CIS 602 Fundamentals of Deep Learning
Fall 2025

Project 5
TinyTransformer & Morse Code Character Recognition
Due: December 12, 2025

Michael Evan 02081213
Graduate Computer Science Department
UMassDartmouth

Abstract

Project 4 implements an encoder-only PyTorch TinyTransformer to classify single-character Morse code from audio. A synthetic, balanced dataset of dits and dahs was generated at 16 kHz, converted to mel-spectrograms, and augmented with noise and tempo jitter; data were split 80/10/10 (train/val/test) and normalized using training statistics. The encoder maps time-frequency frames to character logits and is trained with Adam Optimizer, cross-entropy loss, gradient clipping, and validation-based checkpointing; test results show a strongly diagonal confusion matrix and high accuracy on the synthetic data. Future work includes collecting human-recorded Morse to close the sim-to-real gap and adding light temporal language modeling for multi-character sequences.

Introduction

Relevance & Importance

Being a Ham Operator who uses Morse code, so building a Transformer that decodes variable-speed, noisy synthetic data matters for real-world, on-air decoding and practice. Accurate, on-device decoding could be utilized for improved offline emergency communications, accessibility, and provides a reproducible dataset/method for robust low-resource morse code communications.

Task Definition

Input: Input to the model is .wav audio turned into a log-mel spectrogram represented in a 2D array where rows are short time slices (every 10 ms at 16 kHz) and columns are mel frequency bins (128 bands). Each sound clip is converted to this time x frequency array, converted to decibels, then centered/padded so every sample has the same shape and normalized using training statistics so the model sees consistent inputs. This representation captures the tone energy of morse code dots and dash's ("dits" and "dahs), while ignoring raw waveform quirks, letting the Transformer learn to map each spectrogram to one of the 36 target characters: a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9.

Output: Accuracy of predictability of test data set of 36 unique synthetic morse code character samples including additional statistical metrics for evaluation.

Goal: Train a compact TinyTransformer Encoder to map log-mel spectrograms of single-character Morse Code signals to one of 36 labels (a–z, 0–9). The project emphasizes robustness to speed and noise and demonstrates near-perfect performance on the synthetic dataset (see mel examples and statistical metrics), with the end goal of reliable, offline decoding for ham radio morse operations.

Major Contributions

- Generated a balanced synthetic Morse dataset (36 classes: a–z, 0–9) with noise and tempo jitter.
- Converted audio to log-mel spectrograms and implemented train/val/test splits (80/10/10) with train-stats normalization.
- Built and trained a compact Transformer encoder (TinyTransformer) to classify single-character spectrograms.
- Added validation checkpointing and saved best checkpoints (best_model.pt).
- Evaluated performance with confusion matrices, per-class metrics, and debug visualizations (spectrogram samples, true vs predicted plots ...).

Methodology

Data set: The synthetic dataset is generated with the JupyterNotebook consisting of 36 individual morse code characters, a-z & 0-9, that are in the form of individual .wav files. The script generates 1000 unique files per character with jitter and slight random timing implemented to help simulate human morse code transmission.

Data prep: Load each .wav from per-class folders (one folder per label a-z,0-9), convert the audio to a log-mel spectrogram (a time × frequency “picture”), then center-crop or pad every spectrogram to the same frame length so the Tiny Transformer gets fixed-size inputs. Normalize using the training set mean/std, save that norm plus the exact index -> token mapping (vocab) and then create stratified

80/10/10 train/val/test splits which ensures inference using the exact same preprocessing and labels the model was trained with.

Neural Network Architecture: PyTorch TinyTransformer Encoder was utilized for training, validation, and testing. Compact, standard Transformer encoder (lightweight d_model/ff and only 2 layers) tuned for fast training and local on-device inference.

- **Front end**: linear projection from input mel bins (N_MELS, e.g. 128) to d_model (128).

- **Positional encoding**: used a learned positional embedding of shape 1 × POS_LEN × d_model.

- **Core**: TransformerEncoder with nlayers = 2 layers; each layer = multi-head self-attention (nhead typically 4, sometimes 8 if d_model divisible) + feed-forward of size ff = 256. TransformerEncoderLayer uses dropout (0.1) by default.

- **Readout**: mean-pool over time -> linear head (d_model -> nclass, 36 classes) -> logits (**softmax** for classifier or log-probs for CTC use).

- **Training** : **Adam optimizer**, gradient clipping (clip_grad_norm_), validation checkpointing (save **best_model.pt**), and data augmentation (additive noise + tempo jitter) used as regularization; no convolutions or extra LSTM layers.

Training:

- **Split:** stratified 80/10/10 (train/val/test).
- **Optimizer** Adam Optimizer, lr = 1e-3.
- **Batch size**: 32.
- **Epochs**: trained up to 20 (typical runs: 5 - 20); validate each epoch and save best checkpoint.
- **Loss:** CrossEntropyLoss for the classification setup (no class weighting used because training set is balanced).
- **Regularization**: TransformerEncoderLayer default dropout ~0.1, data augmentation (additive noise, jitter), gradient clipping via clip_grad_norm_.
- **Validation:** per-epoch eval on the val split; best_val checkpoint saved.

