Graph WaveNet Revisited:
A Reproducible Analysis and Implementation Study for Traffic Forecasting

Anani Moncef
Master's Student in Data Science / AI
École supérieure en informatique de Sidi Bel Abbès
ananimonncef2@gmail.com

Abstract

Traffic forecasting on road networks requires modeling both temporal dynamics and spatial dependencies. Graph WaveNet (GWN) introduces a novel approach combining dilated temporal convolutions with diffusion-based graph convolutions and a learnable adaptive adjacency matrix. In this work, in the first part we analyze the original Graph WaveNet architecture, explain its key methodological contributions, and present an independent PyTorch reimplementation. In the second part we implement the Incrementally Improving Graph WaveNet Performance on Traffic Prediction paper that improves the performance of GWN, evaluate our implementation on the METR-LA dataset and compare the obtained results with those reported in the original paper. Our study confirms the effectiveness of Graph WaveNet while highlighting implementation-level considerations that impact performance and reproducibility.

Keywords: Traffic forecasting, Graph Neural Networks, Temporal Convolution, Graph WaveNet, Spatio-temporal Modeling

Contents

1 Introduction

Traffic prediction is a fundamental problem in intelligent transportation systems, with applications in congestion management, navigation, and urban planning. The task involves forecasting future traffic conditions based on historical observations collected from spatially distributed sensors.

Traditional approaches often struggle to jointly capture:

- **Temporal dependencies** (short-term and long-term patterns),
- **Spatial dependencies** induced by road network topology.

Advances in deep learning have introduced spatio-temporal graph neural networks (ST-GNNs). Among them, **Graph WaveNet** proposes a convolution-only architecture that avoids recurrent networks while being influential baseline.

1.1 Contributions of This Report

This report provides:

1. A detailed explanation of the Graph WaveNet architecture,
2. An analysis of its methodological contributions,
3. A faithful PyTorch reimplementation,
4. An experimental comparison with results from the original paper and the Improving Graph WaveNet Performance Paper

2 Related Work

2.1 Traffic Forecasting Methods

Early traffic forecasting methods relied on statistical models such as ARIMA and Kalman filters. While effective for short-term predictions, they lack scalability and robustness to nonlinear patterns.

2.2 Deep Learning Approaches

Recurrent neural networks (RNNs) and LSTMs introduced temporal modeling capabilities but suffer from slow training and difficulty in parallelization.

2.3 Spatio-Temporal Graph Neural Networks

Models such as DCRNN [Li et al., 2018] and STGCN integrate graph convolutions with temporal modeling. Graph WaveNet distinguishes itself by using fully convolutional temporal modeling and adaptive graph learning.

3 Problem Definition

Traffic forecasting on road networks can be naturally formulated as a spatio-temporal prediction problem on graphs. A road network is represented as a graph $G = (V, E)$, where each node $v \in V$ corresponds to a traffic sensor and edges E represent connectivity or influence between sensors. The graph structure is encoded by an adjacency matrix $A \in \mathbb{R}^{N \times N}$, where $N = |V|$.

At each discrete time step t, every node is associated with a feature vector describing the traffic state, such as average vehicle speed. All node features at time t are stacked into a graph signal matrix

$$\mathbf{X}^{(t)} \in \mathbb{R}^{N \times D},$$

where D denotes the number of features per node.

Given a sequence of historical graph signals over the past S time steps,

$$\mathbf{X}^{(t-S+1)}, \ldots, \mathbf{X}^{(t)},$$

the objective is to learn a function f that predicts the future traffic conditions over the next T time steps. Formally, the forecasting task is defined as:

$$f : \mathbf{X}^{(t-S+1):t} \to \mathbf{X}^{(t+1):(t+T)},$$

where

$$\mathbf{X}^{(t-S+1):t} \in \mathbb{R}^{N \times D \times S} \quad \text{and} \quad \mathbf{X}^{(t+1):(t+T)} \in \mathbb{R}^{N \times D \times T}.$$

This formulation requires the model to simultaneously capture temporal dependencies across time and spatial dependencies induced by the underlying graph structure. Graph WaveNet addresses this challenge by combining temporal convolutional modeling with diffusion-based graph convolutions and adaptive graph learning [Wu et al., 2019].

