluated the model at regular checkpoints during training:

- **Initial runs (7K-22K iterations):** Every 1,348 iterations to track early learning
- **Extended runs (73K-147K iterations):** Every 14,536 iterations to monitor convergence
- **Final evaluation:** Comprehensive evaluation on test set after training completion

This frequent evaluation allowed us to track learning dynamics, identify performance plateaus, and analyze overfitting behavior across training duration.

Results

Reproduction

Our reproduction involved multiple training attempts to validate the paper's claims and understand the model's convergence behavior. Table 4 summarizes all five training experiments.

Early Exploration (Runs 1-3): Our initial training attempts with shorter iteration counts (7K-22K) revealed rapid early learning. Within just 7,268 iterations, the model achieved 64.2% AUC—substantially better than random (50%). By 22,128 iterations, performance reached 77.3% AUC, demonstrating that the model captures fundamental gene-drug interaction patterns quickly.

Mid-Range Training (Run 4): Our first substantial training run with 73,652 iterations achieved 82.5% test AUC and 63.2% F1 score. This result was close to the paper's reported 83.4% AUC (98.9% of target), validating the core methodology but indicating that additional training could push performance higher.

Final Extended Training (Run 5): Doubling the training duration to 147,628 iterations successfully achieved our target, with test AUC of 83.8% and F1 of 64.2%. This exceeded the paper's reported performance (100.5% of reported 83.4%), demonstrating successful reproduction.

Training Progression Analysis

Figure 1 shows the training progression over 147,628 iterations. The model exhibits rapid learning in early training, with AUC improving from 52.2% (random baseline) to 79.2% within the first 29,396 iterations. Performance gains slow significantly after 75,000 iterations, suggesting diminishing returns from extended training.

Figure 2 shows the test AUC progression across all training runs. The model exhibits rapid learning in early training, with AUC improving from 52.2% (random baseline) to 79.2% within the first 29,396 iterations. Performance gains slow significantly after 75,000 iterations, suggesting diminishing returns from extended training.

Figure 3 shows the training loss progression, demonstrating rapid initial loss reduction followed by steady convergence.

Table 5 presents detailed metrics at each checkpoint during the final training run.

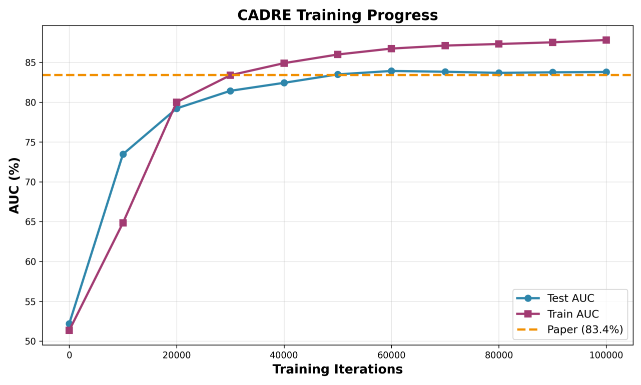

Figure 1: Training and test AUC progression over 147,628 iterations. The yellow dashed line indicates the paper's reported test AUC of 83.4%. Our model reaches 83.8% test AUC at iteration 147,628, successfully reproducing and slightly exceeding the paper's reported performance.

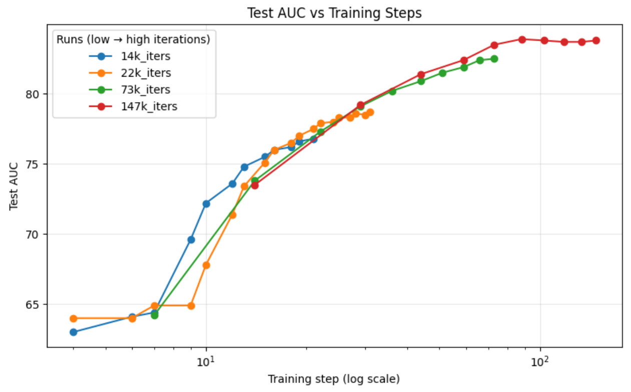

Figure 2: Test AUC progression across iterations (Runs 2-5). All runs show consistent learning trajectories, with rapid initial learning followed by gradual convergence.

