



UNIVERSITÀ DEGLI STUDI DI GENOVA

Department of Naval, Electrics, Electronics and Telecommunication
Engineering

Master's Degree in Internet and Multimedia Engineering

A Semantic-Driven Control Framework for Open RAN

Candidate: Mohamed Amine Belabbes

Supervisor: Roberto Bruschi

Co-Supervisor: Somreeta Pramanik

Co-Supervisor: Chiara Lombardo

14/10/2026

Abstract

Fifth generation (5G) mobile networks expose a programmable radio access network (RAN) through the Open RAN (O-RAN) architecture, enabling software-defined xApps to exercise closed-loop control over base station behaviour. Existing xApp designs operate directly on raw Key Performance Indicator (KPI) streams — a high-dimensional, high-frequency data source that each application must interpret independently, without any shared understanding of the overall network condition. This thesis proposes and evaluates a Semantic State encoder for the 5G NR RAN: a dual-output neural architecture that maps a short window of raw KPI measurements to a compact set of binary semantic labels describing both the current network state and its predicted state one step ahead (20 ms ahead).

Six binary semantic labels are defined from threshold rules applied to 13 KPI streams — covering signal quality, link reliability, channel efficiency, interference, load-induced latency, and edge processing delay. The framework is trained and evaluated on real 5G NR KPI measurements from an indoor single-cell deployment. Three configurations are compared in a controlled progression: A1 (raw KPIs only, $F=13$), B1 (KPIs plus past semantic labels, $F=19$, single shared encoder), and B2 (same input, dual independent encoders — the proposed architecture). The dual-encoder design specialises one encoder for current-state classification and one for future-state prediction, fusing their outputs into a 32-dimensional semantic representation from which two classification heads produce binary predictions in a single 2.710 ms forward pass.

Experimental results demonstrate that B2 achieves a current-state F1 Macro of 0.9279 ± 0.0134 and a future-state F1 Macro of 0.7232 ± 0.0067 (tuned: 0.7446 ± 0.0055) over five independent seeds, outperforming B1 on both tasks across all seeds. A factorial ablation study reveals a strong interaction effect: past semantic labels and dual independent encoders are complementary design decisions — neither alone produces the gain that their combination achieves ($+0.035$ future F1 Macro). A state persistence analysis (Pearson $r = 0.856$ between label autocorrelation and future F1) explains per-label predictability and identifies Edge Processing Delay as structurally unpredictable due to its 20 ms median episode duration matching the prediction horizon. A comparison against standard sequence models (GRU, LSTM, Vanilla Transformer) over five seeds confirms that all architectures converge to a similar future F1 ceiling (0.72-0.75), identifying the dataset as the limiting factor rather than the architecture.

The proposed framework enables 58.5% preemptive control decisions and reduces unnecessary control messages by 11.5%, while satisfying the Near-RT RIC 100 ms latency budget by a factor of 37 .

Keywords: 5G NR, O-RAN, semantic communication, xApp, radio access network, time series classification, Transformer, iTransformer, PatchTST, multi-label classification, Near-RT RIC, event-driven control.

Table of Contents

Abstract	2
1 Introduction	6
2 State of the Art	9
2.1 O-RAN Architecture and the Near-RT RIC	9
2.2 The xApp and rApp Ecosystem	11
2.3 5G NR KPI Characterisation.....	12
2.4 Semantic Communication.....	15
2.5 Machine Learning for RAN Control	16
2.6 Transformer Architectures for Time Series	18
2.6.1 iTransformer.....	19
2.6.2 PatchTST.....	20
2.7 Explainability in RAN Intelligence	21
3 Semantic-Aware RAN Control Framework	22
3.1 Problem Formulation.....	23
3.2 Semantic Label Design	24
3.3 Encoder Architecture	27
3.3.1 Encoder Building Blocks	28
3.3.2 B1 — Single Shared Encoder (Baseline).....	30
3.3.3 B2 — Dual Independent Encoders	31
3.4 Event-Driven Decision Logic	33
3.4.1 Semantic Head Groupings.....	33
3.4.2 Decision Logic.....	33
3.4.3 Evaluation on Validation Data	34
4 Training Methodology	35
4.1 Dataset	35
4.2 Loss Function	36
4.3 Class Imbalance Handling	37
4.4 Window Construction and Normalisation	37
4.5 Optimisation and Regularisation.....	38
4.6 Per-Label Threshold Tuning	38
4.7 Evaluation Metrics	38
5 Results and Analysis	40
5.1 Overall Current-State Classification Performance	41

5.2 Overall Future-State Classification Performance.....	42
5.3 Per-Label Current-State Analysis.....	43
5.4 Per-Label Future-State Analysis	45
5.5 Rare-Label Case Study: Edge Processing Delay	46
5.6 Computational Profile.....	48
5.7 Ablation Study	49
5.8 Event-Driven Decision Logic Evaluation.....	51
5.9 Comparison with Baseline Architectures	52
6 Conclusions	59
References	62

Introduction

The fifth generation of mobile networks (5G) represents a fundamental shift in how wireless infrastructure is designed and operated. Beyond the well-known improvements in throughput and latency, 5G introduces a programmable, software-defined radio access network (RAN) architecture that makes the network itself a platform for intelligence. The Open Radio Access Network (O-RAN) initiative, driven by an industry alliance of major operators and vendors, takes this a step further by disaggregating the RAN into standardised, interoperable components connected through open interfaces. This disaggregation exposes the internal state of the network to external software — and with it, the opportunity to deploy machine learning directly inside the control plane. The Near-Real-Time Radio Intelligent Controller (Near-RT RIC), operating on a timescale between 10 milliseconds and 1 second, is the component designed to host this intelligence in the form of modular software applications called xApps. The Near-RT RIC can collect Key Performance Measurements (KPMs) from E2 nodes, execute inference and decision logic, and issue control commands within near-real-time operating timescales. However, the state provided to most existing RAN-control applications remains predominantly low-level and KPI-centric. Measurements such as Signal-to-Interference-plus-Noise Ratio (SINR), Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ), Channel Quality Indicator (CQI), Block Error Rate (BLER), Physical Resource Block (PRB) utilisation, throughput, and delay are typically processed individually or concatenated into a numerical feature vector. Although these measurements are essential, they do not directly express whether the network is experiencing a stable operating condition, progressive service degradation, interference risk, congestion, or an imminent reliability violation.

The objective is to transform a high-dimensional sequence of raw KPIs into a set of operationally meaningful semantic labels or representations and the network's present condition across six semantic dimensions and its predicted condition one step ahead are inferred. For example, combinations of strong signal quality, low BLER, and stable throughput may indicate a reliable transmission condition, whereas decreasing SINR, increasing retransmissions, and rising delay may jointly indicate interference risk or progressive degradation. Because several conditions may occur simultaneously, the network state cannot generally be represented by a single mutually exclusive class. It is more appropriately formulated as a multi-label semantic-state representation, in which each label describes a distinct but potentially coexisting property of the observed RAN condition.

Current xApp designs predominantly operate directly on raw Key Performance Indicator (KPI) streams — a paradigm referred to as KPI-driven control. This approach has three fundamental limitations. First, raw KPIs are high-dimensional and arrive at high frequency: a 5G base station reports measurements covering signal quality, channel condition, resource utilisation, and latency at periodic interval (10ms-100ms) per user equipment, generating data volumes that no single control algorithm can efficiently process and interpret within the Near-RT RIC latency budget. Second, individual KPIs are not independently meaningful — a drop in SINR only signals a problem when combined with RSRQ and other readings that confirm whether the cause is interference, distance, or channel congestion. Acting on individual KPI thresholds without capturing these joint dependencies leads to false alarms and missed degradation events. Third, KPI-driven control is inherently reactive: it can only respond to conditions that have already manifested in the measurements, with no mechanism to anticipate state changes before they affect service quality. Together, these limitations mean that KPI-driven control — while straightforward to implement — is fundamentally misaligned with the proactive, context-aware intelligence that O-RAN was designed to enable.

Existing xApp designs address individual control objectives such as handover triggering or interference mitigation in isolation, each processing the raw KPI stream independently. No existing component provides a unified, compact representation of the overall network state that all downstream xApps could share — and none simultaneously predicts how that state will evolve in the next control interval.

Semantic communication has emerged as an important research direction for AI-native networks. Rather than preserving all transmitted bits with equal importance, semantic communication seeks to preserve information that is relevant to the intended meaning or downstream task. This thesis does not consider semantic communication as an application-payload encoding problem. Instead, it applies semantic abstraction to the RAN monitoring and control plane. The objective is to transform a high-dimensional sequence of raw KPIs into a set of operationally meaningful semantic labels. For example, combinations of strong signal quality, low BLER, and stable throughput may indicate a reliable transmission condition, whereas decreasing SINR, increasing retransmissions, and rising delay may jointly indicate interference risk or progressive degradation. Because several conditions may occur simultaneously, the network state cannot generally be represented by a single mutually exclusive class. It is more appropriately formulated as a multi-label semantic-state representation, in which each label describes a distinct but potentially coexisting property of the observed RAN condition.

This thesis proposes a semantic abstraction to the RAN monitoring and control plane that maps raw RAN KPI sequences to compact semantic representations, from which the network's present condition across six semantic dimensions and its predicted condition one step ahead are inferred. The intended use of these representations — an event-driven control loop in which the RIC acts only on semantic state transitions or predicted degradation — is outlined as part of the framework design and left to future work; this thesis evaluates the encoding layer that such a loop would depend on. The Semantic- state encoder operates as a preprocessing xApp inside the Near-RT RIC, consuming the raw KPI stream from the E2 interface and producing, in a single forward pass, two outputs: a classification of the current network state into six binary semantic labels, and a forecast of those same labels twenty milliseconds into the future. The six labels — Excellent Link, Reliable Transmission, Efficient Channel State, Interference Risk, Moderate Load High Latency, and Edge Processing Delay — are defined through threshold rules applied to combinations of KPIs, translating the raw measurement space into a human-interpretable semantic representation. The Semantic state encoder architecture combines two complementary attention mechanisms: an inverted Transformer (iTransformer) that attends across KPI features to capture joint signal conditions, and a patch-based Transformer (PatchTST) that attends across time to capture short-term temporal dynamics. Two separate encoder-head pairs are trained jointly with a weighted loss, allowing the model to specialise each head for its respective prediction task while sharing a common representational backbone.

The Semantic state encoder is trained and evaluated on real 5G NR KPI measurements collected from an indoor single-cell deployment. The proposed architecture meets the latency requirements of the Near-RT RIC and produces a compact semantic representation that significantly reduces the volume of information that downstream xApps need to process. A systematic experimental evaluation examines the effect of several design choices on both current and future state prediction performance and identifies the factors that bound the achievable accuracy at the 20ms prediction horizon. To demonstrate that the encoder output can close the RAN control loop, a simple event-driven decision logic is also implemented and evaluated, showing that the semantic state predictions can drive timely and preemptive network control actions. Full experimental details and results are presented in Chapter 5.

The work is subject to a number of scope limitations. The dataset comes from a single indoor deployment, and validation on outdoor, multi-cell, or mobility scenarios is left as future work. The decision logic uses generic placeholder actions; the mapping to specific RAN control commands — such

as handover triggers or Modulation and Coding Scheme (MCS) adjustments — is the primary direction for future work, along with end-to-end integration on a live O-RAN testbed.

The remainder of this thesis is organised as follows. Chapter 2 surveys the relevant literature across four areas: O-RAN architecture, machine learning in the xApp ecosystem, 5G NR KPI characterisation, and Transformer architectures for time series. Chapter 3 describes the semantic encoding framework — the semantic label definitions, the encoder architecture, and the event-driven decision logic. Chapter 4 describes the training methodology and dataset. Chapter 5 presents the experimental results, including the ablation study, the comparison against baseline models, and the analysis of future-state prediction limits. Chapter 6 concludes the thesis and outlines directions for future work.

2 State of the Art

Open RAN creates both an opportunity and a constraint for network intelligence. The opportunity is a standardised interface through which any software component can consume live KPI streams and influence RAN behaviour. The constraint is that any such component must act within tens of milliseconds, on raw measurements that were never designed to be human-readable. Bridging this gap — between raw KPI data and actionable semantic understanding — is the central problem of this thesis. This chapter reviews the existing work on which the proposed solution builds, organised across four areas: the Open RAN architecture that defines the deployment context, the role of machine learning within the xApp ecosystem, the 5G NR measurements that form the input to the encoder, and the Transformer architectures that underpin its design. For each area, we identify what is established in the literature and where the gaps remain that this thesis addresses.

2.1 O-RAN Architecture Overview

Mobile networks have historically been built around tightly integrated, vendor-proprietary stacks, where hardware, software, and interfaces were supplied by a single manufacturer. This coupling simplified deployment but restricted operators to a single vendor's roadmap and made it difficult to introduce third-party intelligence or experiment with new control algorithms. The Open Radio Access Network (O-RAN) initiative, driven by the O-RAN Alliance, addresses this by disaggregating the RAN into standardised, interoperable components connected through open interfaces [1].

The reference architecture, shown in Figure 2.1, separates the RAN into three functional units. The O-RAN Central Unit (O-CU) handles the upper protocol layers, including the Packet Data Convergence Protocol and Radio Resource Control, and can be deployed on general-purpose cloud infrastructure. The O-RAN Distributed Unit (O-DU) manages the lower layers of the protocol stack, specifically the Medium Access Control and Radio Link Control layers and is typically co-located with or close to the radio hardware. The O-RAN Radio Unit (O-RU) is the outermost node, responsible for radio frequency processing and the physical antenna. The fronthaul interface between O-RU and O-DU follows the O-RAN 7.2x split specification, placing tight latency requirements on the transport layer connecting them.

Control intelligence is hosted in two Radio Intelligent Controllers (RICs) that operate at different timescales, reflecting the different response times required by different classes of network decisions. The Non-Real-Time RIC (Non-RT RIC) sits in the management and orchestration layer and operates on a timescale longer than one second. It hosts rApps, which are applications responsible for long-horizon optimisation tasks such as policy generation, model training, and performance analysis. The Non-RT RIC communicates with the Near-RT RIC through the A1 interface, over which it sends policy guidelines and model updates [2].

The Near-Real-Time RIC (Near-RT RIC) is the component most relevant to this thesis. It operates on a timescale between ten milliseconds and one second, hosting xApps that perform closed-loop control decisions based on live network measurements. The Near-RT RIC receives KPI streams and event reports from the RAN nodes through the E2 interface, which connects it to the O-CU and O-DU. Each E2 message carries one or more KPI measurements per reporting interval — in the dataset used in this thesis, one measurement vector is delivered every 20 milliseconds. The Near-RT RIC processes these streams, runs inference on any deployed xApp models, and sends control instructions back to the RAN through the same E2 interface. The O1 interface connects both RICs to the Service Management and Orchestration layer for configuration and fault management purposes [1][2].

