A Lightweight Multimodal Baseline for Readmission Risk Prediction on MIMIC-III

Author: James Zeng, Paul Grigas, Qiran Dong
Affiliation: University of California, Berkeley
Contact: j_zeng@berkeley.edu, pgrigas@berkeley.edu, qirandong@berkeley.edu
Date: May 15th, 2025

Abstract

This project explores a simplified, multimodal approach to 30-day readmission prediction using MIMIC-III. We implement three baselines: a text-only GRU model, a tabular-only Feedforward Neural Network (FFNN), and a multimodal fusion model combining both. To maximize interpretability and minimize compute time, we freeze unimodal encoders and train only a lightweight fusion head. Our results show that while tabular features alone achieve strong AUC (0.833), fusion yields improved accuracy (up to 0.8235) but modest AUC (0.571), showing that naive fusion may not outperform the strongest modality. All code runs on Google Colab with PyTorch, pandas, and scikit-learn.

---

1. Introduction

Preventing avoidable hospital readmissions is a critical healthcare challenge. With the growing availability of structured and unstructured data from Electronic Health Records (EHRs), predictive modeling can offer real-time tools to flag at-risk patients. However, balancing accuracy, simplicity, and clinical interpretability remains difficult.

This project builds a minimalist multimodal readmission risk predictor inspired by the EMERGE framework, but without knowledge graphs or retrieval components. We compare three models:

- A text-only GRU baseline using discharge summaries
- A FFNN using aggregated lab, input, and output features

- A multimodal fusion model with frozen encoders and a weighted fusion layer

---

2. Methods

2.1 Dataset and Preprocessing

We use the MIMIC-III v1.4 database. From `NOTEEVENTS.csv`, we extract discharge summaries. For tabular features, we aggregate time-series values from:

- `LABEVENTS.csv` (e.g., sodium, creatinine)
- `INPUTEVENTS_MV.csv` and `INPUTEVENTS_CV.csv` (fluid intake)
- `OUTPUTEVENTS.csv` (urine output)
- `PROCEDUREEVENTS_MV.csv` (procedures)

We join all features by `HADM_ID` and align them with readmission labels extracted from `ADMISSIONS.csv`.

Preprocessing Steps:

- Texts are lowercased and tokenized by space.
- Numerical features are normalized using:

$$x\_scaled = (x - mean) / std$$

- All samples with missing values are dropped.
- Labels are downsampled to ensure 1:1 positive-to-negative ratio.

2.2 Text Baseline: GRU Model

We use a shallow GRU to encode text:

- Embedding: 256-dim
- GRU hidden size: 128
- Classifier: Linear(hidden → 1)

Let `h_T` be the GRU's final hidden state. We compute logits as:

$$logits = Linear(h\_T)$$

Loss is computed using `BCEWithLogitsLoss` with clipped `pos_weight = min(N_neg / N_pos, 20)` to handle imbalance.

2.3 Tabular Baseline: FFNN

We build a 2-layer FFNN:

- Input: Normalized lab/input/output aggregates
- Architecture:
  Linear(input → 64) → ReLU → Linear(64 → 1)

Intermediate embeddings are extracted before the final layer for use in fusion. Loss and imbalance strategy mirror the text model.

2.4 Fusion Model: Frozen Encoders + Weighted Head

We concatenate the two learned representations:

- GRU final hidden state: `h_T` (text)
- FFNN hidden layer output: `h_tab` (tabular)

Each is normalized using LayerNorm. Then we apply learned scalar weights:

$$h\_fused = [\alpha\_text * LayerNorm(h\_T);\ \alpha\_tab * LayerNorm(h\_tab)]$$

The fused vector is passed to:

Linear → ReLU → Linear → output logit

Only the fusion weights and MLP are trained; GRU and FFNN encoders are frozen. This ensures simplicity and interpretability.

---

3. Training Setup

- Framework: PyTorch 2.0, pandas, scikit-learn
- Hardware: Google Colab CPU
- Train/val split: 80/20, stratified
- Batch size: 32
- Epochs:
  - GRU and FFNN: 50–200
  - Fusion head: 200
- Evaluation:
  - Accuracy
  - ROC-AUC (scikit-learn's `roc_auc_score`)

---

4. Results

Model	Accuracy	ROC-AUC
Text-only GRU	0.5606	0.591
Tabular FFNN	0.4286	0.833
Fusion (Frozen)	0.8235 → 0.6471	0.571

4.1 Key Findings:

- FFNN yields the best AUC, suggesting tabular features are stronger predictors.
- GRU performs moderately, indicating signal in discharge notes.
- Fusion initially improves accuracy but not AUC, and later plateaus or declines.
- Naive fusion is not always additive; modality weighting is critical.

---

5. Discussion

This project shows that:

- Downsampled data + simple FFNN yields strong discriminative performance.
- Fusion must balance modality contributions carefully.
- Lightweight models with frozen encoders are ideal for interpretable rapid prototyping.

Failure Modes:

- Small positive sample size limits stability.
- AUC may fluctuate due to overfitting in fusion.
- GRU and FFNN representations may be misaligned.

Future Work:

- Try softmax-normalized fusion weights.
- Explore multimodal dropout or gating mechanisms.
- Add early stopping based on validation AUC.

6. Conclusion

We present a fast, interpretable, and reproducible pipeline for multimodal readmission prediction. Our tabular-only FFNN baseline outperforms others in AUC. Fusion adds value in accuracy but requires careful handling. This pipeline is extensible to larger EHR tasks and future real-time deployment efforts.