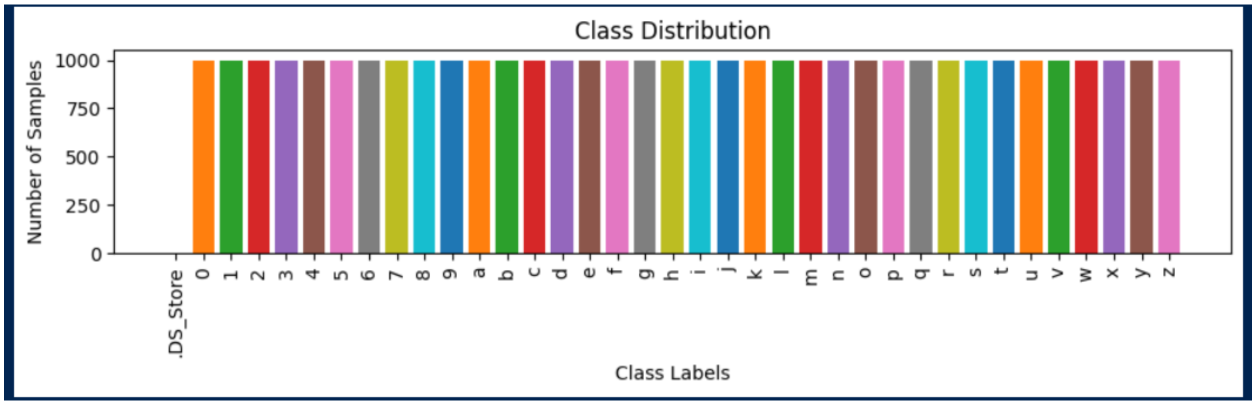
  note: no explicit early stopping beyond saving best epoch.

Figures for evaluation

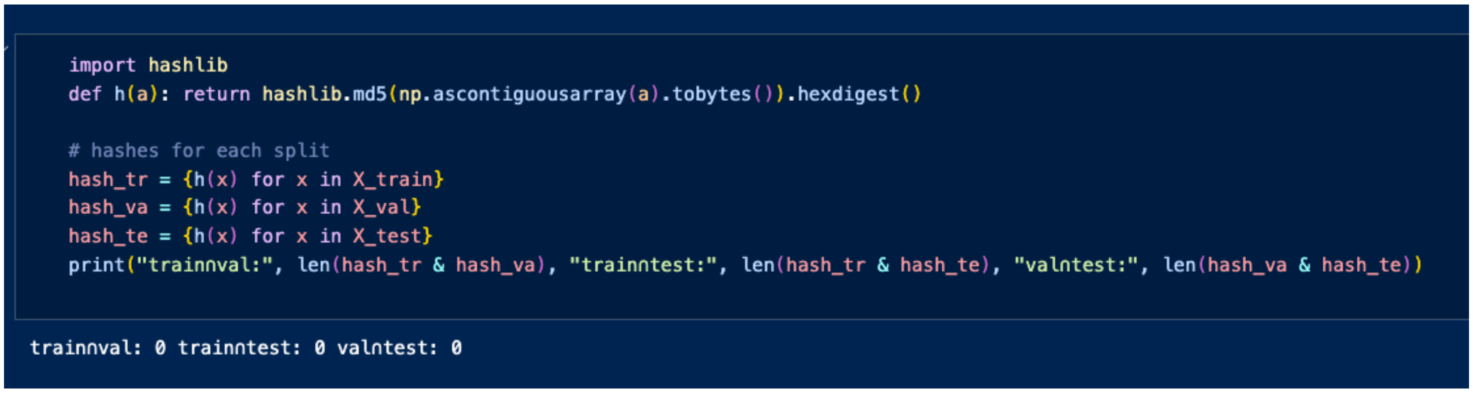
- **Class Distribution Bar Graph:** verifies dataset created; each class a-z & 0-9; class labels vs number samples.
- **Spectrogram / Audio Waveform comparison**: visual spectrogram; frames/ mel bins.
- **MD5 hashing algorithms:** verify no duplicates across class directories & train/val/test splits
- **Training/Validation accuracy**: plots Epoch/Loss for train_loss & val_loss
- **Validation Metric**: plots val_acc & val_f1 for each Epoch.
- **Confusion Matrix**
- **Scatter Plot**: True vs Predicted for each class with incorrect predictions noted as "x" including interactive 3D rotatable scatter plot.
- **Classification report table:** (precision / recall / F1 / support)
- **Cumulative variance plot**
- **PCA/UMAP 3D interactive plot**

Some examples below:

Dataset verification



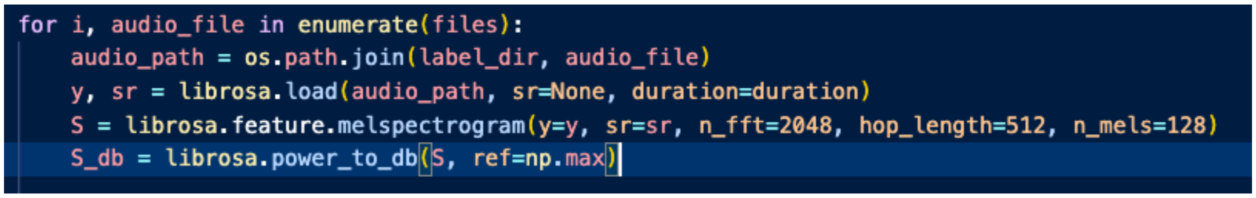
MD5 hashing algorithm



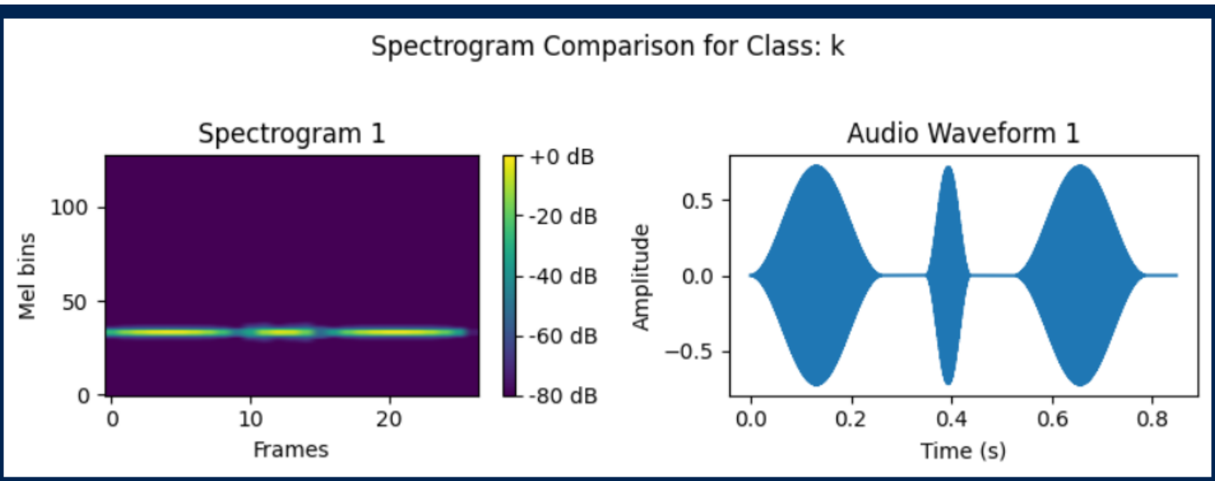
```
import hashlib
def h(a): return hashlib.md5(np.ascontiguousarray(a).tobytes()).hexdigest()

# hashes for each split
hash_tr = {h(x) for x in X_train}
hash_va = {h(x) for x in X_val}
hash_te = {h(x) for x in X_test}
print("trainnval:", len(hash_tr & hash_va), "trainntest:", len(hash_tr & hash_te), "valntest:", len(hash_va & hash_te))
```

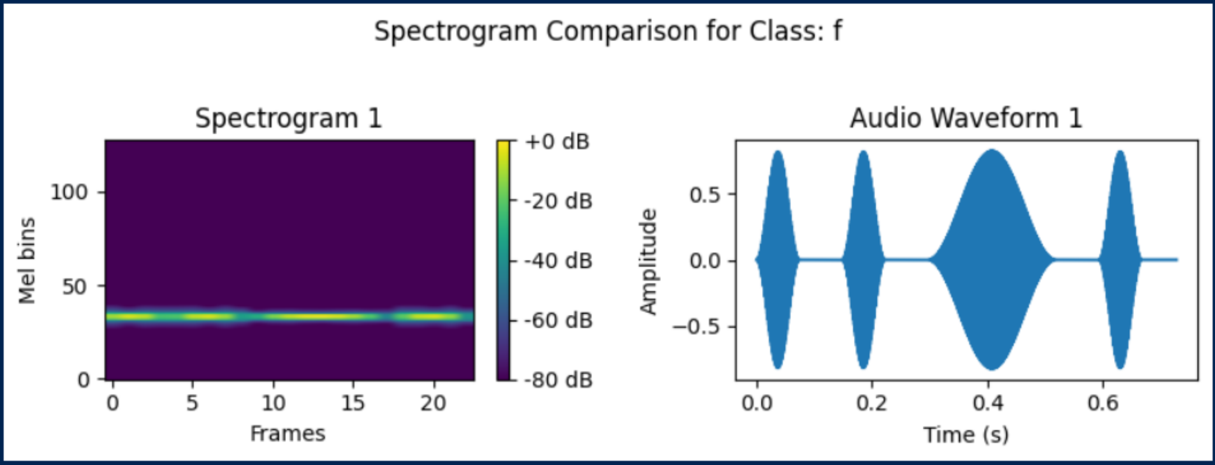
```
trainnval: 0 trainntest: 0 valntest: 0
```

Mel Spectrogram / Audio Waveform

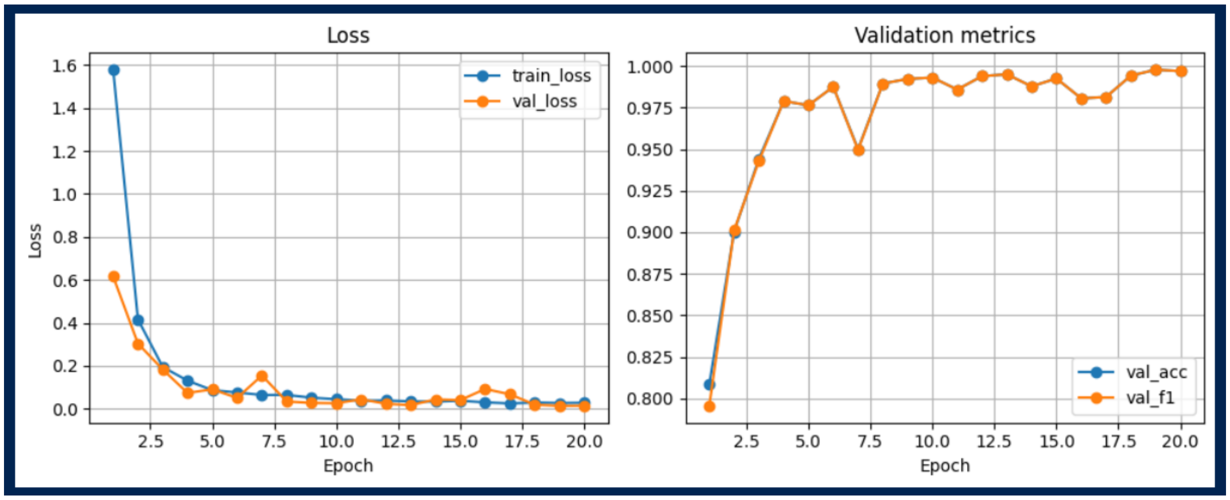


```python
for i, audio_file in enumerate(files):
    audio_path = os.path.join(label_dir, audio_file)
    y, sr = librosa.load(audio_path, sr=None, duration=duration)
    S = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=2048, hop_length=512, n_mels=128)
    S_db = librosa.power_to_db(S, ref=np.max)
```