From a deployment perspective, the Semantic state classifier described in this thesis is designed to operate inside the Near-RT RIC as a semantic preprocessing xApp. Its role is to receive raw KPI vectors from the E2 interface, classify the current network state into a set of human-interpretable semantic labels, and forecast the state one step ahead. This semantic representation can then be consumed by downstream decision-making xApps, separating the sensing and inference functions from the control

logic. The decision block itself — which would translate semantic state changes into concrete control actions such as handover triggers or power adjustments — is identified as future work and is not implemented in this thesis.

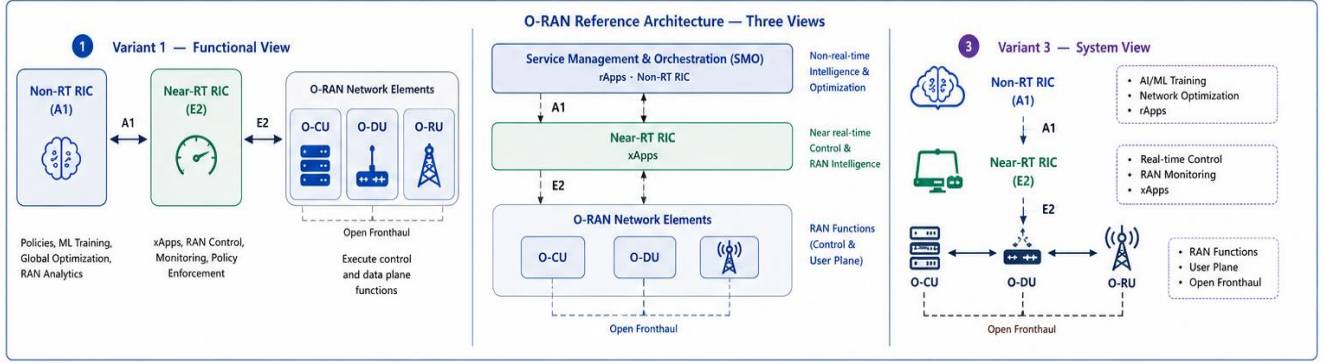


Figure 2.1: O-RAN reference architecture showing the disaggregated RAN components, the Near-RT RIC hosting xApps, and the key interfaces: E2 (RAN control), A1 (policy), and O1 (management).

2.2 The xApp and rApp Ecosystem

The O-RAN architecture was designed from the outset with machine learning in mind. The separation of control intelligence into rApps and xApps creates a natural layered structure for deploying AI: long-horizon models that require substantial computation run as rApps in the non-RT RIC, while latency-sensitive models that must respond within the near-real-time window run as xApps in the Near-RT RIC. This distinction matters because it constrains the complexity of any model deployed as an xApp inference must complete in well under 100 milliseconds to allow time for the E2 control message to reach the RAN node before the network state changes again [3].

Santos et al. [3] provide a detailed treatment of the xApp development lifecycle, from the structure of the E2 service models that define how KPIs are reported, to the subscription mechanisms an xApp uses to receive measurement streams, to the actuation primitives available for sending control instructions back to the RAN. Their work establishes that an xApp operates within a closed-loop cycle: it subscribes to E2 KPI reports, processes the incoming measurements, runs its inference model, and

issues a control command if the model output warrants one. The cycle repeats at the reporting interval of the subscribed KPI stream.

Polese et al. [4] demonstrate this cycle in practice on the ColO-RAN testbed, an open experimental platform built on the O-RAN Software Community stack. They show that xApps for traffic steering and interference management can be trained offline and deployed into the Near-RT RIC, where they interact with real radio nodes over live E2 connections. Their work makes clear that the inference component of an xApp need not be complex — what matters is that it produces a decision at the right timescale with sufficient reliability. It also highlights that most existing xApps are single task, handling one control objective such as handover or scheduling without a unified view of the overall network state.

This is the gap that motivates the present thesis. Rather than building yet another single-task xApp, we propose a Semantic State Classifier that sits upstream of any decision logic and produces a compact, multi-dimensional representation of the network state. This encoder becomes a shared service that any downstream xApp can consume, reducing the need for each xApp to independently process the raw KPI stream. The architecture follows naturally from the O-RAN design philosophy: modular, reusable components with well-defined interfaces.

2.3 5G NR Key Performance Indicators

At the physical and MAC layers of a 5G NR base station, the network continuously generates a stream of measurement quantities that collectively describe the condition of the radio link between the base station and each connected user equipment. These quantities are referred to as Key Performance Indicators, and they are the raw material from which any RAN intelligence must be built [5]. Understanding what each KPI measures physically, and why it varies, is a prerequisite for understanding the semantic labels that this thesis defines over them.

The **Downlink Signal-to-Interference-plus-Noise Ratio (DL SINR)**, expressed in dB, is the fundamental measure of received signal quality. It quantifies how much stronger the desired signal is compared to the combined effect of interference from neighbouring cells and thermal noise at the receiver. A higher SINR indicates a cleaner radio channel and directly enables the use of higher-order modulation schemes.

The **Reference Signal Received Power (RSRP)**, measured in dBm, captures the received power of the cell-specific reference signals. It is the primary quantity used for cell selection and handover decisions. Unlike SINR, RSRP does not account for interference — it measures only the strength of the signal from the serving cell. The **Reference Signal Received Quality (RSRQ)** complements RSRP by incorporating the effect of interference: it is computed as the ratio of RSRP to the total received wideband power, making it sensitive to the presence of interfering transmissions from neighbouring cells. The **Received Signal Strength Indicator (RSSI)** captures the total wideband power received, encompassing signal, interference, and noise.

The **Downlink Block Error Rate (DL BLER)** is the fraction of transmitted transport blocks that are received in error at the UE before any retransmission. It reflects the quality of the MAC-layer transmission and is directly related to SINR — when the channel quality degrades, the BLER rises as the current modulation and coding scheme becomes too aggressive for the prevailing conditions. The **Channel Quality Indicator (CQI)** is the UE's feedback to the base station about the current channel quality, expressed as an integer index between 0 and 15 that maps to a recommended modulation and coding scheme. The **Physical Modulation and Coding Scheme (MCS)** is the actual scheme selected by the scheduler, which may differ from the CQI recommendation depending on scheduler policy. The **Spectral Efficiency (SE)**, expressed in bits per second per Hertz, quantifies how efficiently the available bandwidth is being used and is determined jointly by the MCS and the signal quality.

Resource utilisation is captured by the **number of Physical Resource Blocks (PRBs)** allocated to the UE in the downlink. High PRB utilisation indicates that the scheduler is heavily loaded and has little spare capacity. The **Rank Indicator (RI)** and **Precoding Matrix Indicator (PMI)** are Multiple Input Multiple Output (MIMO) feedback quantities: the RI indicates how many independent spatial streams the channel can support, and the PMI selects the best precoding matrix to exploit the spatial structure of the channel. The **Uplink Buffer Occupancy** measures the amount of data queued at the base station for transmission in the uplink direction. Finally, the **end-to-end packet delay** records the latency in milliseconds between packet generation and reception; a value of -1 in the raw data indicates that no measurement was available for that interval, which is replaced by zero in preprocessing.

Table 2.1 — KPIs used in this work

KPI Name	Unit	Physical Meaning	Role in Semantic Labels
DL SINR	dB	Signal-to-interference-noise ratio at UE	Excellent_Link,Reliable_Trans thresholds
RSRP	dBm	Received power of cell reference signals	Excellent_Link threshold (joint with SINR)
RSRQ	dB	Signal quality relative to total interference	Interference_Risk threshold
DL BLER	%	Block error rate before retransmission	Reliable_Trans threshold
CQI	index	Channel quality feedback from UE (0–15)	Efficient_Ch threshold (joint with SE)
SE	bps/Hz	Spectral efficiency of current transmission	Efficient_Ch threshold (joint with CQI)
PRB count	count	Physical resource blocks allocated ($\div 25 \rightarrow \%$)	Mod_Load_Latency and Edge_Delay thresholds
Delay	ms	End-to-end packet latency	Mod_Load_Latency and Edge_Delay thresholds
MCS	index	Modulation and coding scheme selected	Auxiliary — correlated with CQI/SE
RI	rank	MIMO rank indicator	Auxiliary — spatial channel quality
PMI	index	Precoding matrix indicator	Auxiliary — MIMO precoding selection
UL Buffer	bytes	Uplink data queued at base station	Auxiliary — uplink load indicator
RSSI	dBm	Total received wideband power	Auxiliary — interference context

2.4 Semantic Communication and Network State Representation

The concept of semantic communication emerged from a recognition that the classical information-theoretic view of communication — which treats all bits as equally important regardless of their meaning — is ill-suited for systems that need to act on data rather than simply store or transmit it. Shannon's framework [6] optimises for the faithful reproduction of a source signal at a receiver, but in many practical settings what matters is not that every bit is reproduced exactly, but that the meaning conveyed by those bits reaches the receiver in a form that allows it to take the right action. Semantic communication frameworks therefore shift the optimisation objective from bit-level fidelity to task-level utility.

Applied to wireless networks, this perspective has a natural interpretation. A base station generates raw KPI measurements because they are the quantities that physical-layer and MAC-layer processes expose directly. They are not, however, the quantities that a network operator or a control algorithm needs to reason about. A scheduler does not need to know that SINR is 18.4 dB — it needs to know whether the link is in a state that warrants a change in MCS, or whether interference has risen to a level that requires a control response. The gap between the raw measurement space and the decision-relevant state space is precisely where semantic compression adds value.

Recent work by Gündüz et al. [7] has explored semantic representations for wireless systems, primarily in the context of reducing the communication overhead between sensors and controllers in IoT or vehicular networks. The general approach involves training an encoder to produce a compact latent representation of the input that preserves only the information relevant to a downstream task, discarding everything else. In the RAN context, this idea translates naturally to the problem of compressing the E2 KPI stream: instead of forwarding all thirteen KPI values at every reporting interval, the Near-RT RIC could forward a small semantic vector that describes the qualitative state of the link.

What distinguishes the approach taken here from a simple threshold-based classifier is the introduction of a learned intermediate representation. Rather than applying the threshold rules directly to the raw KPIs at inference time — which would be trivial — the Semantic State Classifier is trained

to learn a 32-dimensional semantic state vector that compresses the temporal KPI window into a representation from which both current-state labels and future-state labels can be decoded. This learned representation captures the joint structure of the KPI measurements in a way that pure threshold rules cannot, and it provides a continuous embedding that can be used for downstream tasks beyond simple classification.

2.5 Machine Learning Approaches for RAN State Classification

A growing body of work has applied machine learning to the problem of classifying or predicting network states from RAN measurement data. The approaches range from classical supervised methods applied to individual KPI features, to recurrent neural networks that model temporal dependencies in KPI sequences, to Transformer-based architectures that attempt to capture more complex structural relationships. We review representative works from each category, identifying their strengths and the gaps that this thesis addresses.

Classical supervised learning methods — decision trees, support vector machines, and multilayer perceptrons — have been applied to anomaly detection and state classification in 5G networks. Alves et al. [8] evaluate MLP, decision tree, and SVM classifiers for detecting network anomalies in O-RAN deployments, using KPI datasets with up to 25 input features. They report accuracy above 90% across the tested configurations and demonstrate that such models can be developed and executed within the O-RAN software framework as xApps. The limitation of these approaches is that they treat each measurement vector independently, with no representation of temporal dynamics. A single KPI snapshot may be insufficient to distinguish a transient fluctuation from a sustained degradation, and the absence of temporal context limits the utility of the classifier for proactive control.

Recurrent neural networks address this limitation by maintaining a hidden state across consecutive time steps. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures have been widely used for KPI forecasting in cellular networks, capturing the temporal dependencies that single-snapshot methods miss. These models process KPI sequences step by step and can in principle model complex temporal patterns. In practice, however, their sequential processing means that they cannot directly compare different KPI features at the same time step — a joint condition such as high BLER coinciding with high PRB utilisation, which is what the Mod_Load_Latency label captures, must be discovered implicitly through the sequence of hidden state updates rather than through

explicit cross-feature attention. Furthermore, none of the standard LSTM/GRU applications to RAN data produce multi-label outputs covering multiple simultaneous semantic conditions.

The most directly related work in the literature is GenTrap [9], proposed by Kazi et al. at Dalhousie University. GenTrap addresses radio link failure prediction in 5G RANs using a Transformer-based temporal feature extractor combined with a graph neural network module that aggregates contextual weather observations from neighbouring stations. Evaluated on two real-world datasets comprising 2.6 million KPI data points, GenTrap achieves an F1-score of 0.93 on rural data and 0.79 on urban data. It demonstrates that Transformer architectures can generalise across datasets with different KPI distributions. However, GenTrap predicts a single binary label — radio link failure — and relies on external weather data that may not be available in all deployment scenarios. It does not produce a multi-label semantic state representation, and it does not address the O-RAN xApp deployment context.

Table 2.2 summarises the key characteristics of these approaches in relation to the work presented in this thesis.

Table 2.2 — Comparison of ML approaches for RAN state classification

Work	Method	Task	Strength	Limitation	Multi-label?
Alves et al. [6] (2023)	MLP, SVM, Decision Tree	Anomaly detection in O-RAN KPIs	Fast inference. Over 90% accuracy.	No temporal modeling. Single snapshot only.	No
LSTM/GRU approaches (various)	Recurrent neural networks	KPI time series forecasting	Captures temporal dependencies.	No cross-feature attention. No multi-label output.	No
GenTrap Kazi et al. [7] (2024)	Transformer + GNN	Radio link failure prediction	F1=0.93 (rural). Generalises across datasets.	Single binary label only. Requires weather data	No

Work	Method	Task	Strength	Limitation	Multi-label?
				as extra input.	
Our work	iTransformer + PatchTST (dual-attention encoder)	6-label semantic state classif. + 20ms-ahead forecasting in O-RAN xApp	Multi-label. Current + future prediction. O-RAN ready (<3ms inference).	Single-cell indoor dataset. No decision block (future work).	Yes — 6 simultaneous labels

The gap that emerges from this review is clear: no existing work produces a multi-label semantic state representation from a window of raw KPI measurements within the Near-RT RIC timescale, while simultaneously predicting the next state one step ahead. This combination — current-state classification and near-future forecasting in a single forward pass — is what the architecture proposed in this thesis is designed to deliver.

2.6 Transformer Architectures for Time Series

The Transformer architecture, introduced by Vaswani et al [10]. for natural language processing, fundamentally changed how sequential data is modelled. The central innovation is the self-attention mechanism, which allows any position in a sequence to attend directly to any other position in a single operation, computing a weighted combination of all elements based on their pairwise relevance. Unlike recurrent networks, which must propagate information through a chain of hidden state updates, self-attention creates direct connections across the full sequence length, making it easier to capture long-range dependencies and enabling fully parallel computation during training.