References

1. Jha, A. K., et al. "Use of electronic health records in U.S. hospitals." *NEJM* (2009). https://www.nejm.org/doi/full/10.1056/NEJMsa0900592
2. Johnson, A. E. W., et al. "MIMIC-III, a freely accessible critical care database." *Scientific Data* (2016). https://www.nature.com/articles/sdata201635
3. PyTorch Documentation: BCEWithLogitsLoss https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html
4. PyTorch Documentation: LayerNorm https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html
5. Zhu, Y., et al. “EMERGE: Enhancing Multimodal EHR Predictive Modeling with Retrieval-Augmented Generation.” *CIKM 2024*. https://arxiv.org/abs/2404.00164
6. Fawcett, T. "An introduction to ROC analysis." *Pattern Recognition Letters* (2006). https://doi.org/10.1016/j.patrec.2005.10.010
7. Rajkomar, A., et al. "Scalable and accurate deep learning with electronic health records." *NPJ Digital Medicine* (2018). https://www.nature.com/articles/s41746-018-0029-1

Data Processing

Data Loading

[ ] ↳ 4 cells hidden

Data Processing

[ ] ↳ 5 cells hidden

Saved 3-column file to /content/drive/MyDrive/Spring2025/IEOR242B/Project/notes_readmit.csv

Dataset DataLoader

[ ] ↳ 4 cells hidden

Training Process

Define Model

GRU

```
import torch.nn as nn

class ReadmitPredictor(nn.Module):
    def __init__(self, vocab_size, embed_dim=128, hidden_dim=64):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=PAD_IDX)
        self.gru = nn.GRU(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        emb = self.embedding(x)                # [B, T] → [B, T, D]
        _, h_n = self.gru(emb)                 # h_n: [1, B, H]
        h_final = h_n.squeeze(0)               # [B, H]
        out = self.fc(h_final)                 # [B, 1]
        return self.sigmoid(out).squeeze(1), h_final  # return both prob and h_T
```

Simple Alternative to GRU

[ ] ↳ 1 cell hidden

Improved Alternative (Imbalance Dataset)

[ ] ↳ 1 cell hidden

Training Loop

[ ] ↳ 5 cells hidden

Save Final Hidden States (h_T) to Disk

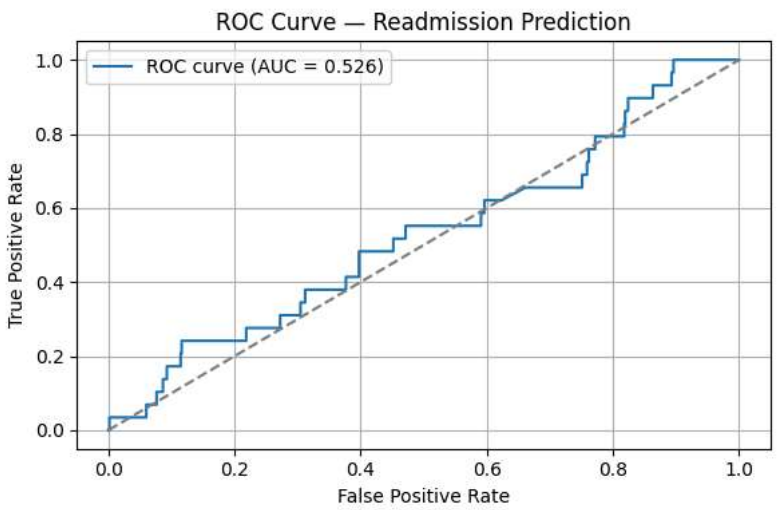
[ ] ↳ 1 cell hidden

Saved h_T vectors and labels to drive.

Plot ROC-AUC on Validation

```
# from sklearn.metrics import roc_auc_score, roc_curve
# import matplotlib.pyplot as plt
```

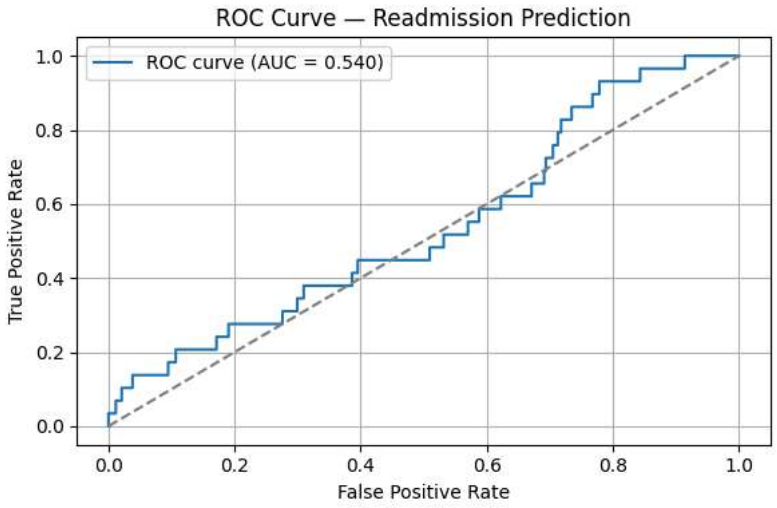
```
# Compute predicted probs for validation set
model.eval()
all_probs = []
all_labels = []
with torch.no_grad():
    for x, y in val_loader:
        x = x.to(device)
        probs, _ = model(x)
        all_probs.extend(probs.cpu().numpy())
        all_labels.extend(y.numpy())

auc = roc_auc_score(all_labels, all_probs)
fpr, tpr, _ = roc_curve(all_labels, all_probs)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.3f})')
plt.plot([0, 1], [0, 1], '--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve — Readmission Prediction')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



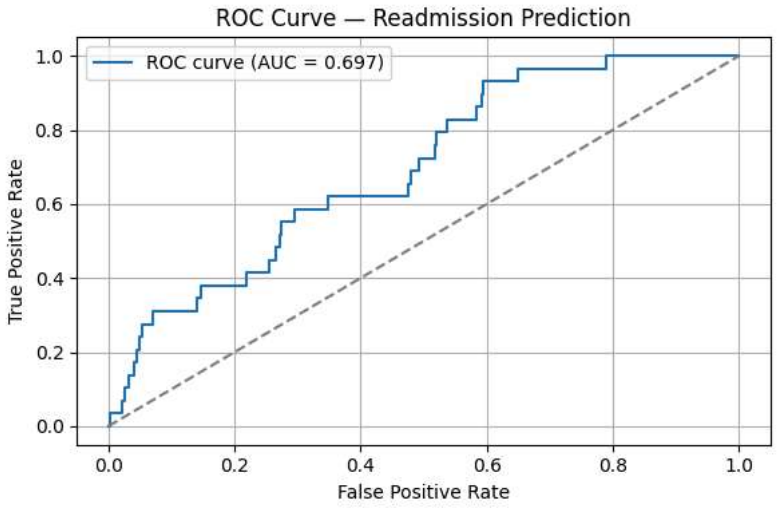