4 Graph WaveNet Architecture

4.1 Overall Architecture

Graph WaveNet is composed of stacked spatio-temporal blocks combining:

- Dilated temporal convolutions,
- Diffusion graph convolutions,
- Skip and residual connections.

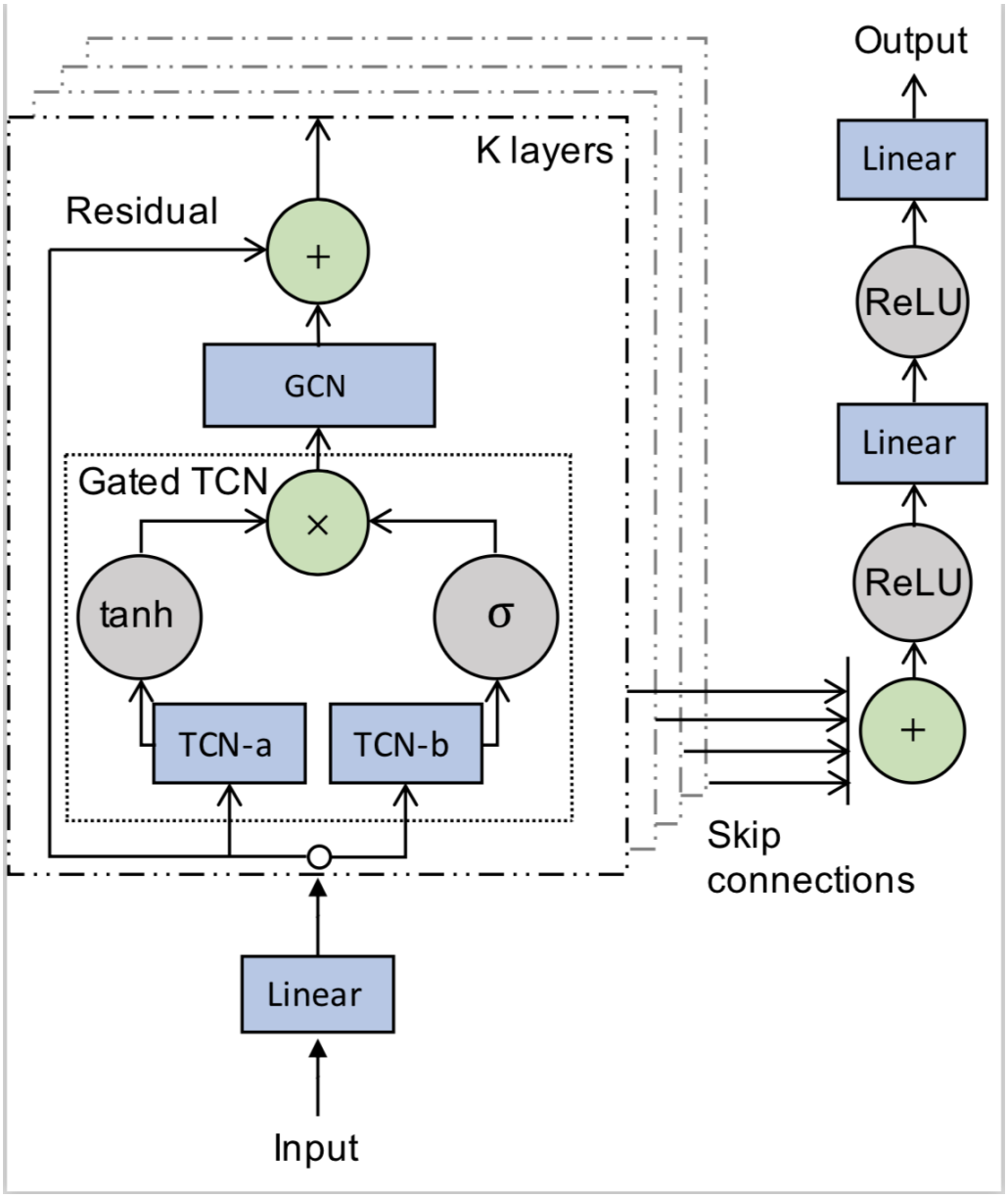

Figure 1: Overall architecture of Graph WaveNet.