The application of Transformers to time series data has, however, required careful thought about what the input tokens should represent. In language, tokens are words or subword units, and the sequence is naturally one-dimensional. In multivariate time series, there are two possible sequence directions: one

can attend across time (comparing timestep t_1 to timestep t_2) or across features (comparing KPI variable i to KPI variable j at the same timestep). These two directions capture fundamentally different kinds of structure, and the choice between them has significant consequences for what patterns the model can discover.

2.6.1 iTransformer — Inverted Attention Across Variables

Standard Transformer-based forecasters form their tokens from all variables at a single timestep — the token for time t is a vector that concatenates the values of all KPI features at t . Attention then operates across time, comparing the joint KPI state at different moments. Liu et al. [11] identify a fundamental problem with this design for multivariate time series: when features with different physical units and measurement scales are concatenated into a single token, the embedding mixes signals that have no natural relationship, and the resulting attention maps can become difficult to interpret or even meaningless.

The iTransformer inverts this convention. Each token is formed from the complete time history of a single variable — the token for KPI i is a vector that encodes the values of i across all timesteps in the input window. Self-attention then operates across variables rather than across time, comparing the history of KPI i to the history of KPI j . This allows the model to discover correlations between different measurement channels — for instance, the relationship between SINR and BLER, or between PRB utilisation and delay — through direct attention rather than through the implicit intermediary of a shared hidden state.

For the semantic labelling problem in this thesis, this property is directly relevant. The six labels are defined by threshold conditions that involve multiple KPIs jointly: `Excellent_Link` requires both $\text{SINR} \geq 15$ dB and $\text{RSRP} \geq -91$ dBm, while `Mod_Load_Latency` requires both $\text{PRB} \geq 60\%$ and $\text{delay} \geq 20$ ms. Discovering these joint conditions from data requires the model to compare the behaviour of different KPI channels, which is precisely what cross-variable attention enables. Liu et al. demonstrate that iTransformer achieves state-of-the-art performance on several multivariate forecasting benchmarks [11], and its architectural motivation aligns well with the structure of the classification problem here.

In the architecture used in this thesis, the iTransformer branch receives the input window of shape (T, F) , where T is the number of timesteps, and F is the number of input features. It transposes this to (F, T) , projecting the time history of each feature into the model dimension, then applies two layers of multi-head attention across the feature dimension before pooling to produce a fixed-size representation.

2.6.2 PatchTST — Patch-based Temporal Attention

While the iTransformer captures cross-variable structure, it does not explicitly model the temporal ordering of events within the input window. A complementary perspective is provided by PatchTST [12], which segments the time series into short contiguous patches before applying attention across those patches. The key insight is that individual time steps are often too noisy and too local to be informative in isolation — what carries meaning is the behaviour of a signal over a short interval rather than its exact value at a single moment.

Nie et al. [12] propose grouping consecutive timesteps into patches of fixed length and treating each patch as a single token. Attention then operates across patches, comparing the aggregate behaviour of the signal over one time interval to its behaviour over another. This has three concrete benefits: it reduces the number of tokens the attention mechanism must process (improving computational efficiency), it encourages the model to learn from local temporal patterns rather than individual point values, and it allows the model to capture dynamics that unfold over multiple timesteps — such as a gradual SINR decline leading to a BLER increase — more effectively than point-by-point attention.

In this thesis, we adapt PatchTST to a multi-scale configuration with two patch sizes, $p_1 = 1$ and $p_2 = 2$ timesteps, corresponding to 20 ms and 40 ms windows respectively. The single-step patch captures per-row events, while the two-step patch captures short-term trends over 40 ms. The outputs of the two patch branches are concatenated with the iTransformer output and projected through a fusion MLP to produce the final semantic state vector. This dual-attention design — cross-variable structure from iTransformer and cross-time structure from PatchTST — is what distinguishes the encoder from simpler alternatives and is motivated by the observation that the semantic labels depend on both the joint state of multiple KPIs and the temporal evolution of those KPIs over the input window, as illustrated in Figure 2.2.

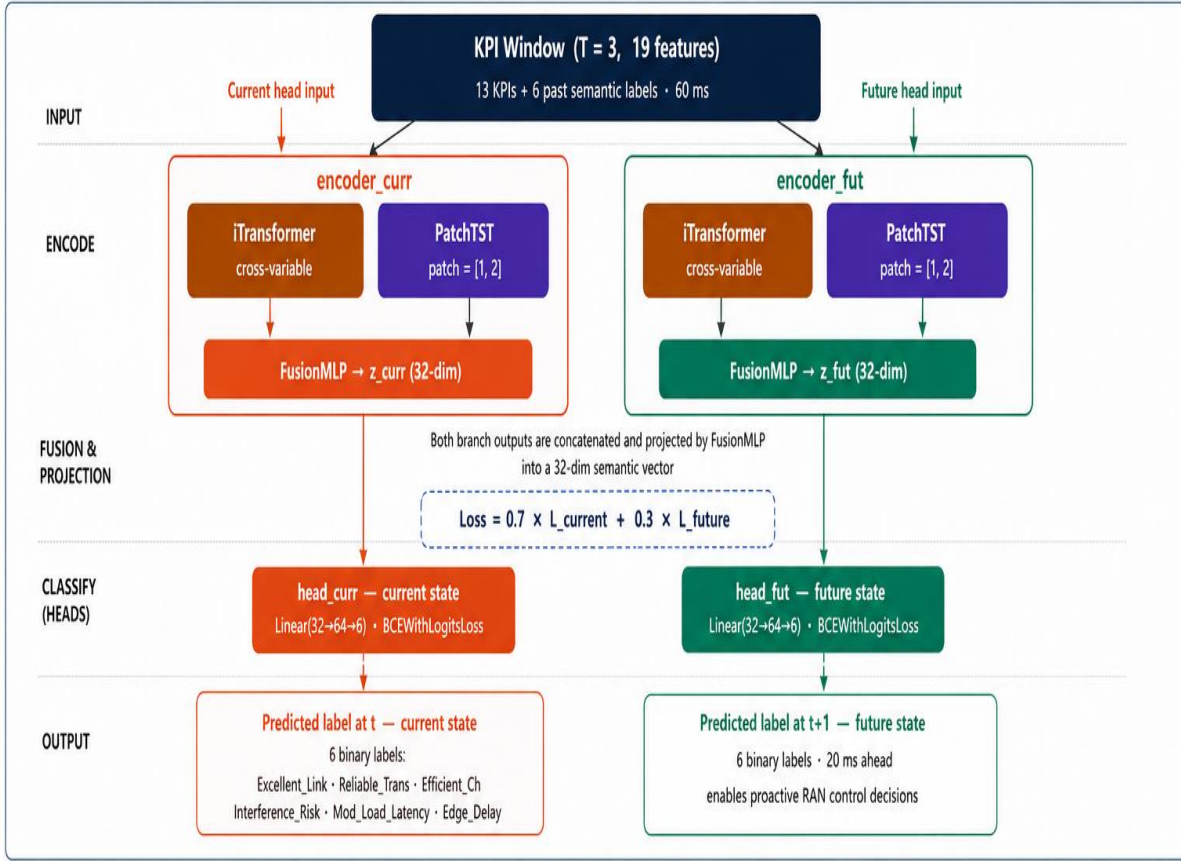


Figure 2.2: Dual-attention encoder combining iTransformer (cross-variable) and multi-scale PatchTST (cross-time) branches, fused into a 32-dimensional semantic state vector.

2.7 Explainability in RAN AI Systems:

The deployment of machine learning models in production radio access networks raises a question that goes beyond predictive accuracy: network operators need to understand why a model produced a particular output before they can trust it enough to act on it. A model that correctly classifies 93% of network states but provides no explanation for its decisions is difficult to validate, harder to debug when it fails, and may not satisfy regulatory or operational requirements that demand human oversight of automated control systems.

Two broad approaches to explainability have been explored in the RAN and wireless networking literature. The first relies on post-hoc attribution methods, of which SHapley Additive exPlanations (SHAP) is the most widely applied. SHAP assigns to each input feature a value that represents its marginal contribution to the model's output, computed as a weighted average over all possible subsets of

features. In the KPI context, a SHAP analysis can identify which measurements — SINR, BLER, PRB utilisation — contributed most to a particular state classification, giving an operator a feature-level explanation for each decision. Tassie et al. [13] apply this approach in an O-RAN setting, demonstrating that SHAP-based explanations can identify which KPIs are most predictive of performance degradation and can be used to reduce the number of indicators that an xApp needs to monitor, lowering E2 interface overhead without sacrificing classification accuracy. The second approach exploits the attention weights that are a native by product of Transformer inference. **Zerveas et al. [14]** demonstrate on multivariate time series that the attention maps produced during inference can highlight which input variables and time steps the model relied on most, providing a natural and computationally free form of interpretability that complements post-hoc methods such as SHAP. Because attention weights quantify how much each input token influences each output, they provide a natural form of explainability that does not require any additional computation beyond the standard forward pass. In a cross-variable Transformer such as the iTransformer used in this thesis, the attention weights over the feature dimension indicate which KPI pairs the model considered most strongly correlated when producing its output. A high attention weight between SINR and BLER, for example, would be consistent with the known physical relationship between link quality and block error rate.

It is worth noting that attention-based explanations have been subject to debate in the broader machine learning literature: high attention weights do not always imply causal influence, and the relationship between attention and explanation fidelity depends on the architecture and training objective. In the context of this thesis, attention weight analysis is identified as a direction for future work. The encoder produces attention maps as a byproduct of inference, and visualising these maps across different network states would allow a practitioner to verify that the model is attending to KPI relationships that are physically plausible, adding a layer of interpretability that complements the quantitative evaluation presented in the results chapter.

3 Semantic-Aware RAN Control Framework

This chapter presents the semantic-state classification and forecasting framework proposed in this thesis: a neural architecture that maps a short window of raw 5G NR KPI measurements to a compact set of binary semantic labels describing the current and predicted future state of the radio link. The framework is designed to operate inside the Near-RT RIC as a preprocessing layer that transforms high-

dimensional KPI streams into operationally meaningful representations that downstream control applications can act upon directly and efficiently.

The chapter is organised as follows: Section 3.1 states the problem formally; Section 3.2 defines the six semantic labels that constitute the output of the encoder; Section 3.3 presents the three encoder configurations evaluated — A1 (KPIs only baseline), B1 (single encoder with past labels), and the dual-encoder model B2 (KPIs + past labels, dual independent encoders) ; and Section 3.4 outlines the event-driven decision framework that the encoder output is designed to support. The dataset used for training and evaluation, along with the training methodology, are described in Chapter 4.

3.1 Problem Formulation

The RAN state classification problem involves processing sequences of multivariate KPI measurements that exhibit two distinct structural properties: cross-variable dependencies, where semantic conditions emerge from the joint behaviour of multiple KPI channels simultaneously, and temporal dynamics, where the evolution of KPI values over a short window carries information about both the current state and its near-future trajectory. Transformer-based architectures are particularly well suited to this setting because their self-attention mechanism can learn arbitrary dependencies across variables or time steps without imposing a fixed locality or ordering prior, making them effective for capturing the complex relational structure of KPI data.

Let $\mathbf{x}_t \in \mathbb{R}^F$ denote the KPI measurement vector at time step t , where F is the number of KPI channels. At each step, the base station reports one such vector to the Near-RT RIC over the E2 interface. The encoder receives a sliding window of T consecutive measurements:

$$\mathbf{X}_t = [\mathbf{x}_{t-T+1}, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t] \in \mathbb{R}^{T \times F} \quad (3.1)$$

In addition to the raw KPI window, the encoder receives the binary semantic labels predicted for the previous $T - 1$ steps as auxiliary context. Denoting the number of semantic labels as L , the augmented input per timestep has dimensionality $F + L$, giving a total input tensor:

$$\tilde{\mathbf{X}}_t \in \mathbb{R}^{T \times (F + L)} \quad (3.2)$$

The inclusion of past semantic labels as auxiliary input is motivated by the temporal persistence of network states — knowing that a label was active in the previous windows provides informative context for both classifying the current state and forecasting the next one. The contribution of this design choice relative to a KPI-only input is evaluated experimentally in Chapter 4.

The encoder is required to produce two outputs simultaneously from a single forward pass:

1. A **current-state prediction** $\hat{y}_t \in \{0, 1\}^L$ — the binary activation of each of the L semantic labels at the current timestep t .
2. A **future-state prediction** $\hat{y}_{t+1} \in \{0, 1\}^L$ — the binary activation of each label one step ahead, corresponding to the network condition at $t + 1$.

The label at t is never directly observed by the current-state head at inference time — it is masked in the input to prevent the encoder from trivially copying the label rather than learning to classify it from the KPI signal. This design choice, referred to as **current-label isolation**, ensures that the encoder learns a genuine mapping from KPI dynamics to semantic state. The future-state head, by contrast, receives all past labels as context, since they represent genuinely available historical information at inference time.

The two prediction tasks are formulated as independent multi-label binary classification problems. Each label is treated as a separate Bernoulli variable, and the model is trained with a Binary Cross-Entropy loss with logits. Class imbalance — which is significant for rare labels such as Edge Processing Delay — is addressed through per-label positive class weighting computed from the training set distribution.

3.2 Semantic Label Definitions

The six semantic labels are defined through deterministic threshold rules applied to combinations of KPI channels. Each label is a binary indicator of a qualitative network condition that has operational significance — it corresponds to a state that a network operator would recognise as meaningful and that would warrant a specific class of control response. The threshold rules serve as a weak supervision signal, generating binary target labels from the raw KPI stream without requiring manual annotation.

The label definitions are as follows, where all KPI values refer to the measurement at the current timestep t :

Table 3.1: Semantic label definitions — threshold rules applied to KPI channels.

Label	Threshold Rule	KPIs Involved	Operational Meaning
Excellent Link	$SINR \geq 15 \text{ dB}$ AND $RSRP \geq -91 \text{ dBm}$	DL SINR, RSRP	Link quality is strong. Handover not needed.
Reliable Transmission	$DL \text{ BLER} \leq 2\%$	DL BLER	MAC layer transmitting reliably.No retransmission pressure.
Efficient Channel State	$CQI \geq 7$ AND $SE \geq 0.5 \text{ bps/Hz}$	CQI, SE	Channel supports efficient spectral use. High-order MCS can be used.
Interference Risk	$RSRQ < -8 \text{ dB}$	RSRQ	Interference level elevated. Frequency or power adjustment may be needed.
Moderate Load High Latency	$PRB \geq 60\%$ AND $delay \geq 20 \text{ ms}$	PRB count, delay	Scheduler under pressure. Buffer management or

			Resource-reallocation needed.
Edge Processing Delay	$delay \geq 20\ ms\ AND$ $PRB < 80\%$	delay, PRB count	Latency elevated without high load — delay is at edge node, not scheduler.

