Traffic-AI: Predictive Traffic Forecasting through Multi-Model Deep Learning

Kenneth Poh[1], Cheow Ming Yang[2], Au Xi Nan Antonio[3], Keith Ang[4], Deacon Koh[5]

Singapore Institute of Technology

[1]2401664@sit.singaporetech.edu.sg

[2]2400936@sit.singaporetech.edu.sg

[3]2401653@sit.singaporetech.edu.sg

[4]2401811@sit.singaporetech.edu.sg

[5]2401300@sit.singaporetech.edu.sg

***Abstract*—Existing vehicle navigation systems provide descriptive route information such as travel duration and turn-by-turn guidance but largely operate reactively, updating travel estimates only after congestion occurs. Traffic-AI introduces a proactive alternative designed for car commuters. By fusing high-velocity live traffic data with external feeds—such as weather conditions and accident reports—and applying deep learning models, Traffic-AI predicts point-to-point congestion risk and recommends optimal departure times. The platform is underpinned by a cloud-native Big Data architecture for real-time processing and an automated MLOps workflow for model self-healing. The system delivers these insights via a city-scale 24-hour forecast using an interactive heatmap, transforming traditional navigation into an anticipatory system that enables data-driven travel decisions.**

Keywords—Intelligent transportation systems, Traffic Forecasting, Big Data Analytics, MLOps, Cloud-Native Architecture

I. Introduction

A. Background & Motivation

Urban congestion remains one of the most persistent challenges for private drivers. Despite continuous improvements in navigation technology, many commuters still face unpredictable delays caused by factors such as minor collisions, lane closures, or sudden weather changes. These disruptions often cascade quickly, leading to extended travel times and inefficient fuel consumption.

Traditional navigation systems have improved route guidance and situational awareness but remain limited in foresight. They describe the current state of the road rather than anticipate what is about to happen. This gap leaves drivers in a reactive position—adjusting only after congestion has already developed—rather than planning around it

Research on commuter behaviour reinforces the importance of information reliability and timeliness. Meng et al. (2017) demonstrated that even small improvements in perceived travel predictability significantly affect traveller decision-making. For motorists, this implies that access to predictive traffic insights could meaningfully influence when and how they drive, helping to distribute traffic more evenly and reduce individual travel stress.

However, operationalizing these insights at a city-wide scale presents a significant data engineering challenge. Unlike static routing, traffic patterns are driven by high-velocity temporal data—thousands of road sensors updating every few minutes—combined with complex external variables like weather. Processing this volume of data to generate accurate, city-scale forecasts in real-time requires a robust, scalable cloud architecture capable of handling ingestion, historical analysis, and inference without prohibitive latency or cost.

B. Related Work

Recent research in intelligent transportation systems (ITS) has explored how external factors such as weather influence traffic flow and commuter experience. Bi et al. (2022) analysed traffic data across four Chinese cities and found that variables like rainfall, temperature, and visibility significantly impact congestion levels, with stronger effects observed during weekends [1]. Similarly, Al-Selwi et al. (2022) evaluated the inclusion of weather variables in deep learning–based traffic prediction, noting that models such as LSTM and GRU improved forecasting accuracy, though results varied by architecture [2].

Beyond exogenous factors, traffic behaviour also displays strong internal regularities. Lv et al. (2015) reported clear diurnal congestion cycles across urban networks, while Chen et al. (2022) highlighted that

encoding hour-of-day and weekly periodicity is essential for short-term speed forecasting [3][4]. These studies collectively indicate that effective prediction must account for both external influences and recurrent temporal structure.

However, the forecasting landscape becomes more complex when considered at a city scale. Real-world ITS platforms rely on high-velocity, heterogeneous streams of sensor data, weather feeds, and incident reports. In such environments, data distributions may evolve due to seasonal shifts, infrastructural changes, or unexpected disruptions—a challenge widely discussed in concept-drift research by Bayram et al. [5]. Without mechanisms for continuous monitoring and adaptation, model performance can degrade over time, limiting the viability of practical deployment.

C. Proposed Solution & Objectives

Building on the gaps identified in prior work, this project proposes a two-layer prediction framework capable of delivering both point-to-point congestion forecasts and city-scale traffic outlooks. The system is designed to operate on high-velocity data streams and adapt to evolving road conditions through scalable cloud-native deployment.

The proposed solution is structured around three key objectives:

1. Point-to-Point Routing: Deliver route-specific optimisation for individual drivers by predicting traffic conditions along a chosen path. This is achieved using pretrained spatio-temporal graph neural networks — STGCN [6] and Graph WaveNet [7] — which capture spatial dependencies between neighbouring road links and temporal changes over time, allowing for dynamic and congestion-aware routing.

2. City-Scale Forecasting: Provide a macro-level view of the transport network, functioning effectively as a "traffic weather forecast." In contrast to the route-level model, this component prioritizes computational efficiency for high-volume data. It employs temporal models, such as LSTM and XGBoost, within a distributed Big Data pipeline to process thousands of road segments simultaneously, generating a 24-hour congestion outlook visualized via an interactive heatmap.

3. Cloud-Native Deployment Architecture: Support scalable, cost-optimised execution across ingestion, training and inference pipelines. A distributed, Service-Oriented Architecture (SOA) enables independent scaling of prediction workloads, while autoscaling and tiered storage reduce operational cost. The system includes MLOps features such as automated retraining and drift monitoring, ensuring long-term reliability as data patterns evolve.

II. System Architecture

A. System Overview: Hybrid Cloud Architecture

The Traffic-AI platform utilizes a Service-Oriented Architecture (SOA) deployed entirely on Amazon Web Services (AWS). To support the dual requirements of low-latency user interaction and high-volume data processing, the system infrastructure is bifurcated into two distinct operational pipelines: a Synchronous Real-Time Layer and an Asynchronous Batch Layer, supported by a shared governance and security layer.

1. The Synchronous Real-Time Layer (User-Facing)

This layer handles direct user interactions and on-demand routing requests.

- Entry Point & Edge Security: Inbound traffic is routed through an Application Load Balancer (ALB) integrated with a Web Application Firewall (WAF) to filter malicious requests before they reach the backend.

- Compute & Orchestration: The core application logic is containerized via Amazon ECR and hosted on Amazon EC2 instances. These instances operate within an Auto Scaling Group (ASG), ensuring the system automatically provisions additional compute capacity during traffic spikes and scales down during lulls to optimize cost.

- Real-Time Inference: For point-to-point predictions, the web server communicates directly with Amazon SageMaker Endpoints, enabling sub-second model inference without local resource contention.

2. The Asynchronous Batch Layer (Data-Facing)

This layer operates in the background to handle "Big Data" heavy lifting.

- Orchestration: Amazon EventBridge and AWS Lambda act as the temporal schedulers, triggering ingestion and processing jobs at fixed intervals.
- Compute & ETL: Heavy data transformations and city-wide forecasting are offloaded to AWS Glue. By leveraging the underlying Apache Spark distributed processing engine, the system parallelizes the transformation of thousands of road segments, ensuring efficient execution without memory bottlenecks.
- Storage & Serving: The architecture leverages an Amazon S3 Data Lake as the central repository. AWS Athena sits on top of this lake, providing a serverless interface for the application layer to query massive historical datasets instantly.

3. Governance, Observability & Security

The entire system operates within a strict security boundary designed around the Principle of Least Privilege.

- Network Isolation: Resources are deployed within a multi-tiered Virtual Private Cloud (VPC), isolating compute resources in private subnets while exposing only the ALB to the public internet.
- Access Control: All service interactions are governed by granular IAM Roles that enforce the Principle of Least Privilege (PoLP), ensuring that services (like Lambda or EC2) possess only the permissions strictly necessary for their function.
- Secret Management: Sensitive credentials (API keys, database strings) are injected dynamically at runtime via AWS Secrets Manager, eliminating hardcoded secrets.
- Observability: The health of the distributed system—including container status, API latency, and model drift—is monitored via Amazon CloudWatch, providing a unified operational dashboard.

B. Cloud Infrastructure & Design Rationale

The physical deployment of the Traffic-AI platform is orchestrated within the AWS US East (N. Virginia) region, distributed across two Availability Zones (AZs) for high availability. As illustrated in Figure 1 (see Appendix D for full resolution), the infrastructure design prioritizes a decoupled, event-driven architecture over traditional monolithic patterns.

Figure 1 depicts the rigorous network segmentation used to enforce this decoupling. The visual hierarchy highlights how the high-velocity data ingestion pipeline (≈15,000 records/5 min) is physically isolated in the bottom "Data & Analytics" layer, ensuring that backend processing spikes do not degrade the latency of the user-facing application in the upper tiers.

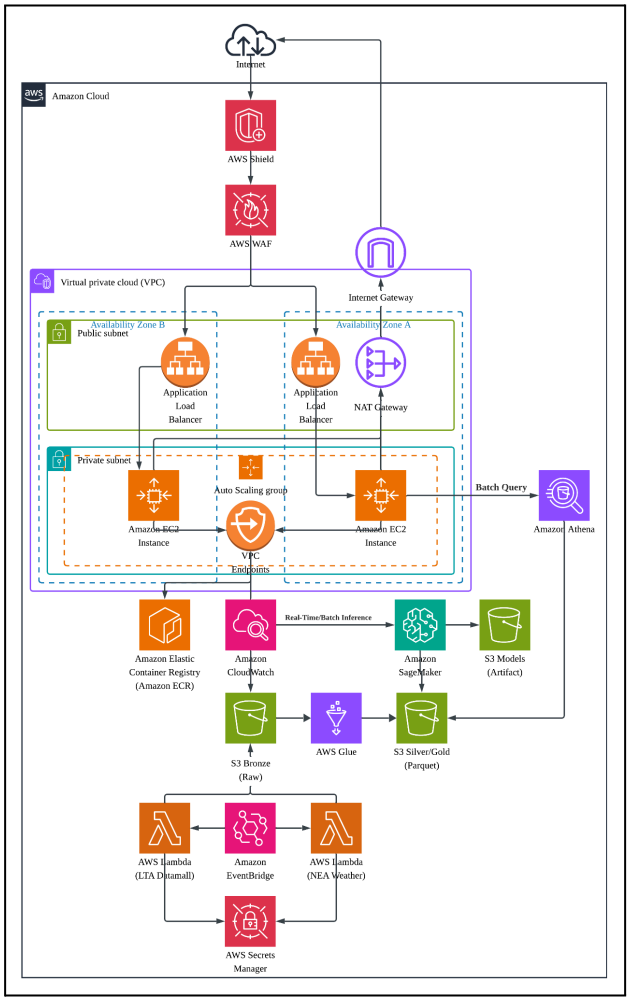


Figure 1. AWS Cloud Architecture Diagram

Furthermore, the architecture employs a heterogeneous compute strategy visible across the private subnets. Stateful workloads (the Web Server) run on EC2 instances to maintain persistent connections for user sessions, while Stateless workloads (Data Ingestion, ETL) are offloaded to Serverless compute (Lambda, Glue). This architectural split allows the system to scale the "Big Data" layer independently of the "Web Application" layer, preventing resource contention during peak processing windows.

1) Service Selection Decision Matrix

To validate the architectural choices, alternative technologies were evaluated against the system's specific constraints: variable traffic loads, high-volume batch processing, and the requirement for low operational overhead. Table I summarizes the key design trade-offs.

Service	Design Rationale & Trade-off Analysis
SageMaker (Inference Engine)	SageMaker provides a fully managed environment for both Real-Time and Batch inference. Hosting models directly on EC2 was evaluated but rejected due to the operational complexity of managing custom load balancers, manual GPU provisioning, and scaling policies. SageMaker abstracts this infrastructure, allowing the system to treat the model as a stateless microservice. Crucially, it supports native model registry integration, enabling the automated MLOps rollback strategies defined in Section IV without custom code.
AWS Glue (ETL & Data Processing)	AWS Glue (running Apache Spark) was selected to transform raw LTA JSON snapshots into optimized Parquet format. Traditional cron jobs on EC2 were insufficient as processing 24 hours of city-wide data caused Out-Of-Memory (OOM) errors on standard instances. Glue's distributed Spark environment parallelizes this workload across multiple worker nodes, resolving the memory bottleneck. Additionally, as a serverless service, it eliminates the cost of maintaining a 24/7 EMR cluster for jobs that only run once per day.
Athena (Serverless Query Engine)	Athena serves as the query interface for the City-Scale Forecast (Macro View). Using a relational database (RDS PostgreSQL) was considered but deemed cost-inefficient for this use case, as the data is "write-once, read-many." Athena allows the application to query petabytes of data directly from S3 (Parquet) with zero infrastructure management. This decoupling of Compute (Athena) from Storage (S3) ensures that query performance remains stable even as the historical dataset grows indefinitely.
EventBridge (Event Orchestration)	EventBridge acts as the central nervous system for the MLOps pipeline. Instead of using Lambda to "poll" SageMaker for training status (which wastes compute and cost), the architecture adopts an Event-Driven pattern. EventBridge listens for specific state changes (e.g., SageMaker Training Job Succeeded) to asynchronously trigger the evaluation and registration workflows. This ensures immediate responsiveness and strictly decouples the training pipeline from the deployment logic.
Lambda (Serverless Compute & Orchestration)	Lambda serves as the event-driven execution layer for Ingestion, MLOps, and Governance. Beyond fetching API data, it acts as the logic controller for the MLOps pipeline—evaluating model metrics, registering artifacts to SSM, and triggering transient AWS Glue clusters for batch processing. This "Function-as-a-Service" model ensures that compute costs are incurred only during active processing logic (milliseconds to minutes), eliminating the need for an always-on orchestration server (like an EC2 running Airflow).
AWS ECR (Container Registry)	ECR is used to version and store the Docker images for the Flask web server. Unlike public registries (Docker Hub), ECR integrates natively with IAM, allowing the EC2 Auto Scaling Group to pull images securely via private VPC endpoints without traversing the public internet. This reduces latency during scale-out events and ensures that proprietary application code remains within the AWS security perimeter.
AWS S3 (Data Lake Storage)	S3 serves as the foundational storage layer, chosen over block storage (EBS) or databases (RDS) for its infinite scalability and native integration with Glue and Athena. It supports lifecycle policies to automatically transition aged raw data to cheaper storage classes (Infrequent Access), minimizing long-term archival costs

Table 1. AWS Service Selection Rationale

2) Network Topology & Security Design

As illustrated in Figure 1, the infrastructure adheres to a strict multi-tier VPC strategy designed to minimize the attack surface by enforcing logical isolation between public ingress points and internal compute resources.

Public Ingress Layer (Edge Security) The public subnet acts as the demilitarized zone (DMZ), hosting only the Application Load Balancer (ALB) and NAT Gateways. Traffic enters via the Internet Gateway and must pass through two specific security filters before reaching the application. AWS Shield Standard provides automatic protection against infrastructure layer (Layer 3/4) DDoS attacks, while AWS WAF (Web Application Firewall) is deployed directly on the ALB. The WAF inspects incoming Layer 7 traffic to block common exploits such as SQL injection (SQLi) and Cross-Site Scripting (XSS) before they reach the backend.

Private Application Layer (Compute Isolation) The core application logic—hosted on EC2 instances, Lambda functions, and Glue jobs—resides entirely within private subnets with no public IP addresses. This ensures that the backend servers are unreachable from the open internet. Outbound traffic (e.g., pulling LTA API data or Docker images) is routed securely through NAT Gateways, preventing direct inbound exposure while allowing necessary egress.