```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

def plot_roc_auc(model, val_loader, device, return_auc=False):
    from sklearn.metrics import roc_auc_score, roc_curve
    import matplotlib.pyplot as plt

    model.eval()
    all_probs = []
    all_labels = []
    with torch.no_grad():
        for x, y in val_loader:
            x = x.to(device)
            logits, _ = model(x)
            probs = torch.sigmoid(logits)
            all_probs.extend(probs.cpu().numpy())
            all_labels.extend(y.numpy())

    auc = roc_auc_score(all_labels, all_probs)
    fpr, tpr, _ = roc_curve(all_labels, all_probs)

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.3f})')
    plt.plot([0, 1], [0, 1], '--', color='gray')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve — Readmission Prediction')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    if return_auc:
```

```
    return auc
```

```
plot_roc_auc(model, val_loader)
```

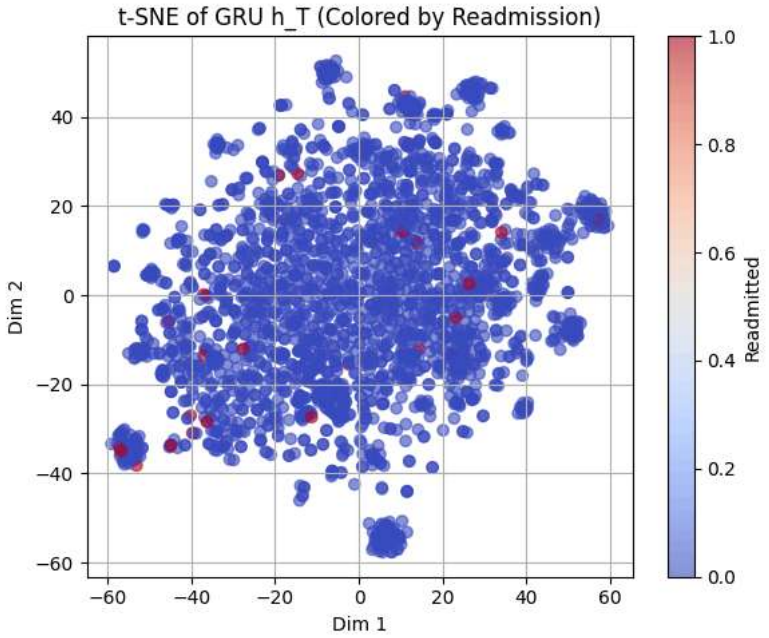



```
plot_roc_auc(model_2, val_loader)
```



Visualize Hidden States (h_T) with PCA / t-SNE

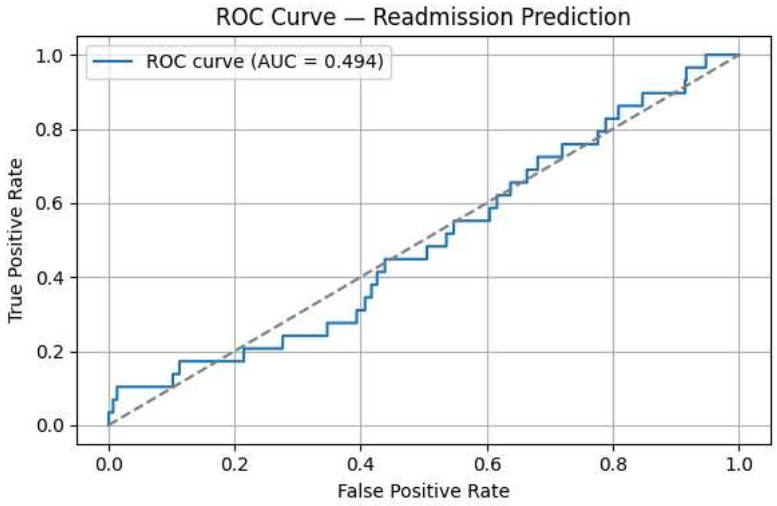
```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

# Load embeddings from disk (or use ht_matrix)
ht = np.load("/content/drive/MyDrive/Spring2025/IEOR242B/Project/val_hidden.npy")
labels = np.load("/content/drive/MyDrive/Spring2025/IEOR242B/Project/val_labels.npy")

# First reduce with PCA to denoise, then t-SNE
pca = PCA(n_components=50).fit_transform(ht)
tsne = TSNE(n_components=2, perplexity=30, random_state=42).fit_transform(pca)

# Plot
plt.figure(figsize=(6, 5))
plt.scatter(tsne[:,0], tsne[:,1], c=labels, cmap='coolwarm', alpha=0.6)
plt.title('t-SNE of GRU h_T (Colored by Readmission)')
plt.xlabel('Dim 1')
plt.ylabel('Dim 2')
plt.colorbar(label='Readmitted')
plt.grid(True)
plt.tight_layout()
plt.show()
```



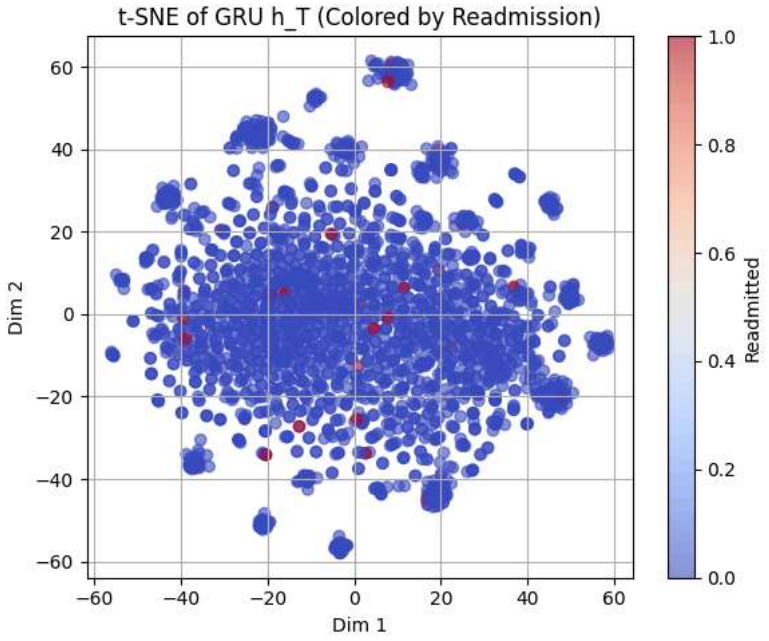

