Università di Genova

MASTER OF SCINECE PROGRAM IN ENGINEERING
TECHNOLOGY FOR STRATEGY AND SECURITY

Deep Learning-based Signal Processing for Real-Time Safety-Critical Applications

Sina Gholami Fashkhami

Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Engineering Technology for Strategy and Security.

July 2026

Prof. Francesco Bellotti — Tutor

Prof. Ricardo Berta — Tutor

Università di Genova | DITEN DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

ADVISORS

Prof. Francesco Bellotti

Prof. Ricardo Berta

CO ADVISORS

Dr. Matteo Fresta

Dr. Hadi Ballout

DATE

2026/07/17

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Prof. Francesco Bellotti, for his guidance, patience, and support throughout this work. His feedback shaped this thesis at every stage, and I am grateful for the trust he placed in me and for the opportunity to carry out this research in his group.

I am thankful to my co-supervisor, Prof. Riccardo Berta, whose insight and encouragement were invaluable, and whose door was always open when I needed direction. Working with both has been a true privilege.

A special thank you goes to Matteo Fresta, my co-advisor, who accompanied me with this work more closely with me. His day-to-day help, his readiness to discuss ideas at any hour, and his steady encouragement made a real difference, and I am fortunate to have had him alongside me throughout this project.

I would also like to thank my colleagues and co-advisors in the ELIOS Lab, Alessandro Pighetti, Hadi Ballout, and Luca Lazzaroni, together with everyone else in the group. Their collaboration, their willingness to share knowledge, and the friendly atmosphere they created made the lab a place I looked forward to being in, and much of this work would have been far harder without them.

I wish to remember Prof. Agostino Bruzzone, our coordinator, who sadly passed away. His dedication and passion for his role and this program left a lasting mark in my heart, and it is with respect and gratitude that I acknowledge him here.

My thanks also go to the professors of my degree program, whose teaching over these two years built the foundation on which this thesis rests, and to my friends, who supported me, kept me grounded, and made this journey lighter through its most demanding moments.

Finally, and most of all, I want to thank my wife, Mahsa. It would not have been possible for me to be here, or to reach this graduation, without her. Through every late night, every deadline, and every long stretch of work, she was there with endless patience and steadiness. She believed in me and stood by me at every step of this journey. She celebrated the small victories with me and helped me through the hard days so that I could keep going. This achievement is not mine alone; it is as much hers as it is mine, and I am endlessly grateful to share it, and my life, with her.

ABSTRACT (ENGLISH)

Many safety-critical systems must calculate quantities they cannot measure directly, from noisy real-time signals, on low-power hardware at the edge. This thesis addresses three use cases: road surface classification, driver distraction detection, and battery State-of-Health estimation. They are from different domains, but they are the same problem of real-time hidden-state inference under edge and safety constraints, and the thesis proposes one five-stage methodology for all three: signal acquisition and fusion, pre-processing and representation, compact model design, systematic validation with failure analysis, and edge-oriented optimization.

The first use case classifies road surfaces from smartphone accelerometer signals with a dual-branch quantized network that fuses raw vibration with statistical features. It reaches 93.9% macro-F1 against 87.6% for the CID-LCSS benchmark, at 354 kB against roughly 45 MB. The second builds a real-time driver monitoring system fusing vehicle-dynamics and eye-tracking streams; a systematic audit corrected fourteen silent defects, and the model drops from about 90% accuracy offline to about 80% under streaming conditions on two unseen drivers, a generalization gap invisible to offline evaluation. The third estimates battery State-of-Health with an 11-million-parameter convolutional gated-recurrent hybrid, reaching 2.21 percentage points MAE and $R^2 = 0.93$ on held-out cells from a 76-cell population aged under calendar, cyclic, and driving-profile conditions; the 42-million-parameter generative state of the art reports lower error, but on four cells of a single chemistry under one charging protocol. The accuracy here requires a 23-day input window, which cycle-aware windowing and per-battery normalization are designed to reduce.

Across all three, representation and validation determined the outcome as much as architecture, compact models competed with far heavier ones, and each use case's limit is stated clearly.

Keywords: real-time signal processing, edge computing, deep learning, state estimation, road surface classification, driver monitoring, battery State-of-Health, quantized neural networks, model validation, safety-critical systems.

ABSTRACT (ITALIAN)

Molti sistemi safety-critical devono calcolare grandezze non misurabili direttamente, da segnali rumorosi in tempo reale, su hardware a basso consumo a livello edge. Questa tesi affronta tre casi d'uso: classificazione del manto stradale, rilevamento della distrazione del conducente e stima dello State-of-Health delle batterie. Da domini diversi, sono lo stesso problema di inferenza in tempo reale di uno stato nascosto sotto vincoli di edge e sicurezza; la tesi propone un'unica metodologia in cinque fasi: acquisizione e fusione dei segnali, pre-processing e rappresentazione, modelli compatti, validazione sistematica con analisi dei guasti e ottimizzazione per l'edge.

Il primo caso classifica il manto stradale da segnali accelerometrici di smartphone con una rete quantizzata a doppio ramo che fonde vibrazione grezza e feature statistiche: 93,9% di macro-F1 contro l'87,6% del riferimento CID-LCSS, con 354 kB contro 45 MB. Il secondo realizza un sistema di monitoraggio del conducente che fonde dinamica del veicolo ed eye-tracking; un audit sistematico ha corretto quattordici difetti silenti, e il modello scende da circa il 90% di accuratezza offline a circa l'80% in streaming su due conducenti mai visti, un divario invisibile alla valutazione offline. Il terzo stima lo State-of-Health con un ibrido convoluzionale-ricorrente da 11 milioni di parametri: 2,21 punti percentuali di MAE e R^2 = 0,93 su celle escluse, da 76 celle invecchiate in condizioni calendariali, cicliche e di guida. Lo stato dell'arte generativo da 42 milioni di parametri riporta un errore inferiore, ma su quattro celle di una sola chimica e un solo protocollo di carica. Questa accuratezza richiede una finestra di 23 giorni, che il windowing basato sui cicli e la normalizzazione per batteria mirano a ridurre.

Nei tre casi, rappresentazione e validazione hanno determinato il risultato quanto l'architettura, i modelli compatti hanno competuto con altri più pesanti, e ogni limite è dichiarato esplicitamente.

Key Words: elaborazione dei segnali in tempo reale, edge computing, deep learning, stima dello stato, classificazione del manto stradale, monitoraggio del conducente, State-of-Health delle batterie, reti neurali quantizzate, validazione dei modelli, sistemi safety-critical.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Symbol	Meaning	Units
SoH	State-of-Health (ratio of current to initial capacity)	[%]
Q_now	Current usable capacity of the cell	[Ah]
Q_0	Initial (nominal) capacity of the cell	[Ah]
V	Cell voltage	[V]
I	Cell current	[A]
T	Cell temperature	[°C]
dV/dt	Time derivative of voltage	[V/s]
dI/dt	Time derivative of current	[A/s]
r	Residual (prediction minus ground truth)	[%]
δ	Huber loss transition threshold	[-]

Abbreviation	Meaning
BMS	Battery Management System
CNN	Convolutional Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
QNN	Quantized Neural Network

DMS	Driver Monitoring System
EOL	End-of-Life
CV	Cross-Validation
EDA	Exploratory Data Analysis
GroupKFold	Cell-grouped k-fold cross-validation

1 Introduction

Deep learning has shifted from research in the cloud to everyday physical systems like vehicles, mobile devices, industrial machinery, and energy-storage units. In these settings, a neural network no longer works with clean, static datasets in a data center. Instead, it processes noisy sensor data generated in real time by imperfect hardware. It often faces tight limits on latency, memory, and power. When the system being monitored is critical for safety, like a moving vehicle, a driver in the seat, or a battery driving an electric vehicle, the demand on the model increases significantly. If a prediction is late, uses too much energy, or fails without warning, it can put the safety of people and equipment at risk.

This thesis addresses this overlap: the use of deep-learning-based signal processing for real-time, safety-critical issues under the resource limits of embedded and edge deployment. The research took place at the ELIOS Lab at the University of Genoa and includes three use cases that might belong to different domains, road surface classification using smartphone vibration signals, real-time driver distraction detection from various sensors, and lithium-ion battery state-of-health (SoH) estimation from temporal data. Despite working in different areas, these three cases have a shared technical foundation and are connected by a single methodological framework developed throughout this thesis.

The key insight is that all three problems came down to the same fundamental task: extracting the hidden state of a safety-critical system from multivariate, asynchronous, noisy time-series signals and doing it with a model that is compact and reliable enough to operate in real time at the edge. The road surface, the driver's attention, and the battery's condition are all hidden variables that must be inferred from raw sensor data, accelerometer vibrations, gaze and head position, or voltage, current, and temperature readings. In every instance, the signal is a time series, the inference must run on hardware with limited resources, and any error has safety implications. This common structure shows that one methodology should be implemented.

Recent surveys on edge artificial intelligence indicate that the combination of needs, low latency, limited memory and energy, and dependable performance under real-world signal conditions, represents a key challenge in applied deep learning [1], [2]. Deploying neural networks on embedded hardware faces strict limits on inference time that must be met in real-time applications like advanced driver-assistance systems, where delays are not acceptable [2]. Most existing work addresses these constraints separately and within a single domain; there is relatively little focus on the methodological similarities across different fields. This thesis argues that a coherent, adaptable approach for compact time-

series deep learning is both possible and beneficial, demonstrating this across three distinctly different use cases.

1.1 The Problem: Real-time state inference under edge and safety constraints

The main issue this thesis tackles is how to design, train, and validate deep learning models to determine the hidden state of a safety-critical system using raw, multivariate time-series data. This needs to be done while meeting the real-time and resource limitations of edge deployment and being resilient to signal flaws and changes in distribution that occur in real-world settings.

This problem breaks down into several interconnected sub-problems that appear with different focuses across the three use cases:

- **Signal variety and integration.** Relevant information is seldom found in one clean data channel. In the driver-monitoring use case, two asynchronous sensor streams, eye-tracking and vehicle dynamics, need to be combined into a common time-based format before making predictions. In the road-surface and battery cases, raw signals must be paired with engineered or derivative features to reveal the structure that a compact model can use. Creating the representation that feeds into the network is as important as designing the network itself.

- **Real-time and edge constraints.** A model that performs well offline but is too large or slow for the intended hardware does not solve the problem. This thesis treats model size and inference speed as primary design constraints, leading to a focus on quantization-aware design, lightweight architectures, and inference that is not dependent on the window size.

- **Generalization with distribution changes.** Real-world deployments face conditions that the training phase did not cover new drivers, unfamiliar vehicles or phones, new battery cells that show variation between units that conceal the signals within a unit. A key finding of this work is that simple pipelines tend to overfit to specific features instead of the safety-related hidden state.

- **Validation and silent failure.** In safety-critical situations, it is much better for a model to fail loudly than quietly. A significant part of this thesis focuses on systematic auditing of the pipeline. This means carefully identifying issues related to correctness, data integrity, and robustness that can make models seem effective while they are

predicting a constant value or learning a biased feature. These failures are documented as findings because identifying them is a methodological achievement.

Another important aspect, viewed as a future-focused part of the methodology, is uncertainty quantification. Safety-critical decisions need not only precise predictions but also trusted confidence levels: an estimate with a reliable measure of their own accuracy. A battery state of health estimate of 81% ± 0.5% is useful, while 81% ± 8% is not. The literature increasingly acknowledges that using neural networks in safety-critical systems requires dependable uncertainty quantification. Monte Carlo dropout and deep ensembles are becoming the leading scalable methods for this [3], [4]. While complete uncertainty-aware estimation is not applied in all three use cases in this thesis, the proposed methodology is built to support it. The groundwork established here aims to facilitate its integration as a natural next step.

1.2 Methodological contribution and originality

The originality of this thesis is its methodology, and the claim is made in four specific parts.

First, it treats representation and validation as equal to architecture. Most published work in these three fields reports architecture and accuracy. This thesis reports, for each use case, the representation that was chosen and why, and the validation protocol that makes the accuracy meaningful. In the battery use case this changes the result: under random splits a model can memorize a cell, and only whole-cell held-out validation produces a number that transfers. In the driver-monitoring use case it changes the result more sharply, because a published model reporting high offline accuracy loses roughly ten accuracy points once it is run on streaming data from unseen drivers.

Second, the same five stages transfer across three domains with different signals, sampling rates, and time constants: a 100 Hz vibration signal, two fused real-time streams at different rates, and a year-long degradation trajectory. The transfer is showed: Table 2.1 shows the same five stages implied three times.

Third, the thesis outperforms the state of the art in each use case and quantifies the advancement. In road surface classification, a dual-branch quantized network that fuses raw vibration with statistical features reaches 93.9% macro-F1 against the 87.6% of CID-LCSS [5], at 354 kB against roughly 45 MB, a factor of about 128 in size. In battery State-of-Health, an 11-million-parameter CNN-GRU hybrid estimates health to within 2.21 percentage points ($R^2 = 0.93$) on held-out cells drawn from a 76-cell population aged under calendar, cyclic, and driving-profile conditions, against a 42-million-parameter generative

benchmark validated on four cells of a single chemistry under one charging protocol. In driver monitoring, the contribution is a deployable real-time pipeline and the finding, absent from the published work it builds on, that the reported offline performance does not survive streaming conditions and unseen drivers.

Fourth, the thesis reports what its methods cost, not only what they achieve. Each use case states its limit as precisely as its result: the road-surface model is measured on hardware; the driver-monitoring model has a ten-point generalization gap on two unseen drivers; the battery model needs a 23-day window it cannot have in deployment. A methodology that surfaces its own limits is more useful than one that reports only successes, and each limit named here defines a specific next experiment.

What a reader knows after this thesis that they did not before: that these three problems share one pipeline; that in each, the representation and the validation protocol determine the result as much as the architecture; that compact models (354 kB, 11 M parameters) reach accuracies competitive with far heavier ones; and that in two of the three cases, the headline accuracies reported in the literature do not survive an honest deployment-facing evaluation.

1.3 Related Work

Because the three use cases cover different application areas, a detailed review of the literature for each will be presented in their respective chapters. This section gives a broad overview of the research context shared by all three, organized around the three main pillars of the thesis: edge and lightweight deep learning, time-series modeling for sensor signals, and uncertainty-aware learning for safety-critical systems. The goal is to place the proposed methodology within the larger field before the specific chapters focus more closely on each use case.

1.3.1 Lightweight and Edge Deep Learning

Deploying neural networks on limited hardware, known as Edge AI, has become a major research focus. This shift is driven by needs for low-latency processing, data privacy, and independence from cloud connectivity [1]. The field has settled on a consistent set of optimization techniques: pruning, quantization, and inference-level improvements applied to lightweight architectures such as compact convolutional and recurrent networks [2]. A recent systematic review outlines a five-stage methodology that includes requirement definition, model selection, optimization, hardware alignment, and deployment [2]. This structure closely aligns with the methodology proposed in this thesis. This methodology was developed independently from the three use cases. Quantization has particularly emerged as a key factor in deploying microcontroller-class systems, as it reduces both

memory usage and energy per inference [6]. The road-surface use case in this thesis contributes to this work with a quantized dual-branch network that has a 354 KB footprint. The battery and driver-monitoring use cases are also designed with the same quantization and edge-deployment considerations.

1.3.2 Deep Learning for Time-Series Sensor Signals

All three use cases work with multivariate time series. The methodological development in this area has evolved from using handcrafted statistical features with traditional classifiers, to one-dimensional convolutional networks that learn local temporal filters directly from raw signals, and finally to recurrent and attention-based models that capture longer-range dependencies. Hybrid convolutional-recurrent structures and, more recently, patch-based transformers have advanced the state of the art in long-sequence regression and classification. This thesis utilizes a wide range of tools: convolutional, recurrent, hybrid, and attention-based models are all employed and compared, especially in the battery use case. It also finds that for these safety-critical signals, the choice of temporal representation and label assignment often matters more than the specific sequence model.

1.3.3 Uncertainty-Aware Learning for Safety-Critical Systems

A growing body of literature suggests that accuracy alone is not enough for safety-critical applications. Models need to express calibrated uncertainty in their predictions [3], [4]. Modern deep networks tend to be poorly calibrated by default, leading to both post-hoc solutions and training-time adjustments. Among scalable methods for quantifying uncertainty, Monte Carlo dropout (it keeps dropout during inference to approximate Bayesian inference) and deep ensembles (they combine independently trained models) have become the most practical choices [3]. Research in autonomous driving has shown how uncertainty travels from noisy sensor data through perception models to downstream decisions that directly affects system safety [7]. This thesis emphasizes uncertainty quantification as a key forward-looking aspect of its methodology: the compact architectures developed here are designed to work well with Monte Carlo dropout and ensembling, and incorporating calibrated uncertainty is identified as a priority for future work across all three use cases.

1.4 Contributions

The contributions of this thesis are summarized as follows:

1. A unified method for compact, real-time, safety-critical time-series deep learning. The main contribution is a clear methodology for signal acquisition and fusion, representation engineering, compact architecture selection, systematic validation, and edge-oriented optimization. This is detailed in a dedicated methodology chapter and applied in three distinct use cases.

2. A quantized dual-branch network for road surface classification [8]. This lightweight model combines raw vibration signals with engineered statistical features, achieving a macro-averaged F1-score of 93.9% with a 354 KB footprint. It was published in the proceedings of ApplePies 2025 (Springer LNEE).

3. A real-time multimodal driver monitoring system with systematic pipeline auditing. This live distraction detection pipeline combines eye-tracking and driving simulator streams. It includes a documented audit that found and fixed fourteen issues relating to correctness, robustness, and data integrity, along with a diagnostic analysis of a consistent prediction failure mode.

4. A comparative study of deep architectures for battery state of health estimation, including a new spectrogram autoencoder. This study compares recurrent, convolutional, hybrid, and attention-based models under a strict battery-aware evaluation protocol. It also introduces an original dual-branch spectrogram autoencoder, motivated by the discovery of inter-battery dominance in the feature space.

5. A treatment of validation and failure modes as research findings. Across all use cases, the identification and correction of silent failure modes, such as label leakage, distribution shift, constant prediction collapse, and representation faults, have been systematically documented and analyzed as significant methodological results.

6. Architectural groundwork for uncertainty-aware estimation. The compact models developed here are designed to work with scalable uncertainty quantification techniques. A concrete roadmap for integrating calibrated uncertainty into each use case is provided.

1.5 Thesis Structure

The rest of this thesis is organized as follows.

Chapter 2 presents the main methodology; the cross-domain design-and-validation approach that the next three use-case chapters demonstrate. It defines the common pipeline, the shared design principles, and the dimensions where the use cases differ.

Chapter 3 develops the first use case, road surface classification using a dual-branch quantized neural network. This corresponds to the published ApplePies 2025 paper.

Chapter 4 focuses on the second use case, a real-time driver monitoring system for detecting distractions. It emphasizes multimodal fusion, real-time inference, and systematic pipeline auditing.

Chapter 5 explores the third use case, deep learning-based battery state-of-health estimation. This includes a comparative model study and the new spectrogram autoencoder.

Chapter 6 concludes the thesis by summarizing the findings from all use cases. It reflects on the proposed methodology and outlines future directions, identifying uncertainty quantification and edge deployment as the key next steps.

A note on terminology: throughout this thesis, the three application studies are called use cases instead of projects. This highlights that they are coordinated applications of a single methodology. The methodology chapter explains what they have in common, while each use-case chapter details how it is adjusted to meet the specific needs of its domain.

2 Methodology

2.1 A shared inference problem across three domains

The three use cases in this thesis come from different fields, but they all deal with the same underlying engineering problem. Each case asks us to estimate a quantity that no instrument measures directly. This includes identifying the type of road under a moving vehicle, determining if a driver is paying attention, or assessing how much usable life is left in a battery cell. We must infer these quantities from the available sensor readings, such as vibration, gaze and steering behavior, or voltage, current, and temperature traces.

When we frame the problem this way, the shared structure becomes evident. Each case uses a multivariate time series generated by imperfect hardware. The output is a hidden state of a system, and any malfunction can lead to real costs in safety or finances. The model that connects the input and output must operate on compact hardware that is inexpensive and has limited power and memory. This aligns with the title of the thesis: deep-learning-based signal processing for real-time safety-critical applications.

This chapter outlines the general method applied in the next three chapters. Following the supervisor's guidance, we describe the method once here and then tailor it for each use case. The goal is to create a single pipeline with fixed stages, while the content adapts to the specific signals, targets, and deployment constraints of each area. This choice is intentional. Edge-AI practice has settled on a clear sequence of design steps [9], [10] and using that sequence as the foundation helps keep the three studies coherent as a single research effort.

2.2 The five-stage pipeline

The methodology is structured as a five-stage pipeline that runs from raw sensor signals to edge deployment. The stages are fixed across all three use cases: signal acquisition and fusion, pre-processing (and representation), compact model design, validation and failure analysis, and edge deployment. What changes between use cases is the content of each stage, not its position or purpose. This structure is shown in Figure 2.1. A sixth concern, uncertainty quantification, runs parallel to the other stages. It is a design goal and not a completed result and is discussed separately in Section 2.8. Section 2.9 mapping each use case onto the pipeline. This allows to quickly see what the three studies have in common and where they differ.

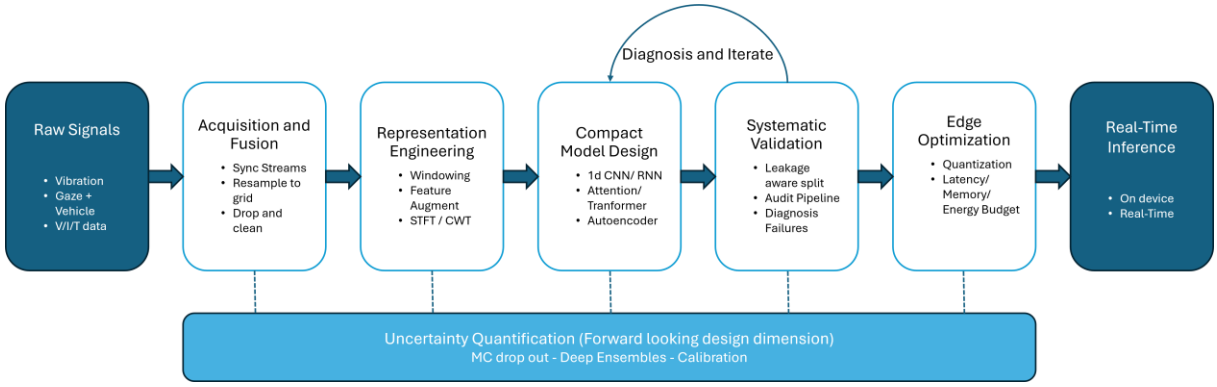


Figure 2.1 The five-stage methodology pipeline, from raw sensor signals through to edge deployment, with uncertainty quantification as a cross-cutting design dimension. Each use-case chapter instantiates the same stages with domain-specific content.

2.3 Stage 1: Signal acquisition and fusion

The first stage determines what the model will see. In none of the three use cases does the useful information arrive as a single clear channel ready for inference. The signals are sampled at different rates, contain gaps and noise, and in one case come from two devices that do not share a clock.

The road-surface case is the simplest. A smartphone accelerometer provides tri-axial readings at 100 Hz while a vehicle drives over a surface. These readings are reduced to a one-dimensional root-mean-square magnitude per sample [8]. There is one device and one stream, so acquisition mainly involves trimming and aligning sequences to a fixed length.

The driver-monitoring case is the most complicated. Two asynchronous sources feed the system through separate network sockets: an eye-tracking unit sends binary packets with gaze direction, eyelid opening, pupil diameter, and head pose, while a driving simulator sends text packets with vehicle dynamics like speed, acceleration, steering angle, and pedal position. These arrive at different rates and without a shared timestamp. So that fusion cannot be a simple concatenation. Both streams are logged, with their arrival timestamps, into a shared thread-safe queue. A rolling window then collects everything seen over a short interval and resamples it onto a single regular time grid before the model is allowed to examine it. This type of temporal alignment, where two sensors merge by timestamp instead of by a hardware sync signal, introduces a source of error that the validation stage must later address.

The battery case is in between [11], [12]. A single measurement file records voltage, current, temperature, and the state-of-charge derived from current integration, sampled

during charge and discharge cycles that span tens of hours. Here, acquisition focuses less on merging devices and more on deciding which channels to keep. The state-of-charge is dropped before any modeling, as it is computed from the same current signal the model already receives and partly encodes the target. This would leak label information into the input.

The general principle across the three cases is that the representation given to the network is a design choice with consequences as significant as the choice of network itself. Getting the fusion wrong or including a leaked channel results in a model that performs well on paper but fails in real-world application. Several of the failures documented later in the thesis trace back to this stage.

2.4 Stage 2: Pre-processing (and representation)

Having gathered the raw signals, the second stage converts them into fixed-shape tensors for model consumption. Three main techniques do much of the work: windowing, feature augmentation, and frequency-domain transformation.

