Phase 1 Master Guide: AI Network Stabilization

Document Version: 1.0
Purpose: This document serves as a comprehensive educational guide for executing Phase 1 of the AI-Based Network Stabilization Tool project. It covers all required foundational knowledge in networking, signal processing, data science, and AI, complete with the relevant mathematical principles and implementation logic.

Table of Contents

Part 1: Foundational Concepts

Chapter 1: Networking for the AI Engineer

This chapter provides the essential domain knowledge required to understand the operational context and the data that will fuel the AI models. It establishes the fundamental characteristics of the hybrid network environment, defines the key metrics that quantify its performance, and categorizes the real-world events that our AI is designed to manage. A clear grasp of these concepts is the bedrock upon which a successful AI stabilization tool is built.

1.1: Introduction to Hybrid Networks: Wired vs. Satellite

A hybrid network is a communication infrastructure that combines two or more different types of connectivity technologies into a single, cohesive system.[1] The primary objective is to leverage the unique strengths of each technology while compensating for their respective weaknesses. This integration creates a more reliable, efficient, and cost-effective network by enabling intelligent, often automatic, switching between different communication paths.[1] For this project, the hybrid network consists of a terrestrial wired path and a Low Earth Orbit (LEO) satellite path, creating a system that balances high performance with global reach. The fundamental trade-offs between these two links are not merely technical details; they are the direct motivation for the dual-AI strategy at the core of this project. The wired path's high stability and low latency establish a clear baseline of "normal" or optimal performance. The satellite path, while providing essential connectivity, introduces known and predictable performance variations, such as those from satellite handoffs or weather effects. These predictable patterns are ideal for a supervised, predictive model designed to learn from historical examples. Conversely, both systems are susceptible to catastrophic and unforeseen failures—such as a physical fiber cut or a latent software bug in the satellite's core

network—that do not follow predictable patterns. These "unknown unknowns" necessitate an unsupervised anomaly detection model that can identify any significant deviation from normalcy without having seen a specific failure type before. Thus, the very nature of the hybrid network dictates that a dual-AI approach is the most robust and comprehensive solution for ensuring network stability.

Wired Networks (Fiber Optic, Cable)

Wired networks form the backbone of modern digital communication, relying on physical cables to transmit data. Among these, fiber optic technology stands out for its superior performance characteristics.

- **Characteristics and Strengths:** Fiber optic cables transmit data as pulses of light through thin strands of glass, a method that is fundamentally faster than transmitting electrons over copper wires.[3] This physical medium grants fiber optics immunity to electromagnetic interference (EMI), a common source of signal degradation in other wired and wireless systems.[3] The principal advantages of a fiber optic connection are its exceptionally high speed, vast bandwidth capacity, and consistently low latency. Speeds can reach up to 100 Gbps, and latency is typically in the range of 5-20 ms, making it the gold standard for applications requiring stable, high-performance connectivity.[3] Under normal operating conditions, fiber networks exhibit minimal packet loss, underscoring their high reliability.[5]
- **Weaknesses:** The primary disadvantages of wired networks are logistical and financial. The initial installation of physical cabling is labor-intensive and can be prohibitively expensive, especially over long distances or in complex terrain.[6] The cables themselves, particularly fiber, are more fragile than their copper counterparts and require careful handling to avoid damage.[8] Furthermore, their physical nature means they offer no mobility, rendering them unsuitable for connecting mobile assets or remote locations where laying cable is impractical.[6]

LEO Satellite Networks (e.g., Starlink)

LEO satellite networks represent a paradigm shift from traditional satellite internet, which relied on single geostationary (GEO) satellites orbiting at over 35,000 km. LEO systems, such as Starlink, employ a large "megaconstellation" of thousands of satellites orbiting much closer to Earth, at an altitude of approximately 550 km.[10] This proximity is the key technological innovation that enables significantly improved performance. Data travels from a user's terminal on the ground to the nearest satellite, potentially through a mesh of inter-satellite laser links, and down to a terrestrial ground station connected to the global internet.[9]

- **Characteristics and Strengths:** The foremost advantage of LEO satellite networks is their ability to provide near-global broadband coverage, reaching remote, rural, and

mobile users who are unserved by terrestrial infrastructure.[1] This low-altitude orbit dramatically reduces the signal travel time, resulting in a much lower latency (typically 25-60 ms) compared to the 600+ ms delays common with GEO satellites.[7] Modern LEO services offer respectable download speeds, often ranging from 50 to 220 Mbps, making them viable for streaming, video conferencing, and even some online gaming.[14]

- **Weaknesses:** The performance of LEO satellite links is inherently more variable than their wired counterparts. While latency is low for a satellite service, it is still higher than fiber and is subject to periodic spikes. The constant, rapid movement of the satellites relative to a stationary user on the ground necessitates frequent **handoffs**, where the user's connection is passed from one satellite to the next. These handoff events are a primary source of performance variability, often introducing brief but sharp increases in latency and jitter.[15] Furthermore, because the signal must travel through the atmosphere, it is susceptible to degradation from environmental factors like heavy rain, snow, or ice—a phenomenon known as **rain fade**.[14]

The following table provides an at-a-glance comparison of the key characteristics of these two network technologies.