```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

def visualize_ht_tsne(model, val_loader, device, save=False, path_prefix=base_dir):
    model.eval()
    all_ht = []
    all_labels = []
    with torch.no_grad():
        for x, y in val_loader:
            x = x.to(device)
            _, h_t = model(x)
            all_ht.append(h_t.cpu())
            all_labels.extend(y.numpy())

    ht_matrix = torch.cat(all_ht).numpy()
    labels = np.array(all_labels)

    # Optional: save for reuse
    if save:
        np.save(f"{path_prefix}val_hidden.npy", ht_matrix)
        np.save(f"{path_prefix}val_labels.npy", labels)
        print(f"Saved embeddings and labels to {path_prefix}*.npy")

    # Dim reduction: PCA → t-SNE
    pca = PCA(n_components=50).fit_transform(ht_matrix)
    tsne = TSNE(n_components=2, perplexity=30, random_state=42).fit_transform(pca)

    # Plot
    plt.figure(figsize=(6, 5))
    plt.scatter(tsne[:, 0], tsne[:, 1], c=labels, cmap='coolwarm', alpha=0.6)
    plt.title('t-SNE of GRU h_T (Colored by Readmission)')
    plt.xlabel('Dim 1')
    plt.ylabel('Dim 2')
    plt.colorbar(label='Readmitted')
    plt.grid(True)
    plt.tight_layout()
    plt.show()


# Model setup
model = ReadmitPredictor(vocab_size, embed_dim=256, hidden_dim=128)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.BCELoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Train model
trained_model = train_model(model, train_loader, val_loader, optimizer, loss_fn, device)

# Plot AUC
plot_roc_auc(trained_model, val_loader, device)

# Visualize latent space
```

```
visualize_ht_tsne(trained_model, val_loader, device, save=False, path_prefix="/content/drive/MyDrive/Spring2025/IEOR242B/Project/")
```

```
Epoch 1 | Train Loss: 30.15 | Val Acc: 0.9888
Epoch 2 | Train Loss: 21.10 | Val Acc: 0.9884
```




```
Saved embeddings and labels to /content/drive/MyDrive/Spring2025/IEOR242B/Project/*.npy
```



Model Saving

```
def save_model(model, vocab, path_prefix):
    import torch
    model_path = f"{path_prefix}_text_model.pt"
    vocab_path = f"{path_prefix}_text_vocab.pkl"

    # Save weights
    torch.save(model.state_dict(), model_path)

    # Save vocab (optional but useful)
    import pickle
    with open(vocab_path, 'wb') as f:
        pickle.dump(vocab, f)

    print(f"Saved model to {model_path}")
    print(f"Saved vocab to {vocab_path}")
```

Explorations

Use Pos_weight

[ ] ↳ 1 cell hidden

Pos_weight Clip

[ ] ↳ 1 cell hidden

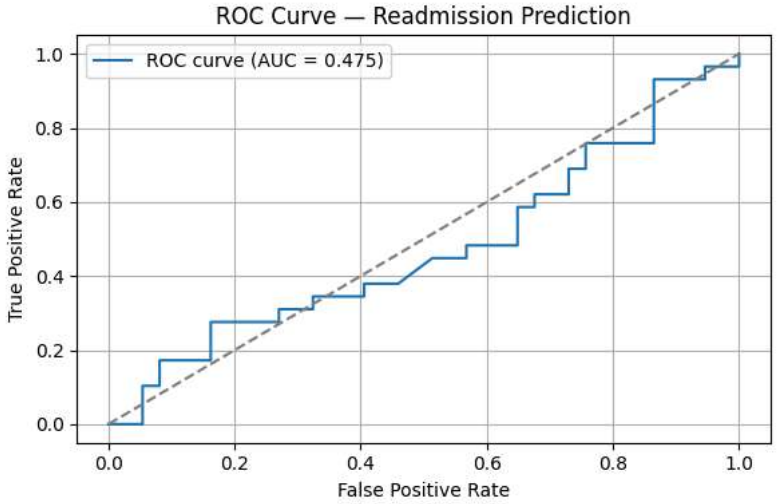
Update Downsampling

```
# Load full notes_readmit.csv
df = pd.read_csv("/content/drive/MyDrive/Spring2025/IEOR242B/Project/notes_readmit.csv")

# Compute pos_weight on full label dist
pos_weight = compute_pos_weight(df['readmitted'].tolist(), max_clip=20.0).to(device)

# Downsample to balanced dataset
df_bal = downsample_negatives(df)

# Proceed with balanced df
train_texts_bal, val_texts_bal, train_labels_bal, val_labels_bal = train_test_split(
    df_bal['text_note'].tolist(), df_bal['readmitted'].tolist(), test_size=0.2, random_state=42
)

# Create vocab, dataset, dataloaders (as before)

# Simple whitespace tokenizer
def tokenize(text):
    return text.lower().split()

# Build vocab (keep tokens with frequency ≥2)
counter_bal = Counter()
for text in df_bal['text_note']:
    counter_bal.update(tokenize(text))

special_tokens = ['<PAD>', '<UNK>']
vocab_bal = {tok: idx for idx, tok in enumerate(special_tokens)}
for word, freq in counter.items():
    if freq >= 2:
        vocab_bal[word] = len(vocab_bal)

vocab_bal_size = len(vocab_bal)
print(f"Vocab size: {vocab_bal_size}")

import torch
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence

MAX_LEN = 256  # truncate long notes
PAD_IDX_bal = vocab_bal['<PAD>']
UNK_IDX_bal = vocab_bal['<UNK>']

class NotesDataset(Dataset):
    def __init__(self, texts, labels, vocab, max_len=MAX_LEN):
        self.texts = texts
        self.labels = labels
        self.vocab = vocab
        self.max_len = max_len

    def __getitem__(self, idx):
        tokens = tokenize(self.texts[idx])
        idxs = [self.vocab.get(tok, UNK_IDX) for tok in tokens[:self.max_len]]
        return torch.tensor(idxs, dtype=torch.long), torch.tensor(self.labels[idx], dtype=torch.float)

    def __len__(self):
        return len(self.texts)

def collate_fn(batch):
    inputs, labels = zip(*batch)
    padded = pad_sequence(inputs, batch_first=True, padding_value=PAD_IDX)
    return padded, torch.stack(labels)