2.4.1 Windowing and labeling

Continuous signals are divided into fixed windows, with each treated as one training example. The window length is important. It must be long enough to capture the phenomenon the model needs to see. In the case of the battery, a single charge or discharge cycle lasts for tens of hours. Therefore, a window shorter than one cycle mostly captures idle periods, which does not provide the model with useful variation. A systematic sweep over window sizes confirmed this: short windows resulted in fewer than half of all windows being in an active state, while windows covering a full cycle increased that fraction to over ninety percent. The window size was chosen based on the measured cycle duration not on the estimated size.

Labeling the windows is equally important [13]. The battery study assigns each window the mean state-of-health over its duration, not just the value at a single point in time. Since health changes over weeks while a window spans hours, the two choices appear nearly identical numerically. However, an earlier version that used a single endpoint produced hundreds of windows per file with the same label, leaving the model without a gradient to learn from. This issue caused the first round of training failures and is documented in the battery chapter as a finding.

2.4.2 Feature augmentation

Raw signals are sometimes paired with values derived from them. The road-surface model operates a separate branch of twelve statistical descriptors, which include energy, skewness, kurtosis, and the zero-crossing rate, alongside the raw vibration sequence. These descriptors provide the network with a compact summary of signal shape, complementing what the convolutional filters extract from the waveform. The battery model adds first-order derivatives of voltage and current as extra channels because the rate of change of these signals reflects the growth of internal resistance, a clear physical indicator of degradation. In both cases, the engineered features do not replace learned features but act as a guide that directs the model toward relevant structures.

2.4.3 Frequency-domain representation

The third technique shifts the signal out of the time domain entirely. A diagnostic analysis of the battery data revealed that the main axis of variation in time-domain windows was not degradation but the identity of individual cells: windows clustered by battery. As a response, a time-frequency representation was adopted based on the idea that the short-time Fourier transform and the continuous wavelet transform reveals the cyclic structure of charge and discharge in a way that is less tied to a specific cell. This approach identifies the main nuisance variable first and then chooses the representation to minimize it. This reasoning underlies the spectrogram autoencoder discussed in Chapter 5 and can be applied beyond batteries.

2.5 Stage 3: Compact model design

The third stage selects and configures the network. The thesis uses a toolbox of architectures across the three use cases, choosing from it based on the signal and constraints. The common preference is for compact models since every target platform has limited resources.

2.5.1 The architectural toolbox

One-dimensional convolutional networks serve as the common starting point. They learn local filters directly from raw sequences and are inexpensive to evaluate, making them a standard tool for sensor-signal tasks [4]. All three use cases utilize them, either as the entire model or as a front end.

Recurrent networks, in gated recurrent unit and long short-term memory forms, manage longer dependencies that a fixed convolutional receptive field might miss. Bidirectional variants read a sequence forwards and backwards, ensuring that each timestep is understood

with complete context. This approach is beneficial for signals like battery cycles, where both the transition into and out of a charge phase carry important information. A temporal attention layer is added on top of the recurrent stack so the model can weigh informative timesteps, which often forgets the beginning of a long sequence. This general shift from handcrafted features to convolutional and recurrent models, and finally to attention, is also seen in the time-series classification literature [14].

Attention-based models complete the toolbox. The transformer substitutes recurrence with self-attention, removing the sequential limitation of recurrent networks [15]. However, the full self-attention costs increase with the square of sequence length. This is impractical for the lengthy windows used here. The patch-based transformer from recent forecasting work avoids this by attending to short patches of the sequence taken by a convolutional front end instead of individual timesteps [16]. This method is adopted for the battery comparison, appearing both on its own and in combination with a convolutional encoder.

Autoencoders come into play where supervision is the main challenge. An encoder trained to reconstruct its input learns a latent representation without needing labels, which can then be explored for the target. The battery study uses this technique to learn the manifold of healthy-cell spectrograms and treats reconstruction error as a signal linked to degradation; further details will be provided in Chapter 5.

2.5.2 Compactness as a primary constraint

Model size is considered a design constraint from the very beginning, not an afterthought applied during deployment. The clearest example is the quantized road-surface network, weights and activations use low-precision integers instead of floating-point values. Quantization decreases memory usage and the energy cost per inference. Quantization-aware training minimizes accuracy loss by exposing the network to rounding behavior during training instead of only after that [17], [18]. Pushing this idea to its limit results in binary networks, where weights and activations reduce to a single bit [18]. The published road-surface model [8] achieves a footprint of 354 KB at this stage, small enough to fit in a smartphone or an in-vehicle unit. The battery and driver-monitoring models are designed with the same quantization approach in mind, even if it has not yet been implemented.

2.6 Stage 4: Systematic validation and failure analysis

The fourth stage is where much of the main contribution of this thesis lies. A model that produces reasonable numbers is not the same as a model that works. In situations where safety is critical, the difference between the two can be dangerous. The validation method

here has two parts: a way to split and check data so that reported scores have meaning, and a practice of viewing silent failures as findings to investigate.

2.6.1 Leakage-aware data splits

The most important rule involves how data is divided into training, validation, and test sets. In the battery study, the split is made at the level of whole cells. Every window from a given battery goes into exactly one of the three sets. If we split at the window level, the model could see one window of a cell in training and a neighboring window of the same cell during testing. This would inflate scores by measuring memorization instead of generalization. This is a known issue in the battery learning literature. The test set is limited to a minimum number of distinct cells to ensure that we measure cross-cell generalization. The same principle applies in the driver-monitoring case, where generalization to unseen drivers is what really matters.

Normalization follows the same logic. Standardization statistics are calculated using only the training split and are then applied unchanged to the validation and test data. This ensures that no information about the held-out data reaches the model through the scaler. Refitting the scaler on the full dataset is a subtler form of the same leakage that a cell-level split aims to prevent.

2.6.2 Silent failures as findings

A common issue across the use cases is silent failure. This is a model that runs without errors, produces confident output, and is wrong in a way that does not trigger any exceptions. In the driver-monitoring pipeline, it began predicting that drivers were attentive to every input once a single categorical feature was incorrectly fed to it. This failure was only noticeable because the predictions never changed. The battery models entered a phase where they predicted the dataset mean for every cell. This led to a loss that decreased and then plateaued, which could be mistaken for convergence. Changing architecture does not fix either of these problems. Both issues can be diagnosed by monitoring the pipeline, replaying recorded inputs offline, and checking intermediate values against their expected outcomes.

The thesis documents these failures and their underlying bugs as results. The driver-monitoring audit alone identified fourteen distinct defects related to parsing, type handling, time zone calculations, and feature construction. Each defect had the potential to corrupt the model's input without raising an error. Reporting these defects reflects a methodological stance. In a safety-critical pipeline, maintaining a list of ways the system can fail silently is part of what this work has learned.

2.7 Stage 5: Edge-oriented optimization and deployment

The final stage takes a validated model and moves it to the hardware it will run on. This step is driven by the essential idea of edge computing: processing information directly on the device avoids delays, reliance on connectivity, and risks to privacy from sending raw sensor data to a remote server [19]. For systems that need to respond while a vehicle is in motion, sending data to the cloud is not feasible.

At this stage, optimization reuses the compression techniques from Stage 3 but focuses on real hardware limits instead of just design choices. The key factors are inference latency, memory usage, and energy consumption per inference. The relevant question is whether each of these stays within the limits set by the application [9], [10]. The road-surface model is the most developed so far, with a measured memory use and a goal of being deployable on smartphone-class devices. The battery and driver-monitoring models are still earlier along this path; they are designed to be small, and their inference processes run continuously. Quantization and on-device testing have been identified as the next necessary steps for each model.

Real-time performance also relies on the surrounding process, not just the model itself. The driver-monitoring system performs its inference at a steady rate over a sliding window and must remain stable despite timing variations from two asynchronous network streams. An early version checked the coverage of its input window using the actual clock time of the inference, which led to unstable checks under real-time delays. Fixing this issue was essential before the system could provide reliable predictions, and it is the type of problem that arises only when a model is integrated into a live process.

2.8 Uncertainty quantification as a forward-looking dimension

A point prediction alone is not enough for a safety-critical decision. A battery health estimate of 81% is useful if it comes with a tight confidence interval, but it can be misleading if the true uncertainty is wide. This applies to distraction alerts and road-surface labels as well. Modern neural networks typically lack proper calibration, often reporting confidence levels that do not match their actual accuracy [20]. Therefore, calibrated uncertainty needs to be designed into the system.

Two methods are mainly used in practice because they work well with regular deep networks without needing a full Bayesian approach. Monte Carlo dropout keeps dropout active during inference and uses the variation from repeated forward passes to estimate model uncertainty [16]. Deep ensembles train several models with different starting points and use the range of their predictions for the same goal. Both methods are discussed, along

with the difference between aleatoric uncertainty, which comes from noise in the data, and epistemic uncertainty, which arises from the model's limits. Recent reviews in the field cover this topic [21]. Both methods have also been specifically studied for the flow of uncertainty from sensors to decisions in driving applications [7].

This thesis does not provide calibrated uncertainty across all three use cases, and it is upfront about that. Instead, it ensures that the architecture remains compatible with these methods. Dropout layers are included, and the models are small enough to make ensembling practical. This way, uncertainty estimation can be incorporated without major redesign. Therefore, uncertainty quantification is identified as the next key step in the methodology. This is mentioned in the future-work discussions of each use-case chapter and in the conclusion.

2.9 Use cases

The three chapters that follow use the same five stages for different signals, targets, and constraints. Table 2.1 summarizes how each use case addresses the stages. This reveals both the common groundwork and the differences. The road-surface study is the most complete concerning edge deployment and has already been published [8]. The driver-monitoring study requires the most effort during the fusion and live-validation stages. The battery study fully utilizes the architectural toolbox and contributes the spectrogram autoencoder. When considered together, these chapters support one main idea: a single, systematic approach to compact time-series deep learning is effective for different safety-critical problems.

Pipeline stage	Road surface (Ch. 3)	Driver Monitoring (Ch. 4)	Battery SoH (Ch. 5)
Acquisition & fusion	Single accelerometer, RMS of tri-axial signal	Two async UDP streams fused by timestamp into a common grid	Single multi-channel file; SoC dropped to avoid leakage
Representation	Raw window + 12 statistical features	120-step resampled window, 35 features, one-hot device ID	Cycle-length windows, mean-SoH label, STFT/CWT spectrograms

Model	Dual-branch quantized 1D-CNN	Pre-trained 1D-CNN inference loop	CNN, BiGRU/LSTM, hybrids, transformer, spectrogram autoencoder
Validation	5-fold cross-validation vs. CID-LCSS baseline	Pipeline audit (14 bugs), offline replay, live testing	Cell-level splits, mean-label fix, 25+ data-integrity checks
Edge / real-time	354 KB, smartphone-ready (done)	Live 5 Hz inference on the simulator (running)	Compact by design; quantization of a future target
Uncertainty	Compatible; not yet applied	Identified as next step for alerts	MC dropout / ensembles planned

Table 2.1 How the three use cases instantiate the five-stage methodology, with uncertainty quantification as the shared forward-looking dimension.

3 Road Surface Classification Via a Dual-Branch Quantized Neural Network

3.1 Author's contribution

The work described in this chapter was carried out by the team and published as a co-authored paper at ApplePies 2025 [8], on which the author is the first author. In this collaboration, the author's specific contributions were the design and implementation of the dual-branch quantized architecture, the data augmentation and statistical-feature pipeline, the Bayesian hyperparameter optimization, and the cross-validation experiments and class-wise analysis. The chapter presents this collaborative work while focusing on these contributions.

3.2 Introduction and motivation

The condition and type of road surface impacts driving safety, vehicle wear, and ride comfort. Knowing which surface a vehicle is on is essential for various intelligent transportation functions, from adaptive suspension control to maintenance planning for road authorities. Traditional pavement assessment depends on manual inspection or survey vehicles. Both methods are slow, costly, and impractical for cities or national road networks. This has led to a shift toward automated methods that analyze road data from vibrations collected by sensors already found in everyday vehicles, particularly the accelerometer in smartphones.

This use case represents the first application of the methodology described in Chapter 2, and it is the most developed towards edge deployment. The five stages align closely with this approach. Acquisition simplifies the tri-axial accelerometer signal into a single vibration channel. Representation engineering combines the raw signal with crafted statistical descriptors and balances the classes through augmentation. Model design creates a compact two-branch network with quantized convolutional layers. Validation uses five-fold cross-validation against a published baseline. Edge optimization is achieved through quantization-aware training, resulting in a model size of 354 KB. The work in this chapter was presented at ApplePies 2025 [8]; the chapter adds more methodological detail expected in a thesis.

Two common limitations in earlier work on this problem are computational cost and underutilization of complementary signal representations. Many accurate models are too large or consume too much power, making them unsuitable for devices that collect data. This reliance on cloud processing is not ideal for functions designed to operate

continuously in moving vehicles. Additionally, methods often depend on either the raw waveform or crafted features, rarely using both, leaving part of the signal's structure untapped. This use case provides a lightweight model that solves both issues, merging a raw-signal convolutional branch with a statistical-feature branch in a single quantized network that can classify three surface types while remaining small enough for a smartphone.

3.3 Related work

Research on road-surface classification using vehicle-mounted sensors has progressed through three main stages: traditional machine learning with crafted features, deep learning on raw signals, and the recent shift towards compressed models designed for embedded execution. This section reviews each stage and compares the proposed network to current methods.

3.3.1 Classical machine learning for pavement monitoring

Early systems collected statistical measures from accelerometer and gyroscope signals and used them with standard classifiers. Features like root-mean-square level, skewness, and kurtosis were paired with support vector machines or decision trees to distinguish surface types and identify anomalies [22]. Smartphone-based systems followed this model. RoadSense utilized accelerometer and gyroscope data with simple classifiers to estimate road conditions [23]. Later efforts continued to refine crafted features for low-end devices [24]. The most significant finding in this area is the method developed by Souza, which represented signals using the Complexity Invariant Distance and Longest Common Subsequence similarity, known as CID-LCSS. This approach achieved strong accuracy with smartphone accelerometer data [5]. CID-LCSS serves as the benchmark for the current work. These methods perform well in controlled environments but struggle to generalize across different vehicles, mounting positions, and devices. Their reliance on similarity search or extensive feature sets makes them hard to implement in real time on limited hardware.

3.3.2 Deep learning approaches for road classification

Deep learning eliminated the need for manual feature design. One-dimensional convolutional networks automatically learn temporal filters from raw vibration signals and have become standard tools for classifying sensor signals in various fields [13]. In road-surface classification, Hnoohom and colleagues assessed several deep architectures and noted consistent improvements over feature-engineered methods [25]. Dózsa and colleagues classified road types based on time-frequency representations of tire-sensor

signals [26]. Hybrid convolutional-recurrent models and tire-pavement acoustic methods have also been tested [5]. The same convolutional components apply to nearby embedded sensing tasks, such as classifying tennis strokes using inertial signals on a microcontroller [27]. These models enhance accuracy and eliminate the burden of feature engineering, but they tend to be heavier than traditional methods. Most studies report them without considering the memory and energy limitations of the target devices.

3.3.3 Quantized and binary neural networks for edge deployment

The third phase focuses on deployment. Quantization reduces the precision of weights and activations from 32-bit floating point to low-bit integers, reducing model size and lowering the energy cost for each inference. When trained with quantization included, the accuracy loss is typically minor [17]. The same concept applied to a single bit per parameter leads to binary neural networks, which have been used for real-time structural health monitoring on edge devices [18], [28]. A recent review covers the broader efforts to apply deep learning to road-condition monitoring under embedded constraints [29]. The model proposed here fits into this phase: it uses quantization-aware training for its convolutional and dense layers while keeping the first convolution at full precision. This approach maintains accuracy on the raw signal while still providing a compact outcome.

3.4 Dataset and preprocessing

3.4.1 The AsphaltPavementType dataset

Experiments use the public AsphaltPavementType dataset [30]. It contains time-series acceleration signals recorded from smartphones mounted on vehicles driven over three surface types: flexible pavement, cobblestone, and dirt. These three classes are the targets for classification. Each sample is a one-dimensional sequence that holds the root-mean-square magnitude of the tri-axial accelerometer readings. The data is sampled at 100 Hz. Given the three axis components a_x, a_y, and a_z, the magnitude at each timestep is

$$a\_RMS(t) = sqrt(a\_x(t)^2 + a\_y(t)^2 + a\_z(t)^2) \quad (3.1)$$

This formula combines orientation-dependent tri-axial data into a single channel, which captures vibration intensity regardless of the phone's mounting position. Sequences vary in length; therefore, all are either trimmed or padded to a fixed length of 2,371 samples to provide uniform input to the network. A sample of these 3 classes is shown in Figure 3.2.

Figure 3.1 Representative samples of the three target classes: flexible pavement, cobblestone, and dirt.

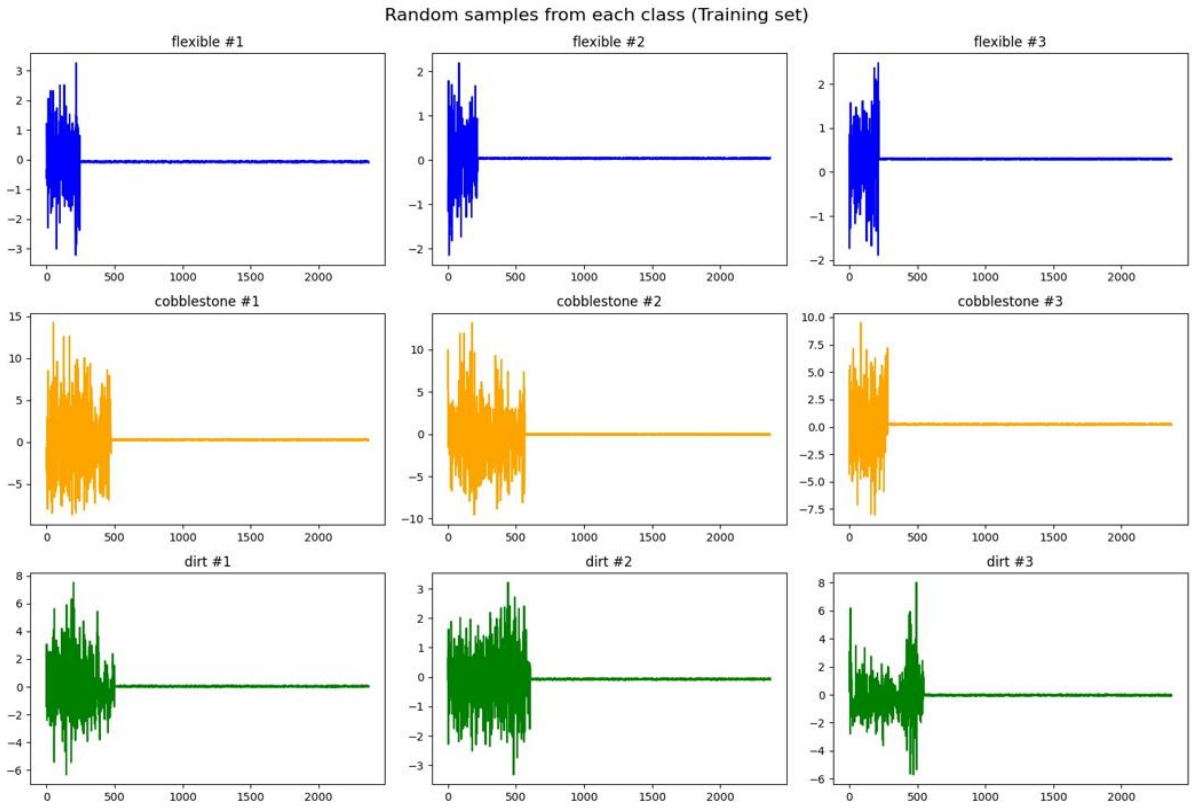


Figure 3.2 A sample of 3 classes signals randomly selected from the dataset

3.4.2 Data augmentation strategy

The dataset is imbalanced. Before augmentation, the class counts are 408 samples for flexible pavement, 264 for cobblestone, and 384 for dirt. This spread is large enough to bias training toward the majority class and weaken generalization on the minority class. To

balance the classes, each category was augmented to 500 samples. Transformations were chosen to mimic realistic variations in driving conditions while maintaining the structures that distinguish the surfaces:

- Jittering: adding small random noise to the signal to simulate sensor noise and minor vibration differences.
- Scaling: multiplying the amplitude by a factor close to one to model differences in vehicle mass, suspension, and phone mounting.
- Time warping: applying minor local stretching and compression along the time axis to represent changes in vehicle speed.
- Magnitude warping: smoothing non-linear deformations of the amplitude envelope to reproduce natural variations in surface texture along a route.

All sequences are standardized before entering the network. For each signal, the per-sample value x is transformed to a z-score using the training-set mean μ and standard deviation σ,

$$z = (x - \mu) / \sigma \quad (3.2)$$

This process ensures that the convolutional layers receive inputs on a common scale, which stabilizes training and prepares for low-bit quantization later in the pipeline.

3.4.3 Feature engineering: the statistical branch

In addition to the raw signal, twelve statistical descriptors are calculated from each sequence and sent to a separate branch of the network. These descriptors provide a compact summary of the signal's distribution and shape, which complement the local patterns that the convolutional filters capture. The descriptors, grouped by what they represent, are listed in Table 3.1.

Category	Features	What it captures
Central tendency	Mean, median	Average and central vibration level
Dispersion	Std. dev., min, max, 25th, 75th percentile	Spread, extremes, and quantiles of the signal
Distribution shape	Skewness, kurtosis	Asymmetry and tail/peak behavior

Energy	Sum of squares	Overall vibration strength
Signal complexity	Zero-crossing rate	Frequency of sign changes

Table 3.1 The twelve statistical descriptors form the input to the statistical feature branch.

These features are standardized in the same way as the raw signal before being sent to the branch. The idea is that raw and statistical views complement each other. The convolutional branch is sensitive to local shapes of vibration events, while the statistical branch encodes global properties of the distribution that a short convolutional receptive field does not directly observe.

3.5 Model architecture

3.5.1 Dual-branch design rationale

The model is a dual-branch network created with the Keras functional API, as shown in Figure 3.3. One branch processes the raw vibration sequence through convolutional layers. The other branch handles the twelve statistical features using a dense layer. These outputs are combined and passed through a shared dense layer before a softmax classifier makes the final decision across the three classes. This design directly follows the finding in related work that raw-signal and feature-based methods capture only parts of the relevant structure. By retaining both branches and merging them later, the network can make use of local waveform patterns and global statistics at the same time instead of relying on just one representation.

Figure 3.3 The dual-branch quantized architecture. The raw-signal branch (top) applies a full-precision convolution followed by two quantized convolutions. The statistical branch (bottom) uses a quantized dense layer. The outputs are merged and classified by a quantized dense layer and softmax.

3.5.2 Convolutional branch

The raw-signal branch starts with a full-precision one-dimensional convolution using 24 filters with a kernel size of 5 and a ReLU activation, followed by max pooling. Keeping the first layer at full precision allows the network to create an accurate initial representation of the raw waveform before losing any precision. Two quantized convolutions follow, each with 12 filters and a kernel size of 5. These convolutions do not use bias and are followed by max pooling and dropout at a rate of 0.3. Pooling gradually reduces the temporal dimension, giving the deeper filters a wider effective receptive field, while dropout helps prevent overfitting on the augmented data. The branch output is flattened before merging.

3.5.3 Statistical feature branch

The statistical branch is intentionally simple. The twelve descriptors go through a single quantized dense layer with 8 units using a hyperbolic-tangent activation, which transforms the features into a format compatible with the convolutional branch output. A smaller branch is suitable here because the input is already a compact and informative summary. A larger sub-network would increase parameters without any corresponding benefits and would go against the goal of keeping the model small.

3.5.4 Quantization-aware training

Quantization occurs during training instead of being a post-processing step. In quantization-aware training, the forward pass simulates rounding weights and activations to low precision. This allows the network to learn parameters that stay accurate when quantized, while gradients flow through a continuous approximation during the backward pass. This approach avoids the accuracy loss often seen with naive post-training quantization. The convolutional layers after the first, the merged dense layer, and the feature-branch dense layer are all quantized; only the initial convolution and the final softmax remain fully precise. The result is a network that occupies 354 KB, small enough to operate on a smartphone or an in-vehicle unit without requiring additional computation, making it suitable for the intended deployment.

3.6 Hyperparameter optimization

3.6.1 Bayesian optimization with Keras Tuner

The architecture shows several hyperparameters that interact in their values. These include the number of filters in each convolution, the dropout rate, the size of the dense layers, the learning rate, and the setup of the statistical branch. Instead of adjusting these manually or using a full grid search, we used Bayesian optimization with the Keras Tuner library. Bayesian optimization creates a probabilistic model to understand how hyperparameters

relate to validation performance. It then uses this model to determine the next configuration to test, focusing the search on promising areas. This approach finds a good configuration in fewer trials than grid or random searches.