The first three labels — Excellent Link, Reliable Transmission, and Efficient Channel State — indicate positive network conditions: when active (value = 1), the network is performing well on that dimension. The remaining three — Interference Risk, Moderate Load High Latency, and Edge Processing Delay — indicate adverse conditions: when active, they signal a potential impairment that may require intervention. This asymmetry is important for the decision logic described in Section 3.4, where a positive label turning off constitutes a degradation, while a negative label turning on constitutes a degradation.

It should be noted that Moderate Load High Latency and Edge Processing Delay share the delay threshold but differ in their PRB condition. This design deliberately separates two distinct causes of elevated latency: scheduler congestion (high PRB utilisation) and edge processing overhead (delay without corresponding load). A classification system that merges these two conditions into a single label would lose the information needed to select the appropriate control response.

3.3 Semantic-state encoder Architecture

Three encoder configurations are designed and evaluated in this thesis, forming a progressive development path from the simplest baseline to the proposed architecture. The starting point, referred to as **A1**, uses the 13 raw KPI measurements as input ($F=13$) and a single shared encoder that feeds both the current-state and future-state prediction heads. A1 represents the baseline condition — the minimal input that any existing xApp already has access to, requiring no semantic augmentation and no architectural specialisation.

The second configuration, referred to as **B1**, augments the input with past semantic labels (F=19: 13 KPIs + 6 past label values from the previous timesteps) while keeping the same single shared encoder structure as A1. The motivation for this addition comes from the high temporal persistence of semantic states observed in the dataset: label autocorrelation at lag=1 reaches 0.934 for Excellent Link and 0.850 for Moderate Load High Latency, meaning that the previous semantic state is a strong predictor of the current and future states. By providing this history as part of the input, B1 allows the encoder to exploit temporal continuity directly from the input representation rather than having to infer it purely from raw KPI dynamics. B1 isolates the contribution of semantic label context independently of any architectural change — the encoder structure is identical to A1, and only the input is enriched.

The third configuration, referred to as **B2**, keeps the same F=19 input as B1 but replaces the single shared encoder with two independent encoder instances — one dedicated to current-state classification and one dedicated to future-state prediction. The motivation is that the two prediction tasks have fundamentally different temporal focuses: the current-state head needs to capture the precise network condition at time t , while the future-state head needs to capture how the network state is evolving across the window. Forcing a single shared encoder to produce one representation that serves both objectives simultaneously creates a representational bottleneck — the encoder must compromise between the two tasks rather than specialising for either. B2 resolves this by allowing each encoder to develop a representation optimised for its respective head. The current-state encoder receives an input window in which the label at the last timestep is masked (current-label isolation), preventing the model from trivially copying the observed label instead of learning to classify from KPIs. The future-state encoder receives the full input window including all past labels, providing complete temporal context for anticipating state transitions. B2 is the proposed architecture and the primary subject of the experimental evaluation in Chapter 5.

Table 3.1 summarises the three configurations and their distinguishing characteristics. The ablation study in Section 5.7 quantifies the contribution of each design decision — the gain from A1 to B1 measures the value of past label context, and the gain from B1 to B2 measures the value of encoder specialisation — providing experimental justification for each step in this progression.

Table 3.1: Summary of the three encoder configurations evaluated in this thesis.

Property	A1	B1	B2 ★ (proposed)
Input features	KPIs only	KPIs + past labels	KPIs + past labels
Input dimension	F = 13	F = 19	F = 19
Encoder structure	Single shared	Single shared	Dual independent
Encoder instances	1	1	2
Current-state head	Shared encoder	Shared encoder	Dedicated encoder (label@t masked)
Future-state head	Shared encoder	Shared encoder	Dedicated encoder (full context)
Attention branches	iTransformer + PatchTST	iTransformer + PatchTST	iTransformer + PatchTST
Design contribution	Baseline — no context	Adds semantic label context	Adds encoder specialisation
Role in evaluation	Ablation reference	Ablation step 1	Proposed architecture

All three configurations share the same internal encoder structure, built from two complementary attention mechanisms that operate on the input window from different perspectives. This shared structure is described in Section 3.3.1, followed by the specific configurations of B1 (Section 3.3.2) and B2 (Section 3.3.3). A1 uses the same internal structure as B1 and B2 and is fully characterised by Table 3.1 — it does not require a dedicated subsection as its only distinction from B1 is the reduced input dimension $F=13$.

3.3.1 Encoder Building Blocks

For A1, the input dimension is $F=13$ (KPIs only). For B1 and B2, the input is augmented with past semantic labels giving $F+L=19$ features. Each encoder branch receives an input tensor of shape $(B, T, F+L)$, where B is the batch size, T is the window length, and $F+L$ is the number of input features. Two parallel branches process this tensor and their outputs are fused into a single semantic state vector of dimension S .

Transformer-based architectures are particularly well suited to this problem for several reasons. First, the semantic labels defined in this work emerge from the joint behaviour of multiple KPI channels simultaneously — Excellent Link requires both SINR and RSRP to meet thresholds at the same time, and Interference Risk is driven by the relationship between RSRQ and other signal quality indicators. A model that processes each KPI independently cannot capture these multi-channel dependencies without explicit feature engineering. The self-attention mechanism addresses this directly by learning which KPI pairs are most informative for each semantic condition from data alone, without hand-crafted interaction terms. Second, network state transitions unfold at more than one timescale: some changes are visible within a single 20 ms measurement interval while others develop gradually over several consecutive windows. A multi-scale patch-based attention approach is therefore better suited than fixed-window recurrent models, which process sequences at a single temporal resolution. Third, unlike recurrent architectures such as LSTM and GRU, Transformer encoders process all timesteps in parallel, which makes them significantly more efficient at inference time — a property that is critical for deployment within the 100 ms latency budget of the Near-RT RIC. The two attention branches described below are designed to exploit each of these properties specifically.

iTransformer Branch — Cross-Variable Attention. The iTransformer branch [11] addresses the joint KPI condition structure of the semantic labels. Labels such as Excellent Link require both SINR and RSRP to meet certain thresholds simultaneously — a condition that involves comparing the behaviour of two separate KPI channels. Standard Transformer architectures form tokens from all features at a single timestep and apply attention across time, which does not naturally expose this cross-feature structure. The iTransformer inverts this convention: each token is formed from the complete time history of a single KPI channel, and self-attention operates across channels rather than across time. Each feature sequence is projected to the model dimension D through a linear layer, and multi-head self-attention with nh attention heads allows each feature to attend to all other features, learning which KPI

channels co-vary and how their joint behaviour encodes the semantic state. The output is averaged across the feature dimension to produce a single vector of dimension D .

PatchTST Branch — Cross-Time Attention. The PatchTST branch [12] complements the iTransformer by capturing temporal dynamics within the window. Rather than attending to individual timesteps, PatchTST segments the input time series into short contiguous patches and treats each patch as a single token. This encourages the model to reason about the behaviour of the signal over a short interval rather than at a single point in time, which is more robust to measurement noise and better aligned with the physical timescales of the network conditions being classified. A multi-scale configuration is used with two patch sizes: a single-step patch and a two-step patch. The two branches run in parallel and their outputs are concatenated. The single-step patch captures instantaneous events at the shortest resolution, while the two-step patch captures dynamics that unfold over twice that interval — patterns that would be invisible to a point-level view.

Fusion Layer. The outputs of the iTransformer branch and the two PatchTST branches — three vectors each of dimension D — are concatenated into a single vector of dimension $3D$ and passed through a FusionMLP. The FusionMLP consists of a linear projection from $3D$ to D , a Gaussian Error Linear Unit (GELU) activation, a dropout layer, a second linear projection from D to S , and a Layer Normalisation (LayerNorm). The output is the semantic state vector $z \in \mathbb{R}^S$, which compresses the KPI window into a compact representation that captures both the joint cross-variable structure and the temporal dynamics of the input.

3.3.2 B1 — Single Shared Encoder (Baseline)

The B1 architecture uses a single instance of the encoder described in Section 3.3.1. This shared encoder receives the full input window — KPIs concatenated with past labels, with the label at t masked — and produces a single semantic state vector z . Two independent linear classifier heads then read from this shared representation to produce the current-state logits and the future-state logits respectively. Each head consists of a linear projection, a GELU activation, a dropout layer, and a final linear projection to L output dimensions.

The motivation for B1 is simplicity and parameter efficiency. A shared encoder produces a single representation that must simultaneously serve two prediction tasks with different temporal horizons. The current-state task requires the encoder to focus on the precise KPI values at the most recent timestep, while the future-state task requires it to extract trend information that generalises beyond the current window. Forcing both tasks to share a single representation imposes a representational bottleneck that may limit the performance of each head individually.

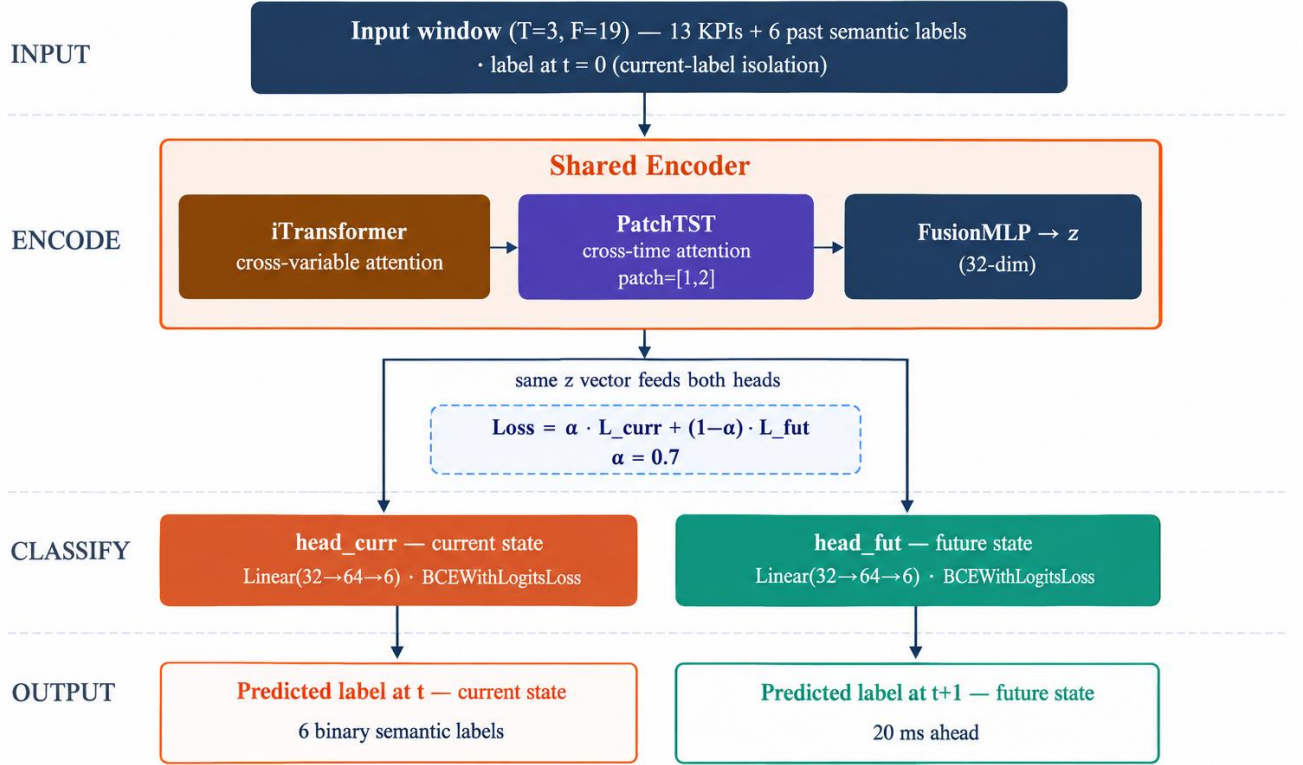


Figure 3.1: B1 baseline architecture

3.3.3 B2 — Dual Independent Encoders

The B2 architecture addresses the representational bottleneck of B1 by replacing the single shared encoder with two independent encoder instances: `encoder_curr`, dedicated to the current-state prediction task, and `encoder_fut`, dedicated to the future-state prediction task. The two encoders share the same

internal structure (iTransformer + multi-scale PatchTST + FusionMLP) but have independent weight sets, allowing each to develop a specialised representation suited to its respective task.

The inputs to the two encoders differ in a specific and deliberate way. The current-state encoder (`encoder_curr`) receives the KPI window with the semantic label at the current timestep t set to zero — the current-label isolation described in Section 3.1. This forces `encoder_curr` to infer the current state from the KPI signal rather than from the label history. The future-state encoder (`encoder_fut`) receives the complete label history without masking, since all past labels are genuinely available at inference time and provide informative context for predicting the next state.

Each encoder produces an independent semantic state vector: z_{curr} and z_{fut} . These vectors feed into their respective classifier heads (`head_curr` and `head_fut`), which have the same structure as in B1. The two heads are trained jointly through a weighted loss function described in Chapter 4. At inference time, both outputs are produced in a single forward pass. The trade-off between parameter count and prediction performance is evaluated in Chapter 5.