# Create loaders
train_bal_ds = NotesDataset(train_texts_bal, train_labels_bal, vocab_bal)
val_bal_ds   = NotesDataset(val_texts_bal, val_labels_bal, vocab_bal)

train_bal_loader = DataLoader(train_bal_ds, batch_size=32, shuffle=True, collate_fn=collate_fn)
val_bal_loader   = DataLoader(val_bal_ds, batch_size=32, shuffle=False, collate_fn=collate_fn)

print(f"Train batches: {len(train_bal_loader)}, Val batches: {len(val_bal_loader)}")
# Train batches: 325, Val batches: 82 without downsample
```

```
Vocab size: 175107
Train batches: 9, Val batches: 3
```

```
# Prepare model and optimizer
model_bal = ReadmitPredictor(vocab_bal_size, embed_dim=256, hidden_dim=128)
optimizer_bal = torch.optim.Adam(model_bal.parameters(), lr=1e-3)

# Compute pos_weight
pos_bal = df_bal['readmitted'].sum()
neg_bal = len(df_bal) - pos_bal

adjusted_weight_bal = min(neg_bal / pos_bal, 20.0)
loss_fn_bal = nn.BCEWithLogitsLoss(pos_weight=torch.tensor([adjusted_weight_bal]).to(device))

trained_model_bal = train_model(model_bal, train_bal_loader, val_bal_loader, optimizer_bal, loss_fn_bal, device, epochs=3)
auc_bal = plot_roc_auc(trained_model_bal, val_bal_loader, device, return_auc=True)
```

```
Epoch 1 | Train Loss: 6.21 | Val Acc: 0.4242
Epoch 2 | Train Loss: 5.39 | Val Acc: 0.4242
```




```
save_model(model_bal, vocab_bal, path_prefix=f"{base_dir}/readmit")
```

```
Saved model to /content/drive/MyDrive/Spring2025/IEOR242B/Project/readmit_text_model.pt
Saved vocab to /content/drive/MyDrive/Spring2025/IEOR242B/Project/readmit_text_vocab.pkl
```

Time-Series FFNN Baseline

Data Extraction

```
import pandas as pd

# Paths
base = "/content/drive/MyDrive/Spring2025/IEOR242B/Project/"
lab_path = base + "LABEVENTS.csv"
input_mv_path = base + "INPUTEVENTS_MV.csv"
input_cv_path = base + "INPUTEVENTS_CV.csv"
output_path = base + "OUTPUTEVENTS.csv"
proc_path = base + "PROCEDUREEVENTS_MV.csv"
label_path = base + "notes_readmit.csv"

# Load core files
labs = pd.read_csv(lab_path, usecols=['SUBJECT_ID','HADM_ID','ITEMID','VALUENUM'])
inputs_mv = pd.read_csv(input_mv_path, usecols=['SUBJECT_ID','HADM_ID','ITEMID','AMOUNT'])
inputs_cv = pd.read_csv(input_cv_path, usecols=['SUBJECT_ID','HADM_ID','ITEMID','AMOUNT'])
outputs = pd.read_csv(output_path, usecols=['SUBJECT_ID','HADM_ID','ITEMID','VALUE'])
procedures = pd.read_csv(proc_path, usecols=['SUBJECT_ID','HADM_ID','ITEMID','VALUE'])
labels = pd.read_csv(label_path, usecols=['patient_id','readmitted'])  # from text baseline

# Preview
print(f"Labs: {len(labs)} | Inputs: {len(inputs_mv)+len(inputs_cv)} | Outputs: {len(outputs)} | Procs: {len(procedures)}")
```

```
Labs: 6089629 | Inputs: 8603563 | Outputs: 1028495 | Procs: 148178
```

Data Preprocessing

[ ] ↳ 1 cell hidden

Merge Features + Labels

[ ] ↳ 1 cell hidden

Build FFNN Dataset

[ ] ↳ 1 cell hidden

Model Construction

[ ] ↳ 1 cell hidden

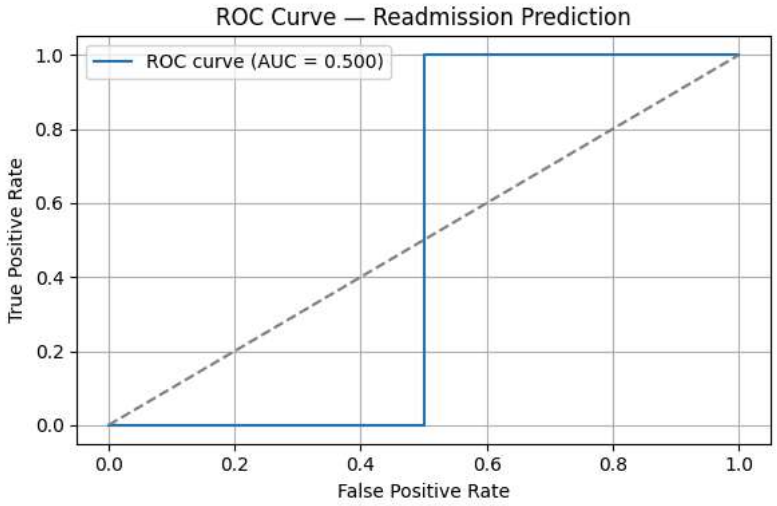
Update Method

[ ] ↳ 2 cells hidden

Training Appraoch

```
input_dim = X_train.shape[1]
model = TabularFFNN(input_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# Reuse pos_weight helper
pos_weight = compute_pos_weight(y_train.tolist(), max_clip=20.0).to(device)
loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

model = train_model(model, train_loader, val_loader, optimizer, loss_fn, device, epochs=2)
plot_roc_auc(model, val_loader, device)
```