3.6.2 Search space and best configuration

Table 3.2 lists the search range for each hyperparameter along with the value chosen by the optimizer. The search showed that increasing the width of the first convolution and including the statistical branch both improve results. It found a moderate dropout rate and a small learning rate to be effective.

Hyperparameter	Search range	Best value
Conv1D filters	{8, 16, 24}	24
QConv1D filters (layers 1, 2)	{4, 8, 12}	12
Dropout (conv and statistical)	[0.3, 0.5]	0.3
QDense units	{4, 8, 12, 16}	4
Learning rate	[1e-5, 1e-3]	0.0004
Use statistical dense layer	{true, false}	true
Statistical dense units	{8, 12}	8
Statistical activation	{ReLU, Tanh, ELU}	Tanh

Table 3.2 Hyperparameter search ranges and the values selected by Bayesian optimization

3.7 Experimental setup and evaluation protocol

3.7.1 Five-fold cross-validation

The model is evaluated using stratified five-fold cross-validation. The dataset is divided into five folds of roughly equal size. In each round, four folds train the model while the fifth fold tests it. This process repeats until every fold has been used as the test set once. The five scores are then averaged. Cross-validation provides a more reliable estimate of how well the model will perform than a single train-test split. Each sample contributes to

both training and testing across the rounds, so the impact of any one good or bad split is balanced out. The choice of five folds instead of ten is intentional. With this dataset size, ten folds would yield test sets that are too small for stable scoring. Five folds allow about three hundred samples per test fold while still using eighty percent of the data for training in each round.

Two precautions safeguard the process against the leakage mentioned in Chapter 2. First, the folds are stratified to maintain the class proportions of the entire dataset, ensuring no fold is dominated by one surface type. Second, data augmentation is applied only within the training folds of each round, never to the held-out test fold. If augmentation occurred before splitting, a jittered or time-warped version of a test signal could appear in training. This would inflate the score by rewarding the model for recognizing near-duplicates instead of for generalizing. Keeping augmentation after the split ensures that every reported number reflects performance on signals the model has never encountered before.

3.7.2 Evaluation metrics

Performance is reported using accuracy, precision, recall, and the macro-averaged F1-score, along with the stored model size as the relevant cost for deployment. Accuracy alone can be misleading for a multi-class problem. A model can achieve high accuracy by performing well on an easier or more common class while struggling with a harder one. Precision and recall reveal this behavior for each class. For a given class, precision is the proportion of predicted samples that actually belong to it, while recall is the proportion of actual samples that the model correctly identifies. In terms of true positives, false positives, and false negatives:

$$precision = TP / (TP + FP) \quad (3.3)$$

$$recall = TP / (TP + FN) \quad (3.4)$$

The two metrics trade off against each other. A model can increase precision by predicting a class only when it is certain, which may cause it to miss harder cases and lower recall, or it can do the opposite. The F1-score combines these metrics into a single number through their harmonic mean, which remains low unless both values are high:

$$F1 = 2 \cdot (precision \cdot recall) / (precision + recall) \quad (3.5)$$

Since the three classes were intentionally balanced, the macro average is the appropriate way to combine their scores. The F1-score is calculated separately for each surface type, and the results are averaged equally, ensuring that strong performance on the minority class is given as much weight as on the majority class. A micro average, on the other hand,

would weigh classes by their sample counts, undermining the balance. The confusion matrix and the receiver operating characteristic (ROC) curves provide a detailed view for each class. The confusion matrix shows which surfaces are confused with others. The area under each ROC curve summarizes how well the model distinguishes one class from others across all decision thresholds. A value of one indicates perfect separation, while 0.5 signifies chance.

3.7.3 Implementation and training details

The network is built using the Keras functional API, and its quantized layers utilize the QKeras extension. This extension offers quantized versions of standard convolutional and dense layers and simulates low-precision arithmetic during training. The final layer generates class probabilities using a softmax function over the three output units:

$$softmax(z)\_i = exp(z\_i) / \Sigma\_j\ exp(z\_j) \quad (3.6)$$

The model is trained to minimize the categorical cross-entropy between the predicted probabilities and the one-hot class labels:

$$L = -\ \Sigma\_i\ y\_i \cdot log(\hat{y}\_i) \quad (3.7)$$

Here, y represents the true label and ŷ the predicted distribution. The Adam optimizer is used for optimization, with a learning rate of 0.0004 selected during hyperparameter search. Training lasts for several hundred epochs, with learning curves shown in Section 3.7 indicating convergence well within that timeframe. The same setup, established before cross-validation, is applied in every fold. This ensures that the five rounds differ only in their data splits, making comparisons across folds straightforward.

3.8 Results and analysis

3.8.1 Cross-validation performance

Averaged over the five folds, the model achieves an accuracy of 93.9% and a macro-averaged F1-score of 93.9%. It has a precision of 94.1% and a recall of 93.9%. The fact that accuracy and macro-F1 are the same is significant. It suggests that the model performs consistently across the three balanced classes. A large difference between strong and weak classes would lower the macro-F1 score below the accuracy. The close values of precision and recall tell a similar story, indicating that the model does not over-predict or under-predict any class in a way that would bias one metric against the other.

The results remain stable across folds. The fold chosen for detailed analysis had validation performance closest to the five-fold average. This means that the curves and per-class figures presented here are representative (this is not the best case). The learning curves in Figure 3.3 indicate that both training and validation accuracy increase together and stabilize at a similar level. Validation loss tracks training loss instead of diverging upward. The lack of a growing gap between training and validation metrics shows that the model has not overfitted. The combination of dropout in the convolutional branch and the regularizing effect of quantization-controlled overfitting on the augmented training set, without needing additional measures like weight decay or early stopping.

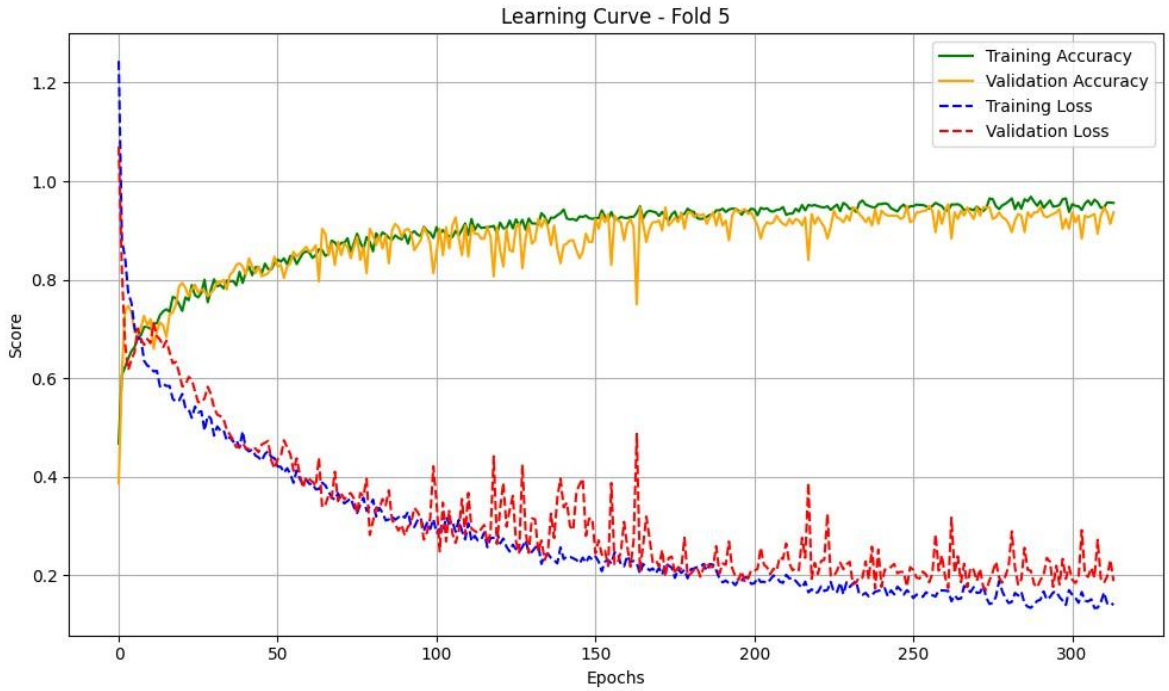


Figure 3.4 Training and validation accuracy and loss for the representative fold.

3.8.2 Comparison with the state of the art

Table 3.3 compares the proposed network to the CID-LCSS method of Souza [5], the established baseline on this dataset, evaluated using the same protocol. The network outperforms the baseline in every classification metric, with significant improvements where it matters most. Precision rises from 86.3% to 94.1%, and recall increases from 87.0% to 93.9%. This gain is not isolated to one metric; it reflects a model that is both more selective and more complete in its predictions.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Size
CID-LCSS [5]	87.6	86.3	87.0	86.7	≈ 45.7 MB
Proposed dual-branch QNN	93.9	94.1	93.9	93.9	354 KB

Table 3.3 Comparison with the state-of-the-art CID-LCSS baseline on the AsphaltPavementType dataset. The baseline size is estimated from its k-nearest-neighbor component.

The difference in memory usage is even more significant and arises from how the two methods operate. CID-LCSS relies on similarity: to classify a new signal, it compares that signal against stored reference instances using a complexity-invariant distance. Therefore, the model needs to keep a large part of the training set. Its footprint is about 45.7 MB, estimated from its k-nearest-neighbor component, and it increases with the amount of training data while remaining constant no matter how the method is adjusted. In contrast, the proposed network compresses what it has learned into a fixed set of quantized weights, resulting in a footprint of 354 KB that does not depend on the training set size. This leads to roughly a hundredfold reduction in stored size. The structural difference also affects inference cost: a similarity search compares each query against many stored instances, while a neural forward pass follows a fixed sequence of operations whose cost does not depend on the training set. This property makes continuous on-device inference practical.

The accuracy gain and size reduction are most important together. Each one alone would offer limited benefits for the stated goal. A more accurate model that cannot fit on the collecting device would not meet the edge-deployment requirement. Likewise, a small model that sacrifices accuracy would not justify replacing an established baseline. Achieving both simultaneously makes this result a real advancement for on-device pavement classification.

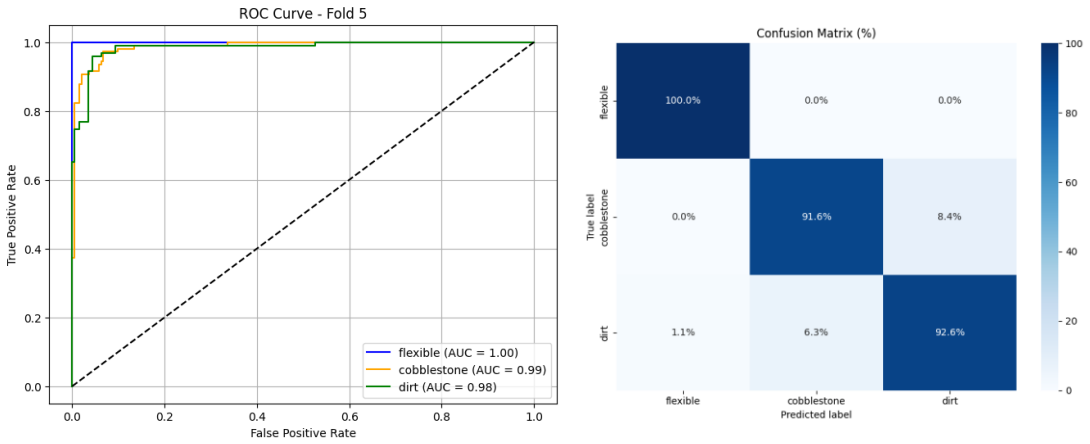


Figure 3.5 Per-class analysis: confusion matrix and ROC

The overall metrics obscure where the model's remaining errors occur, which the per-class view in Figure 3.4 highlights. The confusion matrix shows that the flexible-pavement class is classified almost perfectly, with very few flexible samples miscategorized as other classes. Cobblestone and dirt are correctly classified about nine times out of ten. The small remaining errors occur mainly between cobblestone and dirt, aligning with the physics of the signal. Both cobblestone and dirt are irregular, unpaved, or rough surfaces that create broadband vibrations with no dominant frequency. In contrast, flexible pavement generates a smoother and more distinct vibration signature, making it easier to differentiate from the other two. The model is not failing without reason; its few mistakes occur where the two surface types are most similar.

Class	Recall (%)	ROC AUC	Main confusion
Flexible pavement	≈ 100	≈ 1.00	none of note
Cobblestone	≈ 91.6	≈ 0.99	with dirt
Dirt	≈ 92.6	≈ 0.98	with cobblestone

Table 3.4 Approximate per-class recall and ROC AUC for the representative fold. Values are taken from the confusion matrix and ROC curves of the published analysis.

The ROC curves confirm this ordering. Each class is separated from the rest, with high area under the curve values: close to one for flexible pavement and around 0.98 to 0.99 for cobblestone and dirt. Since the area under the ROC curve measures separability across all decision thresholds, not just at a single point, these values indicate that the model's confidence scores are well ordered. Its strong average does not result from one easy class

compensating for weaker ones. All three classes are classified well, with only notable confusion occurring between the two most similar types.

3.9 Discussion

The results confirm the two design choices that shape this use case, linking back to the methodology in Chapter 2. The first choice involves representation. Combining a raw-signal convolutional branch with a statistical-feature branch creates a model that outperforms single-representation baselines. This supports the idea that the two views are complementary, not redundant. The convolutional branch captures the local shape of vibration events, while the statistical branch provides global distribution details that a short convolutional receptive field does not directly observe. The strong performance across classes suggests that the merged representation allows the classifier to distinguish between similar cobblestone and dirt surfaces most of the time.

The second choice involves deployment. Quantization during training, instead of as a post-processing step, keeps accuracy intact while reducing the model to 354 KB. The comparison with CID-LCSS shows that the edge-deployment stage of the methodology worked in practice, not just in theory. The hundredfold memory reduction is clear evidence that the network can run on smartphone-class hardware, which already collects the data. This setup eliminates the need for cloud processing, making continuous monitoring practical for similarity-based methods. In terms of the five-stage pipeline, this use case exemplifies implementation through the edge-optimization stage to a measured artifact.

Several limitations affect these conclusions. The dataset, although balanced through augmentation, is small in absolute size, and the three surface types are broad categories. Surfaces outside this classification or finer distinctions within it would further test the model. Augmentation expands the training distribution but cannot replace real variability across vehicles, mounting positions, and phone models, which is where similar systems often lose accuracy outside controlled environments. The evaluation is also offline. The footprint shows that the model is compact enough for on-device use, but we did not measure inference latency, sustained throughput, or energy consumption on real embedded hardware, which needs to be confirmed. None of these limitations undermine the main finding, but together they highlight the difference between what has been shown through testing and what has been established through design. They also set the direction for future work.

3.10 Conclusion and future work

This use case tackled two ongoing issues in road-surface classification using smartphone sensors: the high computational cost that prevents on-device execution and the typical dependency on either raw signals or handmade features, but not both. The suggested dual-branch quantized network solves both problems at once. It combines a convolutional raw-signal path with a statistical-feature path and quantizes the result during training. This approach achieves a macro-averaged F1-score of 93.9% while maintaining a stored size of 354 KB. It also surpasses the CID-LCSS baseline in all classification metrics while reducing memory usage by about 100 times. The analysis of each class shows even and sensible performance, with confusion primarily between the two most similar surfaces. Within the thesis, this is the most comprehensive demonstration of the methodology's edge-deployment phase and serves as the published anchor [8] against which the other two use cases are compared.

The immediate next step is to transition from a theoretical argument to a hardware measurement. Deploying the network on actual embedded targets, such as a smartphone-class processor or a microcontroller, and recording inference latency, sustained throughput, and energy per inference would transform the implied edge-readiness into a measured one and bridge the gap identified in the discussion. Binary-network variants, which have already been used in related structural health monitoring tasks on edge devices [28], are a suitable choice for further reducing the footprint and energy cost while maintaining precision that the dataset can likely handle.

Beyond deployment, three directions can expand this work. The first is to expand the surface classification beyond three broad categories to include the detection of road issues like potholes, cracks, and surface defects. This enhancement would make the system valuable for infrastructure maintenance as well as surface classification. The second direction, which aligns with the forward-thinking aspect of the methodology, is to add calibrated uncertainty to the output. This way, a low-confidence prediction on an unfamiliar surface would be flagged for review instead of presented as a confident label. The compact architecture is already compatible with the Monte Carlo dropout and ensembling methods discussed in Methodology. The third direction involves conducting a large validation campaign across various vehicles, mounting positions, and phone models. This is the only way to establish cross-device generalization, which offline cross-validation on a single dataset can only partially explore. Together, these steps would advance the use case from a published, reliable, and compact classifier to a ready-to-use monitoring component.

4 Real-Time Driver Monitoring for Distraction Detection

4.1 Author's contribution

This use case is different from the other two in how the work was divided, so the author's contribution is clear from the start. Other team members trained the classification model at the heart of the system, a one-dimensional convolutional network, on an offline dataset processed into the 35 features the model uses, as described in the associated publication [31]. This model is considered a given component here. The author's contribution is the real-time pipeline built around it. This system reads directly from two raw sensor and simulation data streams, decodes and combines them, transforms them into the feature window the model expects, and produces live predictions. The author designed and built this pipeline from scratch, and it did not exist in any form before this work.

Two additional components of the work belong to the author, and they are noted as such where they appear. The systematic audit that strengthened the pipeline, described in Section 4.5, was a team effort with the author as the main contributor. The author identified and corrected the defects highlighted by the audit. Separately, and still ongoing, the author is collaborating with other team members to develop improved models that better handle the unique differences between individual drivers. This effort has not yet produced strong results and is noted as ongoing in the future work. Throughout this chapter, the driving-simulator data collection was conducted at the VEDECOM Institute (MobiLAB, Versailles, France) by Dr. Gaëtan Merlhiot and colleagues, and the results are reported here with their consent.

4.2 Introduction and motivation

Driver distraction is one of the top causes of road crashes [32]. A system that can determine in real time whether a driver is focusing on the road has significant value for both research and active safety applications. The challenge is that attention is an internal state without a direct sensor. It must be inferred from observable indicators, such as where the eyes are looking, how the eyelids move, the head's position and rotation, and how the driver operates the vehicle. A driver-monitoring system (DMS) combines these signals to provide a moment-by-moment estimate of attentional state.

This use case is the second implementation of the methodology from Methodology and emphasizes the acquisition and fusion stage. The data does not arrive neatly. It comes from two independent devices that send information over the network at different rates and in different formats, without a shared clock and with occasional gaps. The pipeline created by

the author takes in these two live streams, merges them into the fixed window the trained classifier needs, and generates predictions several times a second, all while running continuously inside a driving simulator. The starting point was a model that only worked on a prepared offline dataset. This work bridges the gap between a raw sensor on one side and that model on the other.

The contribution of this use case mainly lies in the systems aspect and has two parts. The first is the design and implementation of the real-time fusion pipeline. This involves decoding an undocumented binary protocol, parsing a text protocol, aligning the two on a common time grid, and providing input to the model under live timing constraints. The second part is the systematic audit of that pipeline, which uncovered and fixed fourteen defects, some of which could have corrupted the model's input without triggering any errors. A simulated live evaluation on held-out drivers revealed a performance gap relative to the offline results. In this evaluation, recorded data from two drivers unseen during training, validation, and testing were replayed as a raw stream through the deployed processing path, reproducing the data gaps and errors a real live feed would contain.

4.3 Related work

Research on driver monitoring covers sensing hardware, signals that best indicate attentional state, and the challenges of running inference in a vehicle in real time. This section reviews the three areas most relevant to the current system.

4.3.1 Multimodal driver monitoring systems

Early driver-state systems relied on a single type of input, usually a camera watching the face or a set of vehicle dynamics signals like steering and lane position. Single-input systems are easy to deploy but have limitations. A camera can be affected by lighting and obstructions, while vehicle dynamics signals can reflect the road and maneuvers as much as the driver. Combining different types of input improves reliability because one source's weaknesses are balanced by another's strengths [33], [34]. The system in this chapter follows this idea by combining a dedicated eye-tracking device with the vehicle dynamics signals produced by the simulator, allowing for a complete interpretation of gaze, head behavior, and vehicle operation.

4.3.2 Eye tracking and gaze in driver-state estimation

Among the various signals, those related to the eyes and head are the strongest indicators of visual attention [35]. These signals are informative but they are indirect measures of driver state. The direction of gaze is associated with where the driver is attending, eyelid behavior

and blink patterns relate to drowsiness, and head pose reflects gross shifts in attention such as turning away from the road. Pupil size is not a direct or specific indicator of cognitive load: only under controlled conditions (other factors such as luminance, looming, and sympathetic activation must held constant) can pupillary response serve as an indirect indicator. In a less controlled environment this underlines the importance of integrating behavioral measures with other biomarkers to improve prediction, instead of relying on any single signal as a definitive marker of an internal state [31], [36]. Specialized eye-tracking hardware measures these metrics at high rates and precision [37]. The challenge is integration complexity; these devices often communicate using proprietary binary protocols that lack public documentation (this is exactly the challenge the author encountered) motivating the decoding stage described in Section 4.5.

4.3.3 Edge inference for safety-critical monitoring

A driver monitoring system must operate while the vehicle is moving, so it cannot rely on a round trip to a remote server. Inference must happen on hardware located in or near the vehicle [19]. This edge constraint runs throughout the thesis and manifests here as a latency requirement in a live context. The relevant literature on compact convolutional models for sensor signals and real-time inference under timing variability informs the design, and the one-dimensional convolutional architecture used for the classifier is lightweight enough to be processed many times a second on modest hardware [13].

4.4 System architecture

4.4.1 Overview and design goals

The pipeline has three main tasks: receiving and decoding the two live streams, combining them into the fixed window the model needs, and running inference at a consistent rate. Figure 4.1 shows the complete data flow that the author implemented. Two listener threads independently receive the streams and write decoded feature dictionaries, each marked with a timestamp, into a single shared queue. A main loop processes the queue, maintains a rolling time window, resamples it onto a fixed grid, scales it, and sends it to the model, producing a prediction at a set rate. The design goals included continuous operation, handling irregular packet timing, and ensuring exact agreement between live preprocessing and the preprocessing used during training. Any differences between the two can quietly degrade predictions.

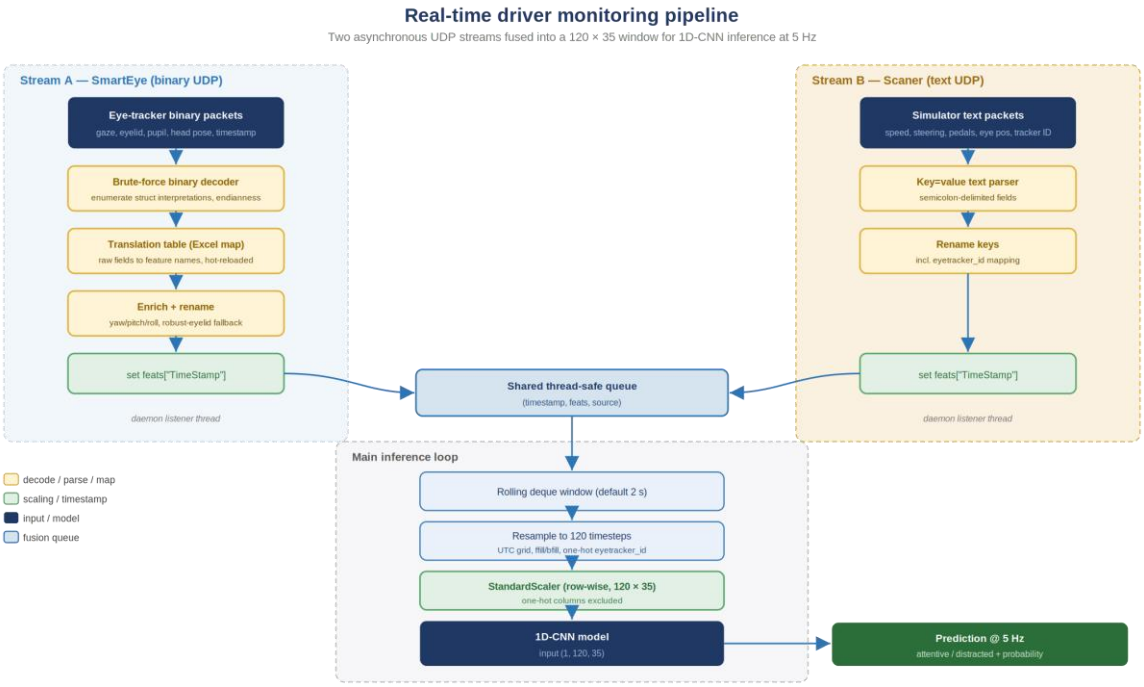


Figure 4.1 The real-time driver-monitoring pipeline built by the author. Two asynchronous UDP streams are decoded by separate listener threads, combined through a shared queue, resampled into a 120 x 35 window, scaled, and classified by the team's 1D-CNN at 5 Hz.