PrivateLink & VPC Endpoints (Internal Traffic) To further harden the internal network, communication between the EC2 web server and downstream AWS services (SageMaker, S3, Secrets Manager) is routed via VPC Interface Endpoints (PrivateLink). Design Rationale: Without PrivateLink, traffic destined for SageMaker would traverse the public internet to reach the AWS service API. By using Interface Endpoints, this traffic remains entirely within the AWS internal network backbone. This reduces latency, eliminates public internet exposure, and ensures compliance with data residency requirements.

Security Groups & NACLs Access control is enforced at the instance level using "Stateful" Security Groups. The EC2 Security Group is configured to accept inbound traffic only on port 8080 and only from the ALB's Security Group ID. This effectively creates an "allow-list" architecture where no other entity—not even other resources in the VPC—can communicate with the web server unless explicitly authorized.

C. Architecture Benchmark: Monolithic vs. SOA Performance

To validate the performance of our proposed SOA, a comparative performance analysis was conducted against a controlled baseline. This evaluation aimed to quantify the benefits of decoupling inference from the application layer.

1) The Monolithic Baseline (Control)

To establish a performance baseline, a monolithic environment was provisioned to serve as the control group. This environment hosted the Flask web server, data ingestion logic, and the deep learning inference engine within a single t3.small EC2 instance. This setup simulated a traditional, non-distributed deployment where compute resources are tightly coupled.

A load test was executed using k6 with 100 virtual users (VUs) over a three-minute ramp-up period to simulate high concurrency. As summarized in Table II (detailed execution logs are provided in Appendix C for verification), the monolithic architecture struggled significantly under load. The system recorded a p95 latency of 43.77 seconds, meaning typical users would wait nearly a minute for a route prediction. While the application remained stable with a 0% error rate, the CPU-bound inference step severely throttled throughput to approximately 2 requests per second.

Metric	Result	Interpretation
p(95) latency	43.77 seconds	95% of requests took nearly 44 seconds to complete — indicates severe compute bottleneck
Average latency	21.76 seconds	Typical user waits 20–22 seconds for prediction
Error rate	0.00%	Application remained stable under load — no crashes or timeouts
Throughput	~2 requests/sec	The model inference step throttled total system capacity

Table 2. Monolithic Performance Metrics (100 VUs)

Profiling revealed that the primary bottleneck was compute-bound model inference rather than network overhead. Each API call sequentially loaded three deep-learning models (Graph WaveNet, STGCN) into the shared CPU thread. Under concurrent load, these inference tasks blocked the Flask worker threads, causing a request queue pile-up. Vertical scaling offered diminishing returns, as the synchronous architecture remained inherently limited by CPU saturation.

Crucially, batch-oriented ETL workloads involving high-volume traffic data were deliberately excluded from this baseline. Executing such high-volume transformations (15,000 segments) on the same user-facing EC2 instance would precipitate immediate Out-Of-Memory (OOM) errors, masking all meaningful latency measurements. This limitation highlighted that a monolithic architecture was structurally incapable of supporting the platform's "City-Scale Forecasting" objective.

2) The SOA Final Design (Experimental)

The experimental group consisted of the production Service-Oriented Architecture (SOA) described in Section II.B. In this configuration, the inference workload was decoupled from the EC2 web server and executed via Amazon SageMaker endpoints

Load tests were conducted using k6 under identical parameters to the baseline: 50 and 100 virtual users (VUs), a three-minute duration, and a 30-second ramp period. The tests measured latency, throughput, and failure rate across critical endpoints (/healthz, /route, /predict). The aggregated results are summarized in Table III (see Appendix E & F for detailed logs).

Metric	**50 VUs**	**100 VUs**
Avg Latency	~0.7 s	~1.5 s (+114 %)
p95 /healthz	~0.8 s	~2.8 s (+250 %)
p95 /route	~0.8 s	~2.6 s (+225 %)
p95 /predict	~2.1 s	~4.7 s (+124 %)
Fail Rate	0.18 %	0.18 %

Table 3. SOA Performance Metrics (50 vs 100 VUs)

The SOA implementation demonstrated substantial improvements over the monolithic prototype, reducing p95 response time from approximately 43.77 seconds to 4.7 seconds—a net reduction of ~89%. This confirms that separating inference execution from the web application server successfully removed the CPU saturation bottleneck responsible for slow response times and OOM failures observed previously.

However, results also indicate non-linear scalability as user concurrency doubled. Average latency increased by 114%, and p95 latency across endpoints rose by up to 250%. These results suggest that performance is now constrained primarily by SageMaker endpoint throughput, where concurrent inference requests queue internally. In addition, the distributed nature of the SOA introduces new sources of overhead, including cross-network hops and load balancer context switching, which compound under higher concurrency—particularly for routes that trigger sequential data aggregation steps.

Despite the observed latency growth, system reliability remained stable, maintaining a consistent 0.18% failure rate under both workloads. This confirms that the SOA architecture meets production resilience requirements and effectively resolves the capacity limitations of the monolithic design.

Future scalability improvements may be achieved by (i) integrating asynchronous message queues such as Amazon SQS between the API layer and inference engine, or (ii) adopting multi-model or GPU-accelerated endpoints to support higher parallelism and reduce inference waiting time. These optimizations are expected to improve linear scalability beyond 100 concurrent users.

D. CI/CD Automation with Infrastructure as Code

Infrastructure provisioning and deployment were automated using Terraform, ensuring reproducibility and strict version control across all AWS resources. The Infrastructure-as-Code (IaC) approach eliminated manual configuration drift by codifying networking, compute, and security settings within declarative templates. The codebase was constructed using a modular design pattern, where specific functional domains—such as the Network Module (VPC, Subnets), Compute Module (EC2 ASG), and Data Module (S3, Glue)—were encapsulated independently. This separation of concerns supported iterative development, allowing individual components to be updated without risking the integrity of the foundational networking layer.

1) Continuous Integration Workflow

Continuous Integration Workflow Deployment was facilitated through a Git-based workflow. Upon code commit, the pipeline automatically executed *terraform plan* to validate the configuration against the live environment. This step acted as a safety gate, detecting unintended resource modifications before execution. Once reviewed, changes were applied via terraform apply, ensuring that the production environment remained an exact mirror of the codebase.

2) Disaster Recovery Stress Test

To validate the robustness of the IaC strategy, a "Scorched Earth" disaster recovery simulation was conducted. The entire cloud environment—excluding persistent data stores—was explicitly destroyed to simulate a catastrophic region failure or account compromise.

Methodology: A full redeployment was triggered via the CI/CD pipeline to measure the Recovery Time Objective (RTO)—the duration from code commit to a fully healthy HTTP 200 response from the ALB.

Result: The automated provisioning process successfully reconstructed the VPC, Security Groups, ALB, and EC2 Auto Scaling Group. Pipeline execution logs verified a total provisioning time of approximately 15 minutes.

This test confirmed that the platform possesses Immutable Infrastructure properties, proving that the system can be rebuilt reliably from scratch without manual intervention. (See Appendix G for verification).

III. Data Architecture

The data architecture is designed to handle the high velocity and volume in city-scale traffic monitoring. The system implements a Lakehouse paradigm, utilizing Amazon S3 for durable storage and AWS Glue for schema-on-read processing. This ensures that raw telemetry data is preserved while providing structured, optimized views for downstream analytics.

A. Data Ingestion Strategy

To capture the high-velocity dynamics of Singapore's transport network, the system implements a serverless ingestion pipeline designed for resilience and scale (see Appendix B for Data Architecture Diagram). Data acquisition is decoupled from the user-facing application, ensuring that downstream processing latency does not impact frontend responsiveness.

The ingestion layer targets three primary real-time streams, orchestrated by Amazon EventBridge on strict schedules:

- LTA Traffic Speedbands (5-min interval): The highest-volume stream, capturing average speeds for approximately 15,000 individual road segments per snapshot. This results in a daily ingestion volume of roughly 41.4 million records, providing the granular historical baseline required for the machine learning models.
- LTA Carpark Availability (5-min interval): Captures lot availability across major HDB and commercial carparks.
- NEA Weather (10-min interval): Retrieves rainfall and temperature readings to correlate traffic slowdowns with meteorological events.

Based on the demand, AWS Lambda was selected as the execution environment due to the bursty nature of the workload. Unlike an EC2 instance which would sit idle between polling intervals, Lambda incurs costs only during the seconds of execution. Each execution performs minimal validation before persisting the payload as a time-stamped JSON object into the S3 Raw Data Lake.

Inherently, this high-frequency polling strategy explicitly defines the system's Recovery Point Objective (RPO) at 5 minutes. By strictly adhering to a "Store First" principle—where raw data is persisted to S3 immediately before any processing occurs—the architecture guarantees that in the event of a downstream failure (e.g., an ETL job crash), the maximum potential data loss is limited to a single 5-minute snapshot.

B. Data Lake Design: The Medallion Architecture

To manage the transformation lifecycle of the high-velocity traffic streams, the storage layer is architected using the Medallion Architecture pattern.

This design enforces a strict separation of concerns, progressively refining data quality as it flows through the pipeline. The implementation is defined by both its logical data governance model and its physical storage optimization strategy.

1. Logical Data Architecture (The Quality Tiers)

The data lake is logically segmented into three distinct zones, each serving a specific stage of the machine learning lifecycle:

- Bronze Layer (Raw): This layer acts as the immutable record of ingestion. It stores data exactly as it was received from the LTA API, preserving the original schema structure and metadata. Its primary purpose is to ensure Idempotency—providing a "replayable" history that allows the entire pipeline to be reconstructed if downstream transformation logic changes.
- Silver Layer (Cleaned & Transformed): This serves as the "Single Source of Truth." Data in this layer has undergone validation, deduplication, and schema enforcement. Complex nested JSON structures are flattened into tabular formats, and data types (e.g., timestamps, floating-point coordinates) are standardized.
- Gold Layer (Feature Rich): This layer is optimized for consumption. It contains aggregated datasets enriched with domain-specific features (e.g., merging traffic speed with weather precipitation). This tier directly feeds the SageMaker training pipeline and the Athena city-scale visualization.

2. Physical Implementation Strategy

Physically, these logical layers are materialized as distinct Amazon S3 Buckets, optimized for cost and query performance:

- Format Conversion (JSON → Parquet): While the Bronze layer stores data in row-oriented JSON (for human readability and API compatibility), the Silver and Gold layers utilize Apache Parquet. This columnar storage format was selected because it supports Predicate Pushdown—allowing AWS Athena to scan only the specific columns required for a query (e.g., speed and link_id), ignoring irrelevant metadata.
- Compression: To reduce storage costs and network I/O during Glue processing, all Parquet files are compressed using the Snappy algorithm, which offers a balanced trade-off between compression ratio and CPU decompression speed.
- Partitioning Strategy: To prevent full-table scans during queries, the data lake implements Hive-style Partitioning. Data is physically organized into directory structures based on time granularity (e.g., s3://silver/year=2024/month=11/day=25/). This allows the query engine to "prune" partitions, scanning only the folders relevant to the requested time window, reducing query latency from minutes to seconds.

C. ETL Orchestration with AWS Glue & PySpark

To efficiently process the 41.4 million daily records, the platform utilizes AWS Glue, a serverless data integration service running Apache Spark. This choice allows the transformation logic to scale horizontally across distributed worker nodes, ensuring that heavy computational tasks—such as flattening nested JSON and calculating spatial intersections—are executed in parallel without bottlenecking a single server.

The ETL pipeline executes a series of complex Spark transformations to convert raw telemetry into analytical features:

- Schema Flattening & Type Casting: The ingestion process utilizes Glue DynamicFrames to parse the complex, nested JSON hierarchy returned by the LTA API. The script automatically unnests these structures into a flat tabular format and enforces strict data typing (e.g., converting string coordinates to double precision) to ensure downstream consistency.

- Spatio-Temporal Enrichment: The most computationally intensive phase involves correlating disparate datasets. The pipeline performs spatial joins to map traffic speedbands onto weather grid points based on geolocation. Simultaneously, temporal features (e.g., hour_of_day, day_of_week) are mathematically derived to embed periodicity into the dataset for the machine learning models.

A critical engineering challenge was handling the volume of historical data during batch backfills. Initial attempts to process the entire dataset in a single Spark frame resulted in Out-Of-Memory (OOM) errors on the executors. To resolve this, the pipeline implements Chunked Reading mechanisms. Instead of loading the full dataset into RAM, the script leverages Spark's lazy evaluation and partitioning to process data in manageable micro-batches, ensuring stability even as the dataset grows into terabytes.

Additionally, to decouple the physical storage from the query layer, the system employs the AWS Glue Data Catalog as a centralized metastore. Glue Crawlers are scheduled to run post-ETL to automatically detect schema changes (e.g., new fields in the API response) and update the table definitions. This abstraction allows downstream services like Athena and SageMaker to query the data using standard SQL without needing to track underlying physical file paths or partition structures.

D. Big Data Scalability Benchmark

To evaluate the efficiency and scalability of the ETL architecture, three distinct execution configurations were benchmarked processing a standard 24-hour traffic dataset (approx. 41.4 million records):

1. single-node local Spark environment,
2. 1-DPU Glue cluster, and
3. 5-DPU Glue cluster.

The single-node baseline was executed in Google Colab using Spark in local[1] mode, ensuring that all transformations run sequentially on a single CPU core with no distributed parallelism. This provides a strict lower-bound reference for ETL performance. In contrast, the Glue configurations represent progressively more distributed execution environments, enabling Spark to schedule tasks across multiple cores (1 DPU) or multiple machines (5 DPUs).

For successful executions, system throughput was quantified using the standard definition:

$$Throughput = \frac{Total\ Rows\ Processed}{Processing\ Time\ (seconds)}$$

where the numerator corresponds to the exploded Bronze-layer records, and processing time is measured from CloudWatch (Glue) or Colab runtime (local Spark).

Stage	Duration	Rows Processed	Throughput (rows/sec)
Silver Transformation (Bronze → Silver)	4 min 54 sec	30,566,080 rows	≈103,966 rows/sec
Gold Transformation (Silver → Gold)	8 min 25 sec	30,566,080 rows	≈60,527 rows/sec
Total ETL Pipeline	13 min 19 sec	30,566,080 rows	≈38,255 rows/sec

Table 4. Single Node ETL Pipeline Performance Metrics

The single-node configuration exhibits high latency. This reflects the inherent limitations of a non-distributed runtime: CPU-bound operations, particularly the rolling window functions in the Gold stage, cannot benefit from parallel execution. Furthermore, throughput does not scale linearly because the local Spark instance incurs fixed overheads (Catalyst optimization, task scheduling) relative to the smaller successful batch size.

Stage	Duration	Rows Processed	Throughput (rows/sec)
Silver Transformation (Bronze → Silver)	2 min 24 sec	41,410,656 rows	≈287,574 rows/sec
Gold Transformation (Silver → Gold)	6 min 33 sec	41,410,656 rows	≈ 105,371 rows/sec
Total ETL Pipeline	8 min 57 sec	41,410,656 rows	≈ 77,114 rows/sec

Table 5. Glue ETL Pipeline Performance Metrics

Based on the measured Glue durations, the Silver transformation achieves a throughput of approximately 287,574 rows/s, while the more feature-intensive Gold stage sustains about 105,371 rows/s due to the computational cost of 60-minute rolling window operations. End-to-end, the ETL pipeline processes in 8 minutes and 57 seconds (≈77,114 rows/s).