```
Epoch 1 | Train Loss: 1.29 | Val Acc: 0.5714
Epoch 2 | Train Loss: 1.27 | Val Acc: 0.5714
```



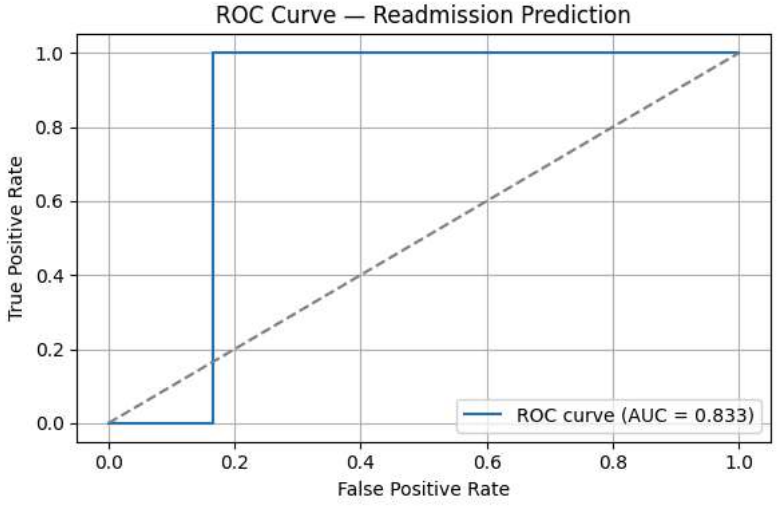

```
# (1) Define and train tabular FFNN
input_dim = X_train.shape[1]
model = TabularFFNN(input_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

# Reuse pos_weight helper
pos_weight = compute_pos_weight(y_train.tolist(), max_clip=20.0).to(device)
loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

model = train_model(model, train_loader, val_loader, optimizer, loss_fn, device, epochs=200)

# (2) Plot ROC-AUC
plot_roc_auc(model, val_loader, device)

# (3) Save tabular embeddings + t-SNE visualization
```

```
save_tabular_embeddings_and_visualize(
    model=model,
    val_loader=val_loader,
    device=device,
    save=True,
    path_prefix="/content/drive/MyDrive/Spring2025/IEOR242B/Project/"
)

# (4) Save model + vocab
save_model(
    model=model,
    vocab={},  # if you have a tabular feature index map, pass it here
    path_prefix="/content/drive/MyDrive/Spring2025/IEOR242B/Project/readmit_tabular"
)
```

```
Epoch 2 | Train Loss: 1.30 | Val Acc: 0.1429
Epoch 3 | Train Loss: 1.30 | Val Acc: 0.1429
Epoch 4 | Train Loss: 1.30 | Val Acc: 0.1429
Epoch 5 | Train Loss: 1.30 | Val Acc: 0.1429
Epoch 6 | Train Loss: 1.30 | Val Acc: 0.1429
Epoch 7 | Train Loss: 1.30 | Val Acc: 0.1429
Epoch 8 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 9 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 10 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 11 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 12 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 13 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 14 | Train Loss: 1.29 | Val Acc: 0.1429
Epoch 15 | Train Loss: 1.28 | Val Acc: 0.1429
Epoch 16 | Train Loss: 1.28 | Val Acc: 0.1429
Epoch 17 | Train Loss: 1.28 | Val Acc: 0.1429
Epoch 18 | Train Loss: 1.28 | Val Acc: 0.1429
Epoch 19 | Train Loss: 1.28 | Val Acc: 0.1429
Epoch 20 | Train Loss: 1.28 | Val Acc: 0.1429
Epoch 21 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 22 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 23 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 24 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 25 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 26 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 27 | Train Loss: 1.27 | Val Acc: 0.1429
Epoch 28 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 29 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 30 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 31 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 32 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 33 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 34 | Train Loss: 1.26 | Val Acc: 0.1429
Epoch 35 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 36 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 37 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 38 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 39 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 40 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 41 | Train Loss: 1.25 | Val Acc: 0.1429
Epoch 42 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 43 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 44 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 45 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 46 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 47 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 48 | Train Loss: 1.24 | Val Acc: 0.1429
Epoch 49 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 50 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 51 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 52 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 53 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 54 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 55 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 56 | Train Loss: 1.23 | Val Acc: 0.1429
Epoch 57 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 58 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 59 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 60 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 61 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 62 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 63 | Train Loss: 1.22 | Val Acc: 0.1429
Epoch 64 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 65 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 66 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 67 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 68 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 69 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 70 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 71 | Train Loss: 1.21 | Val Acc: 0.1429
Epoch 72 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 73 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 74 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 75 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 76 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 77 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 78 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 79 | Train Loss: 1.20 | Val Acc: 0.1429
Epoch 80 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 81 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 82 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 83 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 84 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 85 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 86 | Train Loss: 1.19 | Val Acc: 0.1429
Epoch 87 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 88 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 89 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 90 | Train Loss: 1.18 | Val Acc: 0.1429
```