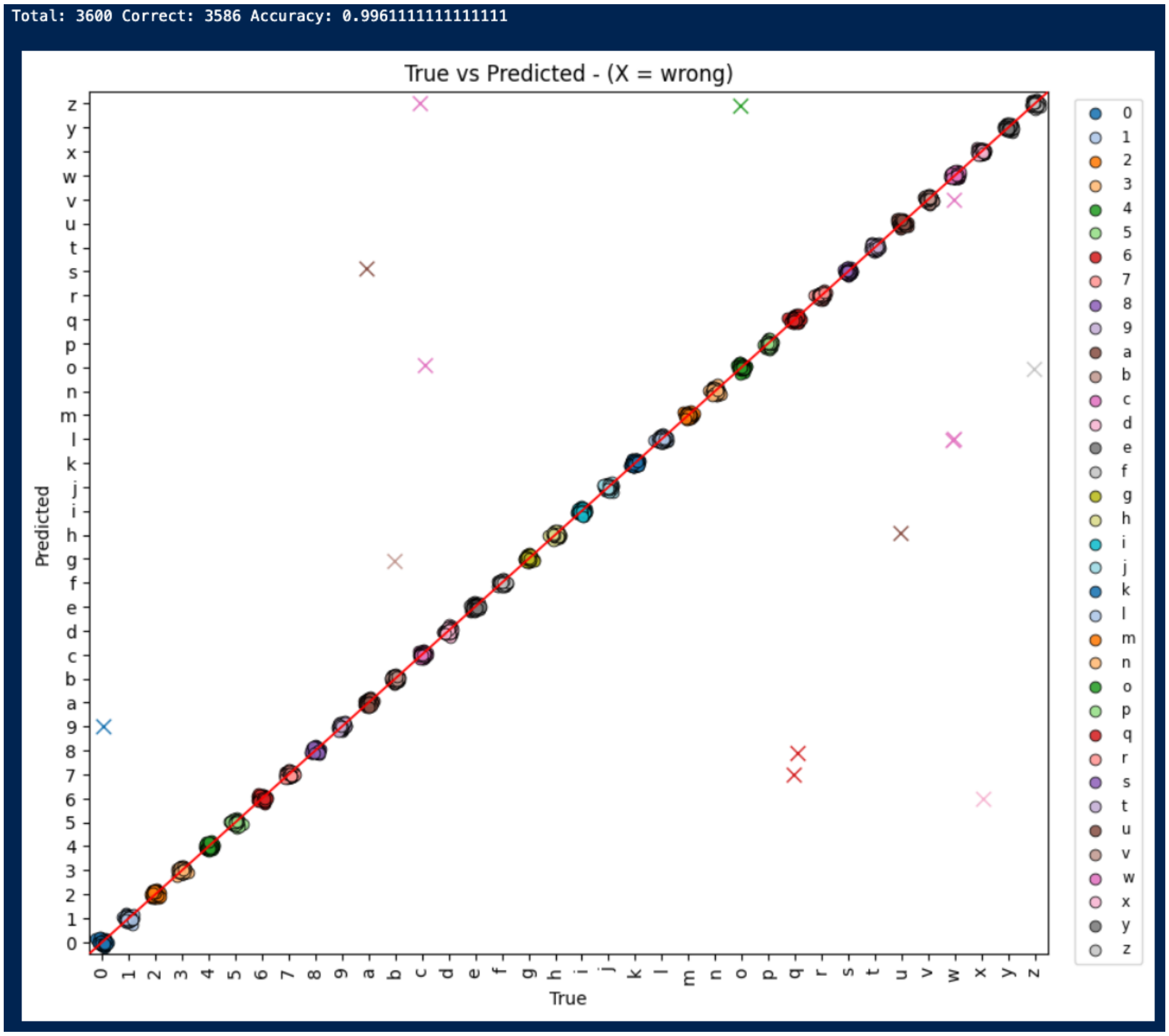




Learning curve / validation metrics

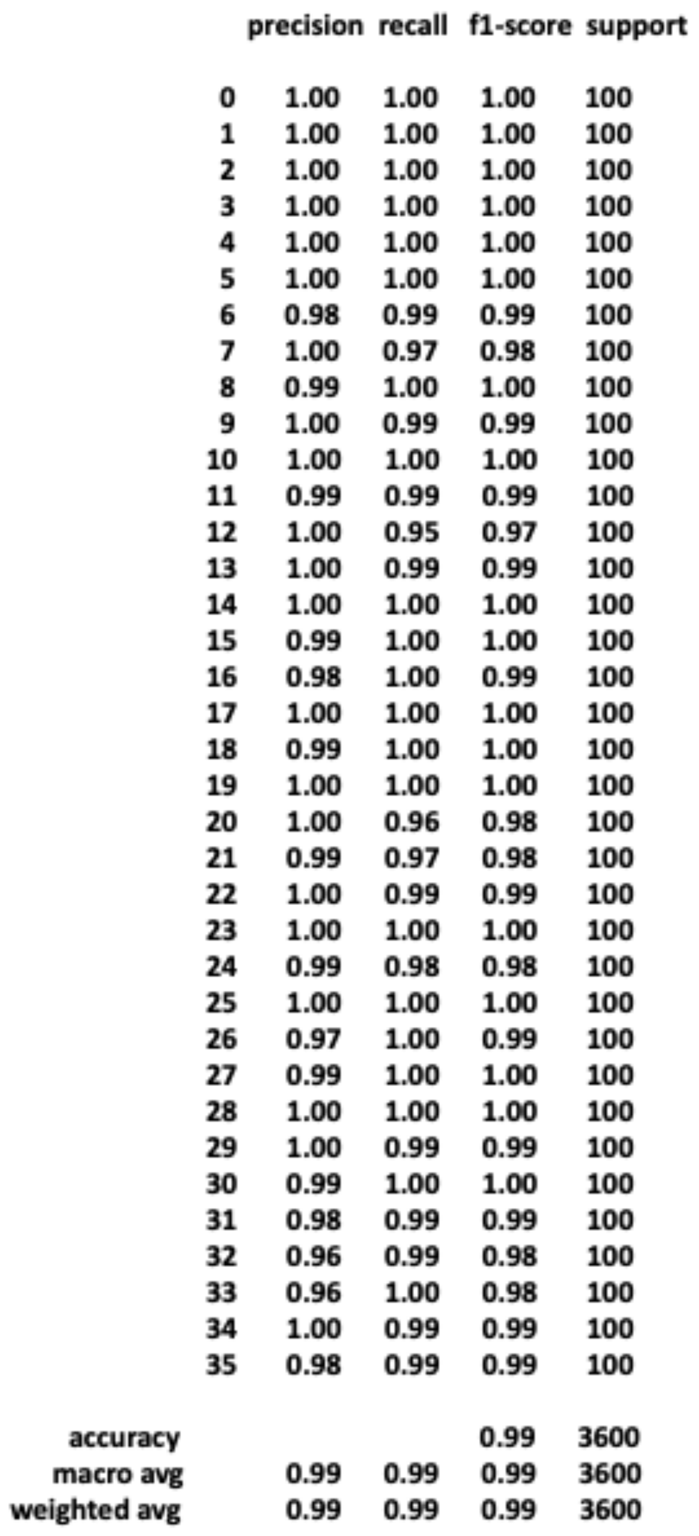