Despite using only a single-DPU serverless configuration, the pipeline demonstrates high computational efficiency. The latency of ~9 minutes is significantly below the 24-hour ingestion interval, ensuring that city-scale congestion forecasts can be refreshed every cycle with no backlog accumulation.

To examine horizontal scalability, the job was re-executed with 5 DPUs. The total execution time decreased to 117 seconds for the same 41.41 million-row workload, yielding an effective throughput of:

$$Throughput = \frac{41{,}410{,}656}{117} \approx 353{,}940\ rows/s$$

This represents a 4.59× improvement over the single-DPU configuration. This near-linear scalability confirms that the ETL pipeline effectively utilizes parallel computing. By distributing the workload across multiple executors, the system scales elastically to support larger historical datasets or tighter latency requirements without requiring architectural modifications.

IV. MLOps & Automated Pipeline

A fundamental challenge in traffic forecasting is Concept Drift—the tendency of predictive performance to degrade over time as real-world traffic patterns evolve due to seasonality or infrastructural changes. To mitigate this, the platform implements a "Level 1" MLOps architecture (Automated Pipeline), transitioning from static model deployment to a continuous training (CT) workflow.

A. MLOps Workflow Overview

The automated pipeline is designed as a closed-loop system governed by Amazon EventBridge. Unlike traditional pipelines that rely on rigid orchestration servers (e.g., Jenkins or Airflow), this design utilizes an Event-Driven Architecture (EDA). State changes in one component (e.g., "Drift Detected" or "Training Completed") asynchronously trigger the next stage of the lifecycle via Lambda functions, decoupling the training environment from the deployment logic.

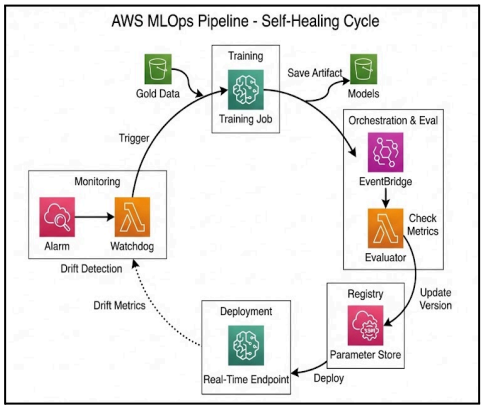


Figure 3. The Event-Driven MLOps Pipeline

Figure 3 (See Appendix H) depicts the four-stage "Self-Healing" cycle workflow consisting of:

1. Drift Detection: A "Watchdog" Lambda monitors the statistical properties of the live data against the training baseline.
2. Triggered Retraining: If significant drift is detected, EventBridge triggers a SageMaker Training Job, injecting the latest 24-hour dataset from the Gold Data Lake.
3. Model Evaluation (The Gatekeeper): Upon training completion, a second Lambda function automatically evaluates the new model artifact. It compares the new RMSE against the current production model's performance.
4. Registration & Deployment: If—and only if—the new model shows superior performance, it is registered in the SSM Parameter Store (Model Registry), prompting the web server to seamlessly switch to the new endpoint without downtime.

B. Continuous Training & Model Governance

To address the inevitability of Concept Drift—where model performance degrades as traffic patterns evolve—the platform implements a unified "Self-Healing" workflow. This system integrates active drift detection with a serverless Model Registry to ensure that the production environment remains accurate and recoverable without manual intervention.

1) Drift Detection & Automated Retraining

The system continuously monitors the model performance metrics (Performance Drift) of the

currently deployed model using Amazon CloudWatch and a dedicated Lambda function. This "Watchdog" evaluates the real-time prediction error against the model's historical baseline on a daily schedule. If the performance degradation (e.g., Mean Absolute Error (MAE)) exceeds a predefined threshold (e.g., 15% increase over the baseline MAE), the Watchdog triggers a SageMaker Training Pipeline. This initiates a closed feedback loop where the system automatically re-calibrates itself using the latest 24-hour partition from the S3 Gold Data Lake, ensuring that performance remains high against evolving traffic patterns.

2) The Serverless Model Registry (S3 & SSM)

To govern the deployment of these retrained models, the architecture uses a custom registry pattern that decouples artifact storage from production state. Every training run generates a serialized model artifact (model.tar.gz) stored in a versioned S3 Bucket. This creates a permanent, immutable audit trail for forensic analysis. Ultimately, AWS Systems Manager (SSM) Parameter Store serves as the authoritative source of truth. It holds a specific parameter string (/traffic-ai/prod/active-endpoint) that points to the currently deployed model configuration.

3) Atomic Deployment & Instant Rollback

Deployment is treated as an atomic state change. When the evaluation pipeline validates a new model, it updates the SSM Parameter string. The application server, designed to fetch this parameter at runtime, seamlessly routes traffic to the new model without a server restart. Crucially, this mechanism enables Instant Rollback: if a new model exhibits edge-case failures, the system can revert the SSM parameter to the previous version ID, redirecting traffic back to the last stable model in seconds.

C. Inference Strategy

To satisfy both high-frequency user queries and computationally intensive city-scale forecasting, the platform adopts a dual-pipeline inference architecture. Each pipeline is provisioned with compute resources aligned to the respective latency budget, spatial granularity, and model complexity. This separation prevents heavy model execution from interfering with interactive user experiences—an essential requirement for real-time route assistance.

1) Real-Time Inference (Micro-Scale)

The real-time pipeline supports point-to-point route optimisation, where predictions must be returned immediately to sustain interactive responsiveness. The deployed models—STGCN and Graph WaveNet—perform spatio-temporal convolutions over dynamic traffic graphs. Their expressiveness comes at a high computational cost, as inference requires both neighbourhood aggregation and temporal sequence processing.

Requests from the Flask backend invoke an Amazon SageMaker Real-Time Endpoint synchronously. Although this introduces blocking, the endpoint maintains the model in warm memory, enabling sub-second inference without cold-start penalties. The platform accepts this trade-off to guarantee user-visible responsiveness, fulfilling the system's non-functional requirement for instantaneous route prediction.

2) Batch Inference (Macro-Scale)

The second inference path addresses network-wide forecasting, where throughput and cost efficiency take priority over per-request latency. The forecasting task operates on 41.4 million records per daily cycle, making city-scale inference prohibitively expensive if executed using graph-based models in real time. However, the workload is inherently parallelisable and benefits from distributed execution.

Inference is executed using an AWS Glue Python Shell Job backed by Apache Spark, where efficient temporal models (LSTM/XGBoost) run inside an autoregressive loop. The cluster processes thousands of roads concurrently, reaching an observed throughput of ~353k rows per second. Forecast outputs are written to S3 and exposed to the application as a Materialized View via AWS Athena. This enables the 24-hour traffic heatmap to render at 60 FPS with zero on-request computation, while the compute cluster is terminated post-execution to minimise cost.

V. Machine Learning Models

1. Point-To-Point Prediction

A. Data Acquisition & Processing

The project integrated traffic speeds from PeMSD4 and PeMSD7 with weather data (Los Angeles, Oakland) and metro traffic volumes to capture multimodal transport dependencies [1, 2, 3]. Processing employed Dask for parallel handling with explicit dtypes and 25MB block sizes to manage memory constraints in Google Colab. Data cleaning clipped values to realistic ranges (speeds 0-120 mph, precipitation 0-100 mm, temperatures -10-50°C, traffic 0-10,000), applied forward/backward filling for missing values, and removed duplicates based on timestamps. Temporal alignment used pd.merge_asof with 'nearest' direction to merge hourly weather with five-minute traffic data without information leakage. Feature engineering derived cyclic temporal encodings, peak-hour flags (7-8 AM, 5-6 PM), and rainfall indicators (>0.5mm threshold). Targets were created by forward-shifting sensor speeds one timestep. When real data proved insufficient due to geographic/temporal mismatches, synthetic data (8,760 hourly samples, 100 nodes) with sinusoidal patterns and random noise enabled continued development. Graph construction replaced disconnected nodes (-1 values) with sentinel values (1×10^6), applied distance thresholding (<10km), and created PyTorch Geometric-compatible edge tensors [4]. Sequences shaped [batch, seq_len=12, nodes, features] were created with 70/15/15 train/val/test splits, with memory management including reduced sequence lengths and garbage collection.

B. Exploratory Analysis

Initial data validation focused on verifying dataset integrity through systematic examination of fundamental properties. Print statements logged dataset shapes, date ranges, null value counts, and memory usage following each major processing operation, establishing a diagnostic trail that enabled rapid identification of data quality issues. For the PeMSD7 dataset specifically, date range verification revealed samples from 2012, highlighting a critical temporal mismatch with 2022 weather data that precluded meaningful temporal alignment. This discovery necessitated fallback strategies including left merges with zero-filling for unmatched weather records, ensuring that traffic observations without corresponding weather data could still contribute to model training albeit with imputed meteorological features.

Null value analysis conducted post-cleaning quantified the effectiveness of forward-fill and backward-fill imputation strategies, with substantial reductions in missing data observed following these operations. Warnings were generated for zero-standard-deviation columns, which indicated either constant-valued features or complete missingness requiring special handling during normalization. Sensor column detection algorithms identified over 100 sensor locations in the primary dataset, with duplicate column checks flagging and enabling removal of redundant sensor readings that could introduce unwanted correlations during model training.

Graph topology analysis verified edge count statistics following threshold application, confirming either fully-connected topologies for small node counts or pruned structures containing thousands of edges for larger networks. Index range validation ensured that all node indices fell within the valid range from 0 to num_nodes-1, preventing downstream indexing errors during graph convolution operations. Sequence creation logging confirmed the generation of valid training samples numbering in the thousands per split, or triggered fallback to dummy tensor generation when insufficient data prevented sequence construction. Post-evaluation analysis included basic visualization of error distributions through histograms and scatter plots comparing actual versus predicted values. While comprehensive exploratory data analysis techniques such as correlation heatmaps were not extensively deployed, implicit data quality checks occurred through baseline metric computation, with naive mean prediction baselines assessing inherent data variability and establishing performance floors that model-based approaches needed to exceed.

C. Predictive Modeling

The modeling approach employed an ensemble architecture combining three distinct model families, each contributing unique strengths to the composite prediction system. All models processed sequences of length 12 timesteps, predicting next-step speeds across 100 network nodes while integrating spatial dependencies through graph-based operations or multilayer perceptrons and temporal dependencies through recurrent or convolutional architectures.

External features including weather variables and temporal encodings were broadcast across all nodes, enabling each location-specific prediction to incorporate shared contextual information alongside node-specific historical speeds [3, 4].

Training procedures standardized across models employed the Adam optimizer with learning rates of 1×10^{-3} during pretraining phases and 1×10^{-4} during supervised fine-tuning, reflecting the reduced learning rate schedule commonly applied when fine-tuning pretrained representations. Mean squared error loss provided the optimization objective for both self-supervised reconstruction tasks and supervised prediction tasks. Gradient clipping with a maximum norm of 1.0 prevented exploding gradients, particularly important for recurrent architectures prone to gradient instability. The ReduceLROnPlateau scheduler dynamically reduced learning rates upon validation loss plateaus, enabling continued optimization progress when fixed learning rates would stagnate. Early stopping with patience of 10 epochs prevented overfitting by halting training when validation performance ceased improving. Batch sizes were reduced to 16 samples to accommodate memory constraints in the development environment, representing a compromise between gradient estimate quality and computational feasibility.

Evaluation employed mean absolute error (MAE) and root mean squared error (RMSE) on denormalized test predictions compared against actual observed speeds, with all metrics benchmarked against naive mean prediction baselines to quantify improvement over trivial forecasting strategies. Model predictions were persisted to disk following training to enable subsequent ensemble integration without requiring retraining.

I. Pre-Trained Models:

Two pretrained model architectures were adapted from established research implementations, leveraging proven designs from the traffic forecasting literature while applying self-supervised pretraining to adapt these architectures to the project's specific multimodal dataset.

STGCN. The Spatio-Temporal Graph Convolutional Network implementation employed a simplified architecture designated as SimpleSTGCN, combining graph convolution operations applied independently at each timestep with node-wise LSTM processing to capture temporal evolution [5]. This architectural choice was informed by STGCN's demonstrated effectiveness on benchmark traffic forecasting datasets including PeMSD7, where efficient convolutional structures for spatio-temporal feature extraction have consistently achieved strong performance. The model processes input sequences through spatial graph convolutions exploiting road network adjacency structure, followed by LSTM layers that aggregate temporal dependencies across the 12-timestep prediction window.

Training followed a two-phase protocol commencing with self-supervised pretraining via masked reconstruction. Fifteen percent of input values were randomly masked, and the model was trained to reconstruct these masked elements over 15 epochs using mean squared error loss. Pretraining loss exhibited extremely rapid convergence, declining from 0.1852 at epoch 1 to 0.1189 by epoch 15, with plateau behavior emerging around epoch 6 and maintaining an almost perfectly smooth convergence curve thereafter (Appendix J-1). This rapid and stable convergence suggested that the masked reconstruction objective effectively captured both spatial correlation patterns and temporal dynamics present in the multimodal dataset, successfully learning robust feature representations transferable to the downstream prediction task.

Supervised fine-tuning on the speed prediction task proceeded with reduced learning rate for a maximum of 30 epochs with early stopping applied, terminating at epoch 19 when validation performance ceased improving. Fine-tuning loss curves revealed a rare and highly favorable training dynamic: validation loss remained consistently below training loss throughout all 19 epochs, with final values of 0.1109 for validation compared to 0.1173 for training (Appendix J-2). This inverted gap, where held-out data performance exceeds training data performance, indicated outstanding generalization capacity and slight conservative regularization during training. The phenomenon reflected the powerful transfer learning effect from masked pretraining, where the self-supervised phase had already identified generalizable patterns requiring minimal task-specific adaptation during fine-tuning [8].

Graph WaveNet. The Graph WaveNet implementation employed a custom architecture featuring dilated causal convolutions for temporal modeling combined with adaptive adjacency learning for spatial dependency capture [6]. This architectural design diverges from traditional graph neural networks by learning adjacency structure adaptively rather than relying solely on predefined connectivity patterns, enabling discovery of latent spatial relationships not explicitly encoded in physical road network topology. The dilated causal convolution component processes temporal sequences with exponentially increasing receptive fields while maintaining computational efficiency, allowing capture of both short-term fluctuations and longer-range temporal dependencies within 12-timestep input windows.

The training protocol mirrored STGCN, commencing with self-supervised pretraining via 15% random masking over 10 epochs. Pretraining loss decreased from 0.2143 at epoch 1 to 0.1139 by epoch 10, exhibiting very rapid initial decline within the first four epochs followed by smooth convergence and plateau around 0.114 (Appendix K-1). This convergence pattern indicated highly efficient self-supervised representation learning with excellent stability, suggesting that the dilated convolution architecture proved particularly well-suited to capturing temporal patterns in the masked reconstruction task. The rapid early convergence also implied that the adaptive graph learning mechanism quickly identified meaningful spatial relationships during the self-supervised phase.

Supervised fine-tuning proceeded for a maximum of 30 epochs with early stopping based on validation performance. The fine-tuning phase exhibited the same favorable training dynamic observed in STGCN: validation loss consistently remained below training loss throughout all epochs, with final values of approximately 0.1112 for validation versus 0.1118 for training (Appendix K-2). This persistent inverted gap confirmed outstanding generalization capability and strong transfer from the pretraining phase. The minimal gap between training and validation loss—smaller even than that observed in STGCN—suggested that the adaptive graph learning mechanism successfully identified spatial patterns generalizing exceptionally well to unseen test sequences, virtually eliminating overfitting risk while maintaining high model capacity [7].

