Reproducing and Analyzing Baselines for Chest X-Ray Report Generation: Insights and Challenges
Jimeng Sun

Written by Hong Wu, Yinzhe Luo
University of Illinois at Urbana Champaign
201 North Goodwin Avenue
Urbana, Illinois 61801
admin@cs.illinois.edu

Abstract

Link to presentation video —
https://github.com/hwu5542/cxr-baselines
Link to Github Repo —
https://github.com/hwu5542/cxr-baselines

Introduction

The writing of radiology reports, especially for chest X-rays (CXRs), is a routine but time-consuming part of clinical practice. Since CXR imaging is widely used, the number of reports generated daily is substantial. These reports require domain-specific expertise, and automating the process could help ease the burden on radiologists while speeding up diagnosis.

The study "Baselines for Chest X-Ray Report Generation" by Boag takes a critical look at how machine-generated reports are typically evaluated. Most of the research in this space has focused on surface-level text metrics such as BLEU or CIDEr, which compare word overlap with reference reports. Although metrics like BLEU and CIDEr are widely used in natural language processing, they do not always capture what really matters in a medical context, namely whether the generated report makes clinical sense. Boag and his teammates tackled this issue by experimenting with a range of baseline models, from basic random output to a neural network that combines convolutional and recurrent layers. To get a clearer sense of whether the generated reports made clinical sense, the authors applied the CheXpert labeler. Rather than just checking for grammatical similarity, this tool looks at how well the content aligns with actual medical findings.

What stood out in their results was how well some of the simpler retrieval-based models performed. In fact, just retrieving the closest matching reports often worked as well as—or in some cases better than—much more complex neural approaches. It is a good reminder that the usefulness of a model in practice does not always match its technical sophistication, especially when the goal is to capture clinical meaning.

In our project, we tried to reproduce the main findings from their study and also explored a small extension by including a BERT-based decoder in the original CNN-RNN pipeline. The idea was to see if adding a modern pre-trained language model would improve how natural the reports sounded or whether it helped with capturing key medical points. We were curious to see whether a pre-trained language model could improve either the readability or the diagnostic usefulness of the generated text. Throughout the project, we paid close attention to the details of implementation and kept track of where things worked as expected and where they didn't—both to understand the reproducibility of the original work and to reflect on our own process.

Citation to the original paper

Boag, W., Hsu, T. M., McDermott, M., Berner, G., Alsentzer, E., & Szolovits, P. (2020). Baselines for Chest X-Ray Report Generation. Proceedings of Machine Learning Research, 116, 126-140. ML4H at NeurIPS 2019.

Scope of Reproducibility

our results were fairly close to what the original paper reported, drawing similar conclusion. We had a consistent setup with the author. And through the project, it gave us a better understanding of the challenges involved.

- **Dataset processing:** Used MIMIC-CXR dataset. we read chest X-ray images in DICOM format and resized them to 224×224 pixels. The images were normalized to match DenseNet input expectations. For the reports, we applied a text-cleaning process that removed sensitive identifiers and standardized formatting, though we used the full report text rather than isolating the "Findings" section. Each image was paired with its corresponding cleaned report, and the dataset was split into training, validation, and test sets.
- **Baseline models:** 4 models, random, n-gram(1-, 2-, and 3-grams), 1-NN, and CNN-RNN. Each was evaluated.
- **Evaluation pipeline:** For evaluation, we used standard NLG evaluation (BLEU, CIDEr) and CheXpert labeler for their clinical accuracy

Methodology

Environment

Python version used: 3.11.2. And dependencies/packages needed for execution:

- **numpy**, **pandas** – for numerical operations and data manipulation.
- **pydicom**, **pillow (PIL)** – for loading and preprocessing DICOM images.
- **torch**, **torchvision** – for implementing and training deep learning models.
- **scikit-learn** – for splitting the dataset and computing evaluation metrics.
- **tqdm** – for displaying training progress bars.
- **matplotlib** – for plotting results and data inspection.

Data

Dataset Access and Download:

We used the MIMIC-CXR v2.1.0 dataset, which contains chest radiographs in DICOM format and associated radiology reports. The dataset is hosted on PhysioNet and requires credentialed access through a data use agreement process. To use the dataset, user must complete the CITI "Data or Specimens Only Research" course and agree to the terms of use before downloading via https://physionet.org/content/mimic-cxr/2.1.0/.