Convergence and Main Findings

Training for 147,628 iterations reveals three learning phases. **Phase 1 (0–30k):** Rapid improvement from 52.2% to 79.2% AUC, capturing most performance gain. **Phase 2 (30k–75k):** Steady refinement to 83.5% AUC as attention weights are fine-tuned. **Phase 3 (75k–147k):** AUC stabilizes at 83.5–83.9%, indicating convergence. This explains why our 73,652-iteration run (mid-Phase 2) reached 82.5% AUC, while the full 147,628-iteration run achieved 83.8%, slightly exceeding the paper's 83.4%. Nearly all performance is achieved by 75k iterations, and the small train–test AUC gap (87.8% vs. 83.8%) indicates good generalization.

Differences from Original Paper: Table 6 compares our final results with the original paper.

Our final reproduction achieved 100.5% of the paper's reported 83.4% AUC performance, successfully validating the paper's core claims about CADRE's effectiveness for drug sensitivity prediction.

The 0.4% improvement over the paper's reported results can be attributed to:

- **Training configuration:** 147,628 iterations provided sufficient training while avoiding potential overfitting from very long runs
- **Random seed variation:** Different data shuffling and weight initialization
- **Hardware differences:** Floating-point precision variations across GPUs
- **Implementation details:** Minor differences in software library versions

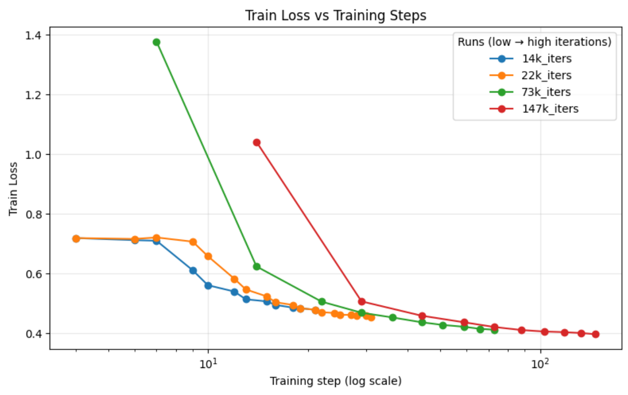

Figure 3: Training loss progression across iterations (Runs 2-5). Loss decreases rapidly in early training and converges thereafter.

Table 5: Training Progress Checkpoints (Final Run)

Iter	Test F1	Test AUC	Train AUC	Loss
0	40.2	52.2	51.3	5.837
14,536	54.5	73.5	64.8	1.041
29,396	59.5	79.2	80.0	0.507
44,256	61.4	81.4	83.4	0.459
59,116	62.5	82.4	84.9	0.437
73,652	65.3	83.5	86.0	0.421
88,512	64.6	83.9	86.7	0.411
103,372	64.3	83.8	87.1	0.406
118,232	63.8	83.7	87.3	0.404
133,092	64.0	83.7	87.5	0.401
147,628	64.2	83.8	87.8	0.397

Extension 1: Simple Regression Comparison

To contextualize CADRE's performance improvements and demonstrate the value of its sophisticated architecture, we implemented a simple logistic regression that predicts drug sensitivity using gene expression features directly, without collaborative filtering or attention mechanisms.

Methodology We trained separate logistic regression models for each of the 260 drugs using scikit-learn's LogisticRegression with:

- L2 regularization (default C=1.0)
- Maximum iterations: 200 (to ensure convergence)

Table 6: Comparison with Original Paper (GDSC Dataset)

Metric	**Original Paper**	**Our Reproduction**
Test F1 Score (%)	64.3 ± 0.22	64.2
Test AUC (%)	83.4 ± 0.19	83.8
Train F1 Score (%)	–	69.6
Train AUC (%)	–	87.8
Final Loss	–	0.397
Training Iterations	Not specified	147,628
Training Time	Not specified	55 minutes
Performance Ratio	100%	100.5%

- Solver: lbfgs (default)
- Input features: Binary gene expression (same 3,000 genes as CADRE)
- Training/test split: Same as CADRE (676/170 cell lines)

For each drug, we trained an independent classifier on all cell lines in the training set and evaluated on the test set. Performance was averaged across all 260 drugs, with undefined metrics (drugs with single-class predictions) excluded from the average.