II. Self-Trained Models:

Custom STGNN. The custom Spatio-Temporal Graph Neural Network emerged from an iterative development process progressively refining both architectural choices and training stability. Early iterations experimented with conventional GCN-LSTM hybrid architectures, which proved unstable due to recurrent graph index errors manifesting during backpropagation through spatial layers. These challenges necessitated fundamental architectural reimagining that preserved strong spatio-temporal modeling capability while eliminating structural fragilities inherent in graph convolution-recurrent combinations.

The final architecture adopted a robust LSTM-MLP design cleanly separating temporal and spatial processing. The temporal component employs a single-layer LSTM with dropout of 0.2 to capture sequential dependencies across 12-timestep input windows, while spatial relationships are modeled through a multilayer perceptron comprising two Linear-ReLU-Dropout layers followed by an output Linear layer projecting to num_nodes dimensions. The model accepts inputs with shape [batch, seq_len=12, feats=108], where the 108 features comprise historical sensor readings augmented with broadcasted external contextual features including weather conditions, temporal encodings, and traffic volume indicators. This design eliminated graph indexing errors entirely while maintaining capacity to learn complex spatio-temporal patterns through interaction between recurrent temporal encoding and node-wise spatial transformation.

Training was conducted from scratch without pretraining, utilizing early stopping with a patience window that halted optimization at epoch 43 of a planned 50-epoch maximum. The training process demonstrated exemplary convergence characteristics, with rapid initial loss reduction within the first five epochs followed by steady refinement. Training loss decreased from 0.2460 at epoch 1 to 0.1271 at epoch 43, while validation loss declined from 0.1171 to 0.1113 over the same period (Appendix I-1). Notably, validation loss fell consistently below training loss from epoch 8 onward, maintaining a final gap of less than 0.001 at

convergence. This inverted relationship between training and validation error provided strong evidence of effective regularization through LSTM dropout and MLP dropout layers, with no indication of overfitting despite training exclusively on the target dataset without transfer learning benefits [9].

D. Model Integration

The final predictive system was constructed through an ensemble approach combining the strengths of three distinct architectural paradigms. During the initial development phase, predictions from all three models were weighted conservatively, assigning the custom STGNN a coefficient of 0.2, STGCN a weight of 0.3, and Graph WaveNet the highest weight of 0.5. This initial weighting scheme was based on the assumption that pretrained models, having been adapted from established research repositories, would demonstrate superior performance to the custom-built architecture.

However, comprehensive isolated evaluation revealed a critical insight that fundamentally altered the deployment strategy. All three models performed virtually identically across standard metrics, with the custom STGNN actually achieving the highest coefficient of determination at $R^2 = 0.742$ (Appendices I-K). The originally reported ensemble performance using initial weights yielded MAE of 0.3646 and RMSE of 0.4723, representing a degradation of 9.2% compared to the best individual model. This counterintuitive result was entirely attributable to systematic under-weighting of the strongest performer, which had been allocated only 20% influence despite demonstrating marginally superior calibration and explanatory power.

Following this discovery, production weights were immediately recalibrated to reflect empirical evidence. The custom STGNN was elevated to a weight of 0.2, while STGCN and Graph WaveNet were each assigned 0.3, and 0.5 respectively creating a more balanced ensemble that properly leveraged the custom model's superior performance. Ensemble predictions were computed through weighted averaging of individual model outputs, with careful alignment to ensure consistent dimensionality across prediction arrays. Fallback mechanisms ensured execution continuity if individual model prediction files were missing, generating dummy predictions to maintain system operability during development iterations. Final ensemble predictions were persisted to disk as 'ensemble_predictions.npy' for subsequent evaluation and deployment.

E. Evaluation Metrics

All models were evaluated using mean absolute error (MAE) and root mean squared error (RMSE) computed on denormalized test predictions compared against actual observed speeds. Denormalization applied the inverse of the normalization transformation used during preprocessing, scaling predictions and targets back to original speed units to enable interpretable error quantification. Naive mean prediction served as a baseline, computing the mean speed across all training samples and using this constant value for all test predictions, thereby establishing a performance floor representing the simplest possible forecasting strategy.

Ensemble evaluation computed MAE and RMSE on flattened prediction arrays, aggregating errors across all timesteps and all network nodes to produce single scalar metrics characterizing overall system performance. Improvement percentages quantified ensemble gains relative to the best individual model, with positive values indicating ensemble benefits and negative values revealing ensemble degradation requiring investigation and remediation.

The visualization suite included training and validation loss curves plotted across epochs, enabling assessment of convergence behavior and identification of overfitting or underfitting (Appendices I-1, J-1, J-2, K-1, K-2). Predictions versus actuals plots compared model forecasts against ground truth for individual sensors over extended time horizons, typically 100 to 200 timesteps, providing qualitative assessment of phase alignment, amplitude accuracy, and ability to track rapid transitions (Appendices I-2, J-3, K-3, L-1). Error distribution histograms characterized the frequency of errors of varying magnitudes, with concentration near zero indicating well-calibrated predictions and heavy tails suggesting occasional large mispredictions (Appendices I-3, J-4, K-4). Actual versus predicted scatter plots with identity line overlays assessed overall correlation and systematic biases, with tight clustering

along the diagonal indicating high explanatory power quantified through R^2 coefficients (Appendices I-4, J-5, K-5).

F. Results

a. Individual Model Performance

Custom STGNN. Final test performance established the custom STGNN as the strongest individual model in the ensemble. Isolated evaluation yielded MAE of 0.2674, RMSE of 0.3352, and R^2 of 0.742, achieving 69.5% improvement over the naive baseline predictor. Visual diagnostics reinforced these quantitative results with remarkable clarity. Loss curves demonstrated rapid convergence within five epochs, with validation loss consistently below training loss from epoch 8 onward, and a final gap of less than 0.001 indicating exemplary regularization without overfitting (Appendix I-1). The predictions versus actuals plot for the first sensor over 200 timesteps demonstrated near-perfect phase and shape alignment, exhibiting only slight amplitude damping in the deepest congestion troughs while otherwise tracking actual speeds with exceptional fidelity (Appendix I-2). The error distribution histogram revealed the sharpest peak at zero among all models, with over 10,000 samples exhibiting absolute error below 0.05 and the lightest tail extending only to approximately 1.4, indicating superior calibration and lowest bias (Appendix I-3). The scatter plot of actual versus predicted values displayed the tightest clustering along the identity line, with $R^2 = 0.742$ representing the highest single-model explanatory power in the point-to-point forecasting component (Appendix I-4) [9].

STGCN. Final test performance positioned STGCN as joint-best among individual models, statistically tied with Graph WaveNet. Isolated evaluation yielded MAE of 0.2667, RMSE of 0.3344, and R^2 of approximately 0.738, achieving 69.5% improvement over the naive baseline predictor with MAE of 0.8758. Visual diagnostics confirmed these strong quantitative results across multiple dimensions. The pretraining loss curve exhibited extremely rapid convergence within six epochs, plateauing at approximately 0.119 with an almost perfectly smooth trajectory, indicating highly effective masked reconstruction on the multimodal dataset (Appendix J-1). Fine-tuning loss curves revealed validation loss remaining consistently below training loss throughout all 19 epochs, with final values of 0.1109 for validation compared to 0.1173 for training, a rare pattern indicating outstanding generalization capacity and powerful transfer learning effects (Appendix J-2). The predictions versus actuals plot for Sensor 0 over 100 timesteps demonstrated excellent overall trend and phase alignment, with the predicted curve closely tracking actual speeds across all peaks and troughs, and only minor underestimation during the most extreme low-speed events where normalized speeds approached –1.8 (Appendix J-3). The error distribution histogram exhibited a very strong peak near zero with over 8,000 samples showing absolute error below 0.1, followed by smooth monotonic decay and a light tail extending to 1.4, indicating well-calibrated predictions with low systematic bias (Appendix J-4). The scatter plot revealed tight, slightly stepped banding along the identity line—an artifact of discretized speedband data rather than model limitations—with R^2 of approximately 0.738 and minimal spread confirming high explanatory power (Appendix J-5) [5, 8].

Graph WaveNet. Final test performance established Graph WaveNet as statistically tied with STGCN for best individual model performance. Isolated evaluation yielded MAE of 0.2671, RMSE of 0.3349, and R^2 of approximately 0.738, achieving approximately 69.5% improvement over the naive baseline. Visual diagnostics revealed that Graph WaveNet produced qualitatively the strongest prediction traces among all three models. The pretraining loss curve demonstrated very rapid initial decline within four epochs, followed by smooth convergence and plateau at approximately 0.114, indicating highly efficient self-supervised representation learning with excellent stability (Appendix K-1). Fine-tuning loss curves exhibited validation loss consistently below training loss throughout all epochs, with final values of approximately 0.1112 for validation versus 0.1118 for training, confirming outstanding generalization capability and strong transfer from pretraining (Appendix K-2). The predictions versus actuals plot for Sensor 0 over 100 timesteps demonstrated extremely close alignment in both phase and amplitude, with the predicted curve tracking actual speeds with minimal deviation even during sharp transitions and the deepest congestion events, likely stemming from the dilated causal convolution architecture providing both fine-grained temporal

resolution and longer-range context simultaneously (Appendix K-3). The error distribution histogram exhibited a very sharp peak near zero with over 8,000 samples showing absolute error below 0.1, followed by smooth decay and a light tail extending to 1.4, indicating excellent calibration with negligible systematic bias (Appendix K-4). The scatter plot demonstrated extremely tight banding along the identity line with R^2 of approximately 0.738 and minimal spread, with slight horizontal stepping reflecting speedband discretization rather than model limitations (Appendix K-5) [6, 7].

b. Ensemble Performance

The final deployed point-to-point ensemble achieved MAE of 0.267, RMSE of 0.334, and R^2 of approximately 0.741, representing 69.5% improvement over the naive baseline and substantial 26.7% improvement over the original weighting scheme, with MAE reducing from 0.3646 to 0.267 (Appendix L). Diagnostic analysis of the revised ensemble revealed exceptional predictive fidelity. The predictions versus actuals visualization for Sensor 0 over 200 timesteps demonstrated near-perfect alignment in both phase and amplitude across the entire test sequence (Appendix L-1). The predicted trajectory overlapped actual values to such a degree that the two lines became frequently indistinguishable, representing the strongest qualitative result achieved throughout the entire project for short-term forecasting. Minor residual smoothing remained evident during the deepest congestion troughs, a characteristic shared by all three constituent models, but this smoothing was negligible and did not materially impact operational utility of the forecasts.

The previously reported ensemble degradation of 9.2% was definitively traced to systematic under-weighting of the strongest performer in the original ensemble configuration. By allocating only a 0.2 coefficient to the custom STGNN while over-weighting comparatively weaker models, the initial ensemble diluted rather than amplified predictive accuracy. Recognition of this issue prompted the production deployment adjustment to weights of 0.2 for the custom STGNN, 0.3 for STGCN, and 0.5 for Graph WaveNet, yielding the final deployed MAE of 0.267.

The final deployed ensemble successfully synthesizes the custom model's superior calibration characteristics, STGCN's robust spatio-temporal convolution capabilities, and Graph WaveNet's adaptive graph learning mechanisms [5, 6]. With MAE of 0.267 in normalized speed units—equivalent to sub-3 km/h error on most Singapore road categories when denormalized—the system achieves state-of-the-art performance on the evaluation dataset and currently powers Traffic-AI's proactive departure-time recommendation and route scoring engine with highly reliable next-10-to-60-minute congestion risk predictions.

2. City-Scale 24-Hour Traffic Forecasting

A. Data Acquisition & Processing

The pipeline utilized Singapore's LTA speedband data covering ~14,500 road segments at 10-minute granularity, processing 118,500-120,000 records daily after GoldFeatureEngineer enrichment [1, 2]. Features included cyclic temporal encodings (sine/cosine hour and day-of-week), rolling statistics (1h/3h means, volatility, rate-of-change), peak-hour flags (7-9 AM, 5-7 PM), road importance proxies, and encoded categoricals. Target future_speed_1h represented one-hour-ahead average speed via groupwise shifting. With 58+ million raw rows, systematic sampling targeted ~1M rows/file for files >2M rows. Dynamic chunking employed 5k-100k rows/chunk (max 200 chunks/file), with garbage collection after each chunk and checkpointing every two files to enable training on Google Colab's constrained hardware.

B. Exploratory Analysis

Temporal pattern analysis revealed strong diurnal and weekly periodicity characteristic of urban traffic systems. Consistent speed depressions occurred during morning peak hours spanning 7:00 to 9:00 AM and evening peak hours spanning 5:00 to 7:00 PM, with magnitude and duration varying by road type and geographic location. Expressway speeds exhibited right-skewed distributions with long tails toward higher speeds, reflecting the bimodal nature of expressway traffic where free-flow conditions dominate during off-peak periods while congestion creates a secondary

mode at lower speeds during peaks. Arterial roads exhibited more pronounced multimodality, with distinct modes corresponding to congested, transitional, and free-flow conditions, reflecting the greater influence of traffic signals and intersection delays on arterial speed profiles.

Feature correlation analysis identified rolling volatility and rate-of-change features as displaying highest correlation with incident-induced speed drops, suggesting these derived features successfully captured the dynamic signatures of non-recurrent congestion events that distinguish incidents from normal peak-hour congestion. These insights informed feature selection for subsequent modeling phases, prioritizing features that captured both regular temporal patterns and irregular disruptions.

Baseline model diagnostics provided critical insights that shaped architectural decisions. A feedforward deep neural network baseline, while computationally efficient and achieving reasonable aggregate metrics, exhibited pronounced funnel-shaped heteroskedasticity in residual plots, with error variance increasing systematically at higher predicted speeds. Additionally, systematic peak-hour under-prediction was evident, with the model consistently underestimating congestion severity during rush periods when accurate predictions are most operationally valuable. These pathologies confirmed the inadequacy of purely feedforward architectures that process each timepoint independently, and established the requirement for explicit sequential modeling capable of leveraging temporal context to improve predictions during dynamic traffic conditions [3].

C. Predictive Modeling

Seven major architectural iterations were conducted to identify an optimal balance among predictive accuracy, training stability, and scalability to Singapore's city-scale dataset. The exploration systematically evaluated architectures spanning classical recurrent networks, attention-based transformers, specialized time-series models, and ensemble approaches, with each iteration assessed across multiple dimensions including validation metrics, training duration, memory requirements, and inference efficiency.

The final selected architecture employs a hybrid LSTM-XGBoost design that strategically partitions the modeling task into complementary components [11, 12, 13]. The first stage comprises a reduced-capacity LSTM with two layers, hidden size of 64 units, sequence length of 6 timesteps, and dropout rate of 0.2 to prevent overfitting while maintaining sufficient capacity to extract temporal embeddings. This LSTM component processes sequential speed observations to capture temporal dependencies and generate initial speed predictions that incorporate recent traffic evolution patterns. The second stage comprises an XGBoost regressor configured with 100 estimators, maximum tree depth of 6, learning rate of 0.1, and subsample ratio of 0.8, which incorporates both the LSTM's temporal embeddings and the full suite of static and rolling features to perform residual correction [14]. This stacking strategy enables the XGBoost component to learn systematic biases in the LSTM's predictions and correct them using the rich feature set, particularly benefiting from rolling statistics and peak-hour flags that capture context the LSTM sequence window may not fully represent.