For evaluation, we also used the CheXpert labeler released by Stanford, which is available at CheXpert GitHub repository via https://github.com/stanfordmlgroup/chexpert-labeler. Our project includes a local version of the labeler under the `evaluate/` folder. It uses NegBio-based pattern matching to extract 14 clinical labels (e.g., Cardiomegaly, Edema, Pleural Effusion) from free-text reports.

Dataset Description:

The MIMIC-CXR dataset contains over 370,000 chest X-ray images from more than 227,000 imaging studies. Each study is accompanied by a free-text radiology report. Our project filtered for Anteroposterior (AP) view images and used the full report text after de-identification, rather than isolating only the "Findings" section.

Attribute	Description
Source	MIMIC-CXR v2.1.0 (PhysioNet)
Format	DICOM images + free-text reports
Views Used	Anteroposterior (AP) only
Image Size	Resized to 224×224
Text Scope	Full de-identified reports (not only "Findings")
Split Method	80/20, grouped by patient ID
CheXpert Labeler	Used to extract 14 clinical findings

LLM Assistance During Preprocessing:

We used ChatGPT (GPT-4) to assist with parts of our data preprocessing pipeline. Our first prompt was:

> `Can you help me write Python code to load DICOM images, resize them to 224×224, normalize them, and pair them with radiology reports?`

The LLM generated a usable function using `pydicom`, `PIL`, and `numpy`. While it provided a useful structure, we adapted parts of the code to better match the dataset format and added custom error handling. Over the course of preprocessing, we consulted the LLM approximately 4–5 times — mainly for guidance on DICOM handling, normalization parameters, and logging utilities. These prompts were helpful for speeding up the development process, though most of the final implementation was tailored manually.

Model

In the paper *"Baselines for Chest X-Ray Report Generation"*, there are four baseline models proposed, random, n-gram, 1-nearest neighbor(retrieval) and CNN-RNN. Therefore in our project, we same Included these models.

1. Random : Implementation in file `randomRetrival.py`. The Random model selects a report $\hat{R}$ uniformly at random from the training set:

$$\hat{R} \sim \mathcal{U}(R_{train})$$

This Model ignores the input and it serves as naive lower bound baseline for our comparison.

2. n-gram : Implementation in file `ngram.py`. The n-gram model tokenizes each report into n-grams (1-gram, 2-gram, or 3-gram) and retrieves the report from the training set that shares the most overlapping n-grams. Equation used:

$$\hat{R} = \arg \max_{R_i \in R_{train}} sim_{n-gram}(R_i, R_I)$$

Here, sim_{n-gram} is a function that counts shared n-gram tokens. We use Unigram, Bigram and Trigram to generate reports based on the successive word occurrence, using the closest 100 training images in DenseNet121.

3. 1-Nearest Neighbor (1-NN): This model is also implemented in `ngram.py`. It converts each report into a vector using method TF-IDF, which captures how often certain words appear. Then, when a new test report comes in, the model compares it to every other reports in the training set using cosine similarity (a measure of how close the vectors are in direction). After that, the report that from the training set is most similar (has the highest cosine score) is returned as the prediction as the final. Both grammatical and clinical accuracy should increase

$$\hat{R} = \arg \max_{R_i \in R_{train}} \frac{f(R_I) \cdot f(R_i)}{\|f(R_I)\| \|f(R_i)\|}$$

Here, $f(R)$ is the vector form of a report. This method is often effective in capturing the medical content of reports, as it try to matches based on shared wording and terminology.

4. CNN-RNN Encoder-Decoder: Implementation in file $\texttt{cnn}_r nn.py, this model consists of two main parts$:

`CNN Encoder`: `project the output from 1024 dimensions to 256 dimensions:`

$$\mathbf{v}_{img} = GAP(DenseNet121(I)) \in R^{1024}$$

`This vector is then projected into a 256-dimensional latent space:`

$$\mathbf{z}_{img} = W_{proj} \cdot \mathbf{v}_{img} + b_{proj}, \quad \mathbf{z}_{img} \in R^{256}$$

RNN Decoder: The decoder is a unidirectional LSTM that receives token embeddings and the image vector. At each time step t:

$$\mathbf{h}_t, \mathbf{c}_t = LSTM(\mathbf{e}_{t-1}, \mathbf{h}_{t-1}, \mathbf{c}_{t-1})$$

The decoder generates vocabulary probabilities using:

$$\mathbf{y}_t = softmax(W_o \cdot \mathbf{h}_t + b_o)$$

Using cross-entropy loss per token as Optimizer. Epoch: 64. Learning Rate: 1 × 103, decaying 0.5 per 16 epochs. Teacher Forcing: feeding ground-truth tokens to decoder initially, replace 5% to feeding predictions per 16 epochs.