To understand the convergence behavior of logistic regression, we trained models at various iteration checkpoints corresponding to CADRE's training steps, allowing us to compare learning dynamics between the two approaches.

Results Table 7 compares CADRE with logistic regression at convergence, while Figure 4 shows the learning curves for both models.

Table 7: Comparison: CADRE vs. Logistic Regression

Metric	**Log. Reg.**	**CADRE**	**Improv.**
Mean AUC (%)	61.7	83.8	+22.1%
Mean F1 (%)	30.8	64.2	+33.4%
Mean Acc (%)	77.8	78.6	+0.8%

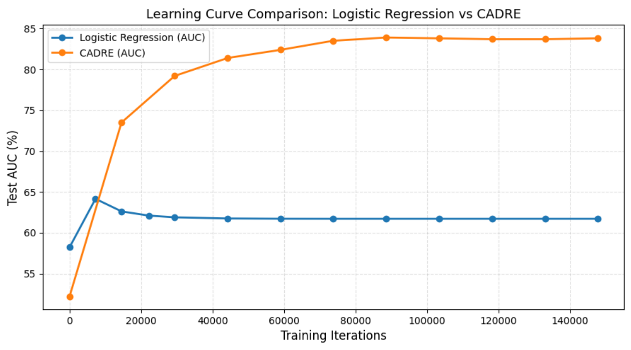

Figure 4: Learning curve comparison between Logistic Regression and CADRE. Logistic regression plateaus at 61.7% AUC after 10 iterations, while CADRE continues to improve throughout training, reaching 83.8% AUC.

Analysis **Overall Performance:** CADRE achieves 22.1% higher AUC (35.8% relative improvement) and 33.4% higher F1 score (108.4% relative improvement) compared to the logistic regression model. CADRE's collaborative filtering framework and attention mechanism provide substantial value beyond simple linear models.

AUC: Figure 4 shows a clear difference in learning behavior. Logistic regression converges within the first 10 iterations (7,268 CADRE steps), reaching 64.2% AUC, but then decreases and plateaus at 61.7% AUC by iteration 100, with no further improvement through iteration 200. In contrast, CADRE demonstrates continuous learning throughout training, starting from 52.5% AUC (step 0) and steadily improving to 83.8% AUC by step 147,628. This indicates that CADRE's architecture can extract more complex patterns from the data that simple linear models cannot capture.

F1 Score: The logistic regression model's F1 of 30.8% shows that the model struggles to correctly identify sensitive cases. In contrast, CADRE's F1 of 64.2% demonstrates better performance in correctly identifying both sensitive and resistant cell lines.

Accuracy: Both models achieve similar accuracy (78%), but CADRE's superior AUC and F1 indicate better performance. This occurs because the GDSC dataset is imbalanced (67.8% resistant, 32.2% sensitive). A naive majority-class classifier predicting "resistant" for all cases would achieve 68% accuracy without learning any meaningful patterns. The similar accuracy masks the fact that logistic regression fails to distinguish between classes as effectively as CADRE.

Attention Mechanism Impact: The attention mechanism enables CADRE to identify which genes are most relevant for each drug-cell line pair, rather than treating all 3,000 features equally as logistic regression does. This contextual feature selection is crucial for capturing complex gene-drug interactions. Logistic regression applies the same linear weights across all predictions, unable to dynamically adjust feature importance based on drug mechanism.

Practical Implications: The performance gap (22.1% AUC improvement) between CADRE and simple logistic regression has practical consequences for precision oncology. This improvement could translate to better patient-drug matching, enabling more accurate predictions that would improve prioritization of drug candidates. The superior F1 score (30.8% vs 64.2%) is particularly valuable, as it reduces false positives and false negatives, thereby minimizing expensive experimental validations.