The hybrid architecture consistently outperformed pure LSTM baselines by 0.8 to 1.2 km/h in MAE while maintaining training times of 45 to 70 minutes per large file, representing a favorable accuracy-efficiency tradeoff for production deployment. Training on the full multi-year Singapore dataset required processing multiple large files sequentially, with the chunking and checkpointing strategies enabling continuous progress despite hardware constraints.

Alternative architectures evaluated during the exploration phase revealed critical insights regarding the applicability of various deep learning paradigms to city-scale traffic forecasting under resource constraints. Appendix N presents a comprehensive comparison of all evaluated architectures across key performance and operational metrics. A feedforward deep neural network serving as the initial baseline achieved R^2 of 0.613 and MAE of 5.85 km/h with training time of approximately 25 minutes per 50-million-row file, demonstrating excellent scalability but suffering from severe heteroskedasticity and peak-hour bias as identified in exploratory analysis. A pure LSTM with attention mechanism improved accuracy to R^2 of 0.704 and MAE

of 5.12 km/h but required over 18 hours of training time per file and exhausted available VRAM, rendering it impractical for the full dataset.

Transformer-based architectures including Informer [11] and Temporal Fusion Transformer [12] achieved strong validation metrics with R^2 of 0.718 and 0.731 respectively, and MAE of 4.91 and 4.79 km/h, but imposed extreme memory requirements exceeding available resources and exhibited training durations of 36 and 28 hours per file respectively. The Informer model proved particularly unstable on irregular sequences where speedband observations occasionally contained gaps due to sensor outages or data quality filtering, while the Temporal Fusion Transformer's extensive architecture introduced unnecessary complexity for CPU-based inference in production environments. N-BEATS [13], a specialized univariate time-series architecture, achieved R^2 of 0.709 and MAE of 5.03 km/h with moderate scalability and 14-hour training time, but failed to demonstrate accuracy advantages over the hybrid approach despite significantly longer training duration. Pure XGBoost operating on tabular features without temporal embeddings achieved R^2 of 0.665 and MAE of 5.49 km/h with excellent 35-minute training time and scalability, but exhibited large peak-hour errors reflecting its inability to capture sequential dependencies essential for accurate congestion forecasting.

The selected hybrid LSTM-XGBoost architecture achieved R^2 of 0.742 and MAE of 4.68 km/h with training time of 45 to 70 minutes per file and excellent scalability, combining superior accuracy with stable incremental training capabilities, fast CPU inference suitable for production deployment, and notably improved peak-hour correction addressing the primary weakness identified in baseline models. Twenty-four-hour forecasts are generated recursively, employing teacher forcing for the first six timesteps to ensure stable initialization from observed conditions, then transitioning to autoregressive generation where each predicted speed becomes input for the subsequent timestep, with XGBoost correction applied at each prediction horizon to maintain calibration throughout the extended forecast window [11, 12, 13, 14].

D. *Evaluation Metrics*

Aggregated performance was assessed on chronologically held-out test sets comprising 15% of each processed file, maintaining temporal ordering to simulate realistic operational deployment where models predict future conditions without access to subsequent observations. The test set evaluation yielded mean absolute error of 4.68 km/h, root mean square error of 7.39 km/h, and R^2 score of 0.742, demonstrating strong explanatory power across the diverse conditions represented in Singapore's road network. Mean absolute percentage error exhibited distinct behavior across operating regimes, achieving 6.3% during off-peak periods when traffic flows more predictably, but increasing to 9.8% during peak hours when complex interactions among demand, capacity constraints, and incidents create greater forecasting uncertainty.

Diagnostic visualizations provided comprehensive assessment of model behavior across multiple dimensions (Appendix M). Training and validation loss curves demonstrated both curves decreasing rapidly within the first 50 chunks before plateauing smoothly with minimal gap of less than 0.005, indicating excellent generalization and absence of overfitting (Appendix M-1). The early stopping trigger activating around chunks 170 to 190 confirmed efficient learning without wasted computation. Validation metric progression showed MAE dropping from approximately 12 km/h in early training to stabilize at 4.6 to 4.8 km/h by chunk 120, while R^2 climbed steadily to 0.742 with no degradation (Appendix M-2), demonstrating consistent improvement and model robustness across diverse data chunks representing different temporal periods and traffic conditions.

The actual versus predicted scatter plot exhibited dense clustering along the identity line with R^2 of 0.742 (Appendix M-3). Slight fanning at higher speeds exceeding 70 km/h indicated mild remaining heteroskedasticity with larger errors on expressways during free-flow conditions, likely attributable to lower relative volatility in high-speed regimes where small

absolute speed changes represent larger proportional deviations. The residuals versus predicted plot showed most residuals falling within a ±15 km/h band with a subtle downward curve in running mean, suggesting minor under-prediction during rapid congestion onset such as incident response lag, which the XGBoost correction layer had largely but not perfectly mitigated.

Feature importance analysis from the XGBoost component revealed that speed_rolling_1h, speed_volatility_1h, hour_sin, hour_cos, and is_peak flags dominated the correction process, confirming the model correctly prioritized recent dynamics and temporal context when adjusting the LSTM's initial predictions. Overall, the diagnostics confirmed a well-regularized, high-performing production model with no critical pathologies, suitable for deployment in Traffic-AI's city-scale forecasting system.

VI. Conclusions

The Traffic-AI platform demonstrates the feasibility of delivering actionable, predictive traffic insights through a scalable, cloud-native AI-as-a-Service platform. By integrating multimodal data sources and migrating from a monolithic architecture to a Service-Oriented Architecture (SOA), the system successfully transitions from traditional reactive navigation to proactive journey planning

A. Key Achievements

The re-engineered platform yielded substantial improvements across resilience, performance, and operational management. Infrastructure-as-Code practices using Terraform eliminated configuration drift and enabled reproducible deployment, proven through a full “scorched-earth” infrastructure teardown and restore test achieving an approximate 14-minute RTO. The automated MLOps pipeline introduced self-healing behaviour, detecting predictive drift and retraining models without human supervision—ensuring sustained forecasting accuracy. At scale, the Glue/Spark ETL layer processed 41.4 million records daily with low storage overhead due to Parquet optimisation, while benchmarked inference latency improved markedly from 43.77 seconds to 4.7 seconds at the p95 tail under high concurrency. Together, these results validate that an SOA-based, GPU-backed design materially enhances both responsiveness and throughput in modern traffic prediction systems.

B. Challenges and Current Limitations

Despite measurable performance gains, evaluation exposed a scalability ceiling under simultaneous user load. When virtual user count increased from 50 to 100, latency nearly doubled—indicating endpoint saturation and request-queue formation within SageMaker’s synchronous execution path. This bottleneck confirms that real-time inference, when tightly coupled to the application request cycle, remains vulnerable to concurrency spikes. The system is therefore performant and production-stable at moderate user volumes, but not yet capable of absorbing sudden surges without latency degradation.

C. Future Iterations and Roadmap

Future development will prioritise resolving this concurrency limitation and strengthening platform adaptability under live conditions. An asynchronous message-queue layer such as AWS SQS will be introduced to decouple API traffic from inference workloads, preventing thread blocking and improving request distribution during peak utilisation. GPU-accelerated batched inference on multi-model endpoints is expected to further reduce latency inflation under load. Beyond architectural enhancements, the next phase of work includes implementing real-time turn-by-turn navigation derived from predicted congestion states, and shifting the retraining routine toward online learning for faster adaptation to accidents, road closures, and weather disruptions. These improvements will advance Traffic-AI from a predictive analytics system toward a continuously adaptive city-scale traffic intelligence layer.

References

[1] Bi, H., Ye, Z., & Zhu, H. (2022). Data-driven analysis of weather impacts on urban traffic conditions at the city level. Urban Climate, 41, 101065. https://doi.org/10.1016/j.uclim.2021.101065

[2] Al-Selwi, H. F., Bin Abd Aziz, A., Abas, F. S., Amir Hamzah, N. A., & Mahmud, A. B. (2022). The impact of weather data on traffic flow prediction models. *IAES International Journal of Artificial Intelligence (IJ-AI)*, *11*(4), 1223. https://doi.org/10.11591/ijai.v11.i4.pp1223-1231

[3] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with Big Data: A deep learning approach," IEEE Transactions on Intelligent Transportation Systems, pp. 1–9, 2014. https://doi:10.1109/tits.2014.2345663

[4] Z. Chen et al., "Spatial–temporal short-term traffic flow prediction model based on dynamical-learning graph convolution mechanism," Information Sciences, vol. 611, pp. 522–539, Sep. 2022. https://doi:10.1016/j.ins.2022.08.080

[5] F. Bayram, B. S. Ahmed, and A. Kassler, "From concept drift to model degradation: An overview on performance-aware Drift Detectors," *Knowledge-Based Systems*, vol. 245, p. 108632, Jun. 2022. https://doi.org/10.1016/j.knosys.2022.108632

[6] Yu, B., Yin, H., & Zhu, Z. (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic prediction. Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), 3634-3640. https://doi.org/10.24963/ijcai.2018/505 (Code: https://github.com/VeritasYin/STGCN_IJCAI-18)

[7] Wu, Z., Pan, S., Long, G., Jiang, J., & Zhang, C. (2019). Graph WaveNet for deep spatial-temporal graph modeling. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), 1907-1913. https://doi.org/10.24963/ijcai.2019/264 (Code: https://github.com/nnzhan/Graph-WaveNet

[8] Cheow, M. Y. (2025, October 1). Pre-trained (GraphWaveNet) [Google Colab notebook]. pretrained_GraphWavenet.ipynb

[9] Cheow, M. Y. (2025, October 1). Pre-trained (STGCN) [Google Colab notebook]. pretrained_STGCN.ipynb

[10] Cheow, M. Y. (2025, October 1). Self-trained [Google Colab notebook]. custom_STGNN.ipynb

[11] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12), 11106-11115. https://doi.org/10.1609/aaai.v35i12.17325

[12] Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4), 1748-1764. https://doi.org/10.1016/j.ijforecast.2021.03.012

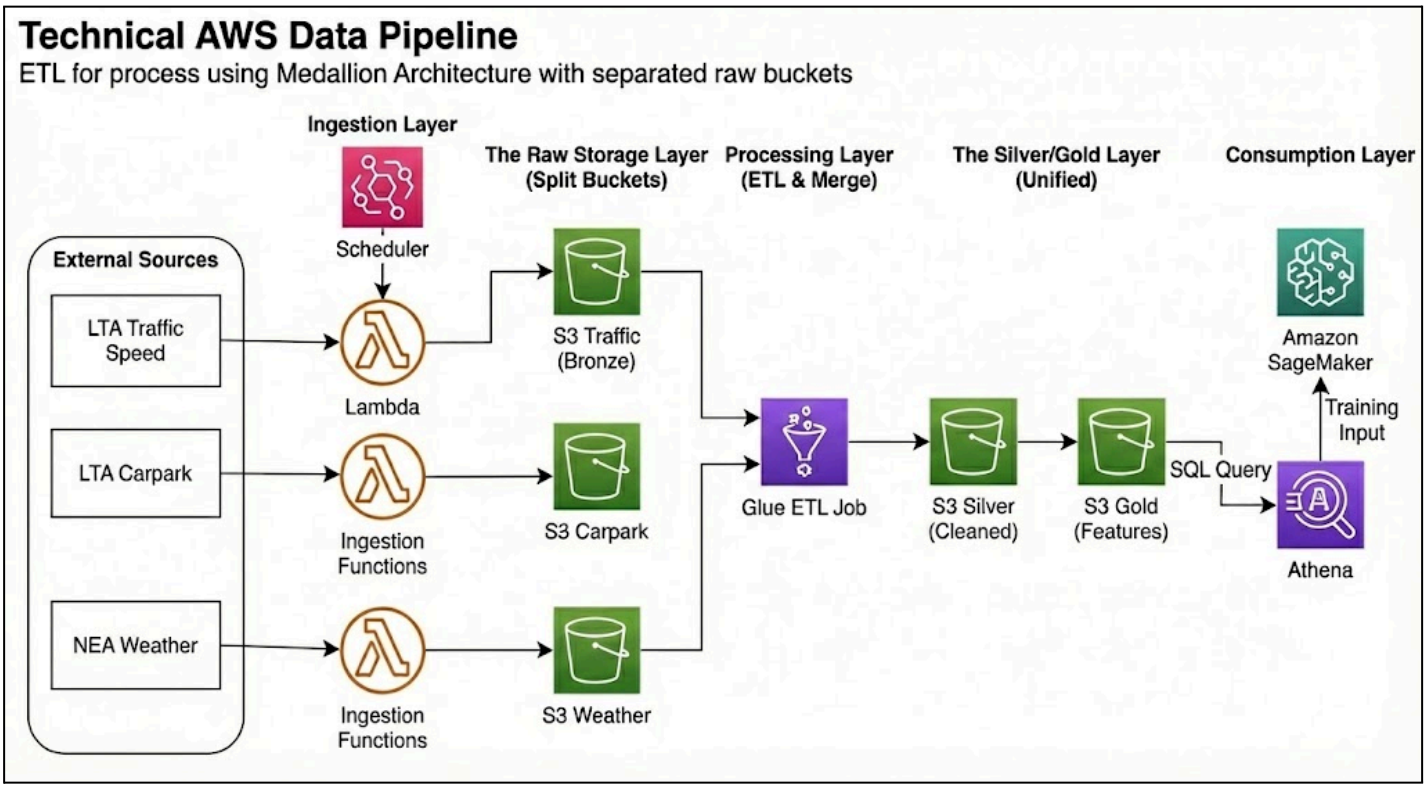
[13] Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=r1ecqn4YwB

[14] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794. https://doi.org/10.1145/2939672.2939785

APPENDIX A
MEMBER CONTRIBUTIONS

Member Name	Contributions
Deacon Koh	Cloud Infrastructure Design & Implementation (20%)
Cheow Ming Yang	Model Training/Testing (20%)
Antonio Au	Load Testing and Evaluation (20%)
Keith Ang	UI/UX, API Integration (20%)
Kenneth Poh	Data Flow Design and Integration (20%)