Characteristic	Wired Network (Fiber Optic)	LEO Satellite Network (e.g., Starlink)
Speed	Extremely high (up to 100 Gbps) [3]	High (50 - 220 Mbps) [14]
Latency	Very low and stable (~5-20 ms) [7]	Low for satellite (~25-60 ms), but variable with spikes [7]
Jitter	Extremely low (<2 ms)	Higher and variable, with significant spikes during handoffs [16]
Packet Loss	Near 0% in normal conditions	Higher, with measurable loss (~1%) correlated with events [16]
Reliability	Very high, consistent performance	Lower, susceptible to atmospheric conditions and handoffs [7]
Key Vulnerabilities	Physical damage to cables, high installation cost [8]	Rain fade, handoff-induced instability, higher upfront equipment cost [14]

1.2: Key Performance Indicators (KPIs) Explained

Key Performance Indicators (KPIs) are the quantifiable, objective measurements used to track and evaluate the performance of a system against its defined goals.[19] For the AI Network

Stabilization Tool, KPIs are the vital signs of the network's health. They are the raw data streams that our AI models will ingest, analyze, and use to make decisions. Understanding what each KPI measures and why it matters is fundamental to interpreting the model's behavior and the state of the network.

Latency (Ping)

- **What it is:** Latency, commonly measured by a "ping" test, is the time it takes for a small data packet to travel from a source device to a destination server and back again. This round-trip time (RTT) is typically measured in milliseconds (ms).[20]
- **Why it matters:** Latency is a direct measure of a network's responsiveness. High latency results in noticeable delays, or "lag," which severely degrades the user experience in interactive, real-time applications such as online gaming, voice-over-IP (VoIP), and video conferencing.[14]
- **Typical Values:**
    - **Wired:** Consistently low, typically ranging from 5 ms to 20 ms.[7]
    - **LEO Satellite:** Low for a satellite connection, averaging 25 ms to 60 ms, but subject to predictable spikes during events like satellite handoffs.[7]

Jitter

- **What it is:** Jitter is the variation in latency over time. It measures the inconsistency of packet arrival times. If latency is the delay of a single packet, jitter is the difference in delay between successive packets.[20] For example, if three packets arrive with delays of 20 ms, 50 ms, and 25 ms, the network is experiencing high jitter.
- **Why it matters:** Jitter is particularly disruptive for streaming media applications like video calls and music streaming. These applications rely on a steady, predictable stream of data to fill a playback buffer. High jitter causes this buffer to either overflow or run empty, resulting in artifacts like choppy video, robotic-sounding audio, or complete dropouts.[20]
- **Typical Values:**
    - **Wired:** Very low, often less than 1-2 ms.
    - **LEO Satellite:** Significantly higher and more variable due to the constantly changing path length to moving satellites and the impact of handoffs, which can cause sharp, temporary spikes.[16]

Packet Loss

- **What it is:** Packet loss is the percentage of data packets that are sent by a source but

never arrive at their intended destination. It is a measure of a network's reliability.[20]

- **Why it matters:** The impact of packet loss depends on the transport protocol. For protocols like TCP (used for web browsing, file transfers), lost packets must be detected and retransmitted, which drastically reduces overall throughput and introduces significant delays.[20] For real-time protocols like UDP (used for VoIP, gaming), lost packets are simply gone, resulting in missing information (e.g., a gap in audio).
- **Typical Values:**
  - **Wired:** Close to 0% under normal conditions.
  - **LEO Satellite:** A higher baseline packet loss rate is expected, with studies showing an average of around 1.4%, often correlated with satellite handover events.[7]

Signal-to-Noise Ratio (SNR)