5. CNN-RNN-BERT Hybrid: Implementation in file cnn_rnn_bert.py, this is a extended model that replaces the original LSTM decoder in the CNN-RNN architecture with a transformer-based decoder using a pretrained BERT model from HuggingFace. This model's goal is to improve linguistic fluency and contextual coherence.

Processed through DenseNet121 (pretrained on ChestX-ray14) and mapped into a vector $\mathbf{z}_{img} \in R^{256}$. Then fused into the input sequence of the BERT model. The BERT decoder takes in the image context along with token embeddings and produces contextualized representations at each step.

$$\mathbf{H} = BERT([\mathbf{z}_{img}, \mathbf{e}_1, \ldots, \mathbf{e}_n])$$

$$\hat{y}_t = softmax(W \cdot \mathbf{h}_t + b)$$

Inputs and Outputs for Retrieval Models (Random, n-gram, 1-nn):

- **Input:** A test image I (text or features optional)
- **Output:** A full report $\hat{R}$ retrieved from the training set

Techniques Used (Retrieval Models):

- Uniform sampling (Random)
- Token level n-gram matching (n-Gram)
- Cosine similarity in sparse vectors (1-NN)

Inputs and Outputs (CNN-RNN):

- **Input:** DICOM image $I \in R^{224\times224}$ and a token sequence $T = (t_1, \ldots, t_n)$
- **Output:** Generated token sequence $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_m)$

Techniques Used (CNN-RNN):

- Image encoding via pretrained DenseNet121
- Projection to compact latent space
- LSTM-based decoder with teacher forcing during training
- Greedy decoding at inference time

Inputs and Outputs (CNN-RNN-BERT):

- **Input:** DICOM image I and a partial token sequence $T = (t_1, \ldots, t_n)$
- **Output:** Predicted sequence $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_m)$

Techniques Used (CNN-RNN-BERT):

- Pretrained DenseNet121 as image encoder
- Tokenization and decoding with HuggingFace TFBertModel
- Image feature fusion with token embeddings
- Teacher forcing during training; greedy decoding at inference

Pretrained Model: The DenseNet121 encoder was initialized with ImageNet weights. During training, all convolutional layers were frozen to preserve general visual features.

LLM Assistance:

We used ChatGPT (GPT-4) to assist with the CNN-RNN model. Our initial prompt was:

> Can you help me build a CNN-RNN model in TensorFlow that takes in an image, extracts features using DenseNet, and decodes a text sequence using an LSTM?

The LLM provided a functional Keras template including DenseNet as a feature extractor and an LSTM-based decoder. We modified the shapes, projection layers, and embedding details to suit our dataset. We prompted the LLM 3{4 times during model implementation and debugging. For the retrieval models, we also used the LLM to refine cosine similarity and TF-IDF-based logic, which we customized for our pipeline.

Training

Hyperparameters:

The CNN-RNN model was trained with the following hyperparameters:

- **Learning Rate:** 0.001. 1×10^{-3}, decaying 0.5 per 16 epochs.
- **Batch Size:** 32.
- **Hidden Size:** 256 LSTM units and 256-d word embeddings.

- **Epochs:** 64 epochs.

Computational Requirements and Feasibility:

- Data Preprocessing: The MIMIC-CXR dataset is around 100GB in size, contains 473,057 chest X-ray images and 206,563 reports from 63,478 patients (Johnson et al., 2019). Through data preprocessing, we filter anteroposterior(AP) views, remove brightness/contracts adjusted duplicate radiographs and select only related sections in the reports to further compress the data size. This will yield a final size of 95,242 images (train: 75,147, test: 19,825) with no patient overlapping between splits. Reducing more than 60 GB of data. We utilize the pretrained DenseNet121 encoder from ChestX-ray14 to extract 1024-dimensional features. Also adding rule-based labeler for 14 clinical categories with around 1 sec/report CPU overhead.
- Hardware of training: The CNN-RNN model training requires RTX 30 series GPU with 16 GB VRAM. The training time will take around 24 to 48 hours.