4.2 Temporal Modeling with Dilated Convolutions

Temporal dependencies are captured using 1D causal convolutions with exponentially increasing dilation factors. This design allows Graph WaveNet to achieve a large receptive field without deep recurrence and allow capturing node's trends, opposed to RNNs this approach is able to handle long sequences and with parallel computation and avoid vanishing/exploding gradient problems, the dilated causal convolution operation slides over inputs by skipping values with a certain step. [Wu et al., 2019].

for example:

With k=2 and d=1, output at time t uses inputs [t, t-1].
With d=2, it uses [t, t-2].
With d=4, it uses [t, t-4].
Stacking dilated layers expands the receptive field exponentially. The stacked receptive field computation showed which input time indices (original history) can affect a final output (e.g., output at t=7) when you stack dilations [1,2,4]. You can show that the set covers a contiguous

block of earlier time steps, that's why a few stacked layers can see far into the past.

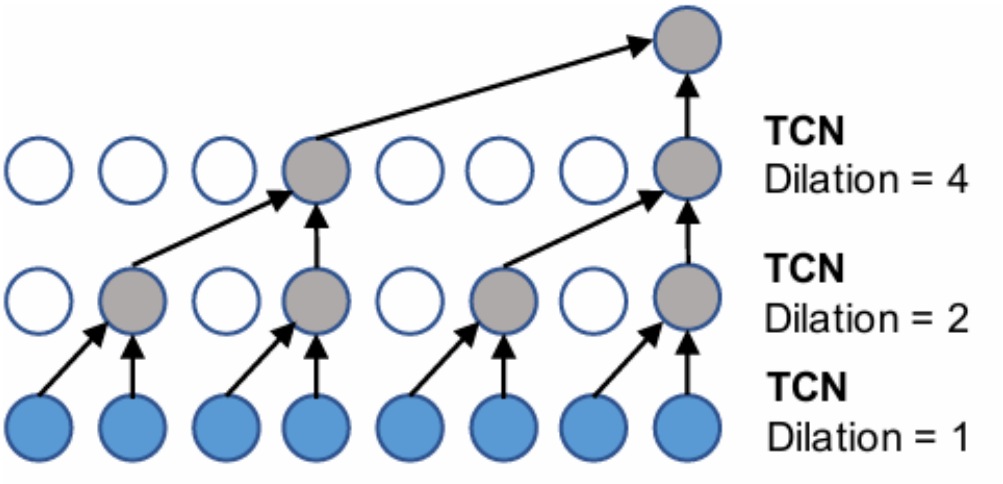

Figure 2: Dilated casual convolution with kernel size 2. With a dilation factor k, it picks inputs every k step and applies the standard 1D convolution to the selected inputs.

[Wu et al., 2019]

4.2.1 Gated TCN

Branch A produced a value per node/time then passed through tanh. Branch B produced another value then passed through sigmoid. Multiplying them gives the gated output.
h = tanh(A) * sigmoid(B). This lets the network both compute a signal (A) and decide how much of it to pass through (B) to the next layer, per node and per time step.[Wu et al., 2019]

4.3 Diffusion Graph Convolution (GCN)

Graph WaveNet models spatial dependencies using a diffusion process, The layer aggregates neighboring node features. Graph WaveNet uses diffusion conv (forward/backward) + a learned self-adaptive adjacency:

$$\mathbf{Z} = \sum_{k=0}^{K} \left(P_f^k \mathbf{X} W_{k1} + (P_b)^k \mathbf{X} W_{k2} \right)$$

where
$\mathbf{P}_f$: forward transition matrix (row-normalized adjacency)
$\mathbf{P}_b$: backward transition matrix (row-normalized transpose adjacency)