- **What it is:** SNR is a physical-layer metric, particularly relevant for wireless and satellite communications, that compares the power level of the desired signal to the power level of background noise.[25] It is measured in decibels (dB). A higher SNR value indicates a clearer, stronger signal that is easier for the receiver to distinguish from noise.
- **Why it matters:** SNR is a critical **leading indicator** of link quality for the satellite path. A degrading SNR is a direct precursor to a rise in bit errors, which in turn leads to increased packet loss and jitter.[26] By monitoring SNR, an AI model can anticipate and react to link degradation
  before it begins to impact the user-facing KPIs. This causal chain is fundamental to building a truly proactive stabilization tool. The sequence of events is as follows: an atmospheric event like rain fade causes the signal to be attenuated, which leads to an immediate drop in measured SNR. This weaker signal makes it harder for the receiver to correctly interpret the data, causing a subsequent increase in packet loss and jitter.
- **Common Causes of Degradation:** The primary cause of SNR degradation in satellite communications is atmospheric interference. Precipitation such as heavy rain, snow, or ice can absorb and scatter the microwave radio frequency (RF) signal, a phenomenon known as **rain fade**.[17] This effect is especially pronounced at the higher frequencies (above 11 GHz) used by modern satellite systems.[17]

1.3: Understanding Network Events: The "Why" Behind the Data

The KPIs provide the "what"—the measurements of network performance. This section explains the "why"—the underlying real-world events that cause these KPIs to change. Each event type produces a distinct "signature" or "fingerprint" in the time-series data. The primary task of our AI models is to learn to recognize and differentiate these signatures. This categorization provides the basis for labeling our training data and defines the clear division

of labor between our two AI models.

Predictable Events

These are routine, expected operational events. Their signatures are consistent and repeatable, making them ideal training targets for our supervised predictive model.

- **Satellite Handoffs:** In a LEO satellite constellation, the satellites are in constant, rapid motion relative to a user on the ground. To maintain a continuous connection, a user's terminal must frequently switch from a satellite that is moving out of view to one that is coming into view.[29] This process is called a handoff.
    - **Signature:** A satellite handoff manifests as a sharp, narrow, and temporary spike in latency and jitter. This is due to the brief period of link re-establishment. In some systems like Starlink, these handoffs occur at highly regular, periodic intervals (e.g., every 15 seconds), and may be accompanied by a small amount of packet loss.[15] The signature is a distinct, short-duration "peak" in the KPI data.
- **Network Congestion:** This occurs when the volume of data traffic on a network segment exceeds its carrying capacity, much like a traffic jam on a highway.[33] Congestion often follows predictable time-of-day patterns, with "internet rush hours" typically occurring in the evenings (e.g., 7 PM to 11 PM) when residential usage for streaming and gaming is highest.[35]
    - **Signature:** Unlike the sharp spike of a handoff, congestion produces a slower, broader "hump" in the KPI data. As network buffers begin to fill, latency and packet loss gradually increase, remain elevated during the peak period, and then slowly decrease as demand subsides.[24]

Unpredictable Failures

These are non-routine, often catastrophic events that represent a true failure state in the network. Their signatures are abrupt and do not follow a predictable pattern, making them the target for our unsupervised anomaly detection model.

- **Configuration Errors:** These are failures caused by human error during the setup or maintenance of network equipment. Examples include a **duplex/speed mismatch** where two connected devices fail to negotiate a common transmission speed, incorrect **Maximum Transmission Unit (MTU)** settings that cause packet fragmentation or drops, or a misconfigured **firewall rule** that suddenly blocks all legitimate traffic.[39]
    - **Signature:** A configuration error typically produces a **step function** signature. A KPI will change abruptly from a normal state to a new, sustained failure state. For example, packet loss might instantly jump from 0% to 100% and stay there, or latency might become effectively infinite.[42]
- **Latent Software Bugs:** These are the most insidious failures, stemming from flaws in

the firmware or operating system of a network device (e.g., a router, switch, or satellite).[43] These bugs can cause bizarre, non-standard, and highly unpredictable behavior.

- **Signature:** The signature of a software bug is its lack of a recognizable signature. It is, by definition, an anomaly that does not fit the pattern of a handoff, congestion, or a simple configuration error. It might manifest as intermittent, random packet loss with no clear cause, or a "silent failure" where the device reports as operational but is subtly corrupting data packets.[47] This chaotic or novel pattern is precisely what the anomaly detection model is designed to catch.

The supervised LSTM model will be trained to recognize and classify the "spike" signature of a handoff and the "hump" signature of congestion. The unsupervised Autoencoder model will be trained on a baseline of normal data that includes these predictable event signatures. Its task is to flag any pattern that it cannot reconstruct well—namely, the "step function" of a configuration error or the chaotic signature of a software bug—as a critical anomaly.

Chapter 2: Introduction to Time-Series & Signal Processing

This chapter introduces the mathematical and conceptual tools needed to process and analyze the raw network data. By treating network performance metrics as signals, we can leverage powerful techniques from the field of Digital Signal Processing (DSP) to clean the data, remove noise, and transform it into a format that highlights the underlying patterns our AI models need to learn.