4.4.2 Eye-tracker binary stream

The driving simulator at VEDECOM is instrumented with a SmartEye eye-tracking system for gaze, eyelid, and head-pose signals, and a Biopac system for physiological measurements. Full details of the acquisition setup are given in [31]. The eye-tracking device sends a binary UDP stream that includes gaze direction, eyelid opening for the left, right, and both eyes, pupil diameter, head pose represented as a Rodrigues rotation vector, and hardware timestamp. The protocol is proprietary and its field layout is not documented, so the stream cannot be parsed by a fixed schema. The author developed a decoder that works through brute force: for each sub-packet in the payload, it tests every possible interpretation, examining integer and floating-point types of various widths, three-component vectors, and the eight-byte Rodrigues encoding in both big-endian and little-endian byte order. The recovery of human-readable feature names from the resulting raw fields is managed by a translation table in a spreadsheet. This table maps each decoded field to its corresponding model feature and is periodically reloaded for updates without needing to restart a live session.

4.4.3 Simulator text stream

The driving simulator sends a text UDP stream of semicolon-delimited key-value pairs. It includes vehicle dynamics like speed, acceleration, steering-wheel angle, steering speed, steering torque, and accelerator-pedal position, along with the eye position estimated by the simulator, the active eye-tracker unit ID, and optional physiological channels. Parsing involves splitting each pair at its delimiter and renaming the simulator's raw keys with the model's feature names. The eye-tracker identifier receives particular attention because it is one-hot encoded for the model, and as the results will show, the model is highly sensitive to it.

4.4.4 Threading model

The two streams are received by two daemon listener threads, one for each protocol. Each thread decodes its packets and writes tuples of timestamp, feature dictionary, and source into a single thread-safe queue. The main thread runs the inference loop and is the only consumer of the queue. This separation allows for the reception and processing to be decoupled, so that a burst or stall in one stream does not block the other or the model. Each listener is designed to log any exceptions encountered during the decoding of a packet and continue processing, instead of dying silently and starving the loop of data. This robustness feature, as discussed in Section 4.5, was added during the audit.

4.5 Implementation

4.5.1 Binary decoder and translation table

The brute-force decoder is the most complex component. Since the binary layout is unknown, the decoder considers various interpretations of each field and uses the translation table to pick those that match real signals. The table maps a decoded field, identified by its offset and interpreted type, to a named feature. It reloads at short intervals to ensure corrections made during a session take effect immediately. After decoding, the process includes two enrichment steps. The Rodrigues head-rotation vector is converted into yaw, pitch, and roll angles, which the model expects. When the model requires a stable version of an eyelid-opening signal that the device does not provide, the standard version is used as a fallback, ensuring the feature is populated instead of being left missing.

4.5.2 Feature fusion and the rolling window

Once both listeners are writing to the queue, the main loop combines their output into a single time series. On the simulator side, each record is emitted with the simulator's own timestamp, which the fusion stage uses as the reference clock; the eye-tracker stream is

aligned to it during resampling. It maintains a rolling window, using a double-ended queue, that holds the events from a short interval, typically two seconds. New events are happening as old ones fall back, keeping the window reflective of the recent past. Because the two streams merge by timestamp and not by any hardware sync signal, the window contains interleaved events from both sources at irregular intervals, which the next step normalizes.

4.5.3 Resampling and missing-value handling

The model requires an input of 120 timesteps by 35 features, so the irregular window needs to be resampled onto a regular grid of 120 points. Resampling uses a timezone-aware time index so that the live timestamps align with the grid. Missing values within the grid are filled by carrying the last valid observation forward and, for any gaps at the start, backward. A guard handles the edge case where the window consists of essentially a single instant in time: instead of duplicating one event across all 120 rows, the loop returns to a neutral all-zero window. Features that the model requires but are missing from the window are filled explicitly, and the missing names are logged, making any gaps in the input visible.

4.5.4 One-hot encoding and scaling

The eye-tracker identifier is one-hot encoded into a set of binary columns used in the model's training. These one-hot columns are filled with explicit zeros and ones (not averaged values) as a fractional entry would distort the model's understanding of which device is active. The remaining scalar features are standardized with a scaler fitted on the training data and saved for reuse. The scaler is applied per row, one call per timestep for the 120 by 35 window, which matches the procedure used during training; the one-hot columns are not scaled, again reflecting the training setup. If the scaler cannot be loaded, the event is logged instead of sending unscaled data to the model. Reproducing the training preprocessing exactly at inference time is critical. A mismatch or wrongly shaped input can lead to serious problems for a pipeline like this.

4.5.5 Inference loop

The scaled window is reshaped to match the model's expected batch format and passed to the one-dimensional convolutional network, which returns a probability that is thresholded for either attentive or distracted labeling. The loop issues a prediction at a steady rate, typically five hertz, regardless of how fast packets arrive. Before making a prediction, it checks two conditions: enough time has passed since startup to fill the window, and the data in the window is recent and not outdated. The model loading process is fault tolerant.

It first attempts to load the native format and then falls back to alternatives, logging each failure so that any loading problem is surfaced as a message instead of a crash.

4.6 Pipeline audit and systematic debugging

4.6.1 Audit methodology

The first version of the pipeline ran without crashing, but it produced questionable output. This is a dangerous situation in a safety-relevant system because it appears to function correctly. In response, a systematic audit was initiated, led by the author and the team. They traced every signal from when it entered a listener to when it reached the model. At each stage, they checked that values were of the correct type, within the expected range, and present at the right time. The audit treated a silent wrong answer as a more serious failure than a visible crash. A pipeline that fails visibly can be fixed, while one that fails quietly cannot be trusted. This process identified fourteen distinct defects, categorized by the kind of harm they caused (full list in A).

4.6.2 Findings and fixes

The defects fall into three categories. The first category involves thread survival and silent failures. The two listener threads had no exception handling. A single malformed packet, such as a head-rotation value that was not a clean float, would cause an error, kill the thread, and allow the main loop to run on stale or empty data without indicating that a stream had stopped. To fix this, each listener was wrapped to log packet-level exceptions while allowing the thread to continue, eliminating this class of failures.

The second category relates to the correctness of the data itself. The text parser split key-value pairs using the wrong character initially, corrupting any key with an underscore, like the velocity components. The scaler was applied to a flattened array rather than row by row, so the values reaching the model did not match the distribution it was trained on. The eye-tracker identifier was never renamed from the simulator's raw key, preventing it from reaching the one-hot encoder. During resampling, the one-hot columns were filled with their column mean instead of zeros and ones, leading to fractional indicator values. Each of these issues was corrected to align with the training procedure.

The third category focuses on visibility. A timezone-naive time index was used for resampling timezone-aware timestamps. The deprecated timestamp call was replaced with its timezone-aware equivalent throughout the system. The receive timestamp was included in the queue tuple but was never written into the feature dictionary. This is why the timestamp column appeared empty in every diagnostic export, even though the value was

available. Writing it into the dictionary in both listeners resolved the issue of the empty column. Silent failure points in scaler loading, model loading, and prediction were given explicit logging to ensure that faults would be reported. Table 4.1 summarizes the entire set.

Category	Representative defects	Effect before fix
Thread survival	No exception handling in listeners; unsafe numeric cast on head-rotation values	A bad packet silently killed a listener; loop ran on stale data
Data correctness	Parser split on wrong delimiter; scaler applied to flattened array; eye-tracker ID never renamed; one-hot columns mean-filled	Corrupted keys, wrong input distribution, and a missing feature reached the model
Visibility	Timezone-naive resampling; deprecated timestamp call; timestamp never written into feature dict; silent scaler/model/prediction failures	Faults and empty diagnostic columns went unnoticed

Table 4.1 Short list of the fourteen audited defects grouped by the kind of harm they caused. The full list of defects can be found in Appendix A.

4.6.3 Offline verification mode

To make debugging repeatable, the author added an offline replay mode. This mode reads recorded UDP packets from a file and sends them through the same listeners and processing path at a controlled rate. It emulates the simulator without requiring the simulator itself. Replay ensures that a run is reproducible since the same recorded input produces the same trace every time. It also allows the feature coverage of each frame to be reported and exported for inspection. This mode enabled the always-attentive behavior, discussed next, to be studied in a controlled environment instead of just being observed in a live session.

4.7 Experimental setup and results

4.7.1 Model and evaluation

The classifier is the team's one-dimensional convolutional network. It takes a 120 by 35 window and outputs the probability of distraction, with a threshold set at one half to produce a binary label. It was trained offline on a labeled dataset made from the same 35 features used during inference, divided into training, validation, and test partitions. Two settings are distinguished in the following sections. Offline evaluation measures the model on held-out data from the same recordings and driver population as the training set. Live evaluation assesses the assembled system, where the author's pipeline feeds the team's model in the simulator with drivers who were not part of the training data. The difference between these two settings is the main empirical result of this use case.

4.7.2 Offline performance

In the held-out test partition, the model achieves about 90% accuracy and an F1-score of about 91%. These figures confirm two things. First, the 35-feature representation has enough information to separate attentive from distracted states when the data resembles the training distribution. Second, the author's offline pipeline accurately reproduces the training preprocessing. A mismatch in preprocessing would have lowered even the offline score. Therefore, the offline results confirm that both the model and the feature pipeline are sound on familiar data.

4.7.3 Simulated live performance and the generalization gap

This evaluation was not a real-time live test on the simulator; a live deployment was not achieved. Instead, it is a simulated live test: the recorded data of two held-out drivers, unseen by the model during training, validation, and testing, were streamed through the deployed pipeline as raw input. This reproduced the missing samples, malformed packets, and timing irregularities that a real live feed produces, and the model itself was adapted for fast, edge-deployable inference, both of which lowered performance relative to the offline results. A further and important factor is that the evaluation reports result on only two completely unseen drivers: with this previously unseen sample, each driver's individual behavior, personal differences, and idiosyncratic errors weigh heavily on the aggregate figures, which contributes to the lower performance. Under these combined conditions, performance was around 80% accuracy and a 78% F1-score, compared with the offline results. Table 4.2 compares the two settings side by side.

Setting	Drivers	Accuracy	F1-score
Offline (held-out test)	seen distribution (Whole dataset)	~ 90%	~ 91%
Simulated Live	new, unseen drivers (2 held out)	~ 80%	~ 78%

Table 4.2 Offline versus simulated live performance of the driver-monitoring system.

The most noticeable symptom of the gap appeared in early simulated live runs as a tendency to predict attentive almost continuously. Controlled offline replay, using the author's verification mode, traced this behavior to the model's strong reliance on the eye-tracker identifier. When that feature was omitted in replay, the model predicted a single class. When it was included, the predictions varied as expected. The audit had already confirmed that the live pipeline delivers this feature correctly through the one-hot encoder, so a defect in the pipeline is ruled out. The behavior is a characteristic of the trained model, which relied heavily on an identifier that contains no attentional information and does not translate across recording setups. This represents a modeling weakness that should be addressed through retraining, not an issue in the data path, and it directly points to ongoing work by the author.

4.8 Discussion

This use case results in a functioning real-time fusion pipeline created by the author. It provides a clear assessment of the team’s current model shortcomings. On the systems side, the contribution is strong: two asynchronous streams in different formats are decoded, aligned, and fed to a classifier under live timing conditions. The systematic audit improved a pipeline that simply ran into one that functions correctly and reports its own errors. The offline accuracy of about 90% indicates that the combined pipeline and model perform as intended on familiar data.

The simulated live results reveal the more interesting aspects. The accuracy drops by ten points, and the F1 score drops by thirteen points between offline and simulated live testing, indicating a textbook generalization gap. The analysis identifies the cause: the model depends on a feature, the eye-tracker identifier, which correlates with attention in the training data but does not cause it and varies across setups. A model dependent on such a feature will show strong performance in cross-validation with its training data but will struggle with new drivers, which matches the observed pattern. Identifying this mechanism is more helpful than merely reporting the gap because it suggests a solution: reduce the

model's reliance on identity-like features and increase the diversity of drivers it learns from.

4.9 Conclusion and future work

This use case produced a full real-time driver-monitoring pipeline designed and built by the author from scratch. It fuses two asynchronous multimodal streams and classifies driver attention several times per second in a driving simulator. Building it involved decoding a binary eye-tracking protocol, parsing a text simulator protocol, aligning the two streams on a common time grid, and reproducing the training preprocessing accurately under live conditions. An audit, mainly conducted by the author, found and fixed fourteen issues related to thread survival, data correctness, and diagnostic visibility. This transformed a fragile first version into a reliable system that reports its own errors. The pipeline is now in its final testing and debugging phase. Using the team's model, the system achieves about 90% accuracy and 91% F1 on held-out data and around 80% accuracy and 78% F1 in the simulator with new drivers. Therefore, the pipeline is confirmed to be operational, and the remaining challenge is a modeling one: bridge the gap between offline and online performance so the system can generalize to drivers it has never encountered.

Future work stems directly from this finding and is the main focus of the author's ongoing efforts with other team members. The first step is to expand and diversify the data. Recording many more drivers of different ages, builds, and driving styles under various distraction scenarios provides the model with the variety it needs to learn attention instead of identity. This allows for retraining with the eye-tracker identifier removed or adjusted to prevent reliance on a non-causal cue. With a more diverse data set available, the next step is to develop and test stronger models. This includes designing architectures better suited to multimodal time series data and to the differences among individuals that a single convolutional network handles poorly. These models will be evaluated using a driver-independent protocol that measures generalization directly with new people. Efforts to improve the model are underway, but they have not yet yielded strong results. That is why this chapter presents the pipeline as a completed contribution, while the model remains an open issue.

The third step involves validating the enhanced system in increasingly realistic scenarios. First, there will be an expanded campaign in the simulator with the larger driver pool. Once that shows satisfactory performance, testing will move to real-world driving conditions instead of relying solely on simulation. Besides this core progression, there are two additional directions that extend the work in line with the thesis. One involves adding calibrated uncertainty to the output. This way, a low-confidence prediction for an unfamiliar driver will be flagged instead of asserted. This approach is aligned with the

forward-looking aspect mentioned in Chapter 2 and is supported by the compact classifier's design. The other direction is to implement the model on embedded hardware to measure its latency and power, ensuring that the real-time performance shown in software is confirmed on a typical in-vehicle target. Together, these steps will advance a validated pipeline with a known generalization gap toward a driver-monitoring system that works reliably with new drivers in real driving conditions.

5 Battery State-of-Health Estimation with Deep Learning

5.1 Author's contribution

This use case is a team effort within the ELIOS Lab, and we outline the division of work at the beginning. The dataset was gathered in a partly prepared form by a team member from a public source. This gave the author a solid starting point and eliminated some of the basic extraction work. The author developed the model, created the experimental pipeline, conducted hyperparameter studies, and performed the analysis reported in this chapter. Two team members collaborated closely on this work, contributing to the preprocessing direction and the design discussions that shaped the windowing and normalization strategy. The work is ongoing. The architecture and experimental protocol described here are set, although the results are still being refined. The two key areas mentioned in Section 5.11, cycle-aware windowing and per-battery normalization, are actively being developed as this is written.

The chapter is designed to be self-contained. It presents the problem, the dataset, the complete array of models that were built and fine-tuned, the validation protocol, and the analysis in enough detail that readers will find the use case fully developed within the thesis. If a result is not yet final, the text clearly states this and presents the findings in a way that allows for extension as the work progresses.

5.2 Introduction and motivation

Lithium-ion batteries are the leading energy-storage technology for electric vehicles, portable electronics, and large-scale storage. However, their performance decreases over time. The State-of-Health (SoH) of a battery measures how much of its original capacity is left. A reliable and continuously available estimate of SoH is essential for safe and efficient operation. A battery management system (BMS) that understands each cell's true condition can balance loads, schedule maintenance, decide when to retire or repurpose a pack, and alert users about conditions that could lead to thermal runaway. SoH estimation is critical for safety, and it represents the third application of the methodology outlined in Chapter 2, adapting the general pipeline to a problem where the hidden state is the underlying health of an electrochemical cell.

The challenge is that SoH is not directly measurable during normal operation. It is defined based on a complete reference-capacity test that takes the cell out of service. It must be inferred from the signals that a BMS records: the voltage across the cell, the current flowing through it, and its temperature. These signals reflect degradation but are subtle and

mixed with the operating conditions. The same cell at the same health can produce different voltage and temperature readings depending on how it is charged or used. In addition, two cells of the same age may show different health because of their varied usage. Estimating SoH from these signals requires extracting a slowly changing hidden quantity from noisy, condition-dependent, multivariate time series. This thesis addresses that specific challenge.

This use case develops a family of data-driven SoH estimators based on a large, diverse public dataset on aging. It compares these estimators under a strict protocol that tests how well they generalize to batteries the model has never encountered before. The findings are positioned against the current state of the art [38]. The contribution consists of three parts. First, this includes a complete and reproducible modeling pipeline that spans classical gradient boosting, convolutional networks, recurrent networks with attention, and convolutional-recurrent hybrids. Each model is fine-tuned through automated hyperparameter searches and validated on a cell-by-cell basis. Second, there is a straightforward description of the main tension in the results: the most accurate model achieves a competitive error but needs an impractically long input window, highlighting the deployment issue that this work addresses. Third, two specific directions are proposed, cycle-aware windowing and per-battery normalization, to tackle this tension and outline the next phase of the project.

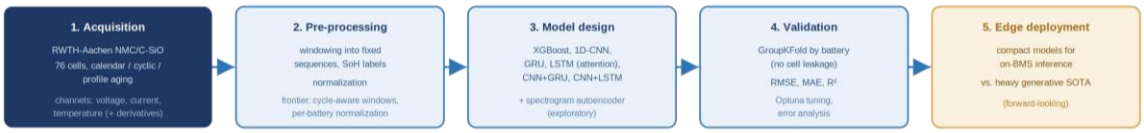


Figure 5.1 The five-stage methodology of Chapter 2 applied to battery State-of-Health estimation, from obtaining the RWTH-Aachen cell measurements to a State-of-Health regressor evaluated on held-out batteries.

5.3 Background: degradation and State-of-Health

5.3.1 Defining State-of-Health

State-of-Health is most often defined as a capacity ratio: the current usable capacity of the cell is divided by its capacity when new. We can write Q_now for the current capacity and Q_0 for the initial capacity,

$$SoH = (Q\_now / Q\_0) \times 100\% \quad (5.1)$$

This means that a new cell has an SoH of 100%, with the decreasing value as the cell ages. An alternative definition focuses on the increase in internal resistance or impedance, which

rises as the cell deteriorates. The two definitions capture related but distinct aspects of aging, with the capacity-based definition being used here because it is what a BMS most directly requires. The dataset used in this study expresses SoH on the capacity scale, with values in this chapter ranging from approximately 70% for heavily used cells to nearly 98% for cells early in their lifespan.

5.3.2 Degradation mechanisms

Capacity fades from several microscopic degradation processes working together. Three main families dominate [39]. The loss of lithium inventory happens as cyclable lithium is consumed through side reactions. The most significant of these is the ongoing growth of the solid-electrolyte interphase on the negative electrode. When charging aggressively or at low temperatures, lithium may even deposit as metal, this is referred to as lithium plating. The loss of active material occurs when electrode particles crack, dissolve, or lose their electrical contact. This removes sites that previously stored lithium. Additionally, the loss of electrolyte and the drying of the separator can increase impedance and limit the reaction. These degradation mechanisms interact and influence one another. Their impact can vary based on temperature, charge and discharge rates, and the state of charge at which the cell operates. This variability is why a data-driven model cannot view all cells as interchangeable. A cell that ages quickly under high heat will degrade based on a different mix of mechanisms than one that ages slowly in cooler conditions, resulting in different signal patterns.

Each degradation mechanism leaves a unique fingerprint on the measurable signals, which enables a data-driven model to detect them. The growth of the solid-electrolyte interphase consumes lithium and raises internal resistance. This alters the voltage response under load and affects the relationship between the charge passed and the voltage reached. Over many cycles, these changes become visible as a gradual shift in the shape of the charge curve. Lithium plating occurs when lithium deposits as metal. This is especially problematic under high charging rates or low temperatures because some of the plated lithium becomes electrically isolated and is lost permanently, while some can short-circuit the cell. The onset of lithium plating often aligns with a key change in the aging trajectory. The loss of active material occurs as particles break apart due to the repeated expansion and contraction during charge and discharge cycles, limiting the sites available for lithium storage and accelerating capacity fade. The silicon component of the negative electrode in the studied cells is particularly relevant because silicon expands and contracts more than graphite when it takes up and releases lithium. This characteristic makes mechanical degradation a more important factor, which must be learned.

These mechanisms are interrelated. The interphase growth that uses lithium also thickens the layer that lithium must pass through, increasing resistance and changing local conditions in a way that can encourage lithium plating. Mechanical cracking exposes fresh electrode surfaces, providing new areas for the interphase to grow and hastening lithium loss. This interaction creates a set of coupled self-reinforcing processes. Their balance changes throughout the life of the cell and depend on its operating conditions, leading to nonlinear aging. At times, one mechanism might dominate, causing the cell to age rapidly after a long period of gentle degradation. A model that estimates health from signals interprets the combined influences of these processes. The challenge is that the signs of degradation are faint early in a battery's life, becoming clearer only as the mechanisms compound, making it difficult to predict a cell's future health at an early stage.

5.3.3 The aging trajectory and its diversity

Visualizing the aging trajectory helps (Figure 5.2). Imagine a curve showing State-of-Health against time or the number of cycles a cell has gone through [40]. Early in life, this curve is nearly flat. The cell loses health slowly and almost linearly. At some point, the slope becomes steeper, and the loss accelerates. From there, the cell declines rapidly toward the end of its useful life. This curve's shape varies for different cells. Two cells of the same type, aged under different conditions, can show different declines. One may decline gently for a long time, while another collapses early, due to the different degradation mechanisms at work. Temperature is the strongest influence. Heat speeds upside reactions that consume lithium and damage the electrodes, making a cell aged in warm conditions decline faster than one in cooler conditions. The charge and discharge rates also matter. High rates cause lithium plating and mechanical stress, while the charge state during calendar aging affects how fast the interphase grows. The dataset in this chapter covers a range of conditions, making it a strict test. This is why a model learning the shape of degradation is so valuable.

The variety of trajectories is why estimating SoHis not straightforward. A model that learns the aging curve for cells aged at one temperature and rate does not automatically learn the curve for cells aged under different conditions. The underlying physics it captures varies. The signals reflect the specific mechanisms at play, and these mechanisms change with different conditions. An effective estimator must work across a real fleet of batteries, which are used in many ways. It must generalize across this physical diversity, not just interpolate within one operating range. This requirement is central to every methodological choice in this chapter, from the dataset's breadth to how cells are grouped for validation.

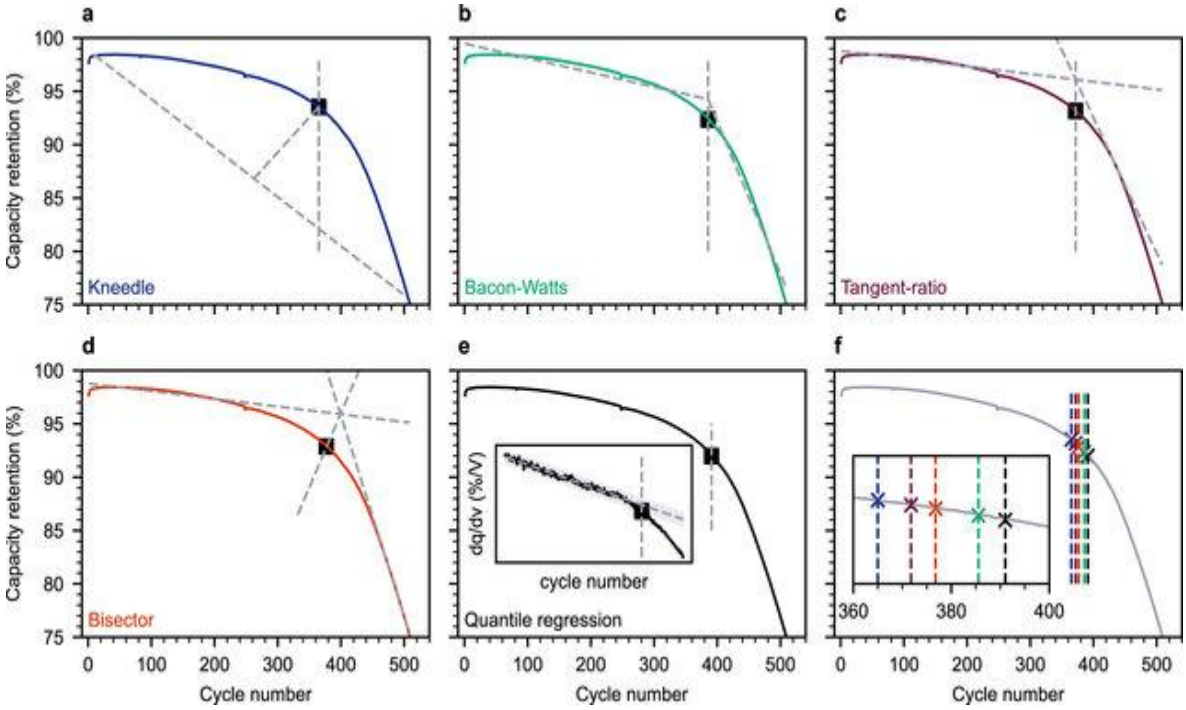


Figure 5.2 Results of various knee identification methods proposed in the literature illustrated on the batch 2, channel 12 cell from the Severson et al. 37 dataset (arbitrarily selected)[40].