```
Epoch 91 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 92 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 93 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 94 | Train Loss: 1.18 | Val Acc: 0.1429
Epoch 95 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 96 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 97 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 98 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 99 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 100 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 101 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 102 | Train Loss: 1.17 | Val Acc: 0.1429
Epoch 103 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 104 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 105 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 106 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 107 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 108 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 109 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 110 | Train Loss: 1.16 | Val Acc: 0.1429
Epoch 111 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 112 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 113 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 114 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 115 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 116 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 117 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 118 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 119 | Train Loss: 1.15 | Val Acc: 0.1429
Epoch 120 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 121 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 122 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 123 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 124 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 125 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 126 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 127 | Train Loss: 1.14 | Val Acc: 0.1429
Epoch 128 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 129 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 130 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 131 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 132 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 133 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 134 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 135 | Train Loss: 1.13 | Val Acc: 0.1429
Epoch 136 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 137 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 138 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 139 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 140 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 141 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 142 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 143 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 144 | Train Loss: 1.12 | Val Acc: 0.1429
Epoch 145 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 146 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 147 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 148 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 149 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 150 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 151 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 152 | Train Loss: 1.11 | Val Acc: 0.1429
Epoch 153 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 154 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 155 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 156 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 157 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 158 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 159 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 160 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 161 | Train Loss: 1.10 | Val Acc: 0.1429
Epoch 162 | Train Loss: 1.09 | Val Acc: 0.1429
Epoch 163 | Train Loss: 1.09 | Val Acc: 0.1429
Epoch 164 | Train Loss: 1.09 | Val Acc: 0.1429
Epoch 165 | Train Loss: 1.09 | Val Acc: 0.1429
Epoch 166 | Train Loss: 1.09 | Val Acc: 0.1429
Epoch 167 | Train Loss: 1.09 | Val Acc: 0.2857
Epoch 168 | Train Loss: 1.09 | Val Acc: 0.2857
Epoch 169 | Train Loss: 1.09 | Val Acc: 0.2857
Epoch 170 | Train Loss: 1.09 | Val Acc: 0.2857
Epoch 171 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 172 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 173 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 174 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 175 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 176 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 177 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 178 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 179 | Train Loss: 1.08 | Val Acc: 0.2857
Epoch 180 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 181 | Train Loss: 1.07 | Val Acc: 0.2857
```

```
Epoch 181 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 182 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 183 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 184 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 185 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 186 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 187 | Train Loss: 1.07 | Val Acc: 0.2857
Epoch 188 | Train Loss: 1.07 | Val Acc: 0.4286
Epoch 189 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 190 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 191 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 192 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 193 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 194 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 195 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 196 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 197 | Train Loss: 1.06 | Val Acc: 0.4286
Epoch 198 | Train Loss: 1.05 | Val Acc: 0.4286
Epoch 199 | Train Loss: 1.05 | Val Acc: 0.4286
Epoch 200 | Train Loss: 1.05 | Val Acc: 0.4286
```




```
Saved embeddings and labels to /content/drive/MyDrive/Spring2025/IEOR242B/Project/val_tabular_emb.npy and /content/drive/MyDrive
Saved FFNN model weights to /content/drive/MyDrive/Spring2025/IEOR242B/Project/readmit_tabular_ffnn_model.pt
Saved vocab to /content/drive/MyDrive/Spring2025/IEOR242B/Project/readmit_tabular_vocab.pkl
```

Multimodal Final

Load pretrained embeddings and labels for train/val sets

```
import torch, pickle, numpy as np, pandas as pd
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import StandardScaler

# Paths
base = "/content/drive/MyDrive/Spring2025/IEOR242B/Project/"
text_model_path    = base + "readmit_text_model.pt"
text_vocab_path    = base + "readmit_text_vocab.pkl"
tab_model_path     = base + "readmit_tabular_ffnn_model.pt"
tab_vocab_path     = base + "readmit_tabular_vocab.pkl"   # if you saved it
notes_path         = base + "notes_readmit.csv"
features_path      = base + "aggregated_tabular_features.csv"        # wherever you saved your aggregated features
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

base = "/content/drive/MyDrive/Spring2025/IEOR242B/Project/"

# Load raw CSVs (use cols to reduce memory)
labs = pd.read_csv(base + "LABEVENTS.csv", usecols=['HADM_ID','VALUENUM'])
inputs_mv = pd.read_csv(base + "INPUTEVENTS_MV.csv", usecols=['HADM_ID','AMOUNT'])
inputs_cv = pd.read_csv(base + "INPUTEVENTS_CV.csv", usecols=['HADM_ID','AMOUNT'])
outputs = pd.read_csv(base + "OUTPUTEVENTS.csv", usecols=['HADM_ID','VALUE'])
procedures = pd.read_csv(base + "PROCEDUREEVENTS_MV.csv", usecols=['HADM_ID','VALUE'])

def aggregate(df, col, prefix):
    agg = df.groupby('HADM_ID')[col].agg(['mean', 'std', 'min', 'max', 'count'])
    agg.columns = [f"{prefix}_{c}" for c in agg.columns]
    return agg

lab_feats = aggregate(labs, 'VALUENUM', 'lab')
input_feats = aggregate(pd.concat([inputs_mv, inputs_cv]), 'AMOUNT', 'input')
output_feats = aggregate(outputs, 'VALUE', 'output')
proc_feats = aggregate(procedures, 'VALUE', 'proc')

# Merge all aggregated features on HADM_ID
features = lab_feats.join([input_feats, output_feats, proc_feats], how='outer').reset_index()

# Load admission labels (from notes_readmit.csv join or separately)
notes_df = pd.read_csv(base + "notes_readmit.csv")
admits = pd.read_csv(base + "ADMISSIONS.csv", usecols=['SUBJECT_ID','HADM_ID'])
labels_df = pd.merge(admits, notes_df[['patient_id','readmitted']], left_on='SUBJECT_ID', right_on='patient_id', how='inner')

# Merge labels into features by HADM_ID
full_df = pd.merge(features, labels_df[['HADM_ID','readmitted']], on='HADM_ID', how='inner')

# Drop rows with missing data (if any)
full_df = full_df.dropna().reset_index(drop=True)

# Normalize features
scaler = StandardScaler()
X = full_df.drop(columns=['HADM_ID','readmitted'])
X_scaled = scaler.fit_transform(X)

# Save for later use
full_df_scaled = pd.DataFrame(X_scaled, columns=X.columns)
full_df_scaled['readmitted'] = full_df['readmitted']
full_df_scaled['HADM_ID'] = full_df['HADM_ID']
full_df_scaled.to_csv(base + "aggregated_tabular_features.csv", index=False)

print("Aggregated tabular features saved to aggregated_tabular_features.csv")
print(f"Shape: {full_df_scaled.shape}")
```

```
Aggregated tabular features saved to aggregated_tabular_features.csv
Shape: (34, 22)
```

Extraction

```
import pandas as pd

base = "/content/drive/MyDrive/Spring2025/IEOR242B/Project/"
```

```
# Load raw text notes + labels
df_notes = pd.read_csv(base + "notes_readmit.csv")  # patient_id, text_note, readmitted