2.1: Viewing Network Data as a Signal

A **time-series** is a sequence of data points collected at successive, equally spaced intervals over time.[50] A stream of network KPI data, such as latency being measured every second, perfectly aligns with this definition.[52] This conceptual framing is powerful because it allows us to move beyond simple statistical analysis and apply a rich toolkit of DSP techniques to our data.[55] By viewing latency, jitter, or SNR not just as a list of numbers but as a continuous *signal*, we can analyze its properties in terms of frequency, shape, and periodicity, unlocking deeper insights into the network's behavior.

2.2: The Problem of "Noise"

In the context of signal processing, **noise** refers to the small, random, high-frequency fluctuations present in the data that are not part of the true, underlying signal we wish to

analyze.[57] This noise can originate from various sources, including minor inaccuracies in measurement equipment, transient electromagnetic interference, or the inherent stochastic variability of complex systems.

The primary problem with noise is that it can obscure the meaningful patterns—the event "signatures" identified in Chapter 1—that are critical for our AI models. A sharp spike in latency caused by a satellite handoff might be partially hidden within a flurry of random fluctuations, making it more difficult for the model to detect and learn the pattern. Therefore, a crucial first step in any robust data preprocessing pipeline is to apply a smoothing or filtering technique to remove this noise, thereby increasing the signal-to-noise ratio and making the underlying patterns more distinct.

2.3: Smoothing Techniques: Savitzky-Golay Filter

While many methods exist for smoothing data, the choice of technique is a critical act of feature engineering. A naive approach, such as a simple moving average, can be detrimental to this project's goals. A moving average filter works by replacing each data point with the average of its neighbors. While this effectively removes noise, it does so at the cost of blurring sharp features in the signal.[59] For our use case, this is a significant drawback, as it would flatten and distort the very latency and jitter spikes that uniquely identify a satellite handoff, effectively destroying a key feature we want our model to learn.

A superior alternative for this project is the **Savitzky-Golay (SG) filter**.

- **How it Works:** The Savitzky-Golay filter is a more sophisticated smoothing method that operates on a sliding window of data. Instead of simply averaging the points within the window, it fits a low-degree polynomial (e.g., a quadratic or cubic curve) to the data points in the window using the method of linear least squares.[61] The new, smoothed value for the central point of the window is then taken from the value of this fitted polynomial at that central point. This process is repeated as the window slides across the entire time-series.
- **Why it is Superior:** The fundamental advantage of the SG filter is its ability to reduce noise while preserving the essential features of the signal, such as the shape, height, and width of peaks.[59] By fitting a curve rather than a flat line (as a moving average effectively does), it can better follow the true trajectory of the data, especially around inflection points. This makes it the ideal choice for our preprocessing pipeline, as it will clean the KPI data without compromising the integrity of the crucial handoff signatures that our predictive model must learn to recognize.

2.4: The Frequency Domain: A New Perspective

Thus far, we have considered our network data in the **time domain**, where we plot the value of a KPI against time. However, any time-series signal can also be analyzed in the **frequency**

domain, which provides a complementary and often highly insightful perspective.

- **The Fourier Transform:** The mathematical tool that allows us to switch from the time domain to the frequency domain is the **Fourier Transform**. At a high level, the Fourier Transform decomposes a complex signal into the sum of the simple sine and cosine waves of different frequencies that make it up.[65] The output of a Fourier Transform, often visualized in a plot called a periodogram or power spectrum, shows the amplitude (or power) of each constituent frequency present in the original signal. It answers the question: "How much of each frequency is in this signal?"
- **Application in Anomaly Detection:** The frequency domain is exceptionally powerful for detecting hidden periodic patterns that may be difficult to see in the time domain. Consider a scenario where a misconfigured network device or a subtle software bug causes a small burst of disruptive packets to be sent out every 10 seconds. In the time-domain plot, these small bursts might be lost in the general noise of the network traffic. However, when we apply a Fourier Transform to this data, this periodic event will manifest as a sharp, distinct spike in the frequency domain at exactly 0.1 Hz (calculated as 1 cycle / 10 seconds).[69] This makes the otherwise hidden periodic anomaly immediately obvious. While not a primary component of the Phase 1 MVP, understanding frequency-domain analysis is a key skill for an AI engineer tasked with more advanced anomaly detection, as it provides a powerful way to engineer features based on a signal's periodicity.

The selection of a preprocessing technique is a deliberate act of feature engineering. Using a Savitzky-Golay filter is a conscious decision to create a feature set that emphasizes the morphological characteristics (the shape) of events, making it ideal for the predictive model that needs to recognize handoff spikes. In contrast, using a Fourier Transform is a choice to engineer features based on periodicity, which would be more suited for an advanced anomaly detector searching for subtle, repeating errors. The AI engineer must choose the right tool to highlight the specific data signatures the models are intended to learn.