True vs Predicted Scatter Plot



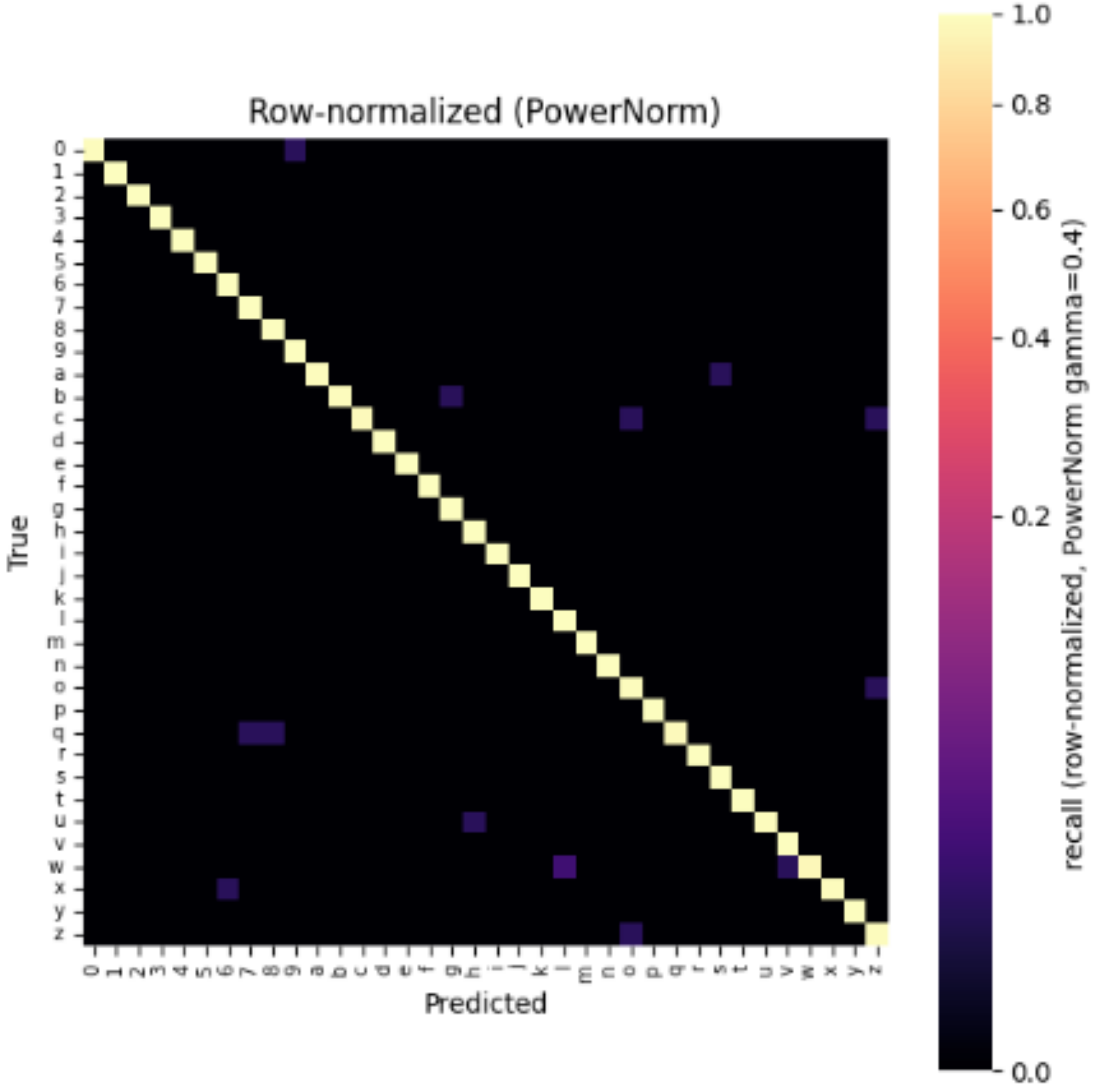
```
Total: 3600 Correct: 3586 Accuracy: 0.9961111111111111
```