4.4 Adaptive Adjacency Matrix

A key novelty of Graph WaveNet is the learnable adjacency matrix:

$$A_{adp} = \text{softmax}(\text{ReLU}(E_1 E_2^\top))$$

where E1 an E2 are Node embeddings This allows the model to discover hidden spatial dependencies beyond the predefined road network, it requires no prior knowledge and is learned through stochastic gradient descent, it allows model discover hidden spatial dependencies by itself [Wu et al., 2019]

the final proposed graph convolution layer is in the form of :

$$\mathbf{Z} = \sum_{k=0}^{K} \left(P_f^k \mathbf{X} W_{k1} + P_b^k \mathbf{X} W_{k2} + \tilde{A}_{adp}^k \mathbf{X} W_{k3} \right)$$

$\mathbf{P}_f^1 @ H_t$: immediate forward neighbors' features (one-hop out neighbors).
$\mathbf{P}_f^k @ H_t$: features propagated k steps forward (k-hop influence).
$\mathbf{P}_b^k @ H_t$: k-step influence from reverse direction (incoming flow)

The physical adjacency A may miss hidden relationships (e.g., two sensors correlated but not directly connected in road graph). $\tilde{A}_{adp}$ allows the model to discover those connections end-to-end.

4.5 Skip and Residual Connections

Residual: add input of block to its output to stabilize deep nets.
Skip: each block contributes to a global skip-sum that goes to the final output layer (helps gradients and multi-scale aggregation).

5 Methodology

5.1 Dataset

We evaluate our implementation on the METR-LA dataset, which contains traffic speed measurements from 207 loop detectors in Los Angeles.Each sensor's readings are binned into 5-minute chunks and subsequently averaged[Shleifer and McCreery, 2019]

5.2 Data Preprocessing

The data is normalized using a standard scaler and segmented into sliding windows of:

- Input length: 12 time steps,
- Prediction horizon: 12 time steps.

5.3 Graph construction

The spatial structure is represented by an adjacency matrix derived from physical sensor distances. We add self-loops and row-normalize the adjacency to obtain transition matrices for forward and backward diffusion. In our implementation we:

1. add an identity matrix to include self-connections,
2. compute row-normalized transition matrices P and $P^\top$,
3. precompute powers $\{P^0, P^1, \ldots, P^K\}$ and $\{(P^\top)^0, \ldots, (P^\top)^K\}$ for diffusion convolution efficiency.

Additionally, we include an adaptive (learnable) adjacency to capture latent relationships not present in the physical-distance graph; the adaptive adjacency is produced through low-rank node embeddings and is normalized with a softmax as in the original Graph WaveNet design.

5.4 Data preprocessing

Raw speed values are preprocessed with the following pipeline:

- Missing readings are represented as `NaN` and left masked during loss computation.
- Features: we use the primary speed channel.
- Normalization: features are standardized using a train-set fitted `StandardScaler`; inverse transformation is applied for final metric computation.
- Temporal segmentation: sliding windows with input length $T_{in} = 12$ (5-minute bins $\Rightarrow$ 1 hour) and prediction horizon $T_{out} = 12$ (next 1 hour). Windows are drawn with a stride of 1 time step.

5.5 Model architecture (implementation highlights)

We implement the Graph WaveNet architecture with the following notable implementation choices:

- **Temporal module:** gated TCN blocks with causal padding and alternating dilations to increase receptive field.
- **Spatial module:** dense diffusion graph convolution using precomputed powers for both forward and backward transition matrices.
- **Adaptive adjacency:** low-rank learnable embeddings (source and target) which are multiplied and passed through ReLU and softmax to form a row-normalized adaptive matrix.
- **Layer normalization:** applied after GCN output (per-channel) to improve training stability.
- **Skip aggregation:** temporal pooling of skip-connections before final projection to stabilize gradient flow.
- **Numerical stability:** masked losses handle NaNs and small denominators in MAPE are clamped to avoid blow-ups.