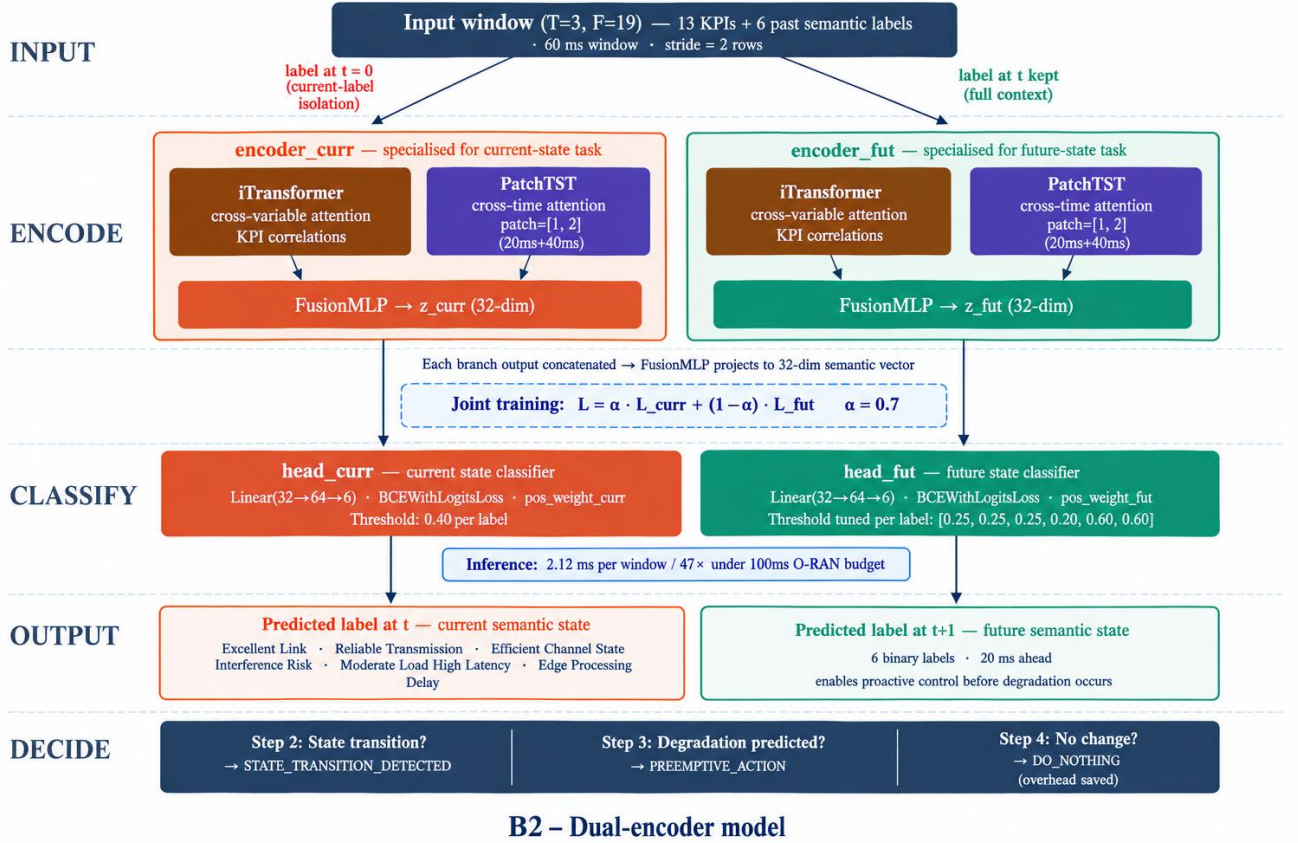


Figure 3.2: B2 — Dual-encoder model architecture

3.4 Event-Driven Decision Framework

The Semantic State Classifier is designed to serve as the sensing and inference layer of a larger event-driven control framework. This section describes the structure of that framework and the decision logic that connects the encoder output to network control actions. The full implementation of specific control actions — such as handover triggers or Modulation and Coding Scheme adjustments — is left as future work; this thesis implements and evaluates the decision logic layer using generic placeholder actions.

3.4.1 Semantic Head Groupings

For the purpose of the decision logic, the six semantic labels are grouped into four semantic heads, each corresponding to a distinct network control domain:

Semantic Head	Labels	Degradation Condition	Control Domain
link quality	Excellent Link, Reliable Transmission	Good label turns off (1 → 0)	Link quality management, handover decisions
scheduler	Efficient Channel State, Mod. Load High Latency	Efficient: 1→0 Mod. Load: 0→1	Scheduler and resource block allocation
interference	Interference Risk	Bad label turns on (0 → 1)	Power control, frequency management
latency	Edge Processing Delay	Bad label turns on (0 → 1)	Buffer management, edge node control

3.4.2 Decision Logic

For each incoming KPI window, the decision logic applies the following four steps:

Step	Action	Description
1	Inference	Feed the KPI window to the trained Semantic State Classifier (B2) . Obtain current_state $\hat{y}_t \in \{0,1\}^6$ and next_state $\hat{y}_{t+1} \in \{0,1\}^6$.
2	State transition detection	Compare current_state to the state predicted at the previous window. If any semantic head changed value, a STATE_TRANSITION_DETECTED event is raised and a reactive control action is triggered for the affected head.
3	Degradation prediction	Compare next_state to current_state. If next_state is worse on any semantic head — a good label predicted to turn off, or a bad label predicted to turn on — a PREEMPTIVE_ACTION event

		is raised and a proactive control action is triggered before the degradation occurs.
4	No change	If no state transition is detected and no degradation is predicted, DO_NOTHING is issued. No E2 control message is sent for this window, saving control plane overhead.

Steps 2 and 3 are not mutually exclusive — both can trigger simultaneously if the network is currently transitioning into a new state and the next state is also predicted to worsen. In that case, the action STATE_TRANSITION_DETECTED + PREEMPTIVE_ACTION is raised.

3.4.3 Evaluation on Validation Data

The decision logic is evaluated by running the full pipeline — encoder inference followed by the four-step logic — on 200 consecutive windows from the validation set, corresponding to 4,000 ms of simulated real-time operation. Table 3.2 summarises the distribution of actions produced.

Table 3.2: Decision action distribution over 200 validation windows (4,000 ms).

Action	Windows	Percentage	Control type
STATE_TRANSITION_DETECTED + PREEMPTIVE_ACTION	76	38.0%	Reactive +Preemptive
STATE_TRANSITION_DETECTED	60	30.0%	Reactive
PREEMPTIVE_ACTION	41	20.5%	Pre-emptive
DO_NOTHING	23	11.5%	No_action— overhead saved

The results show that the Semantic State Classifier output drives a control action in 88.5% of windows. Of these active windows, 58.5% involve a preemptive component — the B2 predicted a degradation before it occurred and triggered a control response ahead of time. This demonstrates the practical value of the future-state prediction capability: a purely reactive system that acts only on

observed state transitions would miss these early intervention opportunities. The 11.5% of DO_NOTHING windows represent a direct reduction in control plane overhead relative to a polling-based approach that sends a control message at every reporting interval.

4 Training Methodology

4.1 Dataset

The Semantic State encoder is trained and evaluated on a dataset of 5G NR KPI measurements collected from an indoor single-cell deployment [15]. The dataset consists of ten independent measurement sessions, each stored as a separate CSV file. Each row corresponds to one measurement interval of 20 milliseconds and contains values for thirteen KPI channels. Across all ten sessions, the dataset comprises 59,798 rows, equivalent to approximately 1,196 seconds of continuous network observation.

The Thirteen KPI channels recorded at each interval are: Downlink Signal-to-Interference-plus-Noise Ratio (DL SINR), Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ), Received Signal Strength Indicator (RSSI), Downlink Block Error Rate (DL BLER), Channel Quality Indicator (CQI), Physical Modulation and Coding Scheme (MCS), Spectral Efficiency (SE), Physical Resource Block count (PRB), Rank Indicator (RI), Precoding Matrix Indicator (PMI), Uplink Buffer Occupancy, and end-to-end packet delay. The delay channel contains the value -1 in intervals where no packet transmission occurred; this value is replaced by zero during preprocessing. The PRB count is expressed as a fraction of the total available resource blocks (25 in this deployment), converted to a percentage.

The ten measurement sessions are partitioned into a training set (sessions 1 through 8, 47,803 rows) and a fixed validation set (sessions 9 and 10, 11,995 rows). The partition is held constant across all experiments to ensure that all reported results are comparable. KPI features are normalised using the mean and standard deviation computed from the training set and applied to both splits — no information from the validation set is used during normalisation.

Table 4.1 summarises the dataset statistics. The class imbalance across labels is substantial: Excellent Link is active in 82.5% of rows, while Edge Processing Delay is active in only 1.8% of rows.

This imbalance is the primary motivation for per-label positive class weighting in the loss function, described in Section 3.5.

Table 4.1: Dataset statistics — 59,798 rows, 13 KPI channels, all 10 measurement sessions.

Label	Active rows	Positive rate
Excellent Link	49,316	82.5%
Reliable Trans.	46,387	77.6%
Efficient Ch.	29,765	49.8%
Interference Risk	46,330	77.5%
Mod. Load Latency	13,285	22.2%
Edge Proc. Delay	1,078	1.8%

Positive rate: fraction of windows where the label is active ($= 1$) across all 10 measurement sessions.

4.2 Loss Function

The model is trained with a weighted combination of two Binary Cross-Entropy losses with logits — one for each prediction head:

$$L = \alpha \cdot L_{\text{curr}}(\hat{y}_t, y_t) + (1 - \alpha) \cdot L_{\text{fut}}(\hat{y}_{\{t+1\}}, y_{\{t+1\}}) \quad (4.1)$$

where L_{curr} is the current-state loss, L_{fut} is the future-state loss, and α is a scalar weight that controls the relative emphasis on each task. The optimal value of α is determined experimentally in Chapter 4; the final configuration uses $\alpha = 0.7$, giving greater weight to the current-state task. This counter-intuitive choice — allocating more loss weight to the easier task — is motivated by the finding that a strong current-state representation acts as a regulariser that also improves future-state accuracy, as demonstrated in the ablation study of Chapter 5.

4.3 Class Imbalance Handling

The dataset exhibits severe class imbalance across labels, particularly for Edge Processing Delay (1.8% positive rate) and Moderate Load High Latency (22.2% positive rate). Without correction, the model would learn to predict the majority class (label = 0) for rare labels, achieving low loss while completely failing to detect positive events.

Each BCE loss is applied with per-label positive class weights computed from the training set. For label i , the positive class weight is:

$$w_i = (N - N_i^+) / N_i^+ \quad (4.2)$$

where N is the total number of training windows and N_i^+ is the number of windows where label i is positive. Separate positive class weights are computed for the current-state and future-state heads, since the window-level label distributions differ slightly between the two tasks.

4.4 Window Construction and Normalisation

Training samples are constructed by sliding a window of $T = 3$ rows over the training data with a stride of $S = 2$ rows. The stride of 2 creates an overlap of one row between consecutive windows, ensuring that every measurement row appears in multiple training samples and preserving temporal continuity at window boundaries. With 47,803 training rows, this produces approximately 23,900 training windows.

KPI features are normalised using the per-channel mean and standard deviation computed across all timesteps in the training set. The same statistics are applied to the validation set without recomputation, ensuring that no information from the validation set influences the normalisation. Semantic label features are not normalised — they are binary $\{0, 1\}$ values and are used as-is.

4.5 Optimisation and Regularisation

The model is optimised with the Adam optimiser [16] at an initial learning rate of 5×10^{-4} . A ReduceLROnPlateau scheduler halves the learning rate when the validation loss does not improve for 4 consecutive epochs. Training runs for a maximum of 60 epochs with early stopping triggered after 10

epochs without improvement in validation loss. Gradient clipping with a maximum norm of 1.0 is applied to prevent gradient explosion. The batch size is 64 and a dropout rate of 0.1 is applied within the encoder branches and classifier heads.

4.6 Per-Label Threshold Tuning

The default decision threshold for converting model output probabilities to binary predictions is 0.4. However, because the optimal threshold depends on the class imbalance and the probability distribution of each label, a post-training threshold search is performed independently for each of the six labels on the future-state head. For each label, eleven candidate thresholds are evaluated on the validation set: {0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60}. The threshold that maximises the per-label F1 score on the validation set is retained.

This procedure requires no retraining — it is a pure post-processing step applied after the model has converged. The optimal thresholds found for the best configuration are [0.25, 0.25, 0.25, 0.20, 0.60, 0.60] for [Excellent Link, Reliable Transmission, Efficient Channel State, Interference Risk, Moderate Load High Latency, Edge Processing Delay] respectively. The lower thresholds for the first four labels reflect that the future head assigns moderate probabilities to true positives for persistent labels; the higher thresholds for the rare labels reduce false alarms.

4.7 Evaluation Metrics

The framework produces binary multi-label predictions — six binary values per window for the current-state head and six for the future-state head. Standard single-label metrics such as accuracy are not appropriate for this setting because they do not account for partial correctness across labels. Four metrics are used throughout Chapter 5, each capturing a different aspect of multi-label classification performance. For each label and each window, a prediction is a **True Positive (TP)** if the label is correctly predicted as active, a **False Positive (FP)** if it is incorrectly predicted as active, and a **False Negative (FN)** if an active label is missed.

Table 4.2: Evaluation metrics used in Chapter 5.

Metric	Formula	Range	Interpretation
F1 Macro	$(1/L) \sum_j F1_j$ where $F1_j = 2TP_j / (2TP_j + FP_j + FN_j)$	$[0, 1] \uparrow$ higher is better	Unweighted average of per-label F1 scores. Treats all labels equally regardless of frequency. Penalises poor performance on rare labels such as Edge Processing Delay. Primary metric throughout Chapter 5.
F1 Micro	$2TP_{\text{global}} / (2TP_{\text{global}} + FP_{\text{global}} + FN_{\text{global}})$ where $TP_{\text{global}} = \sum_j TP_j$	$[0, 1] \uparrow$ higher is better	F1 computed from TP, FP, FN pooled across all labels. Dominated by frequent labels. Reported alongside F1 Macro for completeness. In unbalanced settings, $F1_{\text{Micro}} \geq F1_{\text{Macro}}$.
Exact Match	$(1/N) \sum_i \mathbb{1}[\hat{y}_i = y_i]$	$[0, 1] \uparrow$ higher is better	Fraction of windows where all L labels are predicted correctly simultaneously. The strictest metric — a single wrong label is a complete failure. Measures joint prediction quality across all six semantic conditions.
Hamming Loss	$(1/NL) \sum_i \sum_j \mathbb{1}[\hat{y}_{\{i,j\}} \neq y_{\{i,j\}}]$	$[0, 1] \downarrow$ lower is better	Fraction of individual label predictions that are wrong, averaged over all labels and all windows. Complementary to Exact Match — measures per-label error rate rather than per-window joint accuracy.

Notation: N = number of evaluation windows. L = number of semantic labels ($L = 6$). $\hat{y}_i \in \{0,1\}^L$ is the predicted label vector for window i . $y_i \in \{0,1\}^L$ is the ground-truth label vector. $\hat{y}_{\{i,j\}}$ and $y_{\{i,j\}}$ denote the j -th label of window i . TP_j , FP_j , FN_j are computed per label j independently.

All metrics are computed from binary predictions obtained by applying a decision threshold to the model output probabilities. Two threshold settings are reported in Chapter 5: a **default threshold of**

0.4 applied uniformly to all labels, and **per-label tuned thresholds** found by the procedure described in Section 4.6. For the current-state head, only the default threshold is reported. For the future-state head, both settings are reported to show the improvement achievable through post-training calibration.

5 Results and Analysis

This chapter presents the experimental evaluation of the proposed semantic state classification framework. The evaluation follows a structured progression designed to answer three questions in order: how well the framework classifies semantic states (Sections 5.1–5.4), what limits its performance and what does it cost (Sections 5.5–5.6), and why was each design decision made (Sections 5.7–5.9).