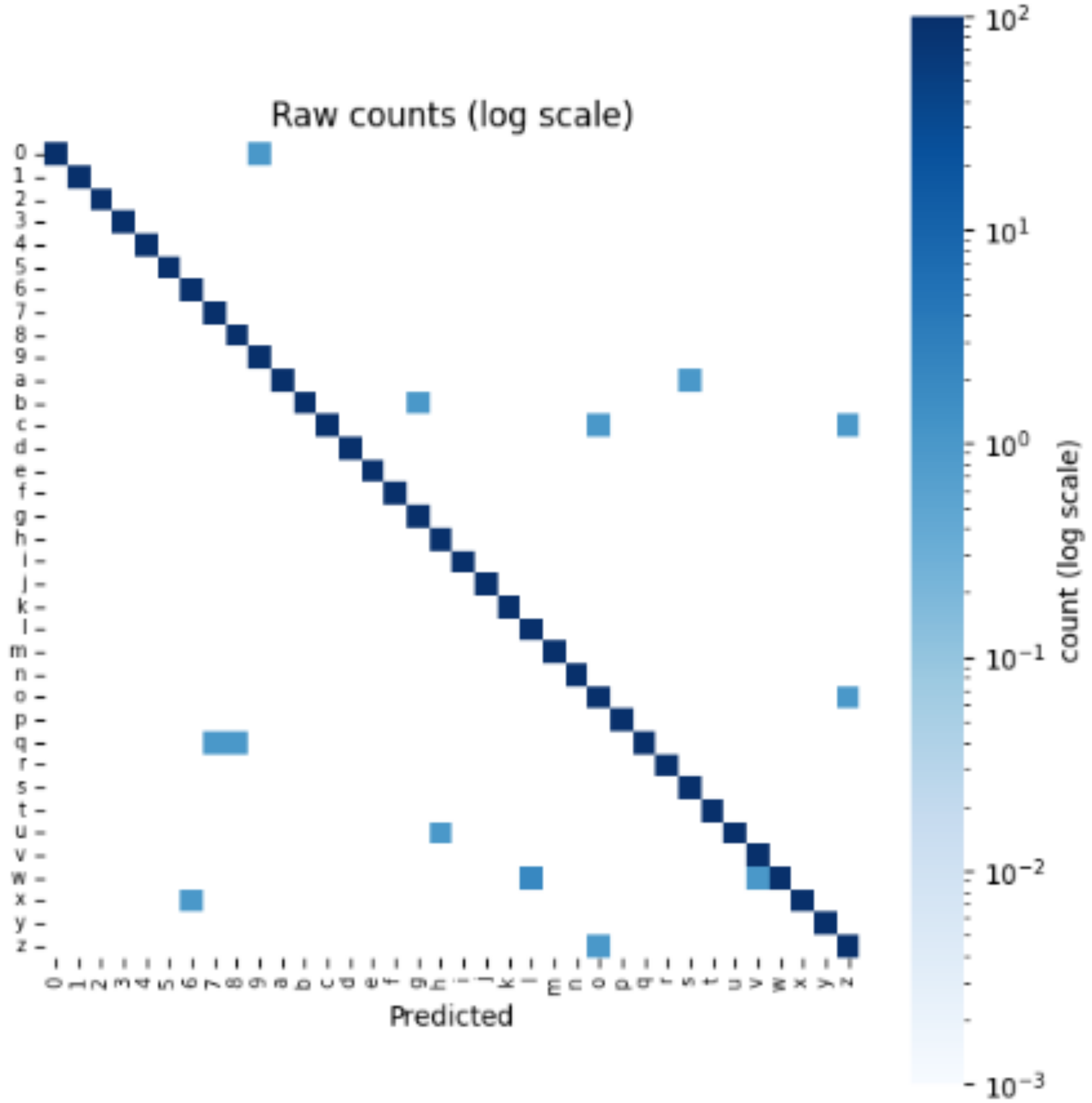
Results & Discussion



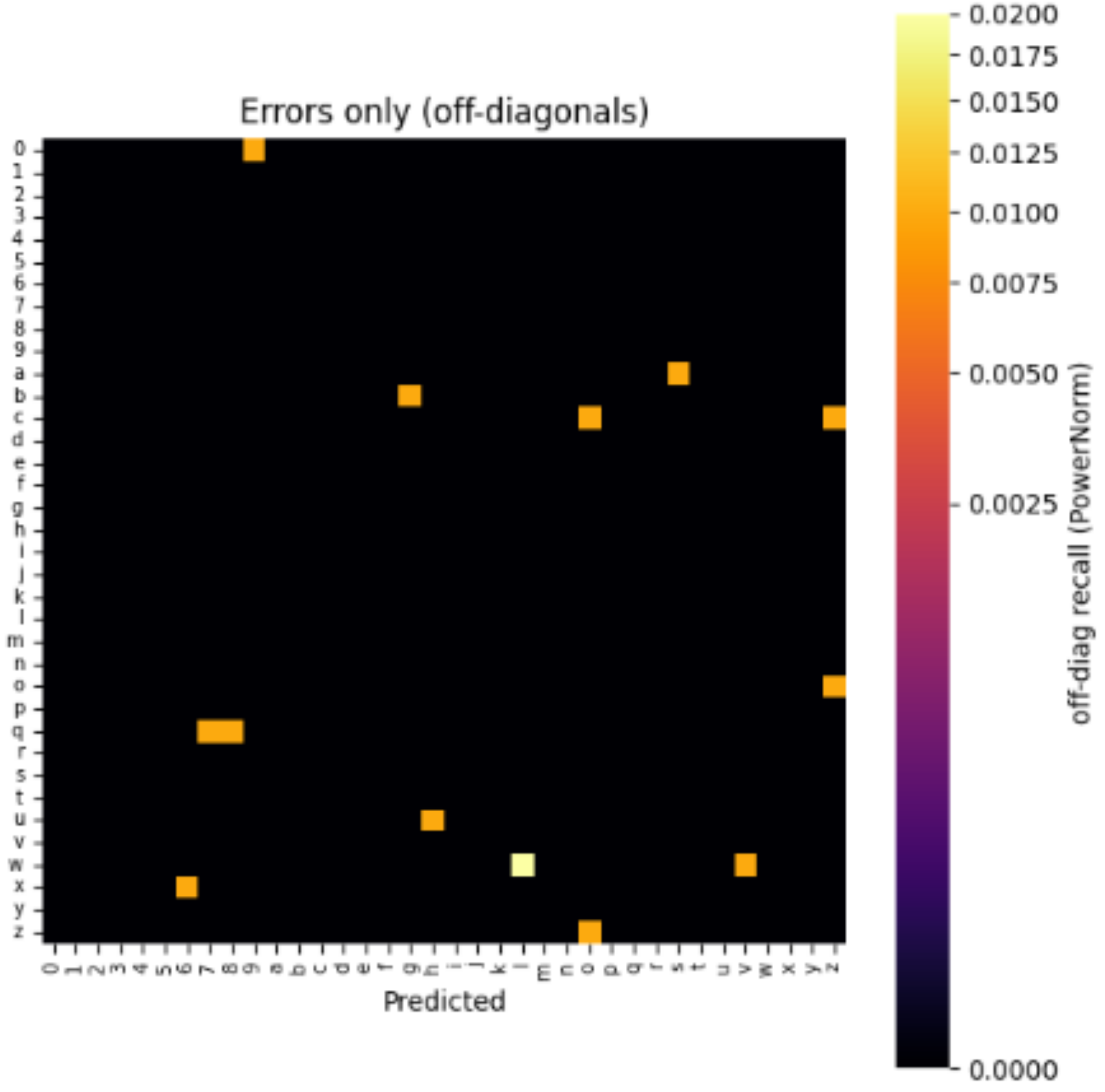
```
              precision  recall  f1-score  support

           0     1.00     1.00     1.00      100
           1     1.00     1.00     1.00      100
           2     1.00     1.00     1.00      100
           3     1.00     1.00     1.00      100
           4     1.00     1.00     1.00      100
           5     1.00     1.00     1.00      100
           6     0.98     0.99     0.99      100
           7     1.00     0.97     0.98      100
           8     0.99     1.00     1.00      100
           9     1.00     0.99     0.99      100
          10     1.00     1.00     1.00      100
          11     0.99     0.99     0.99      100
          12     1.00     0.95     0.97      100
          13     1.00     0.99     0.99      100
          14     1.00     1.00     1.00      100
          15     0.99     1.00     1.00      100
          16     0.98     1.00     0.99      100
          17     1.00     1.00     1.00      100
          18     0.99     1.00     1.00      100
          19     1.00     1.00     1.00      100
          20     1.00     0.96     0.98      100
          21     0.99     0.97     0.98      100
          22     1.00     0.99     0.99      100
          23     1.00     1.00     1.00      100
          24     0.99     0.98     0.98      100
          25     1.00     1.00     1.00      100
          26     0.97     1.00     0.99      100
          27     0.99     1.00     1.00      100
          28     1.00     1.00     1.00      100
          29     1.00     0.99     0.99      100
          30     0.99     1.00     1.00      100
          31     0.98     0.99     0.99      100
          32     0.96     0.99     0.98      100
          33     0.96     1.00     0.98      100
          34     1.00     0.99     0.99      100
          35     0.98     0.99     0.99      100

    accuracy                       0.99     3600
   macro avg     0.99     0.99     0.99     3600
weighted avg     0.99     0.99     0.99     3600
```