# Load tabular features and join with notes by HADM_ID
df_admits = pd.read_csv(base + "ADMISSIONS.csv", usecols=["SUBJECT_ID", "HADM_ID"])
df_features = pd.read_csv(base + "aggregated_tabular_features.csv")  # created earlier

# Merge all on HADM_ID
df_all = pd.merge(df_admits, df_notes, left_on="SUBJECT_ID", right_on="patient_id", how="inner")
df_all = pd.merge(df_all, df_features, on="HADM_ID", how="inner")

# Final clean table
df_all = df_all.dropna().reset_index(drop=True)
print("Final multimodal fusion table shape:", df_all.shape)
```

```
Final multimodal fusion table shape: (84, 26)
```

```
import torch, pickle
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence

# Load vocab and model
with open(base + "readmit_text_vocab.pkl", "rb") as f:
    vocab = pickle.load(f)

PAD_IDX = vocab['<PAD>']
UNK_IDX = vocab['<UNK>']

class TextDataset(Dataset):
    def __init__(self, texts, labels, vocab, max_len=256):
        self.texts = texts
        self.labels = labels
        self.vocab = vocab
        self.max_len = max_len
    def __len__(self): return len(self.texts)
    def __getitem__(self, idx):
        toks = self.texts[idx].lower().split()[:self.max_len]
        ids = [self.vocab.get(tok, UNK_IDX) for tok in toks]
        return torch.tensor(ids, dtype=torch.long), torch.tensor(self.labels[idx], dtype=torch.float)

def collate_text(batch):
    sequences, labels = zip(*batch)
    return pad_sequence(sequences, batch_first=True, padding_value=PAD_IDX), torch.stack(labels)

# Dataset
text_ds = TextDataset(df_all['text_note'].tolist(), df_all['readmitted_y'].tolist(), vocab)
text_loader = DataLoader(text_ds, batch_size=32, collate_fn=collate_text)

# Load trained model
class ReadmitPredictor(torch.nn.Module):
    def __init__(self, vocab_size, embed_dim=256, hidden_dim=128):
        super().__init__()
        self.embedding = torch.nn.Embedding(vocab_size, embed_dim, padding_idx=PAD_IDX)
        self.gru = torch.nn.GRU(embed_dim, hidden_dim, batch_first=True)
        self.fc = torch.nn.Linear(hidden_dim, 1)
    def forward(self, x):
        emb = self.embedding(x)
        _, h_n = self.gru(emb)
        return self.fc(h_n.squeeze(0)).squeeze(1), h_n.squeeze(0)

text_model = ReadmitPredictor(len(vocab)).to("cpu")
text_model.load_state_dict(torch.load(base + "readmit_text_model.pt", map_location="cpu"))
text_model.eval()

# Extract final h_T
all_text_ht = []
with torch.no_grad():
    for x, y in text_loader:
        _, h = text_model(x)
        all_text_ht.append(h)

text_emb = torch.cat(all_text_ht).numpy()
print("Text embedding shape:", text_emb.shape)
```

```
Text embedding shape: (84, 128)
```

```
from sklearn.preprocessing import StandardScaler
import numpy as np
```

```
X_tab = df_all.drop(columns=["SUBJECT_ID", "HADM_ID", "patient_id", "text_note", "readmitted_y"])
y = df_all['readmitted_y'].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_tab)

class TabularFFNN(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim=64):
        super().__init__()
        self.net = torch.nn.Sequential(
            torch.nn.Linear(input_dim, hidden_dim),
            torch.nn.ReLU(),
            torch.nn.Linear(hidden_dim, 1)
        )
    def forward(self, x):
        h = self.net[0](x)
        h = self.net[1](h)
        return self.net[2](h).squeeze(1), h

tab_model = TabularFFNN(X_scaled.shape[1]).to("cpu")
tab_model.load_state_dict(torch.load(base + "readmit_tabular_ffnn_model.pt", map_location="cpu"))
tab_model.eval()

# Extract embeddings
X_tensor = torch.tensor(X_scaled, dtype=torch.float32)
with torch.no_grad():
    _, tab_emb = tab_model(X_tensor)

tab_emb = tab_emb.numpy()
print("Tabular embedding shape:", tab_emb.shape)
```

```
Tabular embedding shape: (84, 64)
```

```
np.save(base + "all_text_emb.npy", text_emb)
np.save(base + "all_tabular_emb.npy", tab_emb)
np.save(base + "all_labels.npy", y)
print("Saved all_text_emb.npy, all_tabular_emb.npy, all_labels.npy")
```

```
Saved all_text_emb.npy, all_tabular_emb.npy, all_labels.npy
```

continue

```
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import numpy as np

# 1) Load the full embeddings + labels (not just the val subset)
text_emb = torch.from_numpy(np.load('/content/drive/MyDrive/Spring2025/IEOR242B/Project/all_text_emb.npy')).float()
tab_emb  = torch.from_numpy(np.load('/content/drive/MyDrive/Spring2025/IEOR242B/Project/all_tabular_emb.npy')).float()
labels   = torch.from_numpy(np.load('/content/drive/MyDrive/Spring2025/IEOR242B/Project/all_labels.npy')).float()

# 2) Stratified train/val split on the full dataset
train_idx, val_idx = train_test_split(
    np.arange(len(labels)),
    test_size=0.2,
    random_state=42,
    stratify=labels.numpy()
)

# 3) Quick sanity check
print("Train positives:", int(labels[train_idx].sum().item()), "/", len(train_idx))
print("Val   positives:",  int(labels[val_idx].sum().item()),    "/", len(val_idx))

# 4) Build Dataset & DataLoader
class FusionDataset(Dataset):
    def __init__(self, text_emb, tab_emb, labels, idxs):
        self.text = text_emb[idxs]
        self.tab  = tab_emb[idxs]
        self.lab  = labels[idxs]
    def __len__(self): return len(self.lab)
    def __getitem__(self, i):
        return self.text[i], self.tab[i], self.lab[i]

train_ds = FusionDataset(text_emb, tab_emb, labels, train_idx)
val_ds   = FusionDataset(text_emb, tab_emb, labels, val_idx)
```