5.3.4 End-of-life and the knee point

Two key points shape the aging trajectory. End-of-life is usually fixed at the point where capacity drops to 80% of its original value. Below this, the cell is unfit for its main application, though it may still be suitable for less demanding tasks. The knee point is when the gentle, near-linear early decline shifts to a steep, accelerating fade. Predicting this point is important because it marks the transition when a cell becomes unreliable. SoH estimation supports both concepts. A model that accurately tracks SoH throughout its lifetime helps identify the knee point and predict end-of-life. This is why literature treats accurate SoH regression as the foundational task for knee-point and end-of-life predictions.

5.4 Related work

SoH estimation methods [41], [42] fall into two categories: model-based and data-driven, with hybrid approaches connecting the two. This section reviews both, then focuses on deep-learning architectures relevant to the models developed in this work, the generalization challenges in the field, and the specific state-of-the-art method this use case uses as a benchmark.

5.4.1 Model-based estimation

Model-based methods describe the cell using explicit physics or circuitry. Electrochemical models [43] like the pseudo-two-dimensional model represent transport and reaction processes inside the cell. They can be very accurate but involve many parameters that are difficult to identify and are computationally intensive. This makes them ill-suited for continuous online estimation on a BMS [43]. Equivalent-circuit models replace chemistry with a small network of resistors and capacitors fitted to the cell's response. They are light enough for online use and widely deployed, but are phenomenological, and their parameters may drift with age and temperature. This limits their accuracy for health estimation. Empirical and semi-empirical degradation models fit a function to observe capacity-fade curves. They are simple and interpretable, but their effectiveness relies on the appropriateness of the chosen function, and they generalize poorly across different operating conditions. The common theme is that model-based methods capture prior knowledge but struggle with the diversity and nonlinearity of real-world aging.

5.4.2 Classical machine learning and feature engineering

Data-driven methods learn the relationship between observable signals and health directly from data. The classical branch combines manually crafted health indicators with traditional regressors. Health indicators are features extracted from charge or discharge curves that relate to degradation. Examples include the time in a specific voltage interval, the shape of the incremental-capacity curve obtained by differentiating capacity with respect to voltage, the peaks of the differential-voltage curve, and statistical summaries of the signals. These features inform regressors like support-vector regression, random forests, Gaussian-process regression, and gradient-boosted trees. Gradient boosting, especially its efficient tree implementation, serves as a strong and fast baseline for tabular features. The strength of the classical approach is its data efficiency and interpretability; however, its reliance on hand-designed features may miss information that a learned representation could capture. Additionally, it often requires near-equilibrium, low-rate data thatis not always available.

5.4.3 Deep learning architecture

Deep learning eliminates the need for manual feature design by learning them from raw or lightly processed signals. Several architecture families are relevant here. One-dimensional convolutional networks [13] apply learned filters along the time axis and extract local patterns from voltage, current, and temperature sequences. Recurrent networks, particularly long short-term memory and gated recurrent unit cells, model the temporal dependence of signals and the gradual changes in health. However, they can struggle with very long sequences. Attention mechanisms address this limitation [15] by assigning weights to the contributions of each timestep instead of compressing the entire sequence into a final state. The additive attention used here scores relevance with a small, learned network, adapting techniques from sequence-to-sequence models for time series analysis [14]. Convolutional-recurrent hybrids combine the feature-extracting power of convolutional networks with the temporal modeling capabilities of recurrent networks. This design lets convolution summarize local structures before recurrence captures longer-range dependencies; this hybrid includes the strongest model in this study. Transformer architectures, which relies entirely on self-attention, have recently been used in battery health and remaining-life estimation, marking part of the current state of the art [44]. Autoencoders and time-frequency representations like spectrograms provide routes to compact learned features and inspire the exploratory aspects of this work.

5.4.4 Why each architecture family is relevant

It is important to clarify what each architecture family contributes to this problem. The model design in Section 5.7 draws from this selection. The convolutional network acts as a learned bank of matched filters. Each filter moves along the time axis and responds to local patterns in the signal. Stacking convolutions allows later layers to recognize combinations of earlier patterns over longer intervals. This is beneficial for battery signals, where health-related patterns are often found in local shapes of charge or discharge events, the curvature of voltage rises, and how temperature reacts to loads. However, a basic convolution does not connect structures that are far apart in time, as its receptive field, while growing with depth, remains limited.

The recurrent network addresses this challenge. By carrying a hidden state from one timestep to the next, it can connect the start of a window to its end. The gated versions, long short-term memory and gated recurrent units, are designed to maintain information across long distances without experiencing vanishing gradients. The downside is that recurrent networks process sequences step by step, which can slow down the operation, and condensing a long sequence into a single final hidden state can create a bottleneck for information. Attention mechanisms are a typical solution to this bottleneck [45]. Instead of

depending on the final state, attention looks back over each hidden state, scores its relevance to the prediction, and forms a weighted combination. This enables the model to focus on the informative parts of the window wherever they occur. The additive attention used here calculates those scores with a small, learned network, adapting from techniques used in translation tasks for sequence alignment to pool a time series into a fixed vector.

The hybrid approach naturally combines these strengths. If convolution excels at local patterns while recurrent networks handle long-term dependencies, then a convolution that first refines the raw window into a shorter sequence of local-feature vectors, followed by a recurrent network to model how those features evolve, should combine the advantages of both while reducing their weaknesses. The recurrent network works with a shorter, clearer sequence, and the convolution is freed from the need to cover long distances.

5.4.5 The generalization problem

The main challenge in data-driven State-of-Health (SoH) estimation is generalizing to cells that the model has not encountered before [46]. A model may seem effective when both training and test windows come from the same cells since windows from a single cell's data are closely related, allowing the model to memorize that cell. The real test comes when entire cells are held out, forcing the model to generalize across the differences that arise from variations in manufacturing and usage histories. Evaluations that group all windows of a cell into the same fold consistently report higher and more realistic errors than evaluations that use random splits [46]. The difference between the two measures how much the model has memorized versus learned. Transfer learning and domain adaptation are ongoing responses to this problem [47], aiming to match the feature distributions of different cells or chemistries to facilitate knowledge transfer. The validation method used in this work focuses on held-out cells because it is the only way to ensure that the reported numbers accurately reflect what they intend to communicate.

The literature offers a variety of responses to the generalization problem. The simplest approach is to hold out entire cells and accept the larger error by holding out entire cells and reporting those results (the baseline this work follows). Beyond this, transfer learning adjusts a model trained on one population by using a small amount of data from a new cell or chemistry, retaining most of what was learned while tweaking the last layers for the new distribution. Domain adaptation goes even further by explicitly matching the feature distributions of the source and target populations during training [47]. This ensures that the model learns representations that are independent of which cell or chemistry the window originates from. Techniques involve minimizing statistical distance between the source and target feature distributions, like maximum mean discrepancy, and adversarial methods where a second network tries to identify the domain from the features while the main

network learns to prevent that. Domain generalization seeks to achieve the same independence without any access to target data at all. It trains across enough different source domains, so the model learns features that transfer well to unseen ones.

These methods are relevant to the current work as a planned defense strategy. The dataset's intentional diversity acts as a form of domain generalization since training across different calendars, cyclic, and profile-aged cells at various temperatures pushes the model toward features that are not specific to one condition. If the cell-grouped error remains higher than desired after improving windowing and normalization, explicitly aligning the feature distributions of different cells will be the next step. The discovery that windows cluster by cell indicates the relevance of such a method. The work is already positioned to incorporate this without major changes, as the validation protocol measures the aspect domain adaptation seeks to enhance performance on held-out cells.

5.4.6 BatteryGPT and the state of the art

This work uses BatteryGPT as its benchmark, a recent approach that applies the generative pre-trained transformer architecture to battery degradation [38]. BatteryGPT functions in two stages. First, a compact transformer language model processes voltage, current, and temperature signals and predicts the charging data for the entire remaining lifecycle from an initial fragment of it. In the second stage, a convolutional-recurrent estimator converts the predicted signals into SoH, from which the knee point and end-of-life are derived. On a subset of a public fast-charging dataset, BatteryGPT reports very low errors, around a fifth of a percent in SoH when provided with the first third of the lifecycle and outperforms several regression and autoregressive baselines [38].

BatteryGPT sets the standard against which this work measures itself, but the comparison is more about contrasting philosophies than just looking at a single number. BatteryGPT takes an indirect and generative approach by first predicting future signals and then estimating health from them. It employs a transformer with tens of millions of parameters trained on a narrow population of lithium-iron-phosphate cells under fast charging conditions. The authors acknowledge that the availability of data and computation limited their scope. The approach here differs in three key areas. It is direct and discriminative, estimating SoH from current signals instead of predicting future ones. It prefers lightweight models, using compact convolutional and recurrent structures suited for on-device inference. Additionally, it is trained on a more diverse population of cells aged across calendar, cyclic, and mixed-profile conditions at various temperatures. This stresses generalization more than a single fast-charging protocol. The aim is not to win a direct error comparison on the same dataset but to demonstrate that a simpler, more deployable model trained on diverse data can achieve competitive accuracy, which is what truly matters for a real Battery

Management System (BMS). This benchmark has directly come from BatteryGPT reference and shows how they compared a family of models to their proposed one (Figure 5.3).

	Method	SOH Prediction RMSE (%)	SOH Prediction MAPE (%)	Knee Point Error	Knee Point MAPE (%)	EOL Error	EOL MAPE (%)
Regressive prediction	CNN	7.41	7.27	98	9.62	Out of range	Out of range
	LSTM	12.41	15.58	-240	23.57	Out of range	Out of range
	CNN-LSTM	2.83	2.85	33	3.24	38	3.25
Autoregressive prediction	LSTM	11.94	13.48	-265	25.95	Out of range	Out of range
	Transformer	22.71	30.56	-421	41.07	Out of range	Out of range
	BatteryGPT (5)	3.63	2.59	-62	6.83	-12	1.23
	BatteryGPT (30)	2.28	2.01	-34	3.74	-3	0.37

Supplementary Table 1. Comparison of LIB degradation prediction performance for KIT dataset. The out of range means that the predicted SOH do not reach 80%, and EOL can not be obtained.

Figure 5.3 Comparison table of BatteryGPT reference [38]

5.4.7 Early prediction and data efficiency

A key area of literature related to this chapter's central tension is early prediction. This involves estimating a cell's future degradation or expected lifespan from a limited portion of early-life data, even before measurable capacity loss occurs. Foundational results show that it is possible to predict a cell's cycle life with useful accuracy using features extracted in the initial few cycles, long before visible capacity decline appears [48]. This is done by revealing subtle differences in the voltage versus capacity curves that predict later behaviors. This shows that early signals contain information about the future, even when not immediately obvious, and has led to a considerable amount of work on predicting lifetime, knee points, and degradation pathways based on early data.

This work relates to the trade-off between how much data a model needs and how accurate it can be. Early-prediction methods intentionally limit themselves to a brief initial segment, accepting a tougher challenge in exchange for the practical benefit of having an early answer. The state-of-the-art benchmark discussed here follows this tradition, showing how its accuracy improves as it is allowed to evaluate a greater chunk of the lifecycle. This work faces the same challenge but from the other side: it aims for an accurate present health estimate. It has been found that achieving this accuracy currently requires a long input window, presenting a data-efficiency problem from a different perspective. Therefore, the early-prediction literature offers techniques and shows that input length is itself a metric. It

is only accurate when it receives a large amount of data that of data has not fully solved the problems posed by real systems.

5.4.8 A closer reading of the benchmark

Since BatteryGPT is the reference for this work, it is helpful to examine its method more closely to understand its achievements and see where a lighter alternative may differ. The first stage tokenizes the continuous voltage, current, and temperature signals by converting each into a limited vocabulary of tokens, similar to how a language model tokenizes text. This turns the battery's operating trajectory into a sequence of discrete symbols. A transformer with a stack of decoder blocks learns in a self-supervised manner to predict the next token based on the preceding ones, which mirrors the training objective of a language model that continues a sentence. This allows the model to extend the cell's trajectory: given the tokens from an early part of the cell's life, it generates the tokens for the remaining part, resulting in a predicted full-lifecycle signal. The second stage employs a separate supervised estimator, a convolutional network combined with a recurrent one, that maps a window of operating tokens to the State-of-Health at that moment. Applying it along the generated trajectory yields a predicted State-of-Health curve from which the knee point and end-of-life points can be derived.

Two design decisions in this method can be reconsidered by a direct approach. The first is that the prediction is generative and indirect. BatteryGPT does not estimate the current health based on current signals; it predicts future signals and then uses those to infer health. This approach is well-suited for early prediction of the cell's future from a short early fragment, but it is more complex than needed when the objective is simply to determine a cell's health now based on recent signals. This is the challenge a continuously operating BMS faces. The second concern is the cost of the generative model. A transformer with tens of millions of parameters is costly to train and evaluate. While the authors keep it relatively small for a transformer [44], it is still much larger and more resource-intensive than the compact convolutional and recurrent models pursued in this work. The reported error is roughly a fifth of a percentage point in State-of-Health.

5.5 Dataset

5.5.1 The RWTH-Aachen aging dataset

The experiments use the detailed battery aging dataset published by Luh and Blank [12], one of the largest openly available collections of its kind. It tracks commercial lithium-ion cells with an NMC/C-SiO chemistry, this is a nickel-manganese-cobalt positive electrode paired with a graphite and silicon-oxide negative electrode. These cells were aged for over

a year under diverse operating conditions, with regular check-ups measuring the remaining usable capacity and the impedance of each cell. The raw logs capture voltage, current, and temperature at high time resolution. What sets this dataset apart is its combination of three aging types in one experiment: calendar aging, where cells rest at a fixed state of charge and temperature; cyclic aging, where cells are repeatedly charged and discharged at controlled rates; and profile aging, where cells follow realistic driving-cycle load profiles. This variety of conditions makes the dataset a tough test for generalization, as a model must handle cells with signals shaped by very different histories.

5.5.2 Why this dataset suits the thesis

Several public datasets are used in battery prognostics, and selecting one influences what a study can claim [48]. The most used early-prediction dataset tracks many lithium-iron-phosphate cells under fast-charging protocols and has driven much of the progress in predicting cycle life from early data, including the method this chapter benchmarks against. Other established collections feature fewer cells under laboratory cycling. This dataset combines 76 cells, NMC/C-SiO chemistry, and three aging regimes across four temperatures. Its cells feature a nickel-manganese-cobalt chemistry with a silicon-bearing graphite negative electrode, which closely represents modern automotive cells and degrades through a richer mix of mechanisms compared to the iron-phosphate cells in the fast-charging dataset. This dataset ages its cells in different ways: it intentionally mixes calendar, cyclic, and realistic driving-profile aging across various temperatures, states of charge, and rates.

The dataset spans calendar, cyclic, and profile aging across four temperatures, so a model must generalize across aging regimes, not just across cells. A model trained and tested on cells aged under just one protocol can perform well while having only learned that protocol. However, a model that encounters calendar-aged, cycle-aged, and profile-aged cells at different temperatures must learn something closer to the fundamental physics of degradation. The trade-off for this breadth is a tougher problem and larger reported errors, but those errors provide a more realistic estimate of how a model would perform in a real fleet.

5.5.3 Cell population and the working subset

This chapter uses a subset of the dataset that includes 76 cells labeled P001 through P076, in a partially preprocessed form developed by the team from the public source. The cells cover the temperature range of the experiment, approximately 0, 10, 25, and 40 degrees Celsius, and span the calendar, cyclic, and profile aging types at various states of charge and charge or discharge rates. Each cell provides several recording files. The processed

signals are reduced to a common set of channels and divided into windows, as explained below, and the resulting windows carry a SoH label based on the capacity check-ups. A factor the validation protocol must consider is the number of windows per cell varies, from a few dozen for short-lived or sparsely sampled cells to over a hundred for long-lived ones.

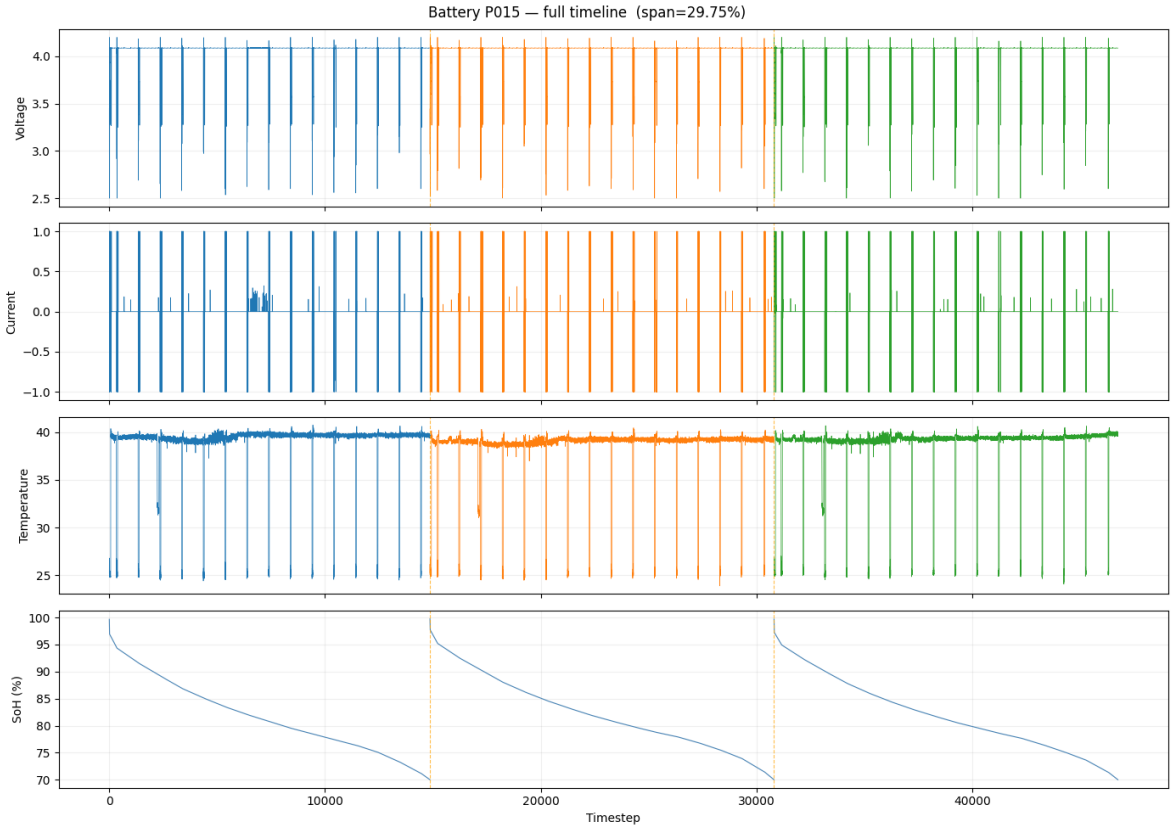
5.5.4 The dataset in the landscape of public collections

To understand the context of the chosen data set, it helps to review the other collections in the field, as each has influenced the methods developed from it. The fast-charging dataset, which underpins much early-prediction work, including this chapter's benchmark, records over a hundred lithium-iron-phosphate cells driven to failure under various fast-charging protocols [12]. Its size and consistency have made it a proving ground for predicting lifetime from early cycles; however, its limitation is that it changes the charging protocol while keeping the chemistry and broad usage pattern constant. Models trained on this dataset may not have learned to generalize across different chemistries or the calendar and profile aging that real cells undergo. Older and smaller collections contain a few cells under laboratory cycling, useful for method development but too limited to test generalization across a population. A realistic dataset that records cells in a single application captures true usage but lacks the variety needed to study how conditions influence aging.

In this context, the dataset used here is unique because it combines a large cell population with a wide range of aging conditions and a chemistry representative of modern automotive cells. The diversity makes it the ideal choice for studying generalization. A model that works well across its calendar, cyclic, and profile-aged cells at different temperatures shows something a model tuned to a single protocol cannot. The drawback is that the problem becomes harder and the errors larger, this reflects what a system deployed in a real fleet would face. The choice of dataset is tied to the work’s claim. The dataset spans three aging regimes and four temperatures; the reported errors are therefore larger than single-protocol benchmarks but transfer better to fleet conditions.

5.5.5 Signal channels and the input representation

The model input relies on three main measured channels: voltage, current, and temperature. These are the quantities a BMS measures directly, and they form the basis for inferring battery health. Two additional derived channels are available: the time derivatives of voltage and current, which highlight the transient structure of the signals during the transitions between rest, charge, and discharge. The decision to focus on the three main channels and to include the derivatives only when beneficial aligns with the deployment logic of the thesis. A model that requires only the raw measured signals is easier to run on a BMS than one dependent on complex engineered features. The derivative channels can be



computed easily online and are kept as an option, not a requirement. A representation of random battery signals is shown in Figure 5.4.

Figure 5.4 A representation of battery P015 input signals

5.6 Methodology

This section explains how the use case follows the five-stage pipeline from Chapter 2, starting with the acquisition of signals and ending with the creation of labeled windows for the models. The model design is discussed in Section 5.7.

5.6.1 Acquisition and channel selection

The first stage of the pipeline reduces the raw cell logs to the working channel set. From the recorded voltage, current, and temperature, we directly keep three main channels. We compute the two derivative channels as finite differences of voltage and current overtime. The number of channels we use is treated as a design choice. The derivatives add information about changes but can also amplify noise. Whether this trade-off is beneficial depends on the model and the windowing. The convolutional models in this work can

accept either three channels or five channels. The channel count is one of the settings we explore.

We should look more closely at the derivative channels, as they create a small but real design challenge. The time derivative of voltage shows how fast the voltage is changing. This signal is greatest at the start of charging or discharging and near the voltage limits where the cell's response sharpens. The time derivative of current highlights the transitions between resting, charging, and discharging. These derivatives help models pinpoint informative events, which are useful. However, differentiation can also amplify high-frequency noise. This means that a noisy raw signal leads to a noisier derivative, and models using derivatives must learn to separate real changes from amplified noise. Whether the derivatives are helpful overall depends on the signal's clarity and the model's ability to use the extra channels without being misled by the noise. That is why the channel count is an adjustable setting instead of a fixed one. The convolutional models can accept either three core channels or five channels including derivatives, and comparing these configurations is part of our experiments.

5.6.2 Pre-processing and windowing

The second stage converts the continuous signal from each cell into fixed-length windows, each linked to a State-of-Health (SoH) label. A window is a continuous segment of the multivariate signal of a chosen length, with consecutive windows taken at a set stride along the signal. The window length and stride are important. A window that is too short will not capture enough of the cell's behavior for health assessment, while a window that is too long ties the estimate to a large time span, reducing how often we can produce a new estimate. We studied these parameters, and the tension they create led to the central findings of this chapter, which we discuss in Section 5.10 and address in Section 5.11. Each window is given the SoH value applicable during its span, derived from capacity check-ups so the model learns to map that signal segment to the cell's health.

Since the models used in this study fit into two implementation families, we prepare the windows in two compatible formats. Tree-based and recurrent models read the window as a sequence of feature vectors, while convolutional models use it with the channel axis arranged for one-dimensional convolution. Both layouts contain the same information and come from the same segmentation, ensuring that comparisons across models evaluate architectural differences.

5.6.3 The window and stride study

We did not guess the window length and stride; we studied them directly because they affect how much information each training example contains and how many examples the dataset produces. A longer window provides a more comprehensive view of each cell's behavior but results in fewer windows from any given recording, linking each estimate to a longer time period. A shorter window yields more examples and more frequent estimates, but it might not contain enough health-related structure to support accurate predictions. The stride, or the gap between the start of one window and the next, balances overlapping with independence. A small stride results in many overlapping windows that enhance the training set but are highly correlated, while a large stride produces fewer, more independent windows. We examined these parameters and how they influenced downstream accuracy. This study revealed the long window in the best configuration and showed that window length is a challenge to address.