Part 2: The AI Toolkit for Phase 1

Chapter 3: The Predictive Steering Model (Supervised Learning)

This chapter provides a detailed examination of the first of our two AI models: the predictive steering model. The objective is to develop a deep, theoretical, and mathematical understanding of the Long Short-Term Memory (LSTM) network, the architecture chosen for its unique ability to process sequential data and learn temporal dependencies. This model will

form the core of our proactive network optimization engine.

3.1: The Goal: Proactive Path Optimization

The task of proactively steering network traffic is framed as a **supervised learning** problem.[72] In supervised learning, a model learns from a dataset containing input features and corresponding correct output labels. For our project, the model will be trained on a large dataset of historical network KPI sequences.

- **Input Features (X):** A sequence of past KPI measurements (e.g., latency, jitter, SNR over the last 60 seconds).
- **Output Label (y):** The known optimal network path (i.e., 'wired' or 'satellite') for the time step immediately following the input sequence.

The model's goal is to learn the complex mapping function that relates patterns in the recent past (the input sequence) to the optimal decision for the immediate future (the output label). By learning the signatures of predictable events like satellite handoffs or congestion, the model can anticipate performance degradation and recommend a switch to the alternative path *before* the user's experience is negatively impacted.

3.2: Introduction to Recurrent Neural Networks (RNNs)

Traditional neural network architectures, such as Multi-Layer Perceptrons (MLPs) or even Convolutional Neural Networks (CNNs), are fundamentally unsuited for this task. These feedforward networks operate under the assumption that all inputs are independent of one another.[74] They have no inherent mechanism for retaining information from one input to the next, meaning they possess no "memory".[76] When presented with a sequence of network data, a feedforward network would process each time step in isolation, completely ignoring the crucial temporal context that defines events like a latency spike or a gradual build-up of congestion.

Recurrent Neural Networks (RNNs) are a class of neural networks specifically designed to overcome this limitation and process sequential data.[77]

- **The Core Concept: The Feedback Loop:** The defining feature of an RNN is its **feedback loop**. Unlike a feedforward network where information flows in only one direction, an RNN's hidden layer feeds its own output from the current time step back to itself as an input for the next time step.[77]
- **The Hidden State: A Form of Memory:** This feedback loop allows the RNN to maintain a **hidden state** (ht), which is a vector that gets updated at every time step.[81] This hidden state acts as a form of memory, serving as a compressed summary of all the information the network has processed in the sequence up to that point.[80] At any given time step
  t, the network's computation is a function of both the current input (xt) and the previous

hidden state (ht−1), enabling it to make decisions based on both present and past context.

3.3: The Challenge of Long-Term Memory

While simple RNNs can effectively capture short-term dependencies, they struggle significantly with learning patterns that span long sequences. This limitation stems from a fundamental issue in their training process known as the **vanishing gradient problem**.[84]
RNNs are trained using an algorithm called Backpropagation Through Time (BPTT). During BPTT, the error calculated at the end of a sequence is propagated backward through the unrolled network, one time step at a time, to update the network's weights. To calculate the gradient for a weight at an early time step, the chain rule of calculus requires repeated multiplication of the gradients from all subsequent time steps.[86]
The problem arises because these calculations involve repeatedly multiplying by the recurrent weight matrix. If the values in this matrix are small (a common scenario, especially after initialization), the gradient signal shrinks exponentially as it is propagated further back in time.[85] By the time the gradient reaches layers corresponding to much earlier time steps, its magnitude can become so infinitesimally small that it has no practical effect on updating the weights. This "vanishing" of the gradient effectively prevents the network from learning from events that occurred far in the past, limiting its memory to only the most recent few time steps.

3.4: Deep Dive: Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are a specialized and highly successful variant of RNNs, architected specifically to overcome the vanishing gradient problem and effectively learn long-term dependencies.[84]

Internal Cell Structure

The key innovation of the LSTM is its complex internal cell structure, which introduces a dedicated memory component and a sophisticated gating mechanism to regulate the flow of information.

- **The Cell State (The "Conveyor Belt"):** At the heart of the LSTM cell is the **cell state** (Ct). This can be visualized as an information "conveyor belt" that runs straight down the entire chain of time steps, with only minor, controlled, linear interactions.[91] Information from previous steps can flow along this belt largely unimpeded. This direct, uninterrupted pathway is the crucial mechanism that mitigates the vanishing gradient problem. Because the error gradient can flow backward along this path without being

repeatedly squashed by activation functions or weight matrices, the network can learn from causes and effects that are thousands of time steps apart.[93]