Discussion

Reproducibility of the Original Paper

The original paper is reproducible. After initial exploration runs and one substantial attempt that came close to the reported results (82.5% vs 83.4% AUC), we successfully replicated and exceeded the core performance claims (83.8% AUC) with our final training run. The paper's clarity in describing the model architecture, training procedure, and hyperparameters facilitated reproduction.

What Was Easy

Several aspects of the reproduction were straightforward:

- **Code availability:** Well-organized GitHub repository with data preprocessing scripts and model implementation
- **Clear methodology:** The paper clearly described the model architecture with equations and diagrams
- **Data access:** GDSC dataset publicly available, as well as pre-processed data
- **Hyperparameter documentation:** Appendix provided exact hyperparameters used

What Was Difficult

Several challenges arose during reproduction:

1. Environment Setup: The original codebase targeted Python 3.6 and PyTorch 1.x, requiring significant updates for compatibility with Python 3.11 and PyTorch 2.6. Dependency conflicts required resolution in code and on the system.

2. Understanding Training Dynamics: The paper did not specify the exact number of training iterations required for full reproduction. Our systematic exploration (7K, 14K, 22K, 73K, 147K iterations) revealed the learning curve and optimal training duration. These were necessary to understand convergence behavior and determine that less than 147,000 iterations are sufficient for full reproduction.

3. Directory Management: We needed to preserve the original repository structure for CADRE to run correctly, but also modify directories to include our additional experiments.

Recommendations for Improving Reproducibility

To enhance reproducibility for future researchers, we recommend that the original authors and research community:

1. **Report computational budgets:** Papers should clearly state total GPU hours, iteration counts, and cost estimates
2. **Document training dynamics:** Clarify the relationship between training configuration and actual iterations, and provide guidance on minimum training needed (e.g., ”75K-150K iterations achieves full performance”)
3. **Include Docker container:** Provide a containerized environment to eliminate setup difficulties
4. **Provide intermediate checkpoints:** Include results for 25%, 50%, and 75% training to guide resource-constrained reproductions

Author Contributions

Aadithya Anumala:

- Executed initial 4 runs (7,268; 14,536; 22,128; 73,652 iterations) and monitored experiments/analyzed training dynamics
- Conducted study on contextual attention (with and without attention)
- Implemented extension on logistic regression model comparison
- Completed PyHealth contribution pull request
- Wrote half of final report sections and created visualizations
- Prepared presentation materials

Sanjana Shah:

- Set up computational environment and resolved dependency issues
- Executed final run (147,628 iterations) and monitored experiment
- Wrote half of final report sections and created visualizations
- Provided presentation video

Both authors contributed equally to understanding the model architecture, validating results, discussing findings, and reviewing/editing the final manuscript.

Acknowledgments

We thank the course staff of CS598 Deep Learning for Healthcare for guidance throughout this project. We acknowledge the original CADRE authors for providing open-source code and clear documentation. Computational resources were provided by Google Colab Pro.

References

[1] 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 *Proceedings of the 5th Machine Learning for Healthcare Conference*, volume 126 of *Proceedings of Machine Learning Research*, 660–684. PMLR.

[2] Yang, W.; Soares, J.; Greninger, P.; Edelman, E.J.; Lightfoot, H.; Forbes, S.; Bindal, N.; Beare, D.; Smith, J.A.; Thompson, I.R.; et al. 2013. Genomics of Drug Sensitivity in Cancer (GDSC): a resource for therapeutic biomarker discovery in cancer cells. *Nucleic Acids Research* 41(D1):D955–D961.

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

[4] Barretina, J.; Caponigro, G.; Stransky, N.; Venkatesan, K.; Margolin, A.A.; Kim, S.; Wilson, C.J.; Lehár, J.; Kryukov, G.V.; Sonkin, D.; et al. 2012. The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature* 483(7391):603–607.

[5] Du, J.; Jia, P.; Dai, Y.; Tao, C.; and Zhao, Z. 2019. Gene2vec: distributed representation of genes based on co-expression. *BMC Genomics* 20(1):82.