In detail, Sections 5.1 and 5.2 report overall current-state and future-state classification performance, across four training data scales of the different architectures (A1, B1, B2) discussed above. Sections 5.3 and 5.4 disaggregate these results per label, revealing which semantic conditions are easy to predict and which are not. Section 5.5 examines the hardest label — Edge Processing Delay — in depth, identifying the structural reason for its low future-state predictability. Section 5.6 reports the computational profile, confirming deployment feasibility within the Near-RT RIC latency budget. Section 5.7 presents an ablation study that isolates the contribution of each design decision — input type, encoder structure, and attention branch — explaining why B2 is designed the way it is. Section 5.8 evaluates the event-driven decision logic that closes the RAN control loop. Section 5.9 compares B2 against three established sequence modelling baselines and the single-encoder B1 variant, situating the proposed architecture within the broader landscape of approaches to this problem.

5.1 Overall Current-State Classification Performance

The first question this evaluation addresses is how accurately the framework can classify the current network semantic state. Three architectural configurations are compared across four training data scales ranging from 2 to 8 files (11,998 to 47,803 training rows). The validation set is fixed across all data scales to ensure a consistent evaluation surface.

Table 5.1: Current-state classification performance across training data scales.

Data Scale	Arch.	F1 Macro $\uparrow$	F1 Micro $\uparrow$	Exact Match $\uparrow$	Hamming $\downarrow$	Δ F1Mac vs A1	Δ F1Mac vs B1
2 files (11,998 rows / 5,997 win.)	A1	0.6598	0.7569	0.1920	0.2720	—	—
	B1	0.6816	0.7546	0.2322	0.2150	+0.0218	—
	B2 ★	0.6864	0.7451	0.1938	0.2400	+0.0266	+0.0048
4 files (23,998 rows / 11,997 win.)	A1	0.8182	0.8403	0.4481	0.1348	—	—
	B1	0.8533	0.8987	0.6147	0.0839	+0.0351	—
	B2 ★	0.8596	0.9098	0.6438	0.0775	+0.0414	+0.0063
6 files (35,847 rows / 17,921 win.)	A1	0.8829	0.8856	0.5425	0.1003	—	—
	B1	0.8873	0.9217	0.6830	0.0663	+0.0044	—
	B2 ★	0.8733	0.9151	0.6633	0.0715	-0.0096	-0.0140
8 files (47,803 rows / 23,899 win.)	A1	0.9435	0.9560	0.7990	0.0376	—	—
	B1	0.9457	0.9482	0.7737	0.0443	+0.0022	—
	B2 ★	0.9395	0.9495	0.7764	0.0434	-0.0040	-0.0062

Analysis: All three architectures improve consistently with more training data. At 2 files, B2 achieves the highest current-state F1 Macro (0.6864), already outperforming both A1 (0.6598) and B1 (0.6816), confirming that the dual-encoder design benefits current-state classification even with limited training data. From 4 files onward all three architectures improve substantially, with B1 and B2 consistently surpassing A1 by increasing margins, confirming that past semantic label context benefits current-state classification when sufficient training data is available. At 8 files, B1 achieves the highest single-seed current-state F1 Macro (0.9457) compared to B2 (0.9395) — however, the dedicated 5-seed multi-seed experiment (Table 5.12) confirms that B2 (0.9279 ± 0.0134) consistently outperforms B1 (0.9130 ± 0.0099) across all seeds, and the 8-file single-seed result reflects statistical variability rather than a systematic advantage of B1. The Δ F1Mac columns show that both B1 and B2 provide meaningful gains over A1 as data scale increases.

5.2 Overall Future-State Classification Performance

Having established current-state performance, the evaluation now turns to the harder and more operationally valuable task: predicting the network semantic state 20 ms ahead. This is the future-state head of B2 — the component that enables the event-driven control loop to act before a degradation occurs rather than after. Two threshold settings are reported: the default threshold of 0.4 applied uniformly to all labels, and per-label tuned thresholds found by searching over eleven candidate values on the validation set for each label independently. The tuned threshold result represents each model at its best achievable performance without retraining.

Table 5.2: Future-state (t+1) classification performance across training data scales.

Data Scale	Arch.	F1 Mac (def.) ↑	F1 Mic (def.) ↑	F1 Mac (tuned) ↑	F1 Mic (tuned) ↑	Exact (tuned) ↑	Hamming (def.) ↓	ΔF1Mac vs A1
2 files (11,998 rows / 5,997 win.)	A1	0.6207	0.7368	0.6702	0.7783	0.1694	0.3072	—
	B1	0.6024	0.6944	0.6724	0.7814	0.1673	0.2743	-0.0183
	B2 ★	0.6854	0.8030	0.7269	0.8376	0.4291	0.1950	+0.0647
4 files (23,998 rows / 11,997 win.)	A1	0.6523	0.7738	0.7090	0.8273	0.3946	0.2023	—
	B1	0.6655	0.7972	0.7321	0.8618	0.5093	0.1756	+0.0132
	B2 ★	0.7103	0.8448	0.7479	0.8830	0.5739	0.1378	+0.0580
6 files (35,847 rows / 17,921 win.)	A1	0.6930	0.8113	0.7355	0.8505	0.4473	0.1708	—
	B1	0.6954	0.8240	0.7450	0.8716	0.5290	0.1540	+0.0024
	B2 ★	0.6966	0.8186	0.7472	0.8818	0.5699	0.1684	+0.0036
8 files (47,803 rows / 23,899 win.)	A1	0.7168	0.8509	0.7711	0.8865	0.5707	0.1297	—
	B1	0.7156	0.8542	0.7751	0.8867	0.5707	0.1280	-0.0012
	B2 ★	0.7316	0.8668	0.7750	0.8974	0.6022	0.1168	+0.0148

Analysis: B2 consistently outperforms both A1 and B1 on future-state prediction across all data scales. At 8 files, B2 achieves F1 Macro=0.7316 versus A1's 0.7168 (+0.015) and B1's 0.7156 (+0.016). This is the most operationally important result: dual independent encoders combined with past semantic label context provide substantially better temporal context for predicting state transitions than either a single shared encoder (B1) or raw KPIs alone (A1). B1 also outperforms A1 on future state at all scales (e.g. +0.000 at 8 files default, +0.004 tuned), confirming that past label context adds value even with a shared encoder — though the gain converges at 8 files where both B1 and B2 approach the dataset-level performance ceiling identified in Section 5.5. The consistent advantage of B2 over B1 is confirmed as systematic by the 5-seed experiment (Table 5.13), which shows B2 winning on every seed. Per-label threshold tuning improves future-state F1 for all three architectures, with B2 reaching 0.7750 tuned at 8 files.

5.3 Per-Label Current-State Analysis

The macro-averaged F1 reported in Section 5.1 treats all six semantic labels equally. However, the labels differ substantially in their positive rates — from 82.5% for Excellent Link to 1.8% for Edge Processing Delay — meaning that the macro average can mask large differences in per-label performance. Table 5.3 disaggregates the current-state macro F1 into individual label scores, revealing which semantic conditions are learned earliest and which require more data to reach reliable performance.

Table 5.3: Per-label current-state F1 across training data scales.

Scale	Arch.	Exc. Link	Rel. Trans.	Eff. Ch.	Int. Risk	Mod. Load	Edge Delay
2 files (11,998 rows / 5,997 win.)	A1	0.8772	0.8236	0.6659	0.7089	0.7197	0.1635
	B1	0.8351	0.6815	0.8188	0.7162	0.7976	0.2407
	B2 ★	0.8830	0.6380	0.8048	0.6501	0.8891	0.2533
4 files (23,998 rows / 11,997 win.)	A1	0.8722	0.9820	0.7356	0.6209	0.9572	0.7413
	B1	0.9833	0.9788	0.8213	0.7006	0.9433	0.6928
	B2 ★	0.9850	0.9652	0.8423	0.7862	0.9081	0.6709

6 files (35,847 rows / 17,921 win.)	A1	0.8662	0.9890	0.8222	0.8174	0.9560	0.8468
	B1	0.9846	0.9808	0.8346	0.8202	0.9542	0.7491
	B2 ★	0.9842	0.9910	0.8311	0.7769	0.9102	0.7465
8 files (47,803 rows / 23,899 win.)	A1	0.9794	0.9950	0.8772	0.9360	0.9677	0.9060
	B1	0.9834	0.9917	0.8742	0.8969	0.9773	0.9507
	B2 ★	0.9848	0.9945	0.8784	0.8967	0.9647	0.9177

Analysis: Excellent Link, Reliable Transmission, and Moderate Load High Latency reach $F1 \geq 0.90$ by 4 files across all three architectures, indicating these conditions are well-represented even with limited training data. B1 and B2 both significantly outperform A1 on Edge Processing Delay (rarest label, 1.8% positive rate): at 8 files, B1 achieves 0.9507 and B2 achieves 0.9177 versus A1's 0.9060, confirming that past semantic label context particularly benefits classification of rare conditions. B1 achieves the highest per-label F1 on most labels at 8 files in single-seed evaluation, consistent with its higher single-seed current-state macro F1 — however, as established by the multi-seed analysis (Table 5.12), B2 is systematically superior across seeds and the single-seed advantage of B1 reflects variability.

5.4 Per-Label Future-State Analysis

The same per-label breakdown applied to future-state prediction reveals a pattern that is qualitatively different from the current-state case. While current-state performance converges across labels as data scale increases, future-state predictability varies dramatically by label and does not converge even at maximum data scale — a divergence that points to a structural rather than a statistical explanation, examined in detail in Section 5.5.

Table 5.4: Per-label future-state F1 across training data scales.

Scale	Arch.	Exc. Link	Rel. Trans.	Eff. Ch.	Int. Risk	Mod. Load	Edge Delay
2 files (11,998 rows / 5,997 win.)	A1	0.8773	0.8280	0.6298	0.7087	0.5636	0.1169
	B1	0.7811	0.6975	0.7023	0.6736	0.6085	0.1511

	B2 ★	0.9153	0.8249	0.8227	0.7091	0.6641	0.1765
4 files (23,998 rows / 11,997 win.)	A1	0.8781	0.9074	0.7278	0.6141	0.6754	0.1109
	B1	0.9732	0.9030	0.7573	0.5989	0.6494	0.1112
	B2 ★	0.9834	0.8991	0.8169	0.7821	0.6726	0.1080
6 files (35,847 rows / 17,921 win.)	A1	0.8676	0.9101	0.7711	0.7761	0.6829	0.1502
	B1	0.9738	0.9052	0.7612	0.7482	0.6677	0.1165
	B2 ★	0.9852	0.9011	0.8176	0.8046	0.5922	0.0789
8 files (47,803 rows / 23,899 win.)	A1	0.9668	0.9094	0.7853	0.8196	0.6606	0.1594
	B1	0.9720	0.9075	0.7935	0.8259	0.6293	0.1657
	B2 ★	0.9860	0.9112	0.8227	0.8055	0.6771	0.1872

Analysis: B2 leads on Excellent Link and Reliable Transmission across all scales, confirming that dual independent encoders better capture temporal dynamics of stable, persistent conditions. Four labels — Excellent Link, Reliable Transmission, Efficient Channel State, and Interference Risk — achieve future-state F1 above 0.80 for B2 at 8 files. B1 also outperforms A1 on all four high-persistence labels at 8 files, confirming that past semantic label context improves future-state prediction regardless of encoder architecture. **Moderate Load High Latency and Edge Processing Delay remain below 0.20 across all three architectures and all scales** — a finding confirmed as structural rather than architectural in Section 5.5. The convergence of all three architectures on these two labels confirms that their low predictability is a data-level constraint, not a model limitation.

5.5 Rare-Label Case Study: Edge Processing Delay

The per-label analysis of Section 5.4 identified Edge Processing Delay as an outlier: its future-state F1 remains consistently near zero across all data scales and both architectures, while its current-state F1 reaches 0.9177 at 8 files. This large gap between current and future performance — unique to this label — raises the question of whether the limitation is architectural (the model is not powerful enough) or structural (the label is inherently unpredictable at 20ms resolution). This section investigates this question through a state persistence analysis.

Table 5.5: Edge Processing Delay F1 across data scales

Data Scale	Arch.	Current F1 ↑	Future F1 ↑	Gap (Curr–Fut)	Est. Positive Windows
2 files (11,998 rows / 5,997 win.)	A1	0.1635	0.1169	+0.0466	72
	B1	0.2407	0.1511	+0.0896	72
	B2 ★	0.2533	0.1765	+0.0768	72
4 files (23,998 rows / 11,997 win.)	A1	0.7413	0.1109	+0.6304	144
	B1	0.6928	0.1112	+0.5816	144
	B2 ★	0.6709	0.1080	+0.5629	144
6 files (35,847 rows / 17,921 win.)	A1	0.8468	0.1502	+0.6966	215
	B1	0.7491	0.1165	+0.6326	215
	B2 ★	0.7465	0.0789	+0.6676	215
8 files (47,803 rows / 23,899 win.)	A1	0.9060	0.1594	+0.7466	287
	B1	0.9507	0.1657	+0.7850	287
	B2 ★	0.9177	0.1872	+0.7305	287

Analysis: The large gap between current-state and future-state F1 for Edge Processing Delay is consistent across all three architectures (A1, B1, B2) and all data scales. At 8 files, all three achieve high current-state F1 (A1=0.9060, B1=0.9507, B2=0.9177) yet near-zero future-state F1 (A1=0.1594, B1=0.1657, B2=0.1872). This architectural invariance confirms that the limitation is not caused by encoder design or input features — neither adding past label context (B1) nor dual independent encoders (B2) closes the gap. The structural explanation for this finding is examined in Table 5.6.

Table 5.6: State persistence analysis — relationship between label temporal persistence and future-state F1

Label	Autocorr. lag=1	Median Duration (ms)	Future F1 (B2, 8 files)	Interpretation
Excellent Link	0.934	580ms	0.9857	Highly persistent
Reliable Trans.	0.706	180ms	0.9128	Moderately persistent
Efficient Ch.	0.742	60ms	0.8305	Moderately persistent
Interference Risk	0.728	80ms	0.8423	Moderately persistent
Mod. Load Latency	0.850	60ms	0.6819	Persistent but complex
Edge Proc. Delay	0.401	20ms	0.1354	Structurally unpredictable

Analysis: The state persistence analysis reveals a strong correlation (Pearson $r=0.856$) between label autocorrelation at lag=1 and future-state F1 Macro across all six labels. Labels with high autocorrelation — Excellent Link (0.934, median duration 580ms) — are nearly trivial to predict one step ahead because the state rarely changes within a 20ms interval, and this holds regardless of which architecture (A1, B1, or B2) is used. Edge Processing Delay, with autocorrelation of only 0.401 and a median episode duration of exactly 20ms — equal to the prediction horizon — is structurally unpredictable: more than half of all episodes end before $t+1$, making the label absent at the predicted timestep regardless of the model. This finding directly explains the architectural invariance observed in Table 5.5: no architecture can overcome a data-level constraint where the target label systematically disappears before the prediction window.