Training Details:

- **Loss Function:** For loss function, We used sparse_categorical_crossentropy. Which we think it is appropriate for sequence generation tasks.
- **Optimization:** Adam optimizer, with exponential learning rate decay.
- **Training Procedure:** The Teacher forcing was used. Feeding ground-truth tokens to decoder initially, replace 5% to feeding predictions per 16 epochs.

LLM Assistance:

We used ChatGPT (GPT-4) to help construct and debug our training loop. Our first prompt was:

> Can you help me write a training loop for a CNN-RNN model in TensorFlow with teacher forcing and sequence masking?

The LLM provided a valid prototype using tf.GradientTape, sequence-level loss computation, and optimizer setup. We adapted the code to match our dataset structure and added a custom learning rate schedule and early stopping logic. We issued 3{4 follow-up prompts for reshaping issues, mask alignment, and Keras callback usage. The LLM was helpful for bootstrapping the boilerplate, especially for the TensorFlow functional API.

Evaluation

Metrics Used:

In this project, we included three metrics for evaluation for the assessment of models. In linguistic and clinical :

- **BLEU-4:** A precision-based metric that compares n-gram overlap (up to 4-grams) between the generated report and the reference report. The Higher BLEU scores, the better linguistic similarity.
- **CIDEr:** Originally designed for image captioning tasks, CIDEr weights n-gram matches using TF-IDF scores and compares a candidate sentence to a set of reference sentences. Contents that words are more informative are more favor and generic language in contrast.
- **CheXpert F1:** CheXpert labeler is used to both generated and ground-truth reports to extract 14 clinical labels. Then it is used to compute the macro-averaged F1 score. This metric focuses on the medical correctness the generated output. It is independent from grammar.

LLM Assistance:

We used ChatGPT (GPT-4) to help design and validate our evaluation code. Our initial prompt was:

> How can I compute BLEU and CIDEr scores for medical report generation using Python?

The LLM returned code snippets using nltk.translate.bleu_score and recommended the pycocoevalcap package for CIDEr. We cross-checked this with the implementations in our evaluate_nlg.py script and verified that the structure matched standard practice.

For the CheXpert F1 score, although the evaluation logic is not fully visible in our codebase, we integrated the official CheXpert labeler to generate structured label predictions and used macro-averaged F1 as the primary clinical metric. We issued two follow-up prompts to confirm how best

to calculate label-level metrics on multi-label classification tasks using external labelers.

Overall, the LLM was useful for confirming that our evaluation pipeline aligns with established best practices in both natural language generation and medical report evaluation.

Results

Reproduced Results

We report our reproduced performance for the original four baseline models. All evaluations were performed on the MIMIC-CXR validation set using BLEU-4, CIDEr, and CheXpert F1 metrics.

Model	BLEU-4	CIDEr	CheXpert F1
Random	0.024	0.10	0.032
1-Gram	0.071	0.26	0.074
2-Gram	0.112	0.35	0.096
3-Gram	0.125	0.42	0.115
1-NN	0.186	0.63	0.185
CNN-RNN	0.212	0.70	0.186

Table 1: Reproduced BLEU, CIDEr, and CheXpert F1 scores for baseline models

These results closely align with the original paper. Notably, the 1-NN model remained competitive across all metrics. This supports the authors' claim that retrieval-based methods may perform comparably to generation in terms of diagnostic relevance.

Extended Model Results: CNN-RNN-BERT

We trained a BERT-based decoder model for 64 epochs and evaluated performance across checkpoints.

Epoch	BLEU-4	CIDEr	CheXpert F1
8	0.170	0.59	0.158
24	0.188	0.64	0.181
48	**0.216**	**0.73**	**0.190**
64	0.210	0.71	0.187

Table 2: Performance of the CNN-RNN-BERT model at selected epochs

The BERT decoder led to small but consistent improvements in fluency scores (BLEU/CIDEr), particularly around epoch 48. However, the clinical accuracy plateaued and did not substantially exceed the original CNN-RNN model.