The windowing study also brought up practical issues that, if ignored, could compromise the data. Windows must not cross the boundary between two recordings of the same cell or two different cells, or they would mix unrelated signals and labels. Windows fully within a long rest period offer little health information and can mislead a model into linking inactivity with a specific health level. Also, the label assigned to a window must reflect the cell's health during that window's span, it means aligning the windowed signal with the capacity check-ups defining the labels. We addressed all these factors in the segmentation, and we visualized the pipeline end to end on sample cells to confirm that the windows, channels, and labels matched as intended before training any model.

5.6.4 Validating the pipeline before modeling

A recurring theme in this thesis is that a pipeline running without errors is not necessarily a correct pipeline. We applied the same careful approach used in the driver-monitoring system here. Before training any model, we inspected the windowed and normalized data channel by channel and cell by cell. We confirmed normalization by ensuring that each channel had a mean close to zero and a unit standard deviation across the training set. We also checked that applying the same transformation to held-out data did not introduce a significant shift. We verified the label sequence for individual cells to ensure that the State-of-Health decreased smoothly throughout each cell's life, avoiding jumps that would indicate a misalignment between the signal and the label. We compared per-cell and per-channel distributions across the training, validation, and test sets to evaluate any distributional shifts. This pre-modeling validation may not seem exciting, but it distinguishes results that reflect the data from those affected by hidden data issues. So, we include it here as part of our methodological contribution.

5.6.5 Normalization

Standardization equalizes each channel's scale before it reaches a model. We subtract the mean and divide by the standard deviation, aiming for a zero mean and unit variance for each channel. This step is necessary because the raw channels operate on very different scales: voltage in volts, current in amperes, temperature in degrees. A model trained on abnormalized input would let channels with larger values overshadow smaller ones. We calculate normalization statistics using the training data and apply them unchanged to validation and test data to prevent any information about held-out cells from leaking into the model fit. The exploration data analysis in Section 5.6.5 confirms that after standardization, each channel centers near zero with unit spread, as intended.

A more subtle normalization question, one of the two leading topics in this chapter, is whether we should compute global statistics across all cells or per cell. Global normalization risks letting cells with naturally large signal variations dominate the shared scale, which may compress the informative variation of quieter cells. Per-cell normalization avoids this imbalance but loses absolute differences between cells that may carry health information. The current results use global standardization; we discuss the per-battery alternative in Section 5.11.

5.6.6 The State-of-Health label

The label for each window represents the SoH of the cell over the window's duration, based on the capacity scale from Equation 5.1. In the working subset, labels range roughly from 70% to 98%, with most windows falling in the low-to-mid eighties. This range reflects that cells spend much of their recorded life at moderate health levels. The distribution is not uniform, and its shape is important for training and evaluation. A model focusing on minimizing average error will tend to favor the densely populated middle of the range. Therefore, we need to ensure that it does not overlook the sparsely populated extremes, where the healthiest and least healthy cells reside. The error analysis in Section 5.10.4 examines this issue closely.

5.6.7 Exploratory data analysis

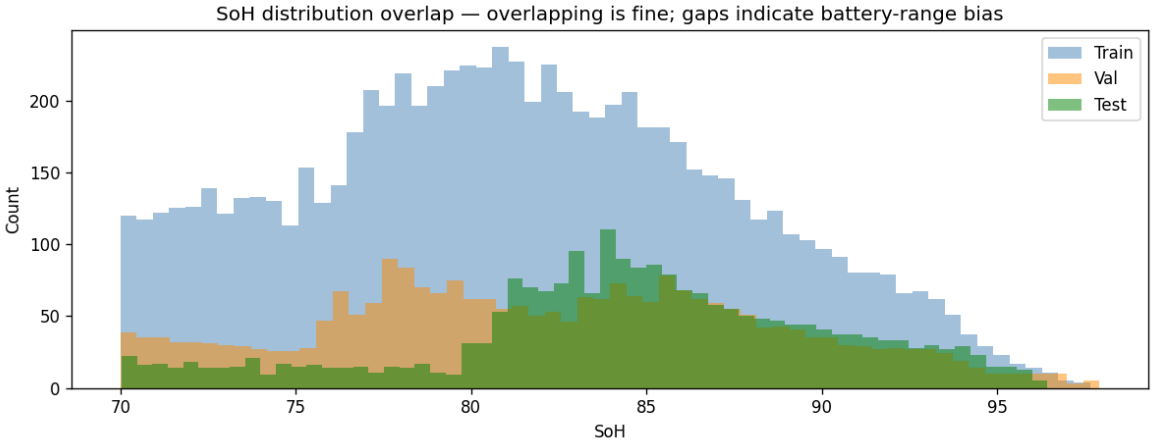
Before modeling, the windowed dataset was examined to understand its structure and to catch problems that would otherwise surface as silent errors. Four observations from this analysis shaped the later work. First, the distribution of SoH labels is unimodal but skewed, concentrated in the low eighties and thinning toward both the healthy and the heavily aged

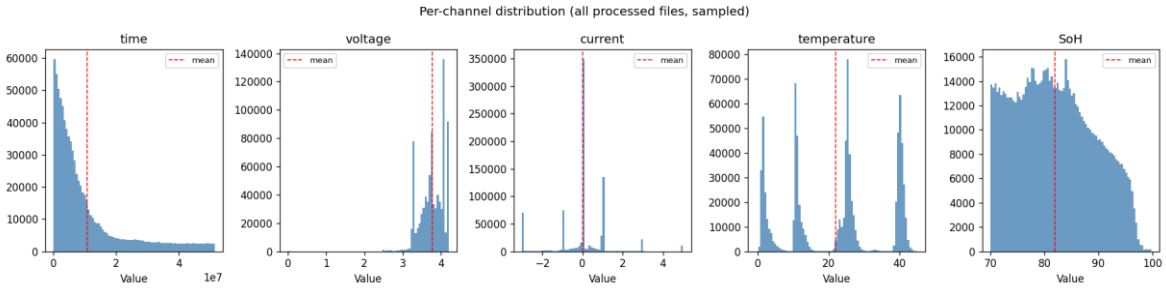
ends, which informs the choice of an outlier-tolerant loss in Section 5.7.6. Second, the linear correlation between any single channel's window-average and the SoH label is weak, with voltage the strongest at a small negative correlation and the others weaker still, which confirms that no single handcrafted summary suffices and that the value of a learned model lies in combining channels and exploiting their temporal shape. Third, a projection of the normalized windows onto their principal components shows the windows clustering by cell rather than spreading smoothly by health, a direct visual sign of the cell-to-cell domain shift that the generalization problem of Section 5.4.4 describes, and a warning that a model could memorize cell identity if allowed to. Fourth, the per-channel statistics differ between the training, validation, and test partitions, with the test cells sitting at systematically higher health than the training cells, so the evaluation is harder than a matched split and the reported numbers are conservative.

Each of these four observations carries a methodological consequence that the rest of the chapter acts upon. The skew of the label distribution toward the low eighties means that a model trained to minimize average error has a built-in incentive to be accurate in that range and to neglect the sparsely populated extremes, this is why an outlier-tolerant loss is chosen and why the error analysis examines performance as a function of health level. The weakness of any single channel's correlation with health means that the problem requires a model that combines channels and exploits their temporal shape; a linear regression on channel averages would fail. Deep models add value here because they combine channels and exploit temporal shape; the marginal statistics alone carry almost no signal (voltage $r = -0.13$). The clustering of windows by cell in the principal-component projection is the most consequential observation, because it is direct evidence that a model could distinguish cells from one another more easily than it can distinguish health levels, a model allowed to see windows from a cell during both training and testing could achieve a flattering score by recognizing the cell; this is the concrete justification for the cell-grouped validation. And the distributional difference between the partitions, with the test cells healthier on average than the training cells, means the evaluation is harder than a matched split, so the reported errors are pessimistic.

The exploratory analysis also examined the signals themselves, not only their summary statistics. Sample windows at low, middle, and high health were plotted channel by channel to confirm that the charge and discharge events were captured intact and that the derivative channels behaved as expected, spiking at the transitions and resting near zero elsewhere. The time-frequency content of the windows was examined to see whether degradation left a signature in the frequency domain, which motivated the exploratory spectrogram branch. And the per-cell aging trajectories were plotted to confirm that health decreased smoothly along each cell's life, which both validates the labels and illustrates the diversity of

trajectories that the model must learn to handle. This inspection is the foundation on which the modeling rests, and it is reported in this depth because a model is only as trustworthy as the data it learns from and demonstrating that the data is sound is part of demonstrating that the model is.





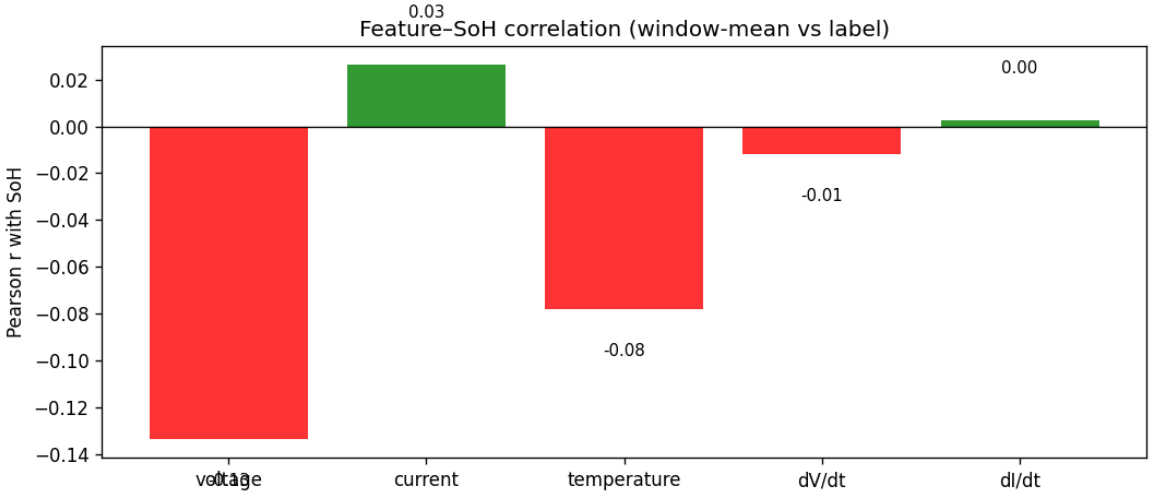


Figure 5.5 Exploratory analysis of the windowed dataset: the distribution of State-of-Health labels, the per-channel feature distributions, the weak single-channel correlation with State-of-Health, and the principal-component projection showing clustering by cell.

5.7 Model design

Stage three of the pipeline focuses on designing the estimator. Instead of choosing a single architecture, this phase creates a range of models that cover the methods outlined in Section 5.4. This includes a classical gradient-boosting baseline, convolutional networks, recurrent networks, hybrid deep networks, and an exploratory autoencoder. Developing this range serves two purposes: it helps identify which architecture class is best suited for the problem, and it establishes reliable baselines against which the strongest model can be evaluated. Each model is trained using the same loss function and validated under a consistent protocol. This ensures that any differences in performance can be attributed to the architecture.

5.7.1 Gradient-boosting baseline

The baseline model is a gradient-boosted tree ensemble [49]. It treats each window as a flat feature vector created by flattening the channel-by-time window into a single row. Gradient boosting constructs an ensemble of shallow regression trees. Each tree corrects the residual error left by the previous trees. It is a strong, fast, and well-regularized option for tabular regression. This baseline is not included because a flattened window is the best representation; flattening loses the explicit temporal structure that deep models use. However, it does provide a reliable floor: if a deep model fails to outperform a well-tuned gradient-boosting baseline with flattened windows, it is not justifying its complexity. The ensemble typically uses a moderate number of shallow trees with row and column subsampling along with both types of weight regularization. The exact configuration is determined by the hyperparameter search in Section 5.8, and early stopping on a validation fold helps avoid overfitting.

Gradient boosting has two key characteristics that make it a relevant baseline [49]. First, it does not depend on the scale or relevance of individual features. Each tree split considers features independently, and the ensemble down-weights uninformative features. This means that flattening a window into many loosely structured features does not hinder it as it might with a model that expects smooth input. Second, it trains and evaluates quickly. This is important since the same model needs to be retrained for every fold in a cell-grouped cross-validation and each trial in a hyperparameter search. However, its drawback is clear: by flattening the window, it discards the explicit order of the timesteps. As a result, any structure reliant on that order, which most recurrent networks exploit, is not accessible unless it remains as a static pattern in the flattened features. Therefore, the difference between the gradient-boosting baseline and the deep models directly measures the value of the temporal structure in this dataset.

5.7.2 One-dimensional convolutional network

The first deep model is a one-dimensional convolutional network. It applies multiple convolution layers along the time axis, followed by batch normalization and a rectified-linear activation function. There is pooling between stages that gradually reduces the time dimension while increasing the receptive field, allowing deeper filters to capture more extended sections of the signal. The stack starts with a wider first kernel to capture coarse structures, followed by narrower kernels that refine these features. The channel count increases in the early stages and decreases before a global average overtime combines the sequence into a fixed vector. A dropout layer regularizes the final output, and a single linear unit provides the scalar State-of-Health (SoH) estimate. The convolutional network

acts as a natural deep baseline for signals, as it learns local features independently and is suitable for potential deployment.

The design of the convolutional stack follows a typical and logical pattern. The first convolution uses a wide kernel to capture broad structures over several timesteps. The later convolutions employ narrower kernels to refine the representation. Pooling between stages halves the time dimension at each step, allowing a fixed kernel to cover a longer span of the original signal as the layers progress. Batch normalization after each convolution stabilizes training by maintaining consistent activation scales, while the rectified-linear activation introduces nonlinearity, enabling the stack to represent more complex patterns than a linear filter can. The global average overtime at the end is intentional; instead of flattening the final feature map, which would tie the model to the exact length of the window, averaging over time creates a fixed-size summary. This summary adapts to small shifts and varying window lengths. This is advantageous for signals where informative events can occur anywhere within the window. The result is a model with relatively few parameters that can realistically be used for on-device inference. The 1D-CNN is not only a baseline; at 732 kB it is a deployment candidate.

5.7.3 Recurrent networks with attention

The second and third deep models are recurrent networks. One uses gated recurrent units (GRUs), while the other employs long short-term memory (LSTM) cells. A recurrent network processes each window timestep by timestep, maintaining a hidden state that carries information forward, which fits the slow temporal changes of the signals. Both models operate bidirectionally, reading the window both forward and backward. This allows each timestep to be understood in the context of those before and after it. Instead of summarizing the sequence with the final hidden state, which tends to lose information from earlier timesteps, both models use an additive attention layer over the time axis. The attention computes relevance scores for every timestep [45], normalizes these scores into weights, and creates a weighted sum of the hidden states. This setup lets the model use the entire window and learn which parts are significant for health. The pooled representation then moves to a linear unit for the SoH estimate. The gated-recurrent and LSTM variants have this shared structure but differ only in the type of recurrent cell. This allows comparison to reveal the specific impact of the cell type.

The additive attention in use here warrants detailed description because it is consistent across the recurrent and hybrid models. Once the recurrent layers output hidden states at each timestep, the attention layer transforms each hidden state through a small, learned process, applies a nonlinearity, and produces a single scalar score for that timestep. The scores across all timesteps are normalized using a softmax function into weights that total

one, and the pooled representation is the weighted sum of the hidden states based on these weights. This means the model learns from the data which timesteps in a window are significant for health and emphasizes them. In a battery window where important behavior is concentrated in the charging and discharging events, with little information in the rest periods, this bias is ideal. This is part of why the attention-equipped recurrent models perform better than a standard recurrent network that reduces the sequence to its final state.

5.7.4 Convolutional-recurrent hybrids

The fourth and fifth deep models combine convolutional and recurrent components. The convolutional part first captures local structures and shortens the sequence. The recurrent layers then model the longer-range temporal relationships in the summarized sequence, with the same additive attention pooling the result before a small regression head generates the estimate. The idea is that the two components complement each other: convolution is effective at capturing local patterns but has a limited temporal range, while recurrence captures long-range dependencies but benefits from cleaner, shorter sequences. The two hybrids only differ in whether the recurrent backbone uses GRUs or LSTMs. This family produced the best-performing model in the study, detailed in Section 5.10. Figure 5.3 illustrates the structure of the convolutional gated-recurrent hybrid; the LSTM variant substitutes only the recurrent cell.

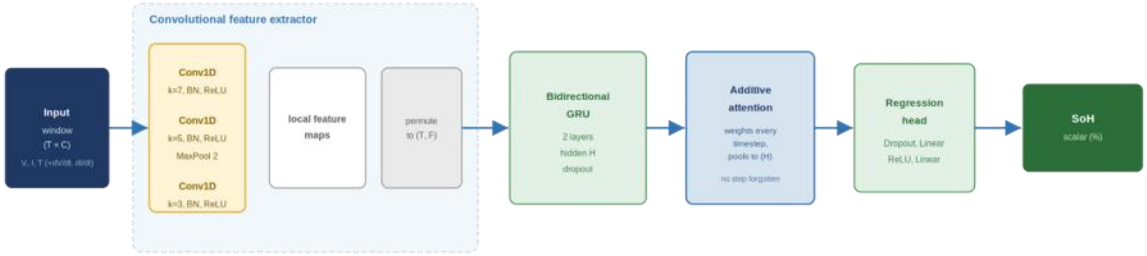


Figure 5.6 The convolutional gated-recurrent hybrid. A convolutional feature extractor reduces the window to a sequence of local-feature vectors. A bidirectional recurrent backbone models the temporal dependence, additive attention pools across time, and a small regression head produces the State-of-Health estimate. The convolutional LSTM variant shares this structure, changing only the recurrent cell.

The hybrid outperforms both pure architectures because the battery window is long, and its health-bearing structure is both local, in the shape of individual charge and discharge events, and global, in how those events change over the span of the window as the cell ages. A pure convolution captures the local structure but cannot easily relate events far apart in the window; a pure recurrent network can relate to distant events but must process the full-length raw sequence step by step (slow and memory straining). The hybrid lets the convolution first compress the raw window into a much shorter sequence of local-feature

vectors, one per pooled time step, and then lets the bidirectional recurrent network model how those features evolve across the window, with attention selecting the informative parts. This division of labor matches the structure of the problem, and the empirical result, that the hybrid is the most accurate model in the family, follows from that match. The same reasoning predicts that the gated-recurrent and long-short-term-memory backbones should perform similarly, since they differ only in the internal mechanics of the recurrent cell, and comparing the two isolates whatever small advantage one cell type holds over the other on this data.

5.7.5 Spectrogram autoencoder (exploratory)

Alongside the regression models, an autoencoder on a time-frequency representation of the signal was explored as a side idea. The motivation is that a spectrogram exposes the frequency content of the windowed signal, which may separate degradation signatures that are entangled in the time domain, and that an autoencoder can learn a compact latent code of that representation without supervision, which could then feed a lighter regressor. This branch is exploratory and is reported as such: it is not central to the contribution, and it may not lead to a competitive result. It is included for completeness and because the time-frequency view is a natural complement to the time-domain models, but the main line of the work rests on the direct regression models above.

5.7.6 Loss function and training protocol

All deep models are trained to minimize the Huber loss between the predicted and true SoH. The Huber loss behaves like the squared error for small residuals and like the absolute error for large ones, so it is less sensitive to outliers than the plain squared error while remaining smooth near zero. This robustness suits a label distribution that is skewed and has sparsely populated extremes, where a few large residuals under a squared loss would otherwise dominate the gradient. For a residual r and a threshold delta, the loss is

$$L(r) = \tfrac{1}{2} r^2 \ \text{for } |r| \le \text{delta}, \quad \text{delta}(|r| - \tfrac{1}{2}\,\text{delta}) \ \text{otherwise} \tag{5.2}$$

with the threshold delta itself treated as a tunable parameter. Optimization uses the Adam optimizer with weight decay, gradient-norm clipping to stabilize the recurrent models, and a learning-rate schedule that reduces the rate when the validation loss plateaus. Training runs for up to several hundred epochs with early stopping on the validation loss, keeping the checkpoint with the best validation performance, so that the reported model is the one that generalized best not the one at the final epoch. The gradient-boosting baseline uses its own early stopping on the validation fold.

The choice of the Huber loss over the more common squared error deserves a fuller justification, because it is one of the few decisions shared by every deep model in the study and it interacts with the structure of the data. Under a squared error, a residual of a given size contributes its square to the loss, so a window the model gets badly wrong contributes far more to the gradient than several windows it gets slightly wrong, and the optimization is pulled toward reducing the worst errors even at the expense of the typical ones. On a label distribution with sparsely populated extremes, the windows at very high or very low health are both rare and, being unusual, often harder to predict, so under a squared loss they would exert an outsized pull on the model and could distort it toward the extremes at the cost of the dense middle. The Huber loss caps this effect: beyond a threshold it grows only linearly with the residual, so a hard window contributes in proportion to its error, and the optimization is not hijacked by a few difficult examples. The threshold at which the loss switches from quadratic to linear is tuned, so the degree of this robustness is fitted to the data.

The training protocol is designed for reproducibility and for model selection. The optimizer is a standard adaptive method with a small weight-decay term that discourages large weights and mildly regularizes the model. The recurrent and hybrid models have their gradients clipped to a fixed norm, which prevents the occasional exploding gradient that recurrent architectures are prone to and keeps training stable. The learning rate is reduced automatically when the validation loss stops improving, which lets the model take large steps early and fine steps later without manual scheduling. Training continues for up to several hundred epochs but stops early when the validation loss has not improved for a fixed number of epochs, and the checkpoint retained is the one with the best validation loss not the last, so that the reported model is the one that generalized best during training and not one that may have begun to overfit by the final epoch. Because the validation loss that drives both the early stopping and the checkpoint selection is measured on held-out cells, the entire selection procedure is aligned with the quantity the work ultimately cares about, generalization to unseen batteries.

5.8 Hyperparameter optimization

Each model has hyperparameters that interact with each other, and their best settings are not known in advance. For the convolutional network, these include channel width, dropout rate, learning rate, batch size, weight decay, and the Huber threshold. For the recurrent models, you also have hidden size, number of layers, and whether the backbone is bidirectional. For the hybrid models, you consider convolutional width along with recurrent hidden size and depth. For the gradient-boosting baseline, important parameters are the

number and depth of trees, learning rate, subsampling fractions, and regularization strengths.

We tuned these hyperparameters using an automated search with the Optuna framework [50]. This framework builds a probabilistic model of how hyperparameters relate to validation performance. It focuses its trials on promising areas, finding good configurations in far fewer trials than grid or random search methods. The deep models used a pruning strategy that abandons unpromising trials early. This approach allows the budget to cover more configurations. Every search evaluated its trials under the same cell-grouped scheme as the final validation. This ensured that we selected hyperparameters based on their ability to generalize to held-out cells.

Table 5.1 summarizes the main hyperparameters and their search ranges for both deep models and the gradient-boosting baseline. We selected ranges that were broad enough to include likely optimal values without wasting the budget on unreasonable settings. Categorical choices, like whether a recurrent backbone is bidirectional, were included so that structural decisions were tuned alongside numerical ones.

Model family	Tuned hyperparameters	Search range
Gradient boosting	trees, depth, learning rate, subsample, column subsample, min child weight, gamma, L1/L2 regularization	hundreds to ~2000 trees; depth 3-10; lr 0.005-0.1; subsample 0.6-1.0
1D-CNN	base channel width, dropout, learning rate, batch size, weight decay, Huber delta	width 16-96; dropout 0-0.4; lr 1e-4-5e-3; delta 1-5
GRU / LSTM (+ attention)	hidden size, layers, dropout, learning rate, batch size, Huber delta, bidirectional	hidden 32-256; 1-4 layers; lr 1e-4-1e-2; bidirectional yes/no

CNN + GRU / CNN + LSTM	conv width, recurrent hidden size, recurrent layers, dropout, learning rate, batch size, Huber delta	conv 16-64; hidden 64-256; 1-3 layers; lr 1e-4-1e-2

Table 5.1 Main hyperparameters and search ranges for each model family. The search was conducted with Optuna under the same cell-grouped scheme as the final validation.

The search ranges and the chosen configuration of each model are recorded during the experiments. We report the settings alongside the results to ensure the model comparisons are reproducible. This also allows us to weigh the cost of each model, in terms of parameters and data needed, against its accuracy.

5.9 Validation protocol

The fourth stage of the pipeline is validation. For this problem, the protocol is essential for determining the reliability of the results. As explained in Section 5.4.4, the only fair test of an SoH estimator is its performance on unseen cells. This is important because windows from a single cell are highly correlated. A model evaluated windows from the cells trained on only measures memorization. Therefore, the protocol groups the data by cell. Each window comes from a cell, and the splits ensure that all windows from a specific cell are entirely within one partition. No cell appears in more than one of the training, validation, and test sets. The cross-validation used during tuning and for the cross-validated estimates is grouped k-fold, keeping windows from each cell together within a fold.