- **The Gates:** The LSTM cell uses three "gates" to carefully regulate the addition or removal of information from the cell state conveyor belt. Each gate is essentially a small, independent neural network composed of a sigmoid activation function and a pointwise multiplication operation. The sigmoid function outputs a vector of values between 0 and 1, which acts as a filter or "gate": a value of 0 means "let nothing through," while a value of 1 means "let everything through".[89]
    1. **Forget Gate (ft):** This gate decides what information should be thrown away from the previous cell state (Ct−1). It looks at the previous hidden state (ht−1) and the current input (xt) and outputs a number between 0 and 1 for each element in the cell state. A 1 represents "completely keep this," while a 0 represents "completely get rid of this".[89]
    2. **Input Gate (it):** This gate decides what new information will be stored in the cell state. This is a two-step process. First, a sigmoid layer (the "input gate") decides which values to update. Second, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state. These two are then combined to update the cell state.[89]
    3. **Output Gate (ot):** This gate decides what part of the cell state will be output as the new hidden state (ht). The cell state is passed through a tanh function (to push the values to be between -1 and 1) and then multiplied by the output of the sigmoid gate, so that only the desired parts of the information are passed on to the next time step and the output layer.[89]

Mathematical Formulation

The operations within an LSTM cell are defined by a set of precise mathematical equations. Understanding these formulas is crucial for a deep comprehension of the model's behavior.[96] Let xt be the input at time step t, ht−1 be the hidden state from the previous time step, and Ct−1 be the cell state from the previous time step. W and b represent the weight matrices and bias vectors for each gate, respectively. The operator ⊙ denotes element-wise multiplication.

1. Forget Gate: The sigmoid layer determines the forget factor ft.

   $$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

   The sigmoid function, $\sigma(z) = 1 + e - z1$, outputs values between 0 and 1, acting as a gate.
2. Input Gate: The sigmoid layer determines the update factor it, and the tanh layer creates the new candidate values C~t.

   $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
   $$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The hyperbolic tangent function, $\tanh(z)=\frac{e^z+e^{-z}}{e^z-e^{-z}}$, outputs values between -1 and 1, scaling the new information.

3. Cell State Update: The old cell state C_{t-1} is updated to the new cell state C_t by first forgetting old information and then adding new information.

   $$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

4. Output Gate: The sigmoid layer determines the output factor o_t, which is then used to filter the updated cell state to produce the new hidden state h_t.

   $$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
   $$h_t = o_t \odot \tanh(C_t)$$

This gated architecture allows the LSTM to learn complex and long-range temporal dynamics, making it an exceptionally powerful tool for the proactive network path optimization required in this project.

Chapter 4: The Anomaly Detection Model (Unsupervised Learning)

This chapter details the second AI model in our dual strategy: the unsupervised anomaly detector. Its purpose is to act as a safety net, identifying major, unforeseen network failures that fall outside the predictable patterns learned by the supervised model. We will explore the architecture and logic of the LSTM Autoencoder, the chosen tool for detecting these "unknown unknowns."

4.1: The Goal: Detecting "Unknown Unknowns"

The problem of detecting novel failures is fundamentally an **unsupervised learning** task.[103] By definition, we do not have a labeled dataset of all possible future failures (e.g., configuration errors, software bugs). Therefore, we cannot train a model to recognize specific failure signatures in a supervised manner.

Instead, the strategy is to train a model to develop a deep and precise understanding of what constitutes **"normal" network behavior**. The core assumption is that anomalies are rare events that deviate significantly from this normal baseline.[103] The model is trained exclusively on a dataset that contains only examples of normal, healthy network operation (including the predictable, routine events like handoffs and congestion).[105] The model's task is to flag any new data that does not conform to this learned representation of normalcy.

4.2: The Concept of Autoencoders

An **autoencoder** is a type of artificial neural network used for unsupervised learning, primarily for learning efficient data codings or representations.[107] Its architecture is elegantly simple and consists of two main components connected by a central bottleneck [108]:

1. **Encoder:** This is the first half of the network. It takes a high-dimensional input (e.g., a sequence of network KPIs) and compresses it down into a low-dimensional representation. This compressed representation is known as the **latent space** or **bottleneck**.[108] The encoder's job is to learn a function that maps the input data to this compact latent representation, capturing the most essential features of the data.
2. **Decoder:** This is the second half of the network. It takes the low-dimensional latent space representation from the encoder and attempts to reconstruct the original, high-dimensional input data from it.[108]

The entire autoencoder network is trained end-to-end with a single objective: to minimize the difference between the original input and the reconstructed output. This difference is quantified by a **reconstruction loss** function, typically Mean Squared Error.[109] The presence of the bottleneck is crucial; it prevents the network from simply learning an identity function (i.e., copying the input directly to the output). Instead, it forces the encoder to learn a meaningful compression of the data that retains enough information for the decoder to perform a faithful reconstruction.