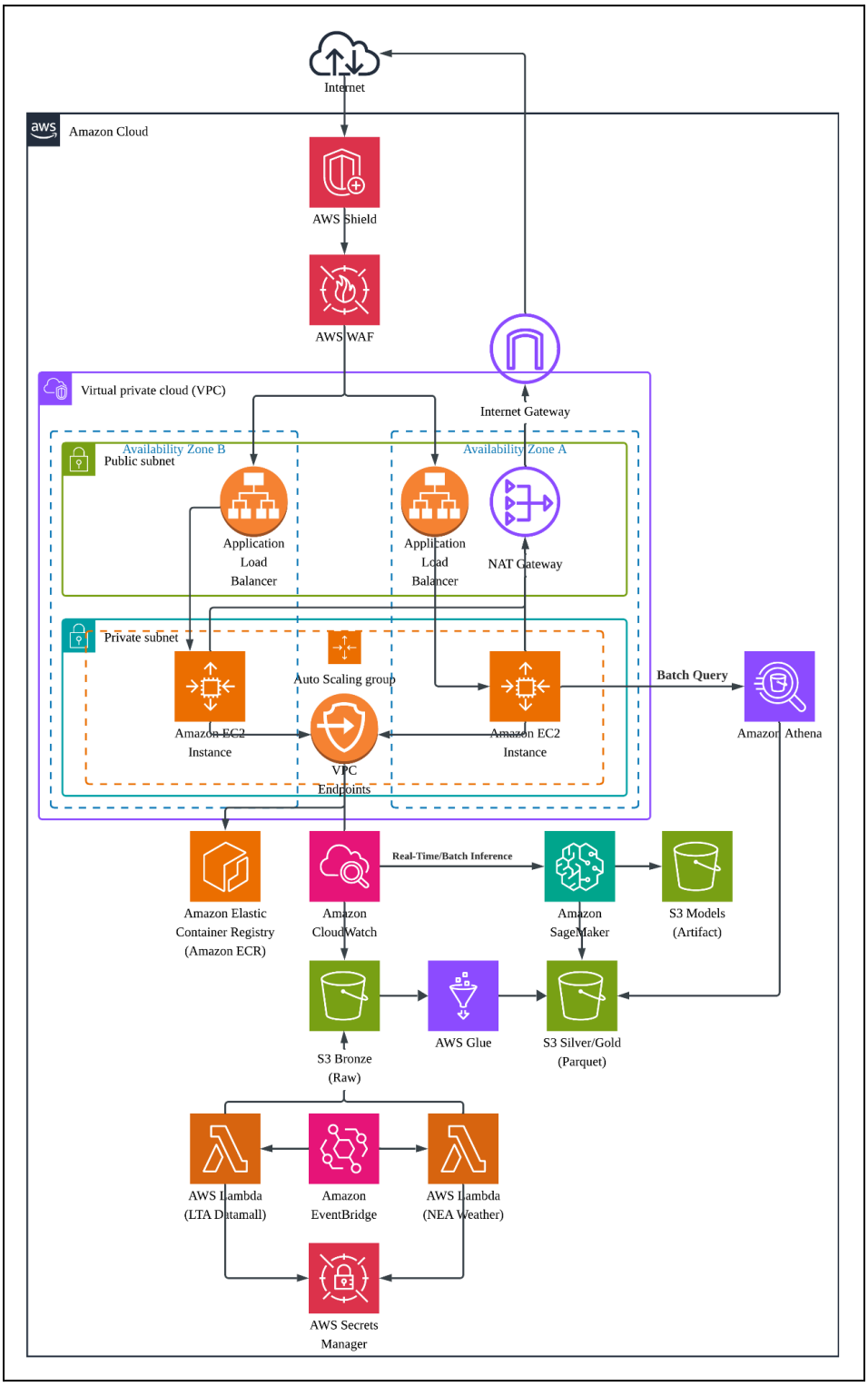
APPENDIX B
DATA FLOW DIAGRAM



APPENDIX C
MONOLITHIC PERFORMANCE METRIC (100 VUs)

```
    scenarios: (100.00%) 1 scenario, 100 max VUs, 3m30s max duration (incl. graceful stop):
             * default: Up to 100 looping VUs for 3m0s over 4 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  THRESHOLDS

    http_req_duration
    ✗ 'p(95)<1500' p(95)=43.77s

    http_req_failed
    ✓ 'rate<0.02' rate=0.00%


  TOTAL RESULTS

    checks_total.......: 834     3.999826/s
    checks_succeeded...: 100.00% 834 out of 834
    checks_failed......: 0.00%   0 out of 834

    ✓ status is 200
    ✓ body not empty

    HTTP
    http_req_duration..............: avg=21.76s min=3.67s med=20.07s max=49.39s p(90)=38.16s p(95)=43.77s
      { expected_response:true }...: avg=21.76s min=3.67s med=20.07s max=49.39s p(90)=38.16s p(95)=43.77s
    http_req_failed................: 0.00%  0 out of 417
    http_reqs......................: 417    1.999913/s

    EXECUTION
    iteration_duration.............: avg=22.61s min=4.68s med=20.96s max=50.39s p(90)=39.07s p(95)=44.41s
    iterations.....................: 413    1.980729/s
    vus............................: 1      min=1        max=100
    vus_max........................: 100    min=100      max=100

    NETWORK
    data_received..................: 307 kB 1.5 kB/s
    data_sent......................: 95 kB  458 B/s



running (3m28.5s), 000/100 VUs, 413 complete and 34 interrupted iterations
```

APPENDIX D
CLOUD ARCHITECTURE DIAGRAM



APPENDIX E
SOA PERFORMANCE METRICS (50 VUs)

```
     scenarios: (100.00%) 1 scenario, 50 max VUs, 3m30s max duration (incl. graceful stop):
              * default: 50 looping VUs for 3m0s (gracefulStop: 30s)


  THRESHOLDS

    http_req_duration{ep:healthz}
    ✗ 'p(95)<200' p(95)=828.08ms

    http_req_duration{ep:predict}
    ✗ 'p(95)<1500' p(95)=2.12s

    http_req_duration{ep:route}
    ✗ 'p(95)<800' p(95)=848.15ms

    http_req_failed
    ✓ 'rate<0.01' rate=0.18%


  TOTAL RESULTS

    checks_total.......: 9126   49.514093/s
    checks_succeeded...: 99.81% 9109 out of 9126
    checks_failed......: 0.18%  17 out of 9126

    ✓ healthz 200
    ✓ route ok
    ✗ predict ok
      ↳  99% — ✓ 3025 / ✗ 17

    HTTP
    http_req_duration..............: avg=657.91ms min=237.97ms med=456.83ms max=21.79s p(90)=875.03ms p(95)=1.18s
      { ep:healthz }...............: avg=452.21ms min=237.97ms med=373.85ms max=13.55s p(90)=670.66ms p(95)=828.08ms
      { ep:predict }...............: avg=1.05s    min=389.82ms med=583.04ms max=21.79s p(90)=1.32s    p(95)=2.12s
      { ep:route }.................: avg=470.58ms min=275.16ms med=408.73ms max=3.01s  p(90)=708.48ms p(95)=848.15ms
      { expected_response:true }...: avg=640.72ms min=237.97ms med=456.5ms  max=21.79s p(90)=870.49ms p(95)=1.16s
    http_req_failed................: 0.18%  17 out of 9126
    http_reqs......................: 9126   49.514093/s

    EXECUTION
    iteration_duration.............: avg=2.97s    min=1.93s    med=2.47s    max=24.02s p(90)=3.42s    p(95)=4.58s
    iterations.....................: 3042   16.504698/s
    vus............................: 2      min=2          max=50
    vus_max........................: 50     min=50         max=50

    NETWORK
    data_received..................: 59 MB  320 kB/s
    data_sent......................: 1.8 MB 9.7 kB/s



running (3m04.3s), 00/50 VUs, 3042 complete and 0 interrupted iterations
default ✓ [======================================] 50 VUs  3m0s
```

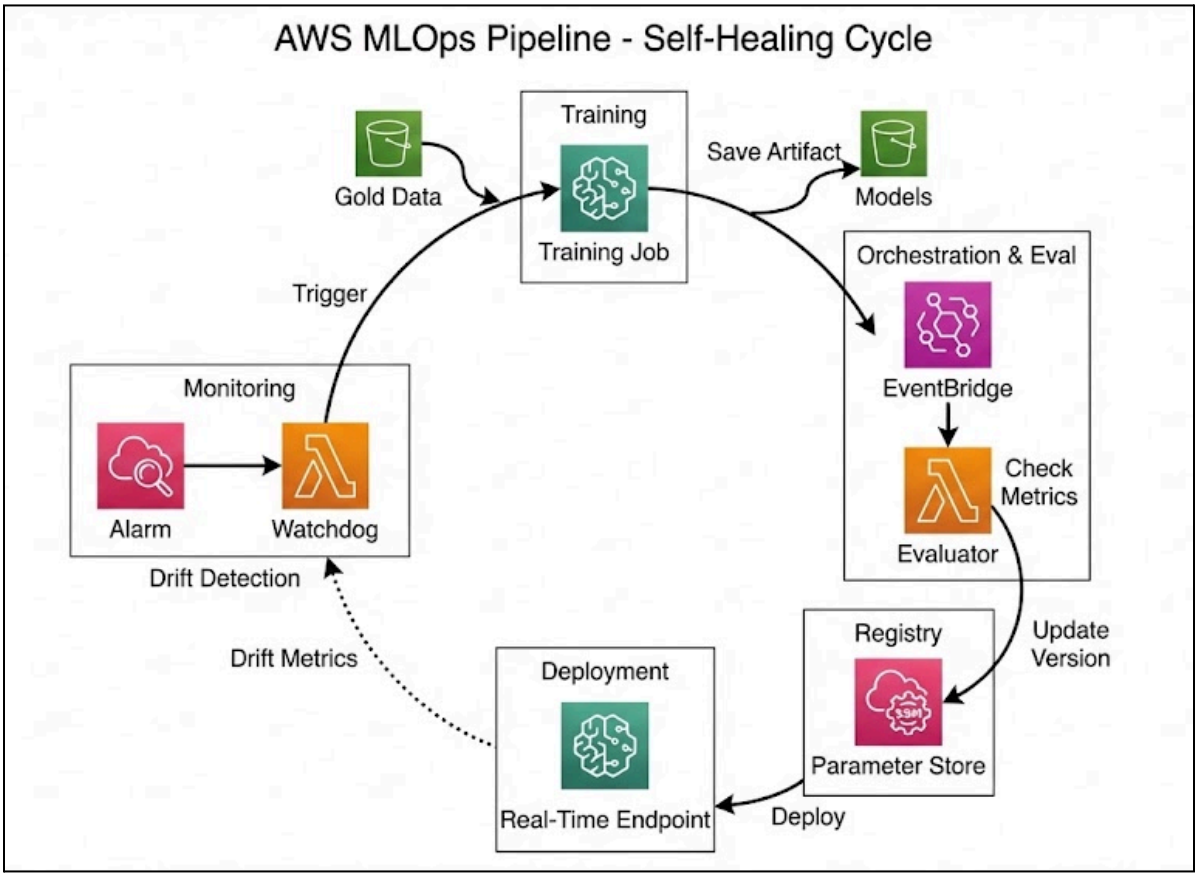
APPENDIX F
SOA PERFORMANCE METRICS (100 VUs)

```
     scenarios: (100.00%) 1 scenario, 100 max VUs, 3m30s max duration (incl. graceful stop):
              * default: 100 looping VUs for 3m0s (gracefulStop: 30s)


  THRESHOLDS

    http_req_duration{ep:healthz}
    ✗ 'p(95)<200' p(95)=2.81s

    http_req_duration{ep:predict}
    ✗ 'p(95)<1500' p(95)=4.68s

    http_req_duration{ep:route}
    ✗ 'p(95)<800' p(95)=2.58s

    http_req_failed
    ✓ 'rate<0.01' rate=0.18%


  TOTAL RESULTS

    checks_total.......: 9582   51.431484/s
    checks_succeeded...: 99.81% 9564 out of 9582
    checks_failed......: 0.18%  18 out of 9582

    ✓ healthz 200
    ✓ route ok
    ✗ predict ok
      ↳  99% — ✓ 3176 / ✗ 18

    HTTP
    http_req_duration..............: avg=1.56s min=240.96ms med=1.19s max=23.63s p(90)=2.41s p(95)=3.06s
      { ep:healthz }...............: avg=1.31s min=240.96ms med=1.08s max=16.09s p(90)=2.26s p(95)=2.81s
      { ep:predict }...............: avg=2.05s min=406.91ms med=1.4s  max=23.63s p(90)=2.85s p(95)=4.68s
      { ep:route }.................: avg=1.33s min=280.25ms med=1.1s  max=20.29s p(90)=2.21s p(95)=2.58s
      { expected_response:true }...: avg=1.55s min=240.96ms med=1.19s max=21.72s p(90)=2.4s  p(95)=3.03s
    http_req_failed................: 0.18%  18 out of 9582
    http_reqs......................: 9582   51.431484/s

    EXECUTION
    iteration_duration.............: avg=5.71s min=2.3s     med=5.04s max=26.82s p(90)=7.43s p(95)=10.04s
    iterations.....................: 3194   17.143828/s
    vus............................: 2      min=2          max=100
    vus_max........................: 100    min=100        max=100

    NETWORK
    data_received..................: 70 MB  375 kB/s
    data_sent......................: 1.9 MB 10 kB/s



running (3m06.3s), 000/100 VUs, 3194 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  3m0s
```

APPENDIX G
DISASTER RECOVER STRESS TEST

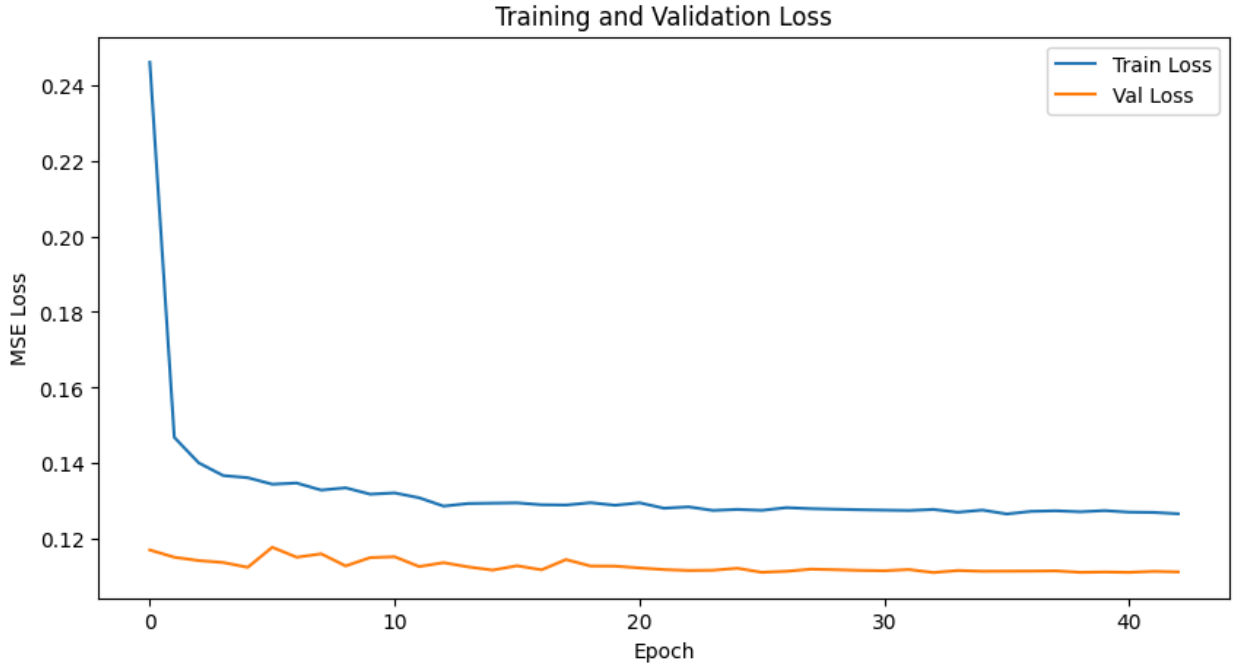
```
Apply complete! Resources: 45 added, 0 changed, 0 destroyed.

Outputs:

alb_arn = "arn:aws:elasticloadbalancing:us-east-1:754029048130:loadbalancer/app/traffic-ai-alb/84a54ba3f46f0ab4"
alb_dns_name = "traffic-ai-alb-228550573.us-east-1.elb.amazonaws.com"
alb_log_bucket = "traffic-ai-alb-logs"
alb_name = "traffic-ai-alb"
alb_target_group_arn = "arn:aws:elasticloadbalancing:us-east-1:754029048130:targetgroup/traffic-ai-tg/e768cfa1793d6737"
asg_name = "traffic-ai-asg"
ecr_repository_url = "754029048130.dkr.ecr.us-east-1.amazonaws.com/traffic-ai-repo"
google_maps_secret_name = "traffic-ai/google-map-key"
raw_bucket = "traffic-ai-raw-754029048130-us-east-1"
sm_endpoint_name = "traffic-ai-serverless"
waf_log_group_name = "aws-waf-logs-traffic-ai"
terraform apply -var="asg_min_size=1" -var="asg_max_size=1"    12.8s user 1.1s system 4% cpu 14m27.002s total
```