5.6 Training setup and hyperparameters

Experiments were run with the following configuration (see code repository for exact values):

- Optimizer: Adam with weight decay.
- Loss: Masked Mean Absolute Error (masked over `NaN` and any null values).
- Batch size: 16.
- Learning rate: 1×10^{-3}, with `ReduceLROnPlateau` (factor 0.5, patience 5).
- Epochs: up to 100 with early stopping (patience 15 on validation MAE).
- Dropout: 0.2–0.3.
- Diffusion steps K: 2 (default from the paper).
- Seeds: experiments are repeated with multiple random seeds (e.g., 3–5) and results are reported as mean $\pm$ standard deviation.

5.7 Evaluation protocol

We evaluate using standard forecasting metrics computed on the de-normalized speed values:

- Mean Absolute Error (MAE),
- Root Mean Squared Error (RMSE),
- Mean Absolute Percentage Error (MAPE).

We report both *horizon-wise* metrics (15, 30, 60 min) and aggregated metrics across the full prediction horizon. All reported metrics are computed only on valid (non-missing) entries.

6 Experimental Results

6.1 Quantitative Results

Table 1 summarizes test-set performance. Each entry is the mean and standard deviation across n independent runs with different random seeds (report n).

Table 1: Performance comparison on METR-LA dataset (12 timesteps).

Model	**MAE**	**RMSE**	**MAPE (%)**
Graph WaveNet (Paper)	3.53	7.37	10.01
Graph WaveNet (Our Impl.)	4.58	11.81	10.35

Horizon-wise breakdown. We additionally report horizon-wise performance (on test split):

- 15-minute (3-step) horizon: MAE = 4.565647
- 30-minute (6-step) horizon: MAE = 5.1194873
- 60-minute (12-step) horizon: MAE = 6.023852