5.6 Computational Profile

Classification accuracy is only one dimension of the evaluation — the framework must also satisfy the strict latency constraints of the Near-RT RIC, which requires that inference complete within 1 sec to stay within its control loop timescale. Table 5.7 reports training time, inference latency, GPU memory consumption, and convergence speed for both architectures across data scales. All measurements were performed on an NVIDIA A100 GPU (GRID A100D-20C). Inference latency is reported as the mean over 500 forward passes on a single window after a 10-pass warm-up to eliminate

cold-start effects. Two complexity metrics are reported alongside timing measurements. **Parameters (Params)** refers to the total number of learnable weight tensors in the model — the sum of all trainable floating-point values across all layers, which determines memory footprint at deployment. **Inference time** is the wall-clock latency for a single forward pass on one input window, measured as the mean over 500 consecutive passes after a 10-pass warm-up on the GPU, reported in milliseconds.

Table 5.7: Computational profile across training data scales.

Scale	Arch.	Train Time (s)	Inference (ms) ↓	GPU (MB)	Epochs	Params	vs 100ms Budget
2 files (11,998 rows / 5,997 win.)	A1	20.9s	1.107ms	36.0	13	322,284	90× under
	B1	23.2s	1.098ms	39.5	15	323,436	91× under
	B2 ★	33.2s	2.137ms	58.9	14	641,868	47× under
4 files (23,998 rows / 11,997 win.)	A1	101.0s	1.126ms	36.0	37	322,284	89× under
	B1	85.5s	1.133ms	39.5	30	323,436	88× under
	B2 ★	93.0s	2.151ms	58.9	22	641,868	46× under
6 files (35,847 rows / 17,921 win.)	A1	182.9s	1.249ms	36.0	48	322,284	80× under
	B1	121.0s	1.219ms	39.5	31	323,436	82× under
	B2 ★	168.0s	2.111ms	58.9	27	641,868	47× under
8 files (47,803 rows / 23,899 win.)	A1	288.5s	1.095ms	36.0	60	322,284	91× under
	B1	225.3s	1.125ms	39.5	45	323,436	89× under
	B2 ★	313.8s	2.373ms	58.9	40	641,868	42× under

Analysis: All three architectures operate well within the 100ms Near-RT RIC latency budget. B1 is the most computationally efficient of the three: 323,436 parameters and ~1.1ms inference — roughly half the size of B2 (641,868 params, ~2.4ms). A1 has a comparable parameter count to B1 (322,284 parameters) reflecting its single-encoder design with F=13 input features; the slight difference from B1 (323,436) arises from B1's additional label input dimensions (F=19). B2's higher parameter count relative to B1 (+318,432) is due to its second independent encoder. B1 converges fastest at all scales because its

single shared encoder receives gradient signal from both heads simultaneously, providing more learning signal per epoch. Despite B2's higher computational cost versus B1, its inference latency of $\sim 2.4\text{ms}$ remains 42 times below the 100ms budget, confirming that the dual-encoder design introduces no deployment constraints inside the Near-RT RIC control loop.

5.7 Ablation Study

Sections 5.1–5.6 established that B2 achieves strong performance and satisfies deployment constraints. This section addresses the question of why B2 is designed the way it is — specifically, whether each design decision genuinely contributes to the observed performance. B2 combines three design choices: (1) using past semantic labels as part of the input ($F=19$ vs $F=13$), (2) employing two independent encoders instead of one shared encoder, and (3) combining iTransformer and PatchTST branches rather than using either alone. A natural question is whether all three choices are necessary, or whether simpler alternatives would perform equally well.

To answer this, six controlled configurations are evaluated on the full 8-file training set (seed=42, $\alpha=0.7$, threshold tuning applied to all). The first three configurations systematically vary two design dimensions: input features (KPIs only vs KPIs+labels) and encoder structure (single vs dual): KPIs only with single encoder (A1 — simplest baseline), past labels with single encoder (B1 — adds label context), and KPIs only with dual encoders (A2 — adds encoder specialisation but without label context). The fourth and fifth configurations isolate each attention branch independently. The sixth configuration is the full B2 combining all design decisions. This structure allows the contribution of each choice to be measured independently and their interaction to be identified.

Table 5.8: Ablation study results — all 6 configurations

Configuration	Params	F1 Mac Curr ↑	F1 Mic Curr ↑	F1 Mac Fut ↑	F1 Mac Fut Tuned ↑	F1 Mic Fut Tuned ↑	Exact Fut Tuned ↑
A1: KPIs only, Single Enc. (F=13)	322,284	0.9435	0.9560	0.7168	0.7412	0.8732	0.5335
B1: Past Labels, Single Enc. (F=19)	323,436	0.9457	0.9482	0.7156	0.7422	0.8738	0.5365
A2: KPIs only, Dual Enc. (F=13)	639,564	0.9520	0.9551	0.6558	0.7213	0.8391	0.4153
iTransformer only (F=19)	218,060	0.7757	0.8196	0.6618	0.6995	0.8290	0.4111
PatchTST only (F=19)	433,228	0.9011	0.9287	0.7173	0.7402	0.8752	0.5295
B2: Past Labels, Dual Enc. (F=19) ★	641,868	0.9395	0.9495	0.7316	0.7565	0.8885	0.5735

Table 5.9: Factorial analysis: future-state F1 Macro (tuned) for each combination of input type and encoder structure. Reveals the interaction effect between past labels and dual encoders.

Input Type	Single Shared Encoder	Dual Independent Encoders
KPIs only (F=13)	0.7412	0.7213 (−0.020 vs single)
KPIs + Past Labels (F=19)	0.7422 (+0.001 vs row above)	0.7565 (+0.035 vs row above) ★

Analysis: The factorial analysis reveals a strong interaction effect: past semantic labels provide negligible benefit with a single shared encoder (+0.001) and dual encoders without label context actually decrease future F1 (−0.020). Only the combination of both design decisions produces a meaningful improvement: B2 achieves +0.035 future F1 Macro over the KPIs-only dual-encoder configuration, confirming that past labels and dual encoders are complementary rather than independent. At the branch level, iTransformer alone is insufficient (F1 Macro current=0.7757) while PatchTST alone performs well (0.9011), confirming that temporal dynamics are the primary discriminating signal — but cross-variable

attention from iTransformer adds a consistent +0.038 on current-state F1 when combined. Together these results provide a complete justification for every architectural decision in B2.

5.8 Event-Driven Decision Logic Evaluation

The preceding sections evaluated the semantic state classifier as a standalone prediction component. This section closes the loop by demonstrating how its dual output — current-state classification and future-state prediction — can drive an event-driven control loop inside the Near-RT RIC. The key property of this loop is that it acts selectively: rather than issuing a control message at every 20ms reporting interval regardless of network conditions, it triggers an action only when a semantic state transition is detected or an impending degradation is predicted by the future-state head. This selective behaviour reduces control plane overhead while also enabling preemptive responses that a purely reactive system cannot provide. The decision logic is evaluated on 200 consecutive windows from the validation set, corresponding to 4,000 ms of simulated Near-RT RIC operation.

Table 5.10: Event-driven decision logic action distribution over 200 validation windows (4,000 ms of simulated Near-RT RIC operation, stride=20ms).

Action Type	Windows	Percentage	Control Nature
STATE_TRANSITION + PREEMPTIVE_ACTION	76	38.0%	Reactive + Preemptive
STATE_TRANSITION_DETECTED	60	30.0%	Reactive
PREEMPTIVE_ACTION	41	20.5%	Preemptive only
DO_NOTHING	23	11.5%	No action — overhead saved
TOTAL	200	100%	200 × 20ms = 4,000ms operation

Analysis: Of the 200 evaluated windows, **177 (88.5%) result in an active control decision**, confirming that the validation deployment is dynamically active and that semantic states change frequently enough to justify continuous monitoring. The 23 DO_NOTHING windows (11.5%) represent

direct overhead reduction relative to a polling-based approach that would transmit a control message at every interval. Most significantly, **117 windows (58.5%) involve a preemptive component** — either `PREEMPTIVE_ACTION` alone (41) or combined with a detected transition (76). In these windows the future-state head identifies an impending degradation before it manifests in current measurements, enabling early intervention that a purely reactive system would miss entirely. This result directly demonstrates the practical value of the dual-output design: the current-state head provides situational awareness while the future-state head enables proactive network management.

5.9 Comparison with Baseline Architectures

The ablation study in Section 5.7 justified B2's design relative to its own architectural variants. This section addresses a different question: how does B2 compare against established sequence modelling architectures that were not designed for this specific task? Three standard baselines are included: a two-layer GRU, a two-layer LSTM, and a Vanilla Transformer encoder. All models are adapted to the dual-output setting (current + future state) and receive the same input (F=19: KPIs + past semantic labels) and the same hyperparameters ($\alpha=0.7$, window size $T=3$). Additionally, B1 (single encoder, F=19) is included to provide a direct architectural reference point within the same framework.

To ensure that results are not affected by random initialisation — a concern identified early in the evaluation when different runs of the same configuration produced a 0.042 gap in Edge Delay F1 — all models are trained with five independent random seeds {0, 1, 2, 3, 4}, with re-seeding applied immediately before each model is constructed. Results are reported as mean $\pm$ standard deviation over the five runs, and per-label threshold tuning is applied equally to all models.

Table 5.11 summarises the architecture and training configuration used for all models in this comparison, ensuring full reproducibility of the reported results.

Table 5.11: Architecture and training settings.

Method	Architecture	Key Dims	Params	LR	Pat.	MaxE p	Batch	Leak Safe	Unc.
GRU	Dual 2-layer GRU	H=64, L=2	91,852	5e-4	10	60	64	Yes	Point
LSTM	Dual 2-layer LSTM	H=64, L=2	119,372	5e-4	10	60	64	Yes	Point
Vanilla Transf.	Dual Transformer Encoder	d=64, nh=4, L=2	211,788	5e-4	10	60	64	Yes	Point
B1 (single enc.)	iT+PatchTST (single shared)	d=64, nh=4, pl=[1,2], sem=32, L=2	323,436	5e-4	10	60	64	Yes	Point
iTransf. only	iTransformer (ablation)	d=64, nh=4, L=2	218,060	5e-4	10	60	64	Yes	Point
PatchTST only	PatchTST (ablation)	d=64, pl=[1,2], L=2	433,228	5e-4	10	60	64	Yes	Point
B2 (ours) ★	Dual iT+PatchTST (proposed)	d=64, nh=4, pl=[1,2], sem=32, L=2	641,868	5e-4	10	60	64	Yes	Point

LR: Adam learning rate. Pat.: early stopping patience (epochs). MaxEp: maximum epochs. Batch: mini-batch size. $\alpha=0.7$: loss weight ($\text{Loss} = \alpha \cdot \text{BCE}_{\text{curr}} + (1-\alpha) \cdot \text{BCE}_{\text{fut}}$). H: hidden size (GRU/LSTM only). nh: number of attention heads (Transformers only). d: model dimension (d_{model}). L: number of encoder layers. pl: patch length(s) (PatchTST only). sem: semantic representation dimension after fusion. Leakage-Safe = Yes: past-only label covariates, contiguous tail split, train-only scaling. Unc. = Point: binary {0,1} predictions from sigmoid threshold.

The observation that B1 outperforms B2 on current-state F1 Macro in the scaling experiment (Tables 5.1–5.2) raised the question of whether this difference reflects a genuine architectural limitation of B2 or a statistical artefact of the single-seed evaluation protocol used in that experiment. To investigate this, a dedicated multi-seed comparison was conducted by running both B1 and B2 on the full 8-file training set with five independent random seeds {0, 1, 2, 3, 4}. Table 5.10 summarises the per-seed results and their mean $\pm$ standard deviation. The results are unambiguous: B2 outperforms B1 on every seed for both

current-state and future-state F1 Macro, with a mean current-state gap of $+0.0148$ (0.9279 ± 0.0134 vs 0.9130 ± 0.0099) and a mean future-state gap of $+0.0235$ (0.7232 ± 0.0067 vs 0.6998 ± 0.0077). The apparent advantage of B1 observed at seed=42 in the scaling experiment was therefore a statistical anomaly of a single run — not a systematic architectural property. These results confirm that B2's dual-encoder design consistently and reliably outperforms B1's single-encoder design on both prediction tasks across all seeds. Table 5.11 summarises the architecture settings and training hyperparameters used for all models in this comparison, ensuring full reproducibility. The complete multi-seed comparison across all baseline models is reported in Tables 5.12 and 5.13.

Table 5.12: Baseline comparison — current-state performance.

Model	F1 Macro $\uparrow$	F1 Micro $\uparrow$	Exact Match $\uparrow$	Hamming $\downarrow$
B1: single encoder	0.9130$\pm$0.0099	0.9274$\pm$0.0072	0.7053$\pm$0.0233	0.0609$\pm$0.0057
GRU	0.9281 $\pm$ 0.0042	0.9463 $\pm$ 0.0022	0.7650 $\pm$ 0.0084	0.0460 $\pm$ 0.0019
LSTM	0.9161 $\pm$ 0.0039	0.9353 $\pm$ 0.0027	0.7300 $\pm$ 0.0087	0.0547 $\pm$ 0.0019
Vanilla Transformer	0.9401 $\pm$ 0.0050	0.9538 $\pm$ 0.0052	0.7910 $\pm$ 0.0182	0.0396 $\pm$ 0.0044
B2: dual encoders (ours) $\star$	0.9279$\pm$0.0134	0.9391$\pm$0.0071	0.7434$\pm$0.0240	0.0517$\pm$0.0061

Table 5.13: Baseline comparison — future-state performance with computational profile.

Model	Future-State Performance					Complexity	
	F1 Mac (def.) $\uparrow$	F1 Mic (def.) $\uparrow$	F1 Mac (tuned) $\uparrow$	F1 Mic (tuned) $\uparrow$	Exact (tuned) $\uparrow$	Params	Infer. (ms) $\downarrow$
B1	0.6998 $\pm$ 0.0077	0.8341 $\pm$ 0.0085	0.7317 $\pm$ 0.0025	0.8624 $\pm$ 0.0027	0.5055$\pm$0.0060	323,436	1.428ms
GRU	0.7268 $\pm$ 0.0047	0.8647 $\pm$ 0.0052	0.7488 $\pm$ 0.0028	0.8848 $\pm$ 0.0024	0.5576 $\pm$ 0.0084	91,852	0.329ms
LSTM	0.7225 $\pm$ 0.0068	0.8615 $\pm$ 0.0046	0.7476 $\pm$ 0.0043	0.8839 $\pm$ 0.0033	0.5600 $\pm$ 0.0086	119,372	0.368ms
Vanilla T.	0.7256 $\pm$ 0.0032	0.8629 $\pm$ 0.0038	0.7469 $\pm$ 0.0052	0.8820 $\pm$ 0.0038	0.5488 $\pm$ 0.0101	211,788	0.935ms
B2 $\star$	0.7232$\pm$0.0067	0.8604 $\pm$ 0.0078	0.7446$\pm$0.0055	0.8800 $\pm$ 0.0064	0.5479$\pm$0.0183	641,868	2.710ms

Analysis: Two findings stand out from the baseline comparison. First, B1 (single encoder) achieves the lowest future-state F1 Macro of all models (0.6998 ± 0.0077) — below GRU, LSTM, Vanilla Transformer, and B2 — confirming that a single shared encoder cannot effectively exploit past semantic label context, consistent with the ablation study finding of Section 5.7.