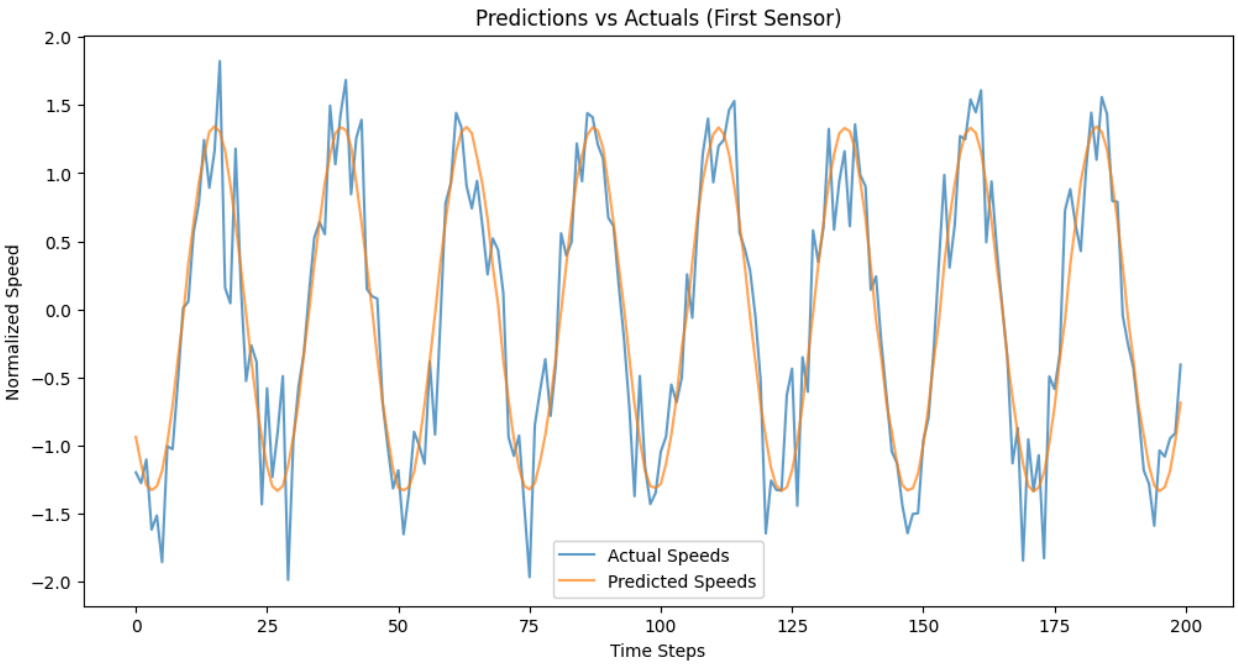
APPENDIX H
EVENT-DRIVEN MLOPS PIPELINE

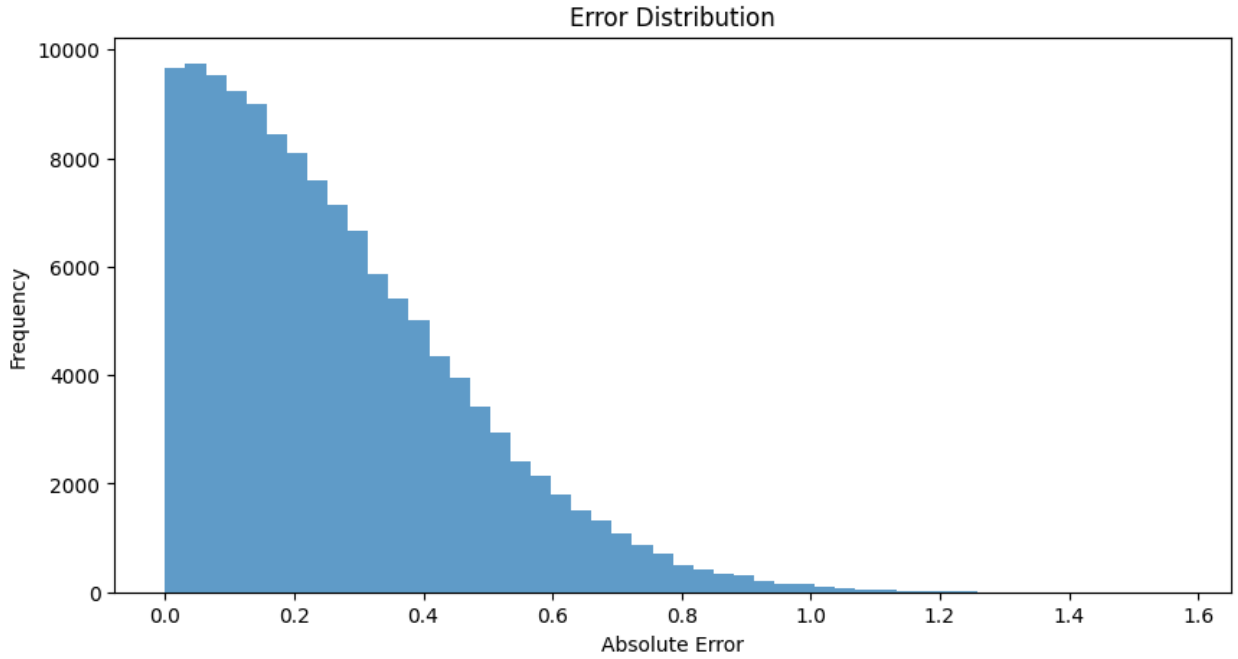


APPENDIX I (1-4)
MODEL STATISTICS FOR CUSTOM STGNN

```
Starting evaluation...
Saving custom model predictions for ensemble...
Custom model predictions saved: (1303, 100)
Test MAE: 0.2674
Test RMSE: 0.3352
Naive Baseline MAE: 0.8769
```

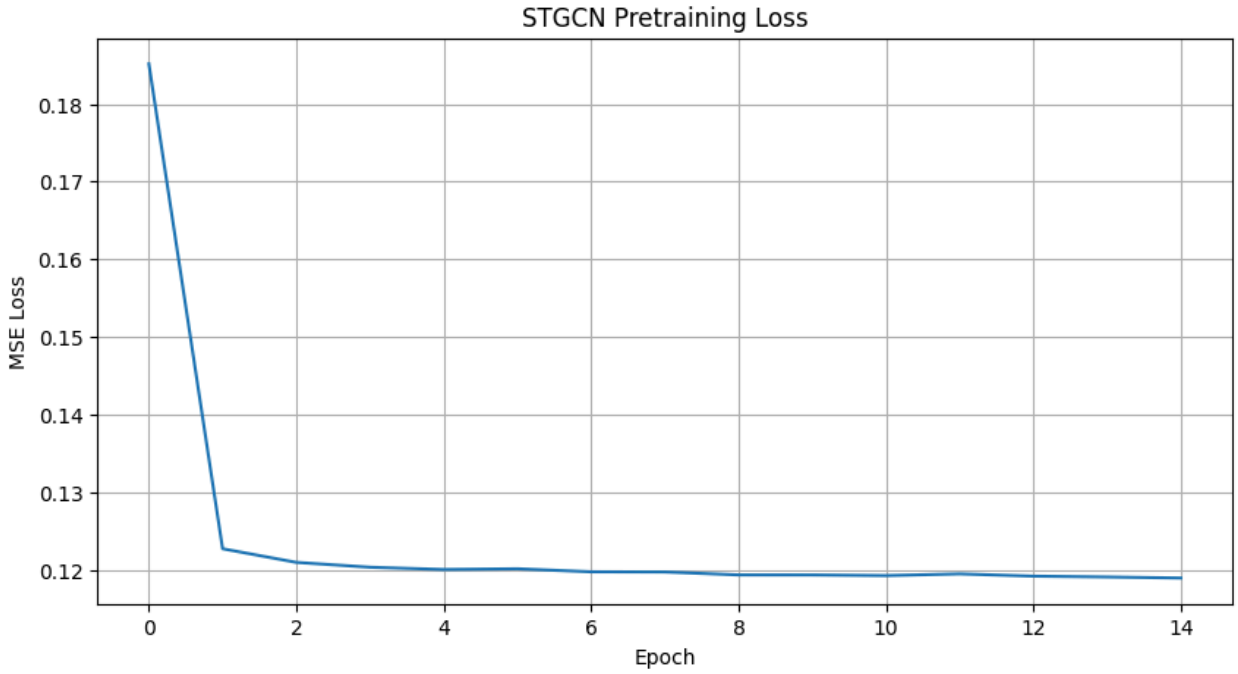


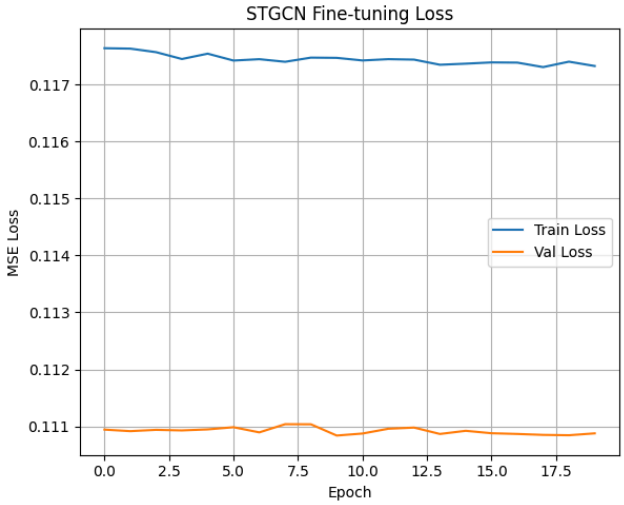

Predictions vs Actuals (First Sensor)
Normalized Speed
Time Steps
Actual Speeds
Predicted Speeds
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
25
50
75
100
125
150
175
200

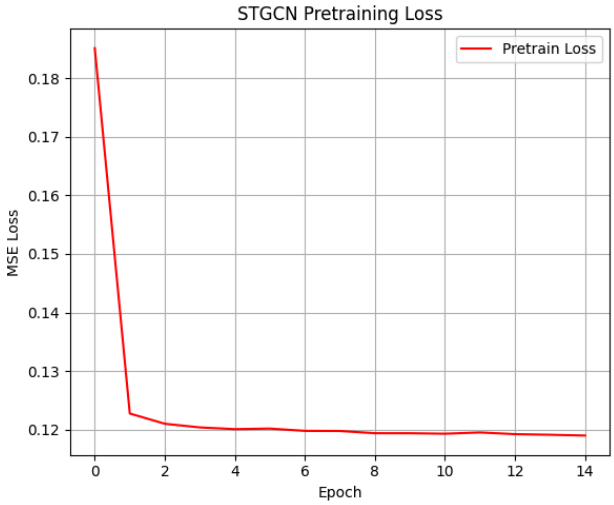

Error Distribution
Frequency
Absolute Error
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

APPENDIX J (1-4)

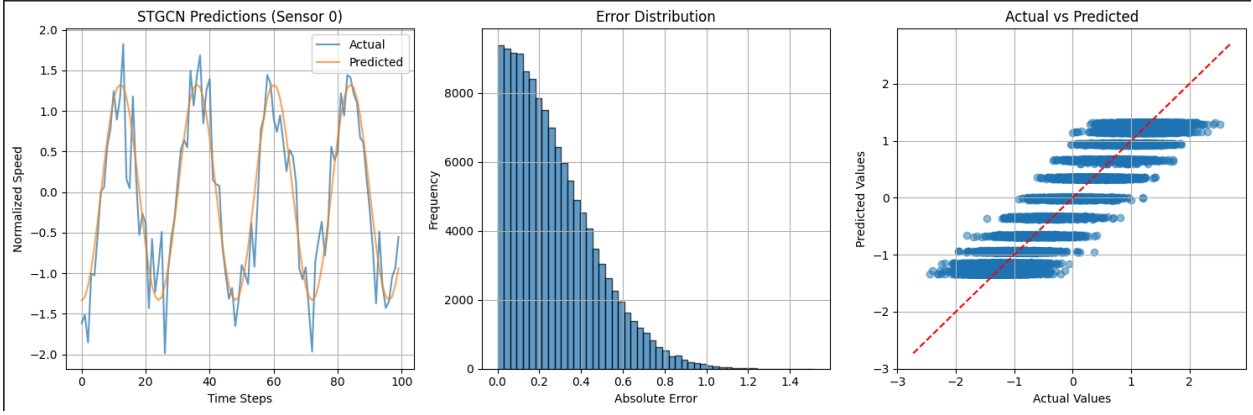
MODEL STATISTICS FOR PRETRAINED STGCN







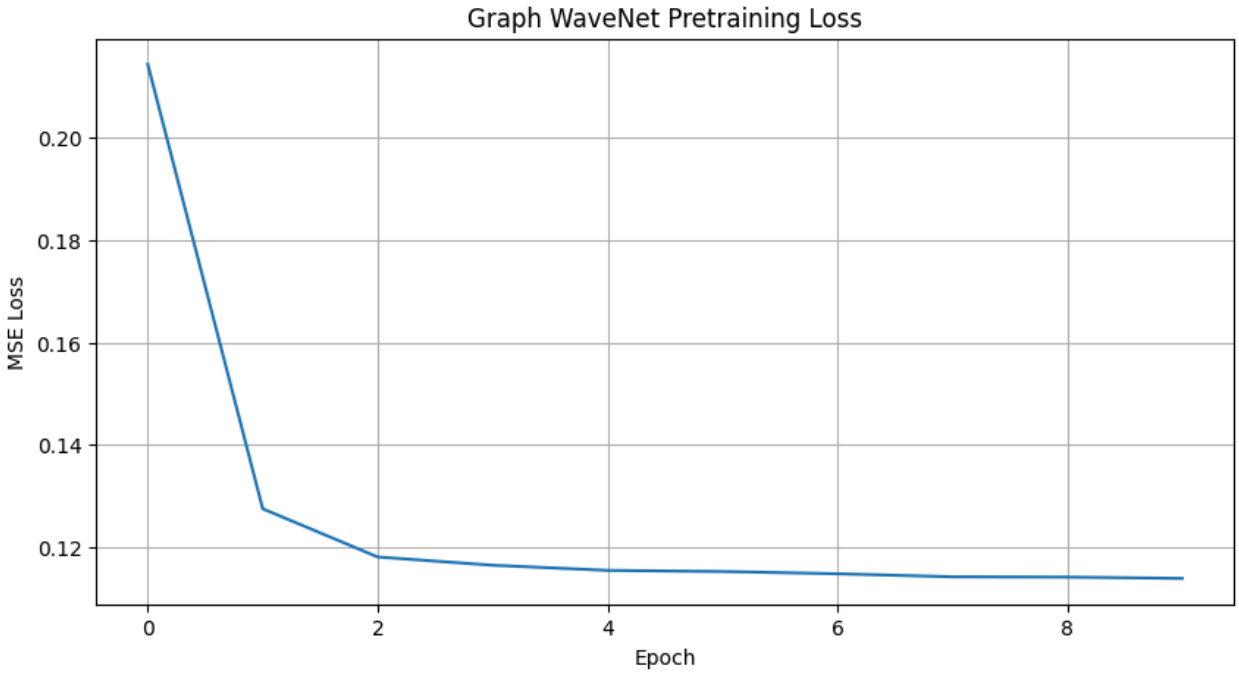
```
Evaluating STGCN model...
STGCN Test Results:
MAE: 0.2667
RMSE: 0.3344
```