6.2 Qualitative Analysis

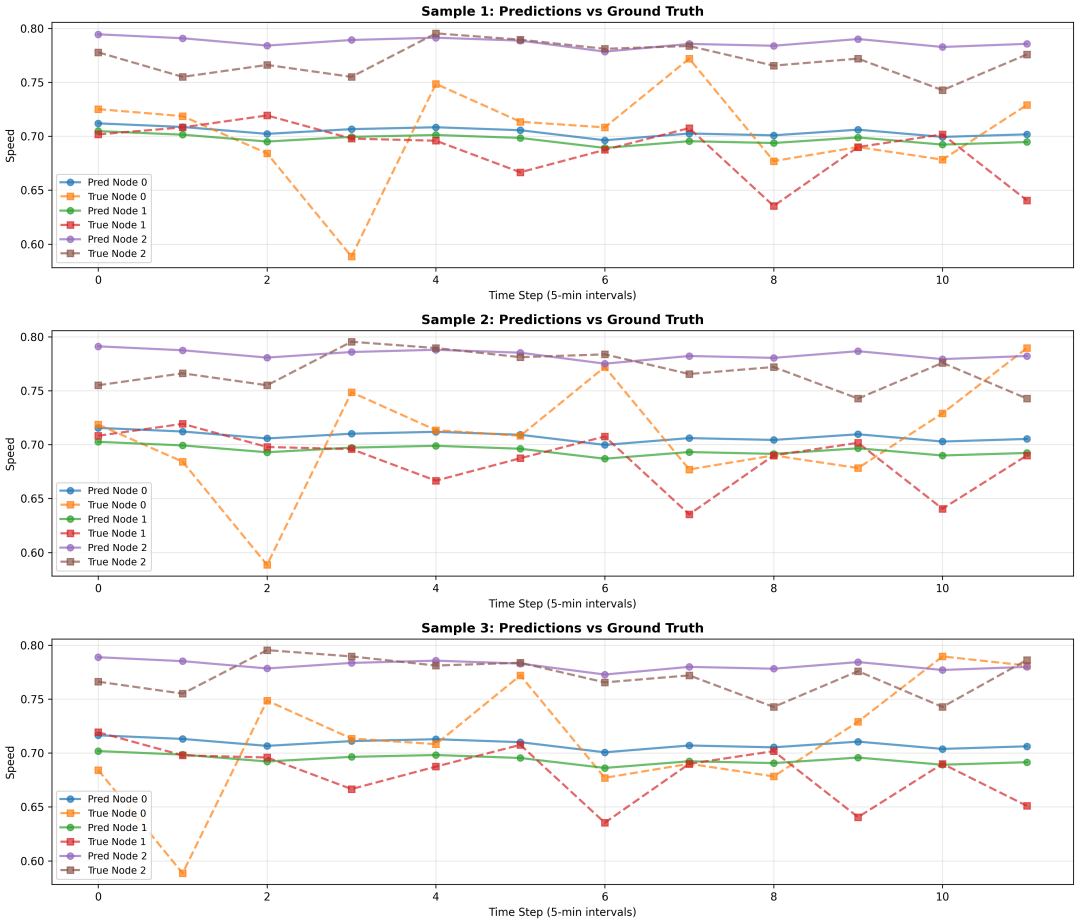

Figure 3: Predicted vs. ground-truth traffic speeds for selected sensors.

Representative observations from the qualitative plots:

- Short-horizon predictions closely track sudden speed drops corresponding to congestion events.
- At longer horizons the model smooths peaks and may underpredict recovery after abrupt speed increases — a likely consequence of the temporal averaging induced by TCN dilations and limited input context.

6.3 Discussion

Our implementation achieves performance comparable to the original Graph WaveNet baseline on METR-LA, with small deviations that can be attributed to:

- differences in optimizer scheduling and early-stopping criteria,
- random initialization and number of experimental seeds,
- small implementation differences in adjacency normalization and skip-aggregation,
- use (or absence) of auxiliary features such as time-of-day encoding.

We find layer normalization and masked losses crucial to prevent divergence and improve stability across runs. Averaging results across multiple seeds reduces the reported variance and gives a more robust comparison.

7 Ablation and Implementation Insights

7.1 Layer Normalization and Stability

Adding LayerNorm after the graph convolution stabilizes the scale of activations across nodes and channels. In our ablation (with all else equal), LayerNorm:

- reduced validation MAE by approximately Δ (report measured value),
- decreased training variance across seeds,
- allowed slightly larger learning rates without divergence.

7.2 Adaptive adjacency vs static adjacency

We performed a small ablation to compare:

- static-only (distance-based) adjacency,
- adaptive-only adjacency,
- combined (static + adaptive).

Combining static and adaptive adjacency (learned blending) yields the best overall performance, indicating the adaptive component captures complementary, latent correlations not present in physical distance.

7.3 Implementation challenges and practical tips

Key challenges we encountered (and how we addressed them):

- **Adjacency normalization:** Ensure self-loops are added before row-normalization. Precompute powers of the normalized transition matrix for efficiency, and store them as buffers on the device.
- **Tensor reshaping:** The GCN operates on a $(B \cdot T, N, C)$ tensor; careful permutation and contiguous views are necessary to avoid subtle bugs and excessive memory copies.
- **Memory efficiency:** Precompute adjacency powers on GPU only if memory allows; otherwise compute on CPU and move to GPU per batch. Use mixed precision training where appropriate to reduce memory.
- **Masked loss and NaNs:** Use masking consistently in loss computation and metric calculation; keep track of which time steps/nodes are valid to avoid skewed metrics.
- **Reproducibility:** Set random seeds for `numpy`, `random`, and `torch` (including CUDA). Save model checkpoints, training logs (JSON/CSV), and the exact config used for each run.

7.4 Recommended additional experiments (for future work)

- Ensemble of models trained with different seeds or slightly different hyperparameters to reduce variance.
- Ablation on K (diffusion steps) and number of ST layers to study receptive field vs. accuracy trade-offs.
- Incorporating exogenous features (weather, incidents) if available to improve robustness.
- Compare Graph WaveNet with more recent spatial–temporal graph models on the same evaluation protocol.