Confusion Matrix Heat Maps

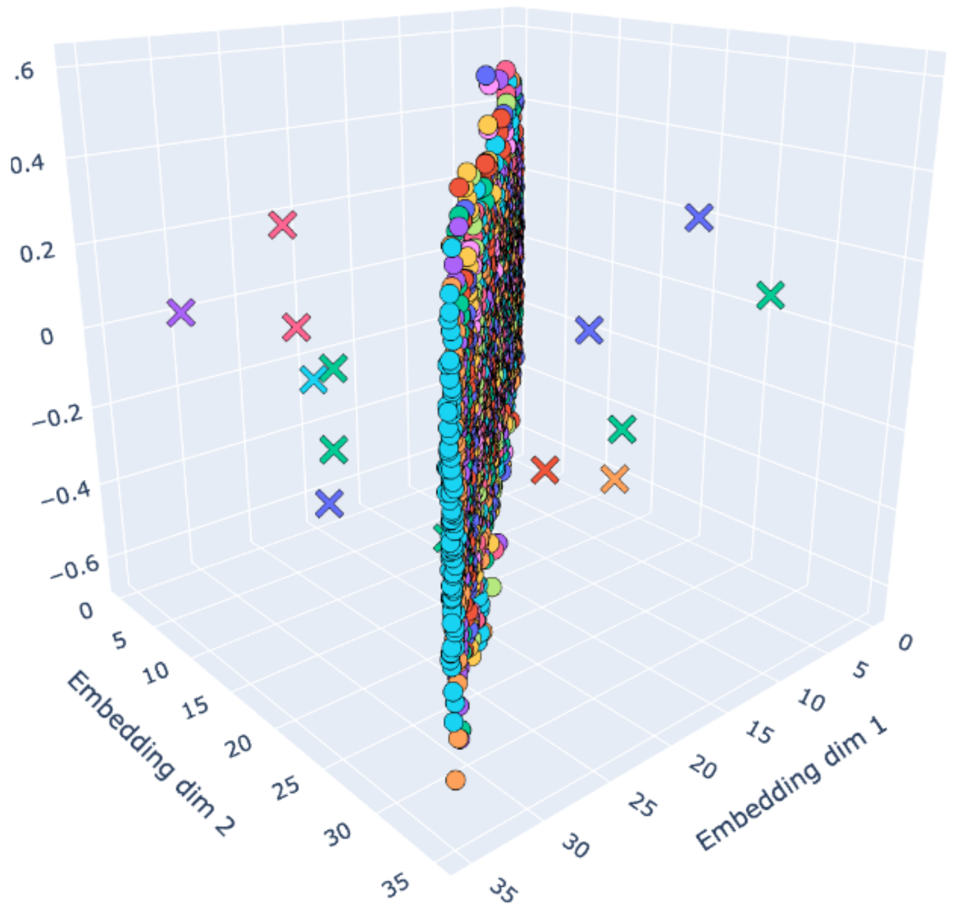




Raw counts (log scale)
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
Predicted
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
count (log scale)


Errors only (off-diagonals)
Predicted
off-diag recall (PowerNorm)
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w x y z

3D silhouette (z = silhouette)



X = prediction errors

Summary

Overall the results are considered good and very strong. Near perfect per-class scores and > 99% accuracy mean the model performs extremely well on the test data set. It must be noted that “Very good” on the test set does not automatically mean the model will be equally strong in every real-world situation. Further testing will be needed to show correlation with human sent morse code.

- Overall accuracy = 0.9961 (3586 correct out of 3600) only 14 errors total.

- Per-class metrics are excellent: most classes show precision/recall/F1 = 1.00; macro and weighted averages ≈ 0.99.

- Dataset appears balanced (support = 100 for each class), so weighted and macro averages are similar.

Successes

- High, consistent per-class performance; nearly every class reaches perfect or near-perfect precision/recall/F1, so the model distinguishes classes cleanly on the evaluation data.

- Balanced test set (100 samples per class) makes the reported averages reliable for the given distribution

- Low variance across classes: only a few classes have slightly lower scores, indicating few weak spots.

Challenges

- Small absolute number of errors (14) can hide issues: those errors may all come from a few specific class pairs or a single confused cluster. A small error count doesn’t rule out meaningful failure modes.

- Possible dataset simplicity or leakage: near-perfect scores sometimes mean the task is easy, the test set is too similar to training, or there was unintended data leakage. Verify there is no overlap and that augmentation/splits were correct.

Conclusion

Created a dataset of 36 morse code characters (a-z & 0-9) as .wave audio. MD5 Hashing checks for duplicates, then turn each audio clip into log-mel spectrogram (128 mel bands, 25 ms window, 10 ms hop), and splitting 80/10/10 (train/val/test) so the test set had 3,600 samples (36 classes × 100). Implementation of a PyTorch Encoder Transformer Classifier (TinyTransCls) that maps input sequences to class logits and is trained with Adam Optimizer and gradient clipping using CrossEntropyLoss. Evaluation of each epoch for validation loss/accuracy/ macro-F1, while saving the best checkpoint. Inference on the test set resulted in 3,586/3,600 = 99.61% accuracy with almost perfect precision/recall.

Bottom line: metrics and embeddings show a very strong model on synthetic data.

Limitations & Future Work

Limited test size per class (100 samples) while balanced, 100 samples may still be too small to capture a real-world variability or morse code sending. Performance on held-out of real deployment data can be lower if the training data isn't representative of actual human sent or synthetic software generated over the air morse code. Even with random seeding results slightly differ on subsequent runs.

Future Work

Inspect the confusion matrix and list of misclassified examples to identify which class pairs are confused. Manually review the 14 errors to check for label noise, ambiguous samples, or systematic patterns. Run cross-validation or evaluate on a separate external dataset to check generalization. If confusions are concentrated, consider targeted data augmentation, more training examples for those classes, or class-specific loss weighting.

The model performs very well on the provided test set (≈99.61% accuracy), but to be confident in a real live scenario, need to investigate the 14 errors, rule out leakage, and validate on additional real-world data.

References

Deep Learning by: Ian Goodfellow, Yoshua Bengio, Aaron Courville

Deep Learning with Python (third edition) by: Francois Chollet, Matthew Watson

Introducing Python by: Lubaovic

An Introduction to Statistical Learning with Applications in Python
by: Gareth James, Daniela WiSen, Trevor HasIe, Robert Tibshirani, Jonathan Taylor

The Hundred-Page Machine Learning Book by: Burkov

Guide to Numpy 2nd Edition by: Travis E. Oliphant PhD