Comparison of Structurally Equivalent GRU-Based Sequence Models

An Empirical Study on Framework-Level Divergence under Autoregressive Inference

1. Introduction

Predicting future player trajectories from NFL tracking data is a challenging spatiotemporal modeling problem that requires capturing both individual motion dynamics and multi-agent interactions. Recurrent neural networks (RNNs), particularly GRU-based encoder–decoder architectures, are widely used for sequence-to-sequence prediction tasks due to their ability to model temporal dependencies.

In practice, the same neural network architecture can be implemented in different deep learning frameworks such as PyTorch and TensorFlow. While these frameworks aim to provide equivalent mathematical formulations, it is often implicitly assumed that structurally identical models trained under the same conditions will exhibit similar behavior.

This study challenges that assumption. We construct structurally equivalent GRU-based encoder–decoder models in PyTorch and TensorFlow, train them on identical data with matched optimization settings, and compare their behavior under autoregressive inference. Despite architectural equivalence, we observe large and statistically significant differences in predictive performance, which we analyze through the lens of established sequence modeling theory.

2. Data and Feature Engineering

2.1 Dataset

We use the NFL Big Data Bowl 2026 player tracking dataset, which provides high-resolution spatiotemporal information for all players on the field prior to the pass. Each play consists of multiple frames, with per-frame measurements including player position, speed, acceleration, orientation, and contextual play metadata.

2.2 Feature Engineering

Raw tracking variables are transformed into a compact representation designed to capture motion, spatial context, and player interactions:

- Play Direction Normalization: All plays are mirrored so that the offense consistently moves in the same direction, reducing symmetry-induced variance.
- Circular Angle Encoding: Directional variables are encoded using sine and cosine transformations to preserve angular continuity.
- Motion Decomposition: Speed and acceleration are decomposed into x/y components aligned with field coordinates.
- Ball Landing Context: Relative displacement and distance to the ball's landing location are included to provide pass-centric spatial context.
- Player Role Information: Player positions are numerically encoded, with an additional binary indicator for the targeted receiver.
- Short-Term Temporal Features: First-order lagged positions, speed, and speed change capture recent movement trends.
- Field Constraints: Distance to the nearest sideline encodes spatial limitations of the field.
- Player Interactions: Distances to the nearest teammate and opponent, along with local player density, approximate defensive pressure.

After feature engineering, each timestep is represented by 26 numerical features, which are scaled and organized into fixed-length input sequences.

3. Model Architecture

Both models follow a sequence-to-sequence encoder–decoder architecture with stacked GRUs:

- Encoder:
  A linear embedding layer followed by a 2-layer GRU encodes historical player trajectories into a latent representation.
- Decoder:
  A symmetric 2-layer GRU generates future player positions autoregressively, starting from an initial zero vector.

- Output Layer:
  A linear projection maps hidden states to 2D field coordinates (x, y).

Special care is taken to ensure structural equivalence between PyTorch and TensorFlow implementations, including matched layer dimensions, parameter counts, activation functions, and decoding logic.

4. Training Strategy

- Teacher Forcing is used during training to stabilize sequence learning.
- Autoregressive Decoding is used during inference to reflect real-world prediction conditions.
- Masked Mean Squared Error (MSE) loss ensures that padded timesteps do not affect optimization.
- AdamW optimizer, cosine annealing learning rate schedule, and early stopping are applied consistently across frameworks.
- Robust feature scaling is applied prior to training.

Random seeds are fixed across libraries to minimize stochastic variability.

5. Results

5.1 Validation Performance

Model	Validation RMSE
PyTorch	1.55
TensorFlow	19.47

Despite structural equivalence, the TensorFlow model performs substantially worse under autoregressive inference.

5.2 Submission-Level Statistical Comparison

Using 5,837 paired predictions:

- Mean Euclidean distance between predictions: 22.65 yards
- 95th percentile discrepancy: 48.51 yards
- Maximum discrepancy: 79.32 yards
- Prediction correlation:
    - x-coordinate: 0.27
    - y-coordinate: 0.50

Paired Wilcoxon and Kolmogorov–Smirnov tests indicate statistically significant distributional differences, confirming that the two models produce meaningfully different outputs rather than noisy variations.

6. Interpretation and Discussion

6.1 Exposure Bias from Teacher Forcing

Teacher forcing introduces a mismatch between training and inference conditions. During training, the decoder always receives ground-truth inputs, whereas during inference it must rely on its own predictions. Bengio et al. (2015) describe this phenomenon as exposure bias, showing that even small prediction errors can compound over time during autoregressive decoding.

In this study, predictions are generated autoregressively for up to 40 timesteps, allowing early deviations to accumulate and amplify framework-level differences.

6.2 Hidden-State Dynamics Mismatch

Professor Forcing (Goyal et al., 2016) demonstrates that RNNs trained with teacher forcing may exhibit fundamentally different hidden-state distributions when run in free-running mode. Although the PyTorch and TensorFlow models share identical equations and parameters, their internal hidden-state trajectories during inference are not constrained to match.

The observed low inter-model correlation and extreme tail errors strongly suggest divergence in hidden-state dynamics.

6.3 Autoregressive Error Accumulation

He et al. (2020) quantitatively show that exposure bias grows with prediction horizon length. The large variance and heavy-tailed error distribution observed in the TensorFlow model are consistent with their findings on long-horizon autoregressive instability.

7. Limitations and Future Work

This study focuses on a single training run per framework. While statistical comparisons are performed at the prediction level, future work could include multiple randomized runs to estimate variance across initializations.

Potential mitigation strategies include:

- Scheduled sampling (Bengio et al., 2015)
- Professor forcing (Goyal et al., 2016)
- Training directly under autoregressive objectives
- Curriculum-based decoding or shorter prediction horizons

8. Conclusion

This work demonstrates that structural equivalence does not guarantee functional equivalence in sequence models. Under autoregressive inference, teacher-forced GRU models implemented in different frameworks can diverge dramatically, producing statistically distinct prediction distributions. These findings align closely with established theoretical results on exposure bias and hidden-state dynamics mismatch, highlighting the importance of inference-aware training strategies for long-horizon sequence prediction.

References

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015).
Scheduled sampling for sequence prediction with recurrent neural networks.
Advances in Neural Information Processing Systems (NeurIPS).

Goyal, A., Lamb, A., Zhang, Y., Zhang, S., Courville, A., & Bengio, Y. (2016).
Professor forcing: A new algorithm for training recurrent networks.
Advances in Neural Information Processing Systems (NeurIPS).

He, T., Liu, J., Neubig, G., & Berg-Kirkpatrick, T. (2020).
Quantifying exposure bias for neural language generation.
International Conference on Learning Representations (ICLR).