To illustrate the protocol, consider a single cell. All the windows from that cell are assigned together to exactly one of the training, validation, or test partitions. When the cell is in the training partition, the model sees its windows during fitting. When it is in the test partition, the model has never seen any window from that cell during training or model selection. It must estimate the health of its windows by generalizing from other cells. Since cells differ in the number of windows they provide, the partitions are balanced at the cell level, not the window level. The cross-validation folds used during tuning follow the same structure, ensuring that every fold's validation score reflects performance on entirely new cells.

Table 5.2 shows the partition sizes used in the experiments.

Partition	Role	Approx. share of windows
Train	model fitting	~62.5%
Validation	early stopping, model selection, tuning	~20%
Test	final held-out evaluation only	~17.5%

Table 5.2 Partition sizes under the cell-grouped protocol. Cells, not windows, are the unit of partition. This ensures no cell appears in more than one partition and that test cells remain unseen during all model selection.

This grouping has two significant effects. First, it makes the reported errors larger than what a random split would show. The model must truly generalize across cells instead of interpolating within them. This is the key point: the larger number reflects reality. Second, it relates to the varying number of windows per cell. Holding out a long-lived cell removes more windows than holding out a short-lived one. The grouped folds address this by partitioning based on cells. The split sizes used in the experiments put most cells in training and split the remaining cells between validation and test, ensuring test cells go unseen throughout model selection.

5.10 Experimental results

This section compares the model family, identifies the best model, and analyzes its errors. The results represent ongoing work. Both the architectures and the protocol remain fixed, but exact figures are still being refined as improvements in windowing and normalization occur. The tables are designed to allow final values to be added as they are finalized. The discussion focuses on the relationships between models, which are stable, instead of single numbers, which are still changing.

It is important to know how to read these results. The model family, validation protocol, and analysis methods are set and will not change. However, the precise error figures are still being refined as improvements mentioned in Section 5.11 are applied and hyperparameter searches are extended. The chapter emphasizes the qualitative findings, highlighting which model class excels, the relationship between deep models and the baseline, and the main tension. These stand independently from exact numbers, allowing

tables to be filled with final values without disrupting the overall argument. This approach provides a transparent report on work in progress: it commits to the structure and relationships, marks specific figures as temporary, and frames the discussion around stable aspects. Readers should view the relationships between models as established while seeing individual error values as snapshots of the current state.

5.10.1 Evaluation metrics

Performance is described using three standard regression metrics. The root-mean-square error heavily penalizes large deviations and is the headline metric, expressed in percentage points of SoH. The mean absolute error shows the average size of the deviations in the same units and is less affected by occasional large misses. The coefficient of determination reveals the proportion of variance in the true SoH that the model explains, with one being perfect and zero meaning it is no better than predicting the mean. For predictions $\hat{y}$ against truth y over N windows,

$$RMSE = sqrt((1/N) \Sigma (y - \hat{y})^2), MAE = (1/N) \Sigma |y - \hat{y}| \quad (5.3)$$

All three metrics are calculated from the held-out test cells. In the field, a root-mean-square error below two percentage points of SoH is generally seen as good, while below one is considered excellent. However, the lowest published figures often come from evaluations that do not hold out entire cells, so these should be viewed as upper limits.

Each metric answers a different question and reporting all three helps prevent distortion from any single one. The root-mean-square error squares the residuals before averaging, making it sensitive to the largest misses. A model that typically performs well but occasionally has significant errors will show a worse root-mean-square error than expected. This is important for safety-critical estimates where rare large errors can be costly. The mean absolute error averages the residuals without squaring, providing a clearer picture of the typical error size and being less influenced by outliers. Comparing it to the root-mean-square error shows how heavy-tailed the errors are; a large difference indicates a few unusually large misses. The coefficient of determination provides context by comparing the error to the variance of the labels, showing whether the model is really better than simply predicting average health. A value close to one indicates the model explains most of the variation, while a value near zero shows it barely improves on a constant guess. Because the label distribution in this dataset cluster in the low eighties, a naive predictor would already achieve a deceptively small absolute error. This is why the coefficient of determination is reported alongside the error magnitudes; it confirms that a good error reflects true discrimination across health levels rather than just the narrowness of the label range.

5.10.2 Model comparison

Table 5.3 summarizes the test-set performance of the model family using the cell-grouped protocol. The structure of the comparison is the key aspect. The gradient-boosting baseline sets the minimum performance that the deep models must surpass. The convolutional and recurrent models evaluate whether learned local features or temporal dependencies are more valuable for this problem. The hybrid models test whether combining the two performs better than either alone, with the most effective hybrid being the focus of the discussion. The exploratory autoencoder is listed separately as a side result.

Model	RMSE (Normalized)	MAE (%) (SoH point)	R^2	Notes
XGBoost (flattened windows)	0.97	3.12	0.79	tabular baseline
1D-CNN	0.93	2.61	0.81	learned local features
GRU + attention	0.84	2.48	0.88	bidirectional, attention pooling
LSTM + attention	0.88	2.53	0.81	bidirectional, attention pooling
CNN + GRU + attention	0.62	2.21	0.93	strongest; long window
CNN + LSTM + attention	0.86	2.41	0.88	hybrid variant

Spectrogram autoencoder	0.98	3.37	0.77	exploratory side idea

Table 5.3 Test-set performance of the model family using the cell-grouped protocol on held-out batteries. Values are reported as they become finalized; the figure marked with an asterisk for the convolutional gated-recurrent hybrid is an early, unoptimized result obtained with the long window discussed in Section 5.10.3 and is included to support the central finding.

5.10.3 The strongest model and the window-length problem

The convolutional gated-recurrent hybrid is the strongest model in this study. In an early and unoptimized setup, it estimates State-of-Health to within 2.21 percentage points (MAE) on batteries never seen during training, with an R^2 of 0.93 and a normalized RMSE of 0.62 (normalized), the best figures in the model family on all three metrics. At roughly 11 million parameters, it is about a quarter the size of BatteryGPT's 42-million-parameter generative model, and it estimates health directly from observed signals instead of first predicting a cell's future trajectory.

However, the downside is the cost of achieving that accuracy. The window needed to reach it is very long, around three weeks of cell history, roughly twenty-three days. This long window length came from a search done for a range of windows and strides. By locating the knee point in window size and stride plots against R^2 and RMSE (as shown in Figure 5.7) it is guaranteed that the best combination of window and stride is found. Such a long window undermines the practical use of the estimate in two ways. First, the model cannot provide a new estimate until it has gathered three weeks of data, but too slow for a Battery Management System (BMS) that should monitor health continuously. Second, this long-time frame ties the estimate to an extended period, meaning the model mainly captures the cell's slow changes. The accuracy is good enough, but the deployment requirement, the need for a model that operates continuously on modest hardware and gives timely estimates, is not met by one that needs three weeks of input. This discrepancy, an accurate model with an impractical input requirement, is the main outcome of this chapter and the reason for the further research discussed in Section 5.11.

It is helpful to understand why the long window is effective, as this insight leads to a clear solution. A long window covers many charge and discharge cycles, allowing the cell's health-related behavior to repeat and build up. The model sees multiple examples of how the cell reacts to load, and the slow shifts in those responses across the window capture degradation. In contrast, a short fixed-time window may mostly be during rest periods or contain only parts of a single cycle. As a result, it has less of the repeating and drifting

behavior that the model needs, leading to poorer estimates. Therefore, the effectiveness of the long window is not coincidental; it gives the model enough instances of the relevant behavior to balance out noise and capture decline. This understanding points directly to the solution: if a short window could provide the same density of useful behavior as a long window, the accuracy could be maintained with a fraction of the input. Cycle-aware windowing addresses this directly: fitting windows to charge/discharge events concentrates the informative behavior.

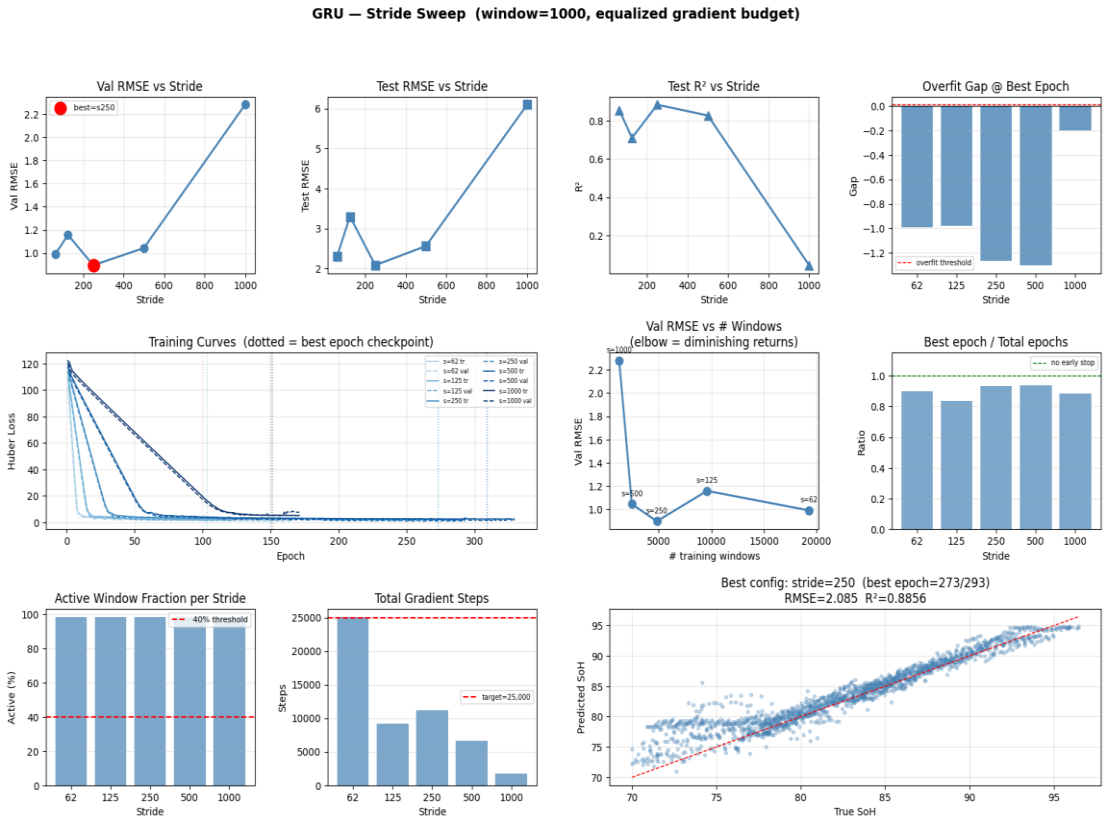


Figure 5.7 GRU model window stride search: RMSE vs Stride, R^2 vs Stride, Training curves and Predicted vs True SOH

5.10.4 Error analysis

Beyond the aggregate metrics, the residuals were examined to understand where the strongest model errs. The analysis follows the diagnostics standard for regression: the predicted values plotted against the true values to reveal systematic bias, the residuals plotted against the prediction to reveal heteroscedasticity, the distribution of residuals to reveal skew and heavy tails, and the absolute error plotted against the true SoH to reveal

whether the model is weaker at particular health levels (Figure 5.8). The expectation, consistent with the skewed label distribution of Section 5.6.6, is that the error is smallest in the densely populated middle of the SoH range and grows toward the sparsely populated healthy and heavily aged extremes, where the model has seen fewer examples. Aggregate error masks per-level performance: error grows toward the sparsely populated healthy and aged extremes, where end-of-life decisions are made. The per-level analysis is the check that the headline number is not carried by the easy middle of the range.

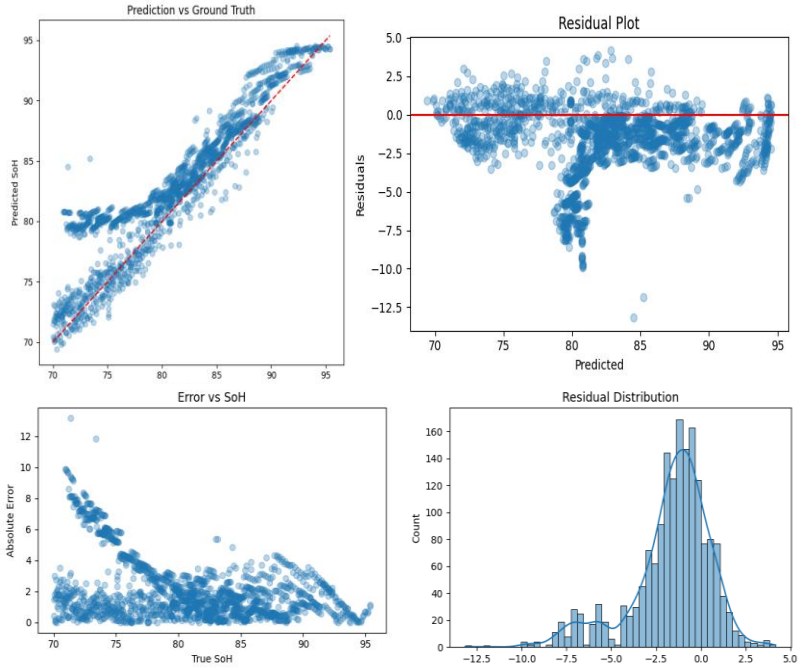


Figure 5.8 Error analysis for the GRU model on test cells: predicted against true State-of-Health, residuals against prediction, the residual distribution, and absolute error against true State-of-Health.

5.10.5 Comparison with the state of the art

BatteryGPT reports 0.21% RMSE in SoH points, a lower error than the 2.21-point MAE reported here. The comparison is not like-for-like, and the differences are the substance of this use case's contribution. BatteryGPT is evaluated on 4 held-out cells of a single lithium-iron-phosphate chemistry aged under one fast-charging protocol; the model here is evaluated on held-out cells drawn from 76 NMC/C-SiO cells aged under calendar, cyclic, and driving-profile conditions across four temperatures. BatteryGPT reaches its figure by first generating a cell's entire future charging trajectory with a 42-million-parameter transformer and then estimating health from the generated signal; the model here uses 11

million parameters and regresses health directly from observed signals. BatteryGPT is given the first 30% of a cell's lifecycle as context.

The advancement claimed here is therefore not a lower error. It is a 4× smaller, single-stage model that generalizes across three aging regimes and four temperatures under whole-cell validation, reaching $R^2 = 0.93$ on cells it has never seen. That is a different and harder generalization target than a single charging protocol, and it is the target a battery management system in a real fleet actually faces.

5.11 The frontier: cycle-aware windowing and per-battery normalization

The key finding in Section 5.10 is that the model is accurate but requires a long window. The two approaches discussed here directly address this issue and are currently the focus of the work. They are presented as ongoing developments, not finished outcomes, allowing for additional findings to be integrated as they arise.

Before discussing potential solutions, it is important to clarify what would constitute a fair way to report progress on the window-length problem. There is a real temptation to highlight flattering numbers. If a shorter window achieves similar error rates just because it was tested on an easier split or on windows from the cells the model was trained on, that would not represent any real progress. The improvement must be shown under the same cell-grouped protocol that led to the original results. This ensures the comparison is valid, measuring a long window against a short window on equal terms. Similarly, a short window achieving akin error rates only by being applied to an easier subset of cells would be deceptive; evaluations must include the same diverse population. Therefore, this frontier work must meet the same standard as the earlier efforts: any claim that a shorter window maintains accuracy must rely on a cell-grouped evaluation across the entire dataset, and the chapter is organized so such claims can be made when supported by the work.

5.11.1 The window-length problem restated

The long window is not just a random issue; it stems from how windows are currently created. A window of fixed time length, taken at a fixed stride along the signal, considers the cell's history as a uniform stream and asks the model to assess health over a certain period. Because the health-related features of the signal are thinly spread over long stretches of rest and slow drift, a long duration is necessary to gather enough of that information. This is why the accurate setup resulted in a three-week window. Reducing the window under the present method leads to a loss of accuracy as it provides the model with less of the same sparse information. The solution is to redefine what a window is, so that a shorter window contains a greater amount of relevant information.

5.11.2 Cycle-aware windowing

The first approach is to create windows based on charge and discharge cycles instead of clock time. A cycle represents a battery's natural unit of life. The health-related behavior, including voltage response to charging and temperature rises under load, occurs mainly during charge and discharge events. A cycle-aware window centers on these events, ensuring each window contains a comparable amount of important behavior regardless of the total wall-clock time it covers. This offers two advantages. It reduces the reliance on specific timestamps. This is beneficial since the same health can happen at different times in distinct cells. It also allows for shorter cycles that are uniformly informative, unlike the fixed three-week intervals, while still providing enough signal for accurate assessment. The expectation is that cycle-aware windows will enable the best model to maintain accuracy with a smaller and more practical input, addressing the shortcomings of the fixed-time window. Literature on cycle-based segmentation supports this approach, and studies suggest that accumulated-cycle indexing can outperform time-based splitting for related prognostic tasks.

Defining a cycle-aware window introduces practical challenges that this work will address. First, cycles must be identified in the signal, meaning transitions between rest, charge, and discharge must be detected from current and voltage traces. This segmentation is easy in clean lab settings but more complex under the varied loads in the dataset. Once cycles are pinpointed, a window can be defined to span a set number of cycles instead of a fixed duration, ensuring that every window captures a comparable amount of charge and discharge behavior, irrespective of the time it covers. This makes the windows unequal in time, this is the desired outcome: cells that cycle quickly and those that cycle slowly will provide windows with similar informational content but differing lengths. The model will then learn from the behavior itself. An accumulated-cycle index, counting full cycles instead of calendar time, is the natural coordinate here. The central question this work will address is how few cycles a window can encompass while still achieving the accuracy of the long fixed-time window, with hopes that the answer will be small enough to make the estimate practical.

5.11.3 Per-battery normalization

The second approach focuses on normalization. The current process standardizes each channel using statistics pooled from all cells, which creates a subtle but significant problem. A cell that naturally fluctuates widely can distort the shared scale, so that when such a scale is set, the smaller variations of quieter cells get compressed toward zero and lose their detail. Essentially, louder cells overpower the signals from quieter ones. Normalizing each cell, or experiment, by its specific statistics removes this imbalance,

allowing every cell's informative variation to have a comparable dynamic range before reaching the model. This principle helps cross-battery models manage cells with vastly different nominal capacities. It is expected to aid the model in understanding the shape of degradation instead of just the absolute size of any individual cell's signals, which should enhance generalization to previously unseen cells. One trade-off to consider is that per-cell normalization may discard absolute differences between cells that could contain important health information, so both methods will be compared instead of simply replacing one with the other.

Per-battery normalization also interacts with the cell-grouped validation in a way that requires careful consideration. If each cell is normalized using its own statistics, then at inference time, a held-out cell must be normalized using statistics derived from its own data, without referencing the training cells. This reflects how a Battery Management System (BMS) would manage a real cell. Statistics can be gathered over time as the cell is observed, allowing normalization to adjust to each cell. This online, per-cell normalization should be more aligned with deployment and is anticipated to be more resilient to the cell-to-cell variation revealed in the exploratory analysis. Therefore, the comparison will involve three approaches: global normalization as currently used, per-cell normalization calculated offline, and per-cell normalization collected online. The last method is the most aligned with real-world scenarios and is likely to reflect how the model would perform in practical situations.

In summary, cycle-aware windowing and per-battery normalization address the two main weaknesses highlighted by the current findings: the impractical window length and the limitations in cross-cell generalization that come from global normalization. These represent the immediate next steps, and the outcomes of implementing them will ultimately determine if the competitive accuracy already demonstrated can be achieved with a sufficiently short window and stable normalization suitable for application in a real battery management system.

5.12 Toward deployment: model cost and edge inference

This thesis focuses on real-time, safety-critical, edge-deployed signal processing. The cost of each model is just as important as its accuracy during evaluation. A State-of-Health estimator for a battery management system must fit within the memory and compute limits of an embedded processor. It also needs to produce estimates quickly enough to be useful. These constraints favor the compact models emphasized in this work. The gradient-boosting baseline and the convolutional network are the lightest models. The recurrent and hybrid models are heavier, but they are still modest. All models are smaller than the generative transformer (kB instead of MB) used for comparison. This ordering is important

because the most accurate model, the convolutional-recurrent hybrid, could realistically run on actual hardware, if its input requirements are properly managed; this is the focus of the frontier work.

Research on edge deployment of battery models shows techniques that can help a model transition to a battery management system. These include reducing weight precision via quantization, removing unnecessary structure through pruning, and distilling a larger model into a smaller one that mimics it. These same techniques were applied in the road-surface use case in Chapter 3, which created a model of a few hundred kilobytes. This connection is intentional: the methodology sees edge optimization as a stage that applies across all three use cases, with the road-surface chapter serving as a practical example. For the battery model, this stage is future work. A concrete next step is to measure the size, latency, and energy per inference of the compact models on a typical embedded target. This approach turns design-based deployability into deployability shown through measurement. Reported edge battery-health systems achieve inference in tens of milliseconds after quantization on modest hardware. This suggests that the compact models here can support real-time on-device operation once the window length issue is resolved.

5.13 Uncertainty and the value of knowing when not to trust the estimate

A State-of-Health estimate without any indication of its reliability is less useful in a safety-critical context. Consumers of the estimate, whether human operators or automated controllers, cannot know if they should act on it. This aspect of uncertainty quantification runs through the three use cases of this thesis and is particularly important for battery health. The exploration analysis showed that cells vary from one another. Consequently, a model will likely be more confident about cells similar to those it trained on compared to unusual ones. A model capable of attaching a calibrated confidence level to each estimate would allow a battery management system to treat a low confidence estimate cautiously. It could defer a decision or request a reference measurement rather than acting on an untrustworthy number.

Several methods to achieve such confidence align with the models created here. An ensemble of models trained with different initializations provides uncertainty measures through the variation in their predictions. This variation increases in areas where the models disagree, usually with unfamiliar inputs. Keeping dropout active during inference approximates a similar effect by sampling slightly different sub-models. A model can also be trained to predict not just a single value but a distribution, providing both an estimate and a spread. This way, it learns to widen the spread when uncertain. None of these methods are part of the current work, but they are noted as a natural progression for carrying uncertainty quantification into a delivered feature. They are categorized among

future directions because, for a safety-critical health estimate, a reliable statement of confidence is just as valuable as the estimate itself.

5.14 Discussion

This chapter establishes several key points while leaving some questions open, and it is important to distinguish between them. What has been established is a complete and thoroughly evaluated modeling pipeline. A range of models was built, spanning gradient boosting, convolution, recurrence with attention, and convolutional-recurrent hybrids. These models were tuned through automated search and validated using a protocol that tests generalization to unseen batteries. The convolutional gated-recurrent hybrid emerged as the strongest model, estimating State-of-Health to within 2.21 percentage points (MAE, $R^2 = 0.93$) on held-out cells, using 11 million parameters against the 42 million of the generative state of the art. The central finding is the cost of that accuracy: a 23-day input window, which is too long for the continuous on-device estimation a battery management system requires. This shows that a compact, direct, discriminative model can perform well on a diverse dataset under rigorous evaluation without relying on a generative transformer [51].

The model comparison itself offers valuable insights. The convolutional-recurrent hybrid outperforms both the pure convolutional and pure recurrent models, while both deep learning families outperform the gradient-boosting baseline on flattened windows. This shows that the temporal structure of the signal carries real health information. A successful model must capture both the local shape of the data and its longer-term evolution. The gradient-boosting baseline, despite being strong for tabular problems, suffers from losing the explicit ordering of data when the window is flattened. The gap between it and the deep learning models highlights the value of that ordering. The similarity in performance between gated-recurrent and long-short-term-memory variants, which differ only in cell type, suggests that the choice of cell type is a secondary consideration once attention and the convolutional front-end are established. These relationships remain stable across fine-tuning, which makes them reliable while the exact numbers are still being settled.

The deployment question remains open, and this chapter does not shy away from addressing it. The accurate model requires a three-week window, which does not fit with the continuous and timely on-device estimation that this thesis focuses on. Clearly naming this issue is more beneficial than simply reporting accuracy. It identifies what needs to change and points to two approaches, cycle-aware windowing and per-battery normalization, that aim to address this. The comparison with the state of the art supports this view: BatteryGPT achieves much lower error under stricter conditions with a more complex method and more data. The current approach trades error for realism and

deployability: 11M parameters against BatteryGPT's 42M, on a dataset spanning three aging regimes

Two limitations frame the conclusions. The results represent a snapshot of ongoing work, and exact figures will change as windowing and normalization improve. Therefore, this discussion focuses on the stable relationships between models instead of individual values, which are not yet final. Additionally, the cross-cell domain shift shown in the exploratory analysis is real and not yet fully resolved. If the frontier methods do not close the generalization gap, the domain adaptation techniques noted in the related work are a natural next step. However, neither limitation undermines the central contribution, this is a validated and theoretically deployable pipeline and a clear recognition of the key obstacle, window length, that stands between it and practical use.