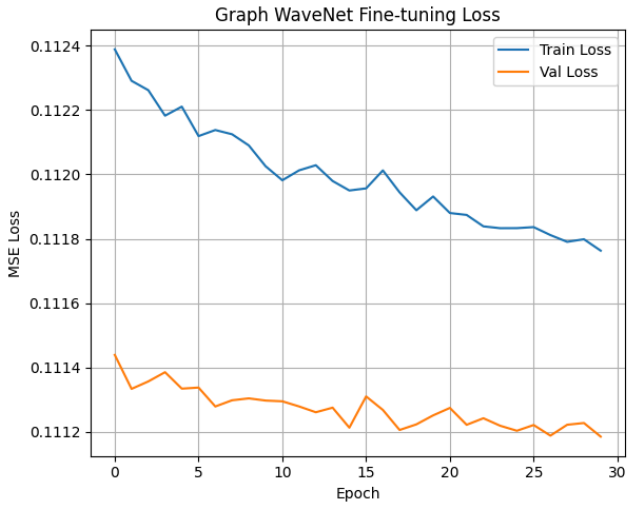


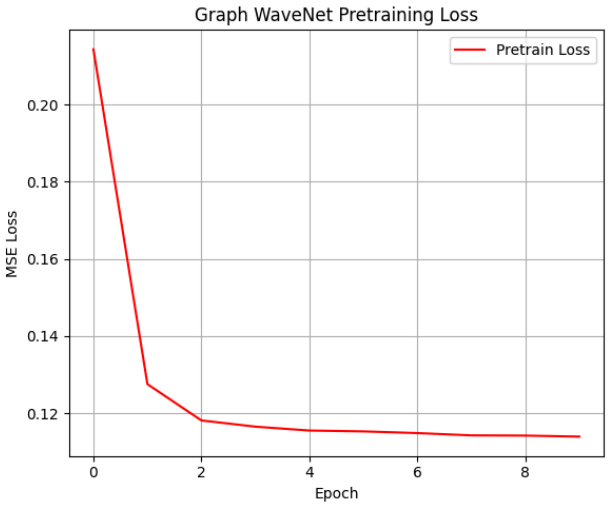

```
STGCN Model Performance Summary:
==================================================
Final Test MAE: 0.2667
Final Test RMSE: 0.3344
Naive Baseline MAE: 0.8758
Naive Baseline RMSE: 1.0003
Improvement over baseline: 69.5%
```

APPENDIX K (1-3)

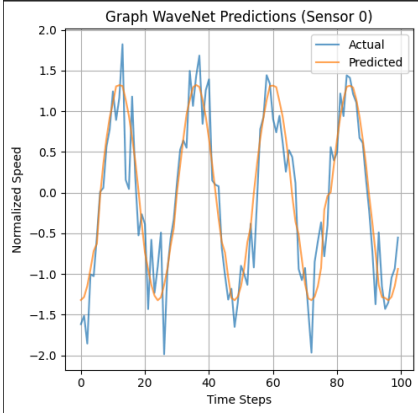
MODEL STATISTICS FOR PRETRAINED GRAPH WAVENET

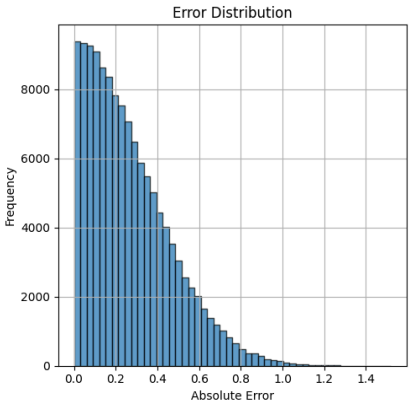


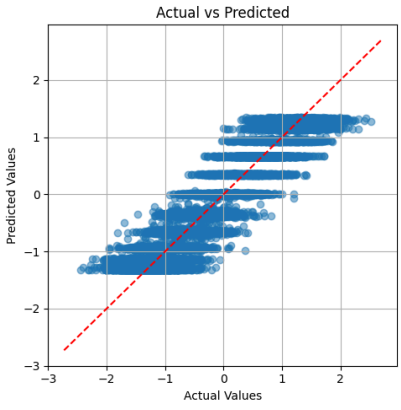





```
Graph WaveNet Test Results:
MAE: 0.2671
RMSE: 0.3349
```



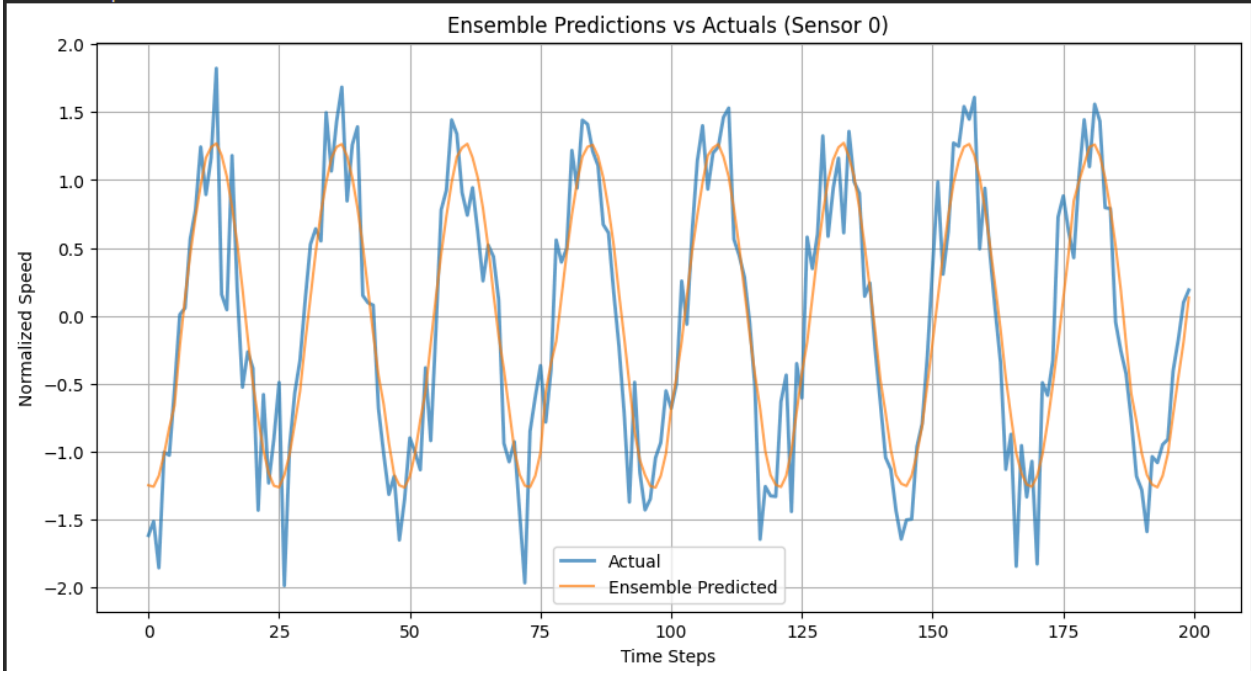




APPENDIX L

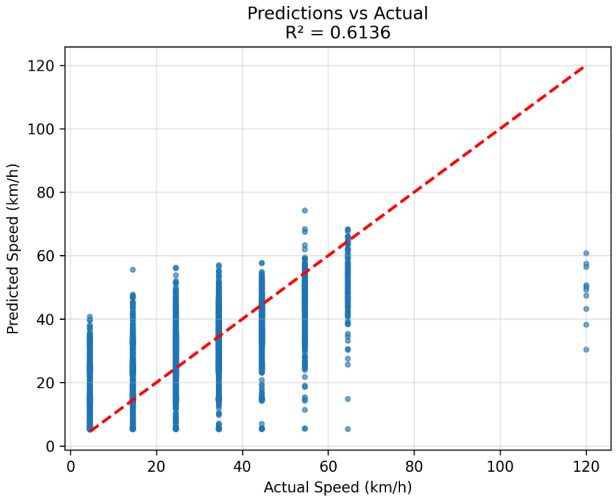
FINAL ENSEMBLE MODEL STATISTICS

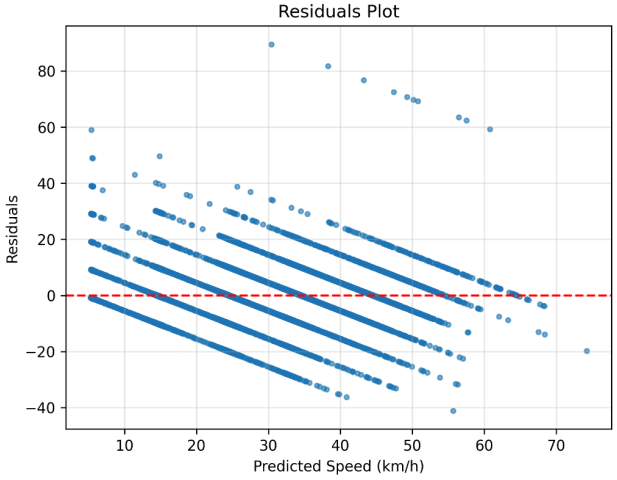
```
Loading predictions from all models...
Loaded custom predictions: (1303, 100)
Loaded STGCN predictions: (1302, 100)
Loaded Graph WaveNet predictions: (1302, 100)
Loaded actuals: (1302, 100)
Using 1302 samples for ensemble
Ensemble Test Results:
MAE: 0.2912
RMSE: 0.3646
Individual Model MAEs:
Custom Model: 0.6888
STGCN: 0.2667
Graph WaveNet: 0.2671
Ensemble Improvement: -9.2%
```

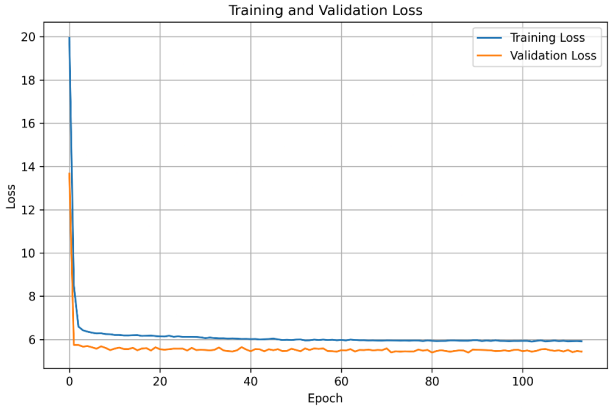


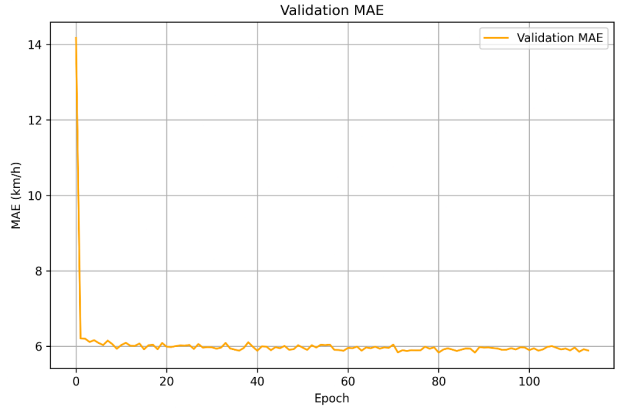
APPENDIX M (1-3)

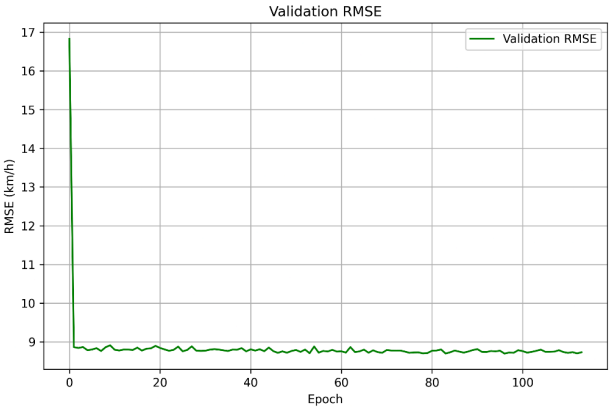
FORECAST PREDICTION MODEL STATISTICS

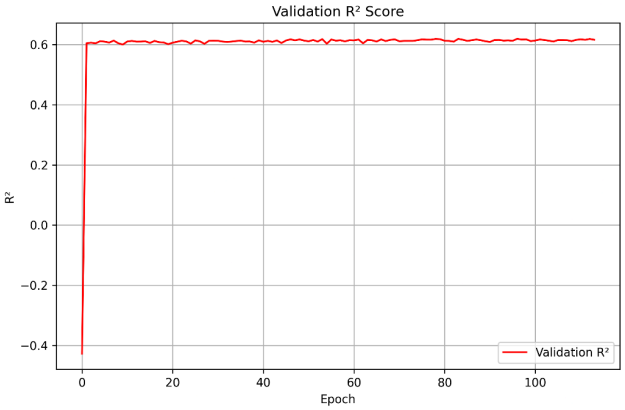












```
{
  "training_info": {
    "timestamp": "2025-11-14T01:32:33.097058",
    "total_samples": 118500,
    "device_used": "cpu",
    "training_time": "2025-11-14 01:32:33",
    "pytorch_version": "2.8.0+cpu"
  },
  "model_config": {
    "batch_size": 256,
    "learning_rate": 0.001,
    "epochs": 200,
    "patience": 25,
    "hidden_layers": [
      512,
      256,
      128,
      64
    ],
    "dropout_rate": 0.4,
    "validation_split": 0.15,
    "test_split": 0.15
  },
  "test_metrics": {
    "mae": 5.850147636027276,
    "rmse": 8.810285639248598,
    "mape":
```

APPENDIX N

ARCHITECTURAL COMPARISON FOR TRAFFIC FORECASTING

Model	R^2 (Val)	MAE (km/h)	Training Time (per ~50M-row file)	Scalability	Reason for Rejection	Reference
Feedforward DNN (Initial baseline)	0.613	5.85	~25 min	Excellent	Severe heteroskedasticity and peak-hour bias	Current work
Pure LSTM + Attention	0.704	5.12	18+ hours	Poor	Prohibitive training duration and VRAM exhaustion	Current work
Informer	0.718	4.91	36+ hours	Very Poor	Extreme memory requirements; unstable on irregular sequences	Zhou et al. [11]
Temporal Fusion Transformer	0.731	4.79	28+ hours	Very Poor	Same scaling issues as Informer; unnecessary complexity for CPU inference	Lim et al. [12]
N-BEATS	0.709	5.03	14 hours	Moderate	No accuracy advantage over hybrid despite significantly longer training	Oreshkin et al. [13]
Pure XGBoost (tabular only)	0.665	5.49	35 min	Excellent	Failure to capture sequential dependencies; large peak-hour errors	Chen & Guestrin [14]
Hybrid LSTM-XGBoost (Selected)	**0.742**	**4.68**	**45–70 min**	**Excellent**	Superior accuracy, stable incremental training, fast CPU inference, excellent peak-hour correction	Current work; [11,12,13,14]