Extension and LLM Integration

Extension Brainstorming with LLM Prompt: ``Suggest possible model extensions for chest X-ray report generation using transformers or retrieval.''

LLM Suggestions:

1. Add a retrieval-augmented decoder that combines 1-NN with generative refinement.
2. Use domain-adapted BERT (e.g., BioBERT, ClinicalBERT).
3. Implement uncertainty-aware loss for negation-sensitive labels.

Chosen Extension: Integrate BERT decoder into CNN-RNN. **Validation:** A promising middle ground between fluency and retrieval. **Iterations Used:** 3 total prompts | 1 idea generation, 2 feasibility checks.

LLM in Extension Implementation Prompt: ``Adapt encoder-decoder captioning model to use a BERT-based decoder with PyTorch.'' **Validation:** Helped prototype decoder structure and tokenizer integration. **Prompt Count:** 4 for architecture, 2 for fine-tuning logic.

Discussion

Our results validate Boag et al.'s key insight: simpler baselines like 1-NN may outperform generative models in clinical accuracy. However, generative models offer advantages in fluency and generalization. Our CNN-RNN-BERT model improved language metrics but did not significantly improve CheXpert accuracy, suggesting that transformer-based fluency gains do not guarantee clinical gains.

What Was Easy? Implementing and testing n-gram and 1-NN models was quick due to their simplicity. Integrating BLEU/CIDEr metrics using nltk was also straightforward.

What Was Hard? BERT decoder integration introduced significant instability due to long sequences, tokenizer alignment, and GPU memory constraints. CheXpert label extraction also required additional handling for regex inconsistency in generated text.

Recommendations for Reproducibility

- Provide complete preprocessing scripts in shared codebases.

- Include example model outputs to standardize labeler behavior.
- Benchmark fluency and clinical correctness separately to guide tuning.

Conclusion

In this work, we reproduced and extended the baseline models from the paper *''Baselines for Chest X-Ray Report Generation''* by Boag et al. (2020). We re-implemented four key models: random retrieval, n-gram generation, 1-nearest neighbor retrieval, and a CNN-RNN encoder-decoder. Our evaluation reproduced the key findings of the original study: simple baselines such as 1-NN can achieve surprisingly strong clinical performance, despite lacking generative flexibility.

We also introduced a novel extension by replacing the LSTM decoder in the CNN-RNN model with a pretrained BERT transformer. While this led to modest improvements in BLEU and CIDEr scores, the gains in clinical accuracy were limited, highlighting that linguistic fluency alone does not ensure medical correctness.

Our experience confirms that reproducibility in medical NLP involves not just replicating code and metrics, but also understanding domain-specific nuances such as image-view filtering, label extraction, and clinical evaluation. We hope our findings and extensions contribute toward a deeper understanding of how to evaluate and improve radiology report generation systems.

Future work may explore ensemble or hybrid systems that combine retrieval with generation, or integrate structured medical knowledge to bridge the gap between fluency and correctness.

Author Contributions

Hong Wu: Project proposal, Data Loading and Preprocessing, data parser, Baseline Models(random retrieval, First Nearest Neighbor, ngram, CNN-RNN Bert), chexpert evaluation, NLG evaluation

Yinzhe Luo: Project paper Draft and presentation

References

Boag, W.; Hsu, T. M.; McDermott, M.; Berner, G.; Alsentzer, E.; and Szolovits, P. 2020. Baselines for Chest X-Ray Report Generation. In *Proceedings of Machine Learning Research*, volume 116, 126--140. ML4H at NeurIPS 2019.

Johnson, A. E. W.; Pollard, T. J.; Berkowitz, S.; Greenbaum, N. R.; Lungren, M. P.; Deng, C.-y.; Mark, R. G.; and Horng, S. 2019. MIMIC-CXR: A large publicly available database of labeled chest radiographs. *arXiv:1901.07042*.

Rajpurkar, P.; Irvin, J.; Zhu, K.; Yang, B.; Mehta, H.; Duan, T.; Ding, D.; Bagul, A.; Langlotz, C.; Shpanskaya, K.; Lungren, M. P.; and Ng, A. Y. 2017. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. *arXiv:1711.05225*.