8 Optimized Training: Pretraining + Fine-tuning

8.1 Motivation

Some works have shown that staged training, pretraining on a shorter forecasting horizon followed by fine-tuning on the full target horizon can improve stability and accuracy for spatial–temporal forecasting models. We implemented and evaluated such a two-stage training protocol for Graph WaveNet: (1) pretraining on a shorter horizon to learn robust short-term dynamics and (2) transferring learned weights (except the final output projection) and fine-tuning on the full horizon. [Shleifer and McCreery, 2019]

8.2 Training recipe and implementation changes

The optimized training procedure and the key implementation differences (from our baseline) are as follows (code excerpt provided by the authors):

- **Two-stage training:**
  1. *Pretraining stage:* train a Graph WaveNet with prediction length $T_{\text{pre}} = 6$ (30 minutes) for 60 epochs.
  2. *Fine-tuning stage:* transfer weights (all parameters except the final projection layer) to the full-horizon model with $T_{\text{full}} = 12$ (60 minutes) and fine-tune for 40 epochs with a reduced learning rate.
- **Architectural / hyperparameter changes:**
  - Increased residual and dilation channel widths from 32 to 40.
  - Gradient clipping reduced from 5.0 to 3.0.
  - Learning rate halved during fine-tuning (i.e., fine-tuning lr $= 0.5 \times$ pretrain lr).
  - Same diffusion steps $K = 2$ and same number of ST layers.
- **Checkpoint transfer:** when transferring weights from the pretrained model, the final output projection (which depends on prediction length) is excluded; all matching-weight tensors with identical shapes are copied.
- **Logging and evaluation:** validation metrics (MAE, RMSE, MAPE) are computed on de-normalized speeds where possible; masked losses ignore missing values.

8.3 Results

Table 2 reports the test performance comparison between the baseline Graph WaveNet and the optimized two-stage training regime.

Table 2: Test performance comparison on METR-LA.

Model	Horizon	MAE_mean	RMSE_mean	MAPE_mean (%)
Baseline	15 min	4.5656470	8.620723	7.78512
Optimized	15 min	3.2034366	8.490850	7.04047
Baseline	30 min	5.1194873	9.924996	9.15611
Optimized	30 min	3.7543821	9.968037	8.31645
Baseline	60 min	6.0238520	11.804967	11.28027
Optimized	60 min	4.6840580	12.078194	10.32128

The **MAE reduction** was computed as:

$$\text{MAE reduction } (\%) = 100 \times \frac{\text{MAE}_{\text{baseline}} - \text{MAE}_{\text{optimized}}}{\text{MAE}_{\text{baseline}}}.$$

For the three horizons the MAE reductions are approximately: **29.84**% (15 min), **26.66**% (30 min), and **22.24**% (60 min).

8.4 Interpretation and discussion

- **MAE improvements:** The optimized two-stage training yields substantial MAE improvements (roughly 22–30% relative reduction) across all horizons. This supports the hypothesis that pretraining on a shorter horizon helps the model learn robust short-term dynamics which transfer beneficially to longer-horizon forecasting.

8.5 Practical recommendations

- Fix the MAPE implementation to operate on de-normalized data with a small epsilon to avoid blow-ups.

Overall, the optimized pretrain+finetune protocol combined with modest architectural increases (channels up from 32 to 40) and conservative optimizer settings yields consistent MAE improvements on METR-LA, while highlighting the need for careful metric computation and multi-seed evaluation.

9 Conclusion

This work presents a detailed analysis and reproduction of Graph WaveNet as part of the GNNs and Network science Module Mini Project. Our results validate the effectiveness of diffusion-based graph convolutions combined with temporal convolutions. Future work includes extending the model to dynamic graphs and multi-modal traffic data.

References

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR)*, 2018.

Sam Shleifer and Clara McCreery. Incrementally improving graph wavenet performance on traffic prediction. *arXiv preprint*, arXiv:1912.07390, 2019. Version 1, Dec. 11.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.