Second, B2 achieves future-state F1 Macro statistically indistinguishable from GRU, LSTM, and Vanilla Transformer (0.7225 – 0.7268 , overlapping standard deviations), confirming a dataset-level performance ceiling rather than an architectural limitation. This ceiling is explained by the state persistence analysis of Section 5.5 (Pearson $r=0.856$): all models plateau at the same level because unpredictable labels such as Edge Processing Delay (median duration= 20ms =prediction horizon) structurally limit future-state macro F1 regardless of architecture. The motivation for B2 over simpler baselines is therefore not a higher macro F1 — which is bounded by the data — but its semantically grounded dual-encoder design, interpretable attention weights, and superior Edge Processing Delay current-state F1 over recurrent baselines (0.8982 ± 0.0396 vs GRU 0.8779 ± 0.0209 and LSTM 0.8637 ± 0.0193).

Beyond classification performance, practical deployment inside the Near-RT RIC requires that inference latency stay within the 100ms control loop budget. Two complementary timing metrics are reported. Per-window latency measures the wall-clock time for a single forward pass on one input window, averaged over 500 passes after a 20-pass warm-up, with `torch.cuda.synchronize()` called before and after the timed block to ensure all GPU operations complete before measurement. Batch throughput measures the average per-window cost when multiple windows are processed simultaneously in a single GPU call — exploiting the parallel architecture of the GPU to amortise fixed overheads across many inputs. In a production xApp, windows may arrive in bursts from multiple UEs, making batch throughput the operationally relevant metric for high-load scenarios. Table 5.13 reports both metrics for all models across batch sizes 1, 8, 16, 32, and 64.

Table 5.14: Per-window inference latency (ms) and batch throughput (ms per window) at different batch sizes

Method	BS=1 (ms) ↓	BS=8 (ms/win) ↓	BS=16 (ms/win) ↓	BS=32 (ms/win) ↓	BS=64 (ms/win) ↓	Speedup BS64 vs BS1	vs 100ms Budget
GRU	0.3218ms	0.0453ms	0.0226ms	0.0116ms	0.0059ms	54.5×	311× under
LSTM	0.4066ms	0.0571ms	0.0286ms	0.0148ms	0.0074ms	54.9×	246× under
Vanilla Transf.	0.7414ms	0.0944ms	0.0465ms	0.0214ms	0.0112ms	66.2×	135× under
B1 (single enc.)	1.0940ms	0.1415ms	0.0709ms	0.0359ms	0.0185ms	59.1×	91× under
iTransformer only	0.8590ms	0.1066ms	0.0542ms	0.0273ms	0.0138ms	62.2×	116× under
PatchTST only	1.5078ms	0.2002ms	0.0990ms	0.0498ms	0.0250ms	60.3×	66× under
B2 (ours) ★	2.2344ms	0.2885ms	0.1453ms	0.0702ms	0.0355ms	62.9×	45× under

Analysis: All models operate well within the 100ms Near-RT RIC latency budget even at single-window inference. B2 achieves a 62.9× throughput speedup at batch size 64 compared to single-window inference — reducing its effective per-window cost from 2.234ms to 0.036ms. All models show similar GPU parallelism gains of 54–66×, confirming that in production deployments where windows arrive in batches from multiple UEs, the latency advantage of simpler models (GRU, LSTM) over B2 becomes negligible — all achieve sub-0.06ms per-window throughput at batch size 64, well within any practical deployment budget.

Table 5.15 reports the Brier score for the future-state prediction head of all models. The Brier score measures how well the model's output probabilities match reality — it is the mean squared error between the predicted probability and the true binary label, averaged over all labels. Lower values indicate better probability calibration.

Table 5.15: Brier score (future head)

Method	Macro ↓	Exc. Link	Rel. Trans.	Eff. Ch.	Int. Risk	Mod. Load	Edge Delay
GRU	0.0876	0.0258	0.1207	0.1084	0.1101	0.0552	0.1057
LSTM	0.0840	0.0258	0.1102	0.1088	0.1246	0.0477	0.0872
Vanilla Transf.	0.0813	0.0232	0.1135	0.1097	0.1158	0.0513	0.0741
B1 (single enc.)	0.0857	0.0281	0.1277	0.1116	0.1191	0.0610	0.0666
iTransformer only	0.1429	0.0814	0.1647	0.1596	0.1644	0.1362	0.1513
PatchTST only	0.0814	0.0221	0.1133	0.1084	0.1068	0.0515	0.0872
B2 (ours) ★	0.0849	0.0216	0.1258	0.1111	0.1421	0.0512	0.0574

Analysis: Vanilla Transformer achieves the best macro-Brier score (0.0813), closely followed by PatchTST only (0.0814) and B2 (0.0849). B2 achieves the best calibration on Excellent Link (0.0216) and Edge Processing Delay (0.0574) — the two most operationally critical labels. iTransformer alone shows the worst calibration (0.1429 macro) consistent with its poor F1 Macro performance, confirming that cross-variable attention alone is insufficient for reliable probability estimation in this task. The small differences in macro-Brier score across the well-performing models (0.0813–0.0876) are consistent with the dataset ceiling identified in Section 5.5.

Table 5.16 extends the baseline comparison of Tables 5.12 and 5.13 by reporting Precision and Recall alongside F1 Macro for future-state prediction across all six models, under both the default threshold (0.4) and per-label threshold tuning. Precision measures how often a predicted state transition is correct; Recall measures what fraction of actual upcoming transitions the model successfully detects. Both metrics are reported as macro-averages over the six semantic labels, mean $\pm$ standard deviation over five independent seeds.

Table 5.16: Baseline comparison — future-state precision and recall.

Model	Default Threshold ($\alpha=0.4$)			Tuned Threshold		
	F1 Macro $\uparrow$	Precision $\uparrow$	Recall $\uparrow$	F1 Macro $\uparrow$	Precision $\uparrow$	Recall $\uparrow$
GRU	0.7268 $\pm$ 0.004 7	0.7099 $\pm$ 0.008 5	0.8590 $\pm$ 0.009 7	0.7488 $\pm$ 0.002 8	0.7189 $\pm$ 0.008 1	0.8599 $\pm$ 0.008 1
Vanilla Transformer	0.7256 $\pm$ 0.003 2	0.7051 $\pm$ 0.009 1	0.8684 $\pm$ 0.011 5	0.7469 $\pm$ 0.005 2	0.7175 $\pm$ 0.009 0	0.8660 $\pm$ 0.007 5
B2: dual encoders $\star$	0.7232 $\pm$ 0.006 7	0.7063 $\pm$ 0.005 4	0.8565 $\pm$ 0.009 8	0.7700 $\pm$ 0.002 7	0.7511 $\pm$ 0.001 4	0.8014 $\pm$ 0.011 2
LSTM	0.7225 $\pm$ 0.006 8	0.7029 $\pm$ 0.009 3	0.8653 $\pm$ 0.004 6	0.7476 $\pm$ 0.004 3	0.7177 $\pm$ 0.007 8	0.8600 $\pm$ 0.007 8
A1: KPIs only	0.7113 $\pm$ 0.007 8	0.7161 $\pm$ 0.008 2	0.8308 $\pm$ 0.005 7	0.7707 $\pm$ 0.004 5	0.7566 $\pm$ 0.005 4	0.7962 $\pm$ 0.007 9
B1: single encoder	0.6998 $\pm$ 0.007 7	0.7066 $\pm$ 0.010 8	0.8238 $\pm$ 0.019 7	0.7613 $\pm$ 0.001 5	0.7388 $\pm$ 0.002 1	0.7968 $\pm$ 0.007 7

Analysis: Three findings emerge from Table 5.16. **First**, under the default threshold, GRU, VT, B2, and LSTM achieve F1 Macro between 0.7225 and 0.7268 with fully overlapping standard deviations and Precision clustered between 0.7029 and 0.7099, consistent with the dataset-level performance ceiling identified in Section 5.5 (Pearson $r = 0.856$). **Second**, B2 achieves the highest future-state Recall under the default threshold (0.8565 ± 0.0098), surpassing A1 (0.8308 ± 0.0057) by $+0.0257$ and B1 (0.8238 ± 0.0197) by $+0.0327$, and the highest Precision after tuning (0.7511 ± 0.0014) with the lowest precision variance — confirming that B2 detects more upcoming state transitions and produces better-calibrated probability outputs than any other model. In the Near-RT RIC deployment scenario, both properties are operationally valuable: high recall minimises missed interventions, while high precision minimises false alarms. **Third**, B1 remains the weakest model on both Recall and Precision under both thresholds, consistent with the ablation finding that a single shared encoder cannot effectively exploit past semantic label context (Section 5.7).

6 Conclusions

This thesis proposed and evaluated a **Semantic state encoder** for the 5G NR radio access network — a dual-output neural architecture that maps a short window of raw KPI measurements to a compact set of binary semantic labels describing both the current network condition and its predicted state one step ahead. The framework was developed through a controlled progression of three configurations. The baseline A1 established what is achievable from raw KPIs alone. B1 introduced past semantic labels as additional input but retained a single shared encoder, revealing that label context without architectural specialisation provides negligible benefit (+0.001 future F1 Macro). B2 resolved this by dedicating one independent encoder per prediction head, producing a +0.035 improvement in future F1 Macro over the equivalent KPIs-only dual-encoder configuration and confirming that past semantic labels and dual encoders are complementary — neither is sufficient alone. At full data scale, B2 achieves **F1 Macro=0.9395** on current-state classification and **F1 Macro=0.7750** on future-state prediction with an inference latency of **2.710 ms** — more than 37 times below the 100 ms Near-RT RIC budget. Applied to 200 consecutive validation windows, the event-driven decision logic triggered a preemptive control response in **58.5% of windows**, demonstrating that the future-state head provides actionable anticipation beyond what a purely reactive system could deliver. The baseline comparison against GRU, LSTM, and Vanilla Transformer confirmed that all architectures converge to statistically indistinguishable future-state F1 values, revealing a dataset-level performance ceiling explained by the structural unpredictability of Edge Processing Delay — a label whose median episode duration of 20 ms equals the prediction horizon, making more than half of its episodes structurally absent at $t+1$ regardless of the model used.

The work is subject to a number of scope limitations that define the boundaries within which the conclusions hold. The dataset originates from a single indoor single-cell deployment, and the semantic label thresholds — calibrated for this environment — may require recalibration for outdoor, multi-cell, or high-mobility scenarios. The future-state head predicts a single horizon of 20 ms ($t+1$), which is appropriate for the Near-RT RIC control loop but insufficient for applications requiring longer-horizon anticipation. The event-driven decision logic triggers generic control actions rather than concrete E2 interface commands, and the mapping to specific RAN operations — handover requests, scheduler parameter adjustments, interference mitigation directives — is left as an integration task for future work. Finally, the sensitivity of the architecture to the choice of window size T and stride was not explored systematically.

Several directions follow naturally from these findings. Validation on multi-cell outdoor deployments with UE mobility is the most immediate extension, needed to establish whether the learned semantic representations generalise across deployment contexts. Extending the future-state head to predict at multiple horizons simultaneously would enrich the anticipation capability of the control loop. An online threshold recalibration mechanism — adapting per-label thresholds to distribution shifts in the live network without full retraining — would improve robustness in production environments. End-to-end integration on an open-source O-RAN testbed such as ColO-RAN, connecting the semantic output to real downstream xApps, would provide the full validation of the semantic abstraction pipeline under operational conditions. Finally, a structured analysis of the attention weights produced by the iTransformer and PatchTST branches could provide label-level explanations for network operators, supporting the interpretability requirements for AI-based control in production networks. Beyond these extensions, the broader contribution of this work is the semantic abstraction layer itself: a reusable component that any downstream xApp can consume without independently processing the raw KPI stream. As the O-RAN ecosystem matures and the number of co-deployed xApps grows, the need for shared, compact state representations will increase. The Semantic State Classifier proposed in this thesis is a step toward an architecture where network intelligence is organised around meaning rather than raw measurement — the semantic communication principle applied not to the radio link, but to the management and control plane of the network itself.

References

- [1] O-RAN Alliance, "O-RAN Architecture Description," Technical Specification O-RAN.WG1.OAD, O-RAN Alliance, 2024.
- [2] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023.
- [3] J. F. Santos, A. Huff, D. Campos, K. V. Cardoso, C. B. Both, and L. A. DaSilva, "Managing O-RAN Networks: xApp Development from Zero to Hero," *IEEE Communications Magazine*, 2023.
- [4] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "CoO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5787–5800, 2023.
- [5] 3GPP, "NR; Physical layer measurements," Technical Specification TS 38.215, 3GPP, 2024.
- [6] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.
- [7] D. Gündüz, Z. Qin, I. E. Aguerri, H. S. Dhillon, Z. Yang, A. Yener, K. K. Wong, and C.-B. Chae, "Beyond Transmitting Bits: Context, Semantics, and Task-Oriented Communications," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 1, pp. 5–41, Jan. 2023.
- [8] P. V. A. Alves, I. Rego, L. Medeiros, et al., "Machine Learning Applied to Anomaly Detection on 5G O-RAN Architecture," *Procedia Computer Science*, vol. 222, pp. 81–93, 2023.
- [9] K. Hasan, T. Trappenberg, and I. Haque, "A Generalized Transformer-based Radio Link Failure Prediction Framework in 5G RANs (GenTrap)," *arXiv:2407.05197*, 2024.
- [10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.
- [11] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "iTransformer: Inverted Transformers Are Effective for Time Series Forecasting," in *Proc. ICLR*, 2024.
- [12] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A Time Series is Worth 64 Words: Long-term Forecasting with Transformers," in *Proc. ICLR*, 2023.
- [13] C. Tassie, B. Kim, J. Groen, M. Belgiovine, and K. Chowdhury, "Leveraging Explainable AI for Reducing Queries of Performance Indicators in Open RAN," in *Proc. IEEE ICC*, 2024, pp. 5413–5418.
- [14] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A Transformer-based Framework for Multivariate Time Series Representation Learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2114–2124, 2021.

- [15] J. Dai, R. Zhao, and L. Liu, "Video Streaming Network KPIs for O-RAN Testing," IEEE Dataport, May 2025, doi: 10.21227/v8p4-v974.
- [16] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in Proc. International Conference on Learning Representations (ICLR), 2015.