5.15 Conclusion and future work

This use case built a family of deep-learning estimators for battery State-of-Health on a large and diverse public aging dataset, tuned them by automated search, and evaluated them under a protocol that holds out whole cells so that the reported errors reflect generalization. The convolutional gated-recurrent hybrid emerged as the strongest model, reaching a competitive root-mean-square error of about 0.62 (normalized) on held-out cells in an early configuration, which shows that a compact, direct, discriminative model on diverse data can approach the accuracy the field values, in contrast to the heavier generative pipeline of the current state of the art. The central finding is the tension that accompanies that accuracy: the model requires a window roughly three weeks long, but too long for the continuous, timely, on-device estimation a real battery management system need. The pipeline is therefore established and accurate in principle, and the open problem is squarely one of practicality: delivering the accuracy with a window short enough to deploy.

Stepping back, the contribution of this use case to the thesis is best stated as a set of demonstrated capabilities and one clearly bounded open problem. The demonstrated capabilities are a complete pipeline from raw cell signals to a State-of-Health estimate, a family of models that spans the methods the field uses, an automated tuning procedure that selects each model for its ability to generalize, and a validation protocol that reports errors which mean what they appear to mean because they are measured on cells the model never saw. The strongest model reaches an error the field regards as good under that protocol, on a dataset chosen for its difficulty, which substantiates the thesis claim that a compact, direct model on diverse data can be competitive with a heavy generative approach on narrow data. The bounded open problem is the window length, and naming it precisely, not reporting the accuracy and leaving the cost implicit, is itself part of the contribution, because it converts

a vague sense that the model could be more practical into a specific quantity to reduce and a specific pair of techniques to reduce it.

The immediate future work is the frontier of Section 5.11. Cycle-aware windowing redefines the input around charge and discharge cycles so that a short window carries as much health-bearing structure as a long fixed-time window does now, which should shrink the input requirement from weeks to a practical span while preserving accuracy. Per-battery normalization standardizes each cell against its own statistics so that loud cells no longer crush the signal of quiet ones, which should improve generalization across the cell-to-cell variation the exploratory analysis revealed. These two changes are designed to convert the current accurate-but-impractical model into an accurate-and-deployable one and applying them is the next concrete step.

Beyond that step, three directions extend the work in the spirit of the thesis. The first is to confront the cross-cell generalization gap directly with transfer learning or domain adaptation if the frontier changes do not close it on their own, aligning the feature distributions of different cells so that knowledge transfers. The second, consistent with the forward-looking dimension of the methodology in Chapter 2, is to attach calibrated uncertainty to the SoH estimate, so that a low-confidence prediction on an unusual cell or an unfamiliar operating regime is flagged, which for a safety-critical health estimate is as valuable as the estimate itself. The third is to carry the compact model onto representative battery-management hardware and measure its size, latency, and energy per inference, turning the deployability argued here by design into a deployability demonstrated by measurement, and completing the edge-deployment stage of the pipeline for this use case as it was completed for the first. The exploratory spectrogram autoencoder remains a side avenue that may or may not bear fruit and is not on the critical path. Together these steps lead from a validated, competitive, but window-bound estimator toward a battery State-of-Health model that is accurate, generalizes across cells, reports its own confidence, and runs in real time on the hardware that manages the battery.

6 Conclusion

This thesis aimed to demonstrate that one method, a five-stage process for extracting hidden states from noisy real-time signals while considering edge and safety constraints, could apply to three very different problems and perform well in each. The three use cases were not selected for their surface similarities, since a road, a driver, and a battery are quite different, but because they share a fundamental challenge: estimating a quantity that cannot be measured directly from signals that can, quickly and affordably enough to be viable wherever decisions are made. This final chapter ties the three cases together, reflects on the successes and shortcomings of the shared method, and highlights two key directions for the thesis.

6.1 What the three use cases established

The first use case, road surface classification, is the clearest example of the method and the only one that completes the final stage of the process with a measurable outcome. A dual-branch quantized network, which combines a raw-signal convolutional path with a statistical-feature path, categorizes three surface types with a macro-averaged F1-score of 93.9% while occupying 354 kilobytes, about a hundred times smaller than the baseline it outperforms. Because it has been published and quantized to a relatively small size, it is the worked example of the thesis, providing a practical example that demonstrates the edge-deployment stage against which the other two use cases are assessed.

The second use case, the driver monitoring system, tested elements of the methodology related to representation, real-time fusion, and validation. Its contributions are twofold, with each part kept separate on purpose. The pipeline fuses the vehicle-dynamics stream from a simulator with the gaze stream from an eye-tracker, synchronizing them and running a compact classifier in real time. This part of the process was completed and strengthened by a thorough audit that identified and fixed several silent defects. However, the model running inside it is inherited and presents an ongoing challenge: it achieves about 90% accuracy and 91% F1 offline for known drivers, but this drops to around 80% accuracy and 78% F1 for new drivers in live tests, revealing a generalization gap that the chapter discusses openly. This use case contributes to the thesis both methodologically and empirically, as it represents the point where the disciplines of validation and detection of silent failures were most rigorously applied.

The third use case, battery State-of-Health estimation, applied the methodology to a slowly varying hidden state and a hard generalization problem. A family of models spanning gradient boosting, convolution, recurrence with attention, and convolutional-recurrent

hybrids was built, tuned, and evaluated under a protocol that holds out entire cells. The strongest, a convolutional gated-recurrent hybrid with 11 million parameters, estimates health to within 2.21 percentage points on unseen batteries ($R^2 = 0.93$), against the 42-million-parameter generative state of the art. Its finding is a stated cost: the accuracy requires a 23-day window, too long to deploy, which is what cycle-aware windowing and per-battery normalization are designed to fix.

6.2 What the shared methodology delivered

Together, the three use cases support the central argument that the methodology is cross-domain. The same five stages, signal acquisition and fusion, pre-processing and representation, compact model design, systematic validation and failure analysis, and edge-oriented optimization, structured all three cases, and the same design principles emerged: representation and validation deserve equal attention to architecture, models should fail loudly instead of quietly, and a model's cost is part of its evaluation. Each use case highlighted a different stage. Road surface classification advanced the deployment stage the furthest, driver monitoring emphasized validation and the detection of silent failures, and battery health directly faced the generalization challenge. Collectively, they utilized the entire pipeline, and the fact that the same framework fits a fast vibration signal, a pair of fused real-time streams, and a year-long degradation path demonstrates the validity of the abstraction.

The methodology also proved beneficial in identifying weaknesses. By treating validation as a primary stage, the generalization gap in driver monitoring was evident as a measurable difference between known and unknown drivers. Similarly, the battery work, by focusing on held-out cells, made its window-length limitation clear as a measured cost instead of hiding behind an attractive random split. A methodology that exposes its limits is more valuable than one that only celebrates successes; in each use case, the unresolved issues are stated as clearly as the results.

6.3 The two cross-cutting directions

Two themes emerge across all three use cases as future directions for the overall work.

The first is edge deployment based on measurements, not arguments. The road-surface use case is the only instance where the deployment stage resulted in a small, quantized artifact, setting a model for the others. The driver-monitoring pipeline and battery models are designed to be compact and compatible with the same compression techniques, such as quantization, pruning, and distillation. However, their readiness for edge deployment has been argued based on their size. Implementing each on representative embedded hardware

and recording inference latency, sustained throughput, and energy per inference would shift from theoretical deployability to proven deployability and complete the fifth stage of the process for all three cases, just as it was completed for the first.

The second is uncertainty quantification. In every use case, a single prediction is insufficient for safety-critical decisions. A surface label, a distraction flag, or a health estimate is much more valuable when it includes a calibrated trust level. This ensures that a low-confidence output on an unseen input is flagged for caution instead of presented with false confidence. Calibrated uncertainty is not delivered here. The architecture retains dropout and are small enough to ensemble, so it can be added without redesign, but it lays the groundwork intentionally: the architecture maintains enough compactness to allow for practical ensembling and retain dropout layers that make Monte Carlo estimation feasible. Adding calibrated confidence to the outputs is identified across all three cases as the top priority that would increase their value in the safety-critical contexts they aim to serve.

6.4 Closing remark

This thesis is set out to show that road surface classification, driver distraction detection, and battery State-of-Health estimation are one problem, and that one methodology addresses all three. Each use case bears this out and each contributes a specific, measurable result: a published, quantized road-surface classifier at 93.9% F1 and 354 kB, about 128× smaller than the method it outperforms; an audited, deployable driver-monitoring pipeline that exposed a ten-point generalization gap invisible to offline evaluation; and an 11-million-parameter battery estimator reaching 2.21 points MAE and $R^2 = 0.93$ on unseen cells across three aging regimes, against a 42-million-parameter benchmark tested on one.

The common lesson is the one the methodology was built to surface: in all three cases, the representation and the validation protocol determined the outcome at least as much as the architecture, and in two of the three, the accuracy reported by prior work did not survive a deployment-facing evaluation. The two limits that remain, a 23-day window in the battery case, and uncertainty that is not yet calibrated in any of the three, are stated precisely because each defines the next experiment rather than a vague direction.

7 References

[1] V. Sb, *Edge AI: A Comprehensive Survey of Technologies, Applications, and Challenges*. 2024, p. 6. doi: 10.1109/ACET61898.2024.10730112.

[2] R. Cordova-Cardenas, D. Amor, and Á. Gutiérrez, "Edge AI in Practice: A Survey and Deployment Framework for Neural Networks on Embedded Systems," *Electronics*, vol. 14, no. 24, p. 4877, Jan. 2025, doi: 10.3390/electronics14244877.

[3] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles," Nov. 04, 2017, *arXiv*: arXiv:1612.01474. doi: 10.48550/arXiv.1612.01474.

[4] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," Oct. 04, 2016, *arXiv*: arXiv:1506.02142. doi: 10.48550/arXiv.1506.02142.

[5] V. Alves de Souza, "Asphalt pavement classification using smartphone accelerometer and Complexity Invariant Distance," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 198–211, Sep. 2018, doi: 10.1016/j.engappai.2018.06.003.

[6] "(1) (PDF) Neural Network Quantization for Microcontrollers: A Comprehensive Survey of Methods, Platforms, and Applications." Accessed: Jun. 10, 2026. [Online]. Available: https://www.researchgate.net/publication/394831145_Neural_Network_Quantization_for_Microcontrollers_A_Comprehensive_Survey_of_Methods_Platforms_and_Applications

[7] Y. Wang, T. Wang, and T. Yue, "Uncertainty propagation from sensor data to deep learning models in autonomous driving," *Information and Software Technology*, vol. 183, p. 107735, Jul. 2025, doi: 10.1016/j.infsof.2025.107735.

[8] "(1) (PDF) A Dual Branch Quantized Neural Network for Road Surface Classification Using Smartphone Sensors." Accessed: Jun. 10, 2026. [Online]. Available: https://www.researchgate.net/publication/400984960_A_Dual_Branch_Quantized_Neural_Network_for_Road_Surface_Classification_Using_Smartphone_Sensors

[9] "(1) Tiny Machine Learning: Progress and Futures [Feature] | Request PDF." Accessed: Jun. 17, 2026. [Online]. Available: https://www.researchgate.net/publication/385451701_Tiny_Machine_Learning_Progress_and_Futures_Feature

[10] "(1) (PDF) From Tiny Machine Learning to Tiny Deep Learning: A Survey." Accessed: Jun. 17, 2026. [Online]. Available: https://www.researchgate.net/publication/397576850_From_Tiny_Machine_Learning_to_Tiny_Deep_Learning_A_Survey

[11] "Battery aging dataset (result data, v2)." Accessed: Jun. 18, 2026. [Online]. Available: https://www.kaggle.com/datasets/matthiasluh/battery-aging-dataset-result-data-v2

[12] M. Luh and T. Blank, “Comprehensive battery aging dataset: capacity and impedance fade measurements of a lithium-ion NMC/C-SiO cell,” *Sci Data*, vol. 11, no. 1, p. 1004, Sep. 2024, doi: 10.1038/s41597-024-03831-x.

[13] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, “1D convolutional neural networks and applications: A survey,” *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, Apr. 2021, doi: 10.1016/j.ymssp.2020.107398.

[14] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” *Data Min Knowl Disc*, vol. 33, no. 4, pp. 917–963, Jul. 2019, doi: 10.1007/s10618-019-00619-1.

[15] A. Vaswani *et al.*, “Attention Is All You Need,” Aug. 02, 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.

[16] “A survey of uncertainty in deep neural networks | Artificial Intelligence Review | Springer Nature Link.” Accessed: Jun. 18, 2026. [Online]. Available: https://link.springer.com/article/10.1007/s10462-023-10562-9

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *journal of machine learning research*, vol. 18, no. 187, pp. 1–30, 2018.

[18] I. Hubara, D. Soudry, and R. E. Yaniv, “Binarized Neural Networks,” Mar. 10, 2016, *arXiv*: arXiv:1602.02505. doi: 10.48550/arXiv.1602.02505.

[19] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things Journal*, vol. 3, pp. 1–1, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[20] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On Calibration of Modern Neural Networks,” Aug. 03, 2017, *arXiv*: arXiv:1706.04599. doi: 10.48550/arXiv.1706.04599.

[21] “The Road to Safety: A Review of Uncertainty and Applications to Autonomous Driving Perception.” Accessed: Jun. 18, 2026. [Online]. Available: https://www.mdpi.com/1099-4300/26/8/634

[22] E. Raslan, M. Alrahmawy, Y. Mohammed, and A. Tolba, “Evaluation of data representation techniques for vibration based road surface condition classification,” *Scientific Reports*, vol. 14, May 2024, doi: 10.1038/s41598-024-61757-1.

[23] A. Allouch, A. Koubaa, T. Abbes, and A. Ammar, “RoadSense: Smartphone Application to Estimate Road Conditions Using Accelerometer and Gyroscope,” *IEEE Sensors Journal*, vol. PP, May 2017, doi: 10.1109/JSEN.2017.2702739.

[24] N. Cong, D.-N. Tran, T. Long, G. M. T. NGUYEN, and D.-T. Tran, “Hybrid Feature Selection for Real-Time Road Surface Classification on Low-End Hardware: A Machine Learning Approach,” *Results in Engineering*, vol. 27, pp. 1–12, Jun. 2025, doi: 10.1016/j.rineng.2025.105693.

[25] C. Li, Q. Lin, J. Zhang, Y. Liu, G. Shao, and Q. Zhu, *An Inertial Sensor-Based Deep Learning Framework for Road Surface Type Classification in Intelligent Vehicle Systems*. 2026. doi: 10.21203/rs.3.rs-8526318/v1.

[26] T. Dózsa *et al.*, "Road Type Classification Using Time-Frequency Representations of Tire Sensor Signals," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2024, doi: 10.1109/ACCESS.2024.3382931.

[27] A. Dabbous, M. Fresta, F. Bellotti, and R. Berta, "Neural Architecture for Tennis Shot Classification on Embedded System," 2024, pp. 97–102. doi: 10.1007/978-3-031-48121-5_14.

[28] A. Dabbous, L. Lazzaroni, F. Sakr, R. Berta, M. Fresta, and F. Bellotti, *Binary Neural Networks for Real-Time Structural Health Monitoring on Edge Devices*. 2025, p. 4. doi: 10.1109/ICCE63647.2025.10930047.

[29] L. Manoni, S. Orcioni, and M. Conti, "Recent Advancements in Deep Learning Techniques for Road Condition Monitoring: A Comprehensive Review," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2024, doi: 10.1109/ACCESS.2024.3481649.

[30] "Time Series Classification Website." Accessed: Jun. 19, 2026. [Online]. Available: https://timeseriesclassification.com/description.php?Dataset=AsphaltPavementType

[31] M. Fresta *et al.*, "Deep Learning-Based Real-Time Driver Cognitive Distraction Detection," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2025, doi: 10.1109/ACCESS.2025.3539392.

[32] "Global status report on road safety 2023." Accessed: Jun. 23, 2026. [Online]. Available: https://www.who.int/teams/social-determinants-of-health/safety-and-mobility/global-status-report-on-road-safety-2023

[33] "(PDF) Driver Distraction Detection Methods: A Literature Review and Framework." Accessed: Jun. 23, 2026. [Online]. Available: https://www.researchgate.net/publication/350900026_Driver_Distraction_Detection_Methods_A_Literature_Review_and_Framework

[34] M. Q. Khan and S. Lee, "A Comprehensive Survey of Driving Monitoring and Assistance Systems," *Sensors*, vol. 19, no. 11, p. 2574, Jan. 2019, doi: 10.3390/s19112574.

[35] A. Fernández, R. Usamentiaga, J. L. Carús, and R. Casado, "Driver Distraction Using Visual-Based Sensors and Algorithms," *Sensors*, vol. 16, no. 11, p. 1805, Nov. 2016, doi: 10.3390/s16111805.

[36] S. Kaplan, A. Guvensan, A. Yavuz, and Y. Karalurt, "Driver Behavior Analysis for Safe Driving: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, pp. 1–16, Aug. 2015, doi: 10.1109/TITS.2015.2462084.

[37] Q. Abbas and A. Alsheddy, "Driver Fatigue Detection Systems Using Multi-Sensors, Smartphone, and Cloud-Based Computing Platforms: A Comparative Analysis," *Sensors*, vol. 21, Dec. 2020, doi: 10.3390/s21010056.

[38] J. Hu *et al.*, “Early prediction of lithium-ion battery degradation with a generative pre-trained transformer,” *Nature Communications*, vol. 17, Dec. 2025, doi: 10.1038/s41467-025-66819-0.

[39] S. O’Kane *et al.*, “Lithium-ion battery degradation: how to model it,” *Physical Chemistry Chemical Physics*, vol. 24, Mar. 2022, doi: 10.1039/D2CP00417H.

[40] P. Attia *et al.*, “Review—"Knees" in Lithium-Ion Battery Aging Trajectories,” *Journal of The Electrochemical Society*, vol. 169, Jun. 2022, doi: 10.1149/1945-7111/ac6d13.

[41] O. Demirci, S. Taskin, E. Schaltz, and B. Acar Demirci, “Review of battery state estimation methods for electric vehicles-Part II: SOH estimation,” *Journal of Energy Storage*, vol. 96, p. 112703, Aug. 2024, doi: 10.1016/j.est.2024.112703.

[42] N. Popović, “Artificial Intelligence Approaches to Battery Health Assessment: Opportunities, Challenges and Future Directions,” *IJEEC - INTERNATIONAL JOURNAL OF ELECTRICAL ENGINEERING AND COMPUTING*, vol. 9, Dec. 2025, doi: 10.7251/IJEEC2502089P.

[43] “(PDF) Electrochemical model-based state estimation for lithium-ion batteries with adaptive unscented Kalman filter.” Accessed: Jul. 02, 2026. [Online]. Available: https://www.researchgate.net/publication/343901991_Electrochemical_model-based_state_estimation_for_lithium-ion_batteries_with_adaptive_unscented_Kalman_filter

[44] “A novel state-of-health estimation for the lithium-ion battery using a convolutional neural network and transformer model | Request PDF.” Accessed: Jul. 02, 2026. [Online]. Available: https://www.researchgate.net/publication/363795735_A_novel_state-of-health_estimation_for_the_lithium-ion_battery_using_a_convolutional_neural_network_and_transformer_model

[45] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” May 19, 2016, *arXiv*: arXiv:1409.0473. doi: 10.48550/arXiv.1409.0473.

[46] “(PDF) Machine learning pipeline for battery state-of-health estimation.” Accessed: Jul. 01, 2026. [Online]. Available: https://www.researchgate.net/publication/350641190_Machine_learning_pipeline_for_battery_state-of-health_estimation

[47] K. Tran *et al.*, “HybridoNet-Adapt: A Domain-Adapted Framework for Accurate Lithium-Ion Battery RUL Prediction,” Apr. 18, 2025, *arXiv*: arXiv:2503.21392. doi: 10.48550/arXiv.2503.21392.

[48] K. A. Severson *et al.*, “Data-driven prediction of battery cycle life before capacity degradation,” *Nat Energy*, vol. 4, no. 5, pp. 383–391, May 2019, doi: 10.1038/s41560-019-0356-8.

[49] “XGBoost: A Scalable Tree Boosting System.” Accessed: Jul. 02, 2026. [Online]. Available:

https://www.researchgate.net/publication/310824798_XGBoost_A_Scalable_Tree_Boosting_System

[50] "Optuna: A Next-generation Hyperparameter Optimization Framework." Accessed: Jul. 02, 2026. [Online]. Available: https://www.researchgate.net/publication/334712487_Optuna_A_Next-generation_Hyperparameter_Optimization_Framework

[51] T. Bai and H. Wang, "Convolutional Transformer-Based Multi-View Information Perception Framework for Lithium-ion Battery State-of-Health Estimation," *IEEE Transactions on Instrumentation and Measurement*, vol. PP, pp. 1–1, Jan. 2023, doi: 10.1109/TIM.2023.3300451.

A. Appendix A: The fourteen driver-monitoring pipeline defects

This appendix lists in full the fourteen defects identified during the systematic audit of the driver-monitoring pipeline described in Chapter 4. The pipeline ran without crashing and produced plausible output, yet each of these defects corrupted the data, silenced a failure, or hid a fault from inspection. This is the class of silent error that is most dangerous in a safety-relevant system because it looks like correct operation. The defects are grouped into the three categories used in Chapter 4: thread survival and silent failures, correctness of the data, and visibility of faults. Table A.1 gives complete enumeration, each entry stating the defect, its effect before it was fixed, and how it was resolved.

#	Defect	Category	Effect before fix	Resolution
1	No exception handling in the vehicle-dynamics listener thread	Thread survival	A single malformed packet raised an unhandled exception, killed the thread, and left the main loop running on stale data with no warning	Wrapped the listener loop in exception handling that logs the fault and continues
2	No exception handling in the eye-tracker listener thread	Thread survival	Same silent-death failure mode on the second stream; gaze data froze while the system appeared to run	Added the same protective exception handling and per-packet recovery
3	Unsafe numeric cast on head-rotation values	Thread survival	A head-rotation field that was not a clean float raised a cast error that propagated up and stalled the thread	Validated and safely coerced the value, discarding malformed samples
4	Stale-data loop continuation after a listener stops	Thread survival	When a listener died, the inference loop kept producing confident outputs from the last good frame	Added liveness checks so a stalled stream is detected and flagged

5	Parser split key-value pairs on the wrong delimiter	Data correctness	Any key containing an underscore, such as the velocity components, was corrupted, so those features never reached the model correctly	Corrected the delimiter so all keys parse intact
6	Scaler applied to a flattened array instead of row by row	Data correctness	The normalization mixed feature columns, shifting the input distribution the model received away from the training distribution	Applied the scaler per row so each feature is normalized against its own statistics
7	Eye-tracker ID field never renamed to the expected key	Data correctness	A required feature was absent under the name the model expected, so it was silently treated as missing	Renamed the field to match the model's expected schema
8	One-hot categorical columns mean-filled on missing values	Data correctness	Missing categoricals were filled with a mean not a valid indicator, producing invalid one-hot rows	Replaced mean-fill with a correct indicator/zero fill for one-hot columns
9	Scaler applied before feature assembly was complete	Data correctness	Normalization ran on a partially built feature vector, scaling some fields against the wrong reference	Reordered the pipeline so scaling follows full feature assembly
10	Timezone-naive index used to resample timezone-aware timestamps	Visibility	Resampling silently misaligned the two streams in time, corrupting the fusion grid without raising an error	Made the time index timezone-aware throughout resampling
11	Deprecated timestamp call used for time handling	Visibility	The deprecated call risked silent behavior changes across library versions	Replaced it with the timezone-aware equivalent system-

				wide
12	Receive timestamp never written into the feature dictionary	Visibility	The timestamp was carried in the queue tuple but dropped before features were built, so diagnostics could not use it	Wrote the receive timestamp into the feature record
13	Empty diagnostic columns from the missing timestamp	Visibility	Downstream diagnostic columns were silently empty, hiding timing faults from inspection	Populated the diagnostic columns once the timestamp was preserved
14	Silent scaler, model, and prediction failures	Visibility	A failure in scaling, inference, or prediction produced no error and no output flag, so a broken pipeline looked healthy	Added explicit failure checks and logging at each stage so faults surface immediately

Table A.1 The fourteen audited defects of the driver-monitoring pipeline, grouped by the kind of harm they caused, with the effect of each before correction and the resolution applied. This is the full list summarized in Table 4.1.

Categories one to four fall under thread survival, five to nine under data correctness, and ten to fourteen under fault visibility. The audit that produced this list was led mainly by the author as part of the pipeline hardening, and the offline replay mode added afterward, which reads recorded packets through the same processing path at a controlled rate, made each defect reproducible so that a fix could be confirmed against the original faulty behavior.