4.3: Measuring Surprise: Reconstruction Error

The mechanism for using an autoencoder for anomaly detection is both clever and effective. The power of this approach comes not from the model's ability to generalize well to all data, but from its intentional inability to generalize to data it has not been trained on.

- **Training on Normal Data:** The critical step is to train the autoencoder *exclusively* on a large dataset of normal, non-anomalous network traffic.[106] Through this process, the encoder learns to find a highly efficient latent space representation for the patterns of normal behavior, and the decoder becomes proficient at reconstructing these normal patterns from their latent codes.
- **Detecting Anomalies:** Once trained, the model is deployed to monitor new, live network data. When a data point representing normal operation is fed into the network, the encoder compresses it effectively, and the decoder reconstructs it with high fidelity, resulting in a **low reconstruction error**. However, when an anomalous data point—representing a pattern the model has never seen before, like a sudden configuration failure—is fed in, the encoder will struggle to map it to the learned latent space. Consequently, the decoder, which only knows how to reconstruct normal patterns, will fail to reproduce the anomalous input accurately. This results in a **high reconstruction error**.[106]

- **The Anomaly Score:** This reconstruction error serves as a direct, quantifiable **anomaly score**.[112] By setting a threshold on this score (typically based on the distribution of errors seen on a validation set of normal data), we can create a powerful detection system. Any data point whose reconstruction error exceeds this threshold is flagged as an anomaly.[114]

4.4: The LSTM Autoencoder Architecture

To apply the autoencoder concept to our sequential time-series data, we must use an architecture that can understand temporal patterns. This is achieved by constructing the encoder and decoder using LSTM layers, creating a model known as an **LSTM Autoencoder**.[115]

- **Encoder:** The encoder consists of one or more LSTM layers. It reads the input time-series sequence step-by-step. The final hidden state (and/or cell state) vector from the last time step of the LSTM serves as the compressed, latent space representation of the entire input sequence.[115] This single vector encapsulates the temporal features of the sequence.
- **Decoder:** The decoder's task is to take the single latent space vector from the encoder and reconstruct the original time-series sequence. To do this, the latent vector must be fed as input to the decoder's LSTM at each time step of the output sequence. This is typically achieved using a RepeatVector layer in frameworks like Keras, which simply duplicates the encoder's final state vector to match the length of the original input sequence.[117] The decoder LSTM then processes this repeated vector to generate the reconstructed sequence.

By training this architecture on normal network KPI sequences, the LSTM Autoencoder learns the characteristic temporal dynamics of a healthy network. Any new sequence of KPIs that violates these learned temporal patterns—such as the abrupt step-function of a configuration error or the chaotic signature of a software bug—will be poorly reconstructed, generate a high reconstruction error, and be correctly identified as an anomaly.

Part 3: Practical Implementation for the MVP

Chapter 5: Building the Hybrid Network Simulator

The foundation of any successful machine learning project is high-quality data. Since obtaining a large, perfectly labeled dataset of real-world network failures is often impractical, we will create a simulator. This chapter provides the design principles and high-level logic for generating synthetic time-series data that accurately models our hybrid network environment. This simulated data will be the fuel for training and validating both the predictive and anomaly detection AI models.

5.1: Design Principles & Data Schema

The simulator must be designed with two core principles in mind: statistical realism and event controllability. The baseline data it generates should reflect the statistical properties of real-world network KPIs, including inherent noise. Crucially, it must allow for the controlled injection of specific, labeled network events (handoffs, failures, etc.) at precise moments in time, which is essential for creating the labeled dataset required for supervised learning.
To structure this data, we will use a clear and consistent schema, which can be saved in a standard format like a CSV file. Each row in the dataset will represent a single time step (e.g., one second) and will contain the following columns:

- timestamp: An ISO 8601 formatted timestamp (e.g., '2025-01-01T12:00:00Z') identifying the specific moment of measurement.
- wired_latency_ms: The simulated latency on the wired path, in milliseconds (float).
- wired_jitter_ms: The simulated jitter on the wired path, in milliseconds (float).
- wired_packet_loss_pct: The simulated packet loss on the wired path, as a percentage (float, 0.0 to 1.0).
- satellite_latency_ms: The simulated latency on the satellite path, in milliseconds (float).
- satellite_jitter_ms: The simulated jitter on the satellite path, in milliseconds (float).
- satellite_packet_loss_pct: The simulated packet loss on the satellite path, as a percentage (float, 0.0 to 1.0).
- satellite_snr_db: The simulated Signal-to-Noise Ratio for the satellite link, in decibels (float).
- event_type: A categorical label identifying the ground-truth event occurring at this time step. This is the primary label for the supervised model. Values can include: 'normal', 'satellite_handoff', 'network_congestion', 'config_error_wired', 'software_bug_satellite'.
- active_path: The path currently being used for traffic in the simulation ('wired' or 'satellite').
- optimal_path: The ground-truth label indicating which path offered the best performance at this time step. This serves as the target variable (y) for the predictive steering model.

5.2: Logic for Baseline & Event Generation

The core of the simulator can be implemented using Python with libraries like NumPy for

numerical operations and Pandas for data manipulation. The logic involves first generating a stable baseline for each KPI and then layering event-specific modifications on top.

Baseline Generation Logic

A realistic baseline for any KPI is not a flat line but rather a value that fluctuates around a mean due to natural system noise. This can be effectively simulated using a normal (Gaussian) distribution.[118]

Python-like Logic:

Python

```
import numpy as np
import pandas as pd

def generate_baseline_kpi(num_timesteps, mean, std_dev):
    """Generates a baseline time-series with Gaussian noise."""
    return np.random.normal(loc=mean, scale=std_dev, size=num_timesteps)

# Example: Generate 1 hour of data (3600 seconds)
timesteps = 3600

# Generate baseline for wired latency (stable, low mean, very low deviation)
wired_latency = generate_baseline_kpi(timesteps, mean=10.0, std_dev=0.5)

# Generate baseline for satellite latency (higher mean, more natural variance)
satellite_latency = generate_baseline_kpi(timesteps, mean=45.0, std_dev=5.0)

# Generate baseline for satellite SNR (e.g., clear sky conditions)
satellite_snr = generate_baseline_kpi(timesteps, mean=20.0, std_dev=0.2)
```

Event Injection Logic

Once the baseline series are created, functions can be defined to inject the specific event signatures discussed in Chapter 1. These functions will modify slices of the baseline arrays to simulate the impact of each event.[121]

Python-like Logic for inject_handoff():

This function simulates the sharp, brief spike in latency and jitter characteristic of a satellite handoff.

Python

```
def inject_handoff(latency_series, jitter_series, start_index, duration=3, spike_factor=3.0):
    """Injects a latency/jitter spike into satellite KPI series."""
    end_index = start_index + duration
    if end_index < len(latency_series):
        # Create a spike shape (e.g., a simple triangular pulse)
        spike = np.linspace(0, 1, duration) * spike_factor * np.std(latency_series)
        spike = spike - np.abs(np.linspace(-1, 1, duration)) * (spike_factor / 2) *
np.std(latency_series)

        latency_series[start_index:end_index] += spike * 2 # Latency spike is higher
        jitter_series[start_index:end_index] += spike      # Jitter also spikes
    return latency_series, jitter_series
```

Python-like Logic for inject_failure():
This function simulates a sudden, catastrophic failure, like a misconfiguration causing total packet loss.

Python

```
def inject_failure(packet_loss_series, start_index):
    """Injects a sustained failure by setting packet loss to 100%."""
    if start_index < len(packet_loss_series):
        packet_loss_series[start_index:] = 1.0 # 100% packet loss
    return packet_loss_series
```

By combining these functions, a complex and realistic training dataset can be generated. For example, a master script could generate a long baseline, then loop through and inject handoffs every 15 seconds, a period of congestion during a simulated "evening," and a few randomly placed catastrophic failures to create a rich dataset for training and testing both the supervised and unsupervised models.

Chapter 6: Data Preprocessing & Feature Engineering

Raw data, even when simulated, is rarely in the perfect format for direct input into a deep learning model. The process of transforming this raw data into clean, structured, and informative features is known as data preprocessing and feature engineering. This chapter

outlines the full, sequential pipeline of steps required to convert the CSV data from our simulator into model-ready tensors that our LSTM models can effectively learn from.

6.1: The Full Pipeline: From Raw Data to Model-Ready Tensors

A robust and repeatable preprocessing pipeline is essential for any machine learning project. The following sequence of operations ensures that our data is cleaned, scaled, and structured correctly before it reaches the models.[1][2][3]

1. **Load CSV Data:** The process begins by loading the entire simulated dataset from the CSV file into a Pandas DataFrame. This provides a convenient structure for data manipulation.
2. **Smooth KPI Data:** Apply the Savitzky-Golay filter to the numerical KPI columns (e.g., wired_latency_ms, satellite_snr_db). This step removes high-frequency noise while preserving the shape of important event signatures, as detailed in Chapter 2.
3. **Normalize Numerical Features:** Scale all numerical features to a common range, typically . This is a critical step for neural network training and will be explained in detail in the next section.
4. **Window the Data:** Transform the flat, two-dimensional time-series DataFrame into a three-dimensional structure of input sequences and corresponding output labels. This "windowing" process creates the (samples, timesteps, features) format required by LSTM layers.
5. **Split Datasets:** Divide the final, windowed data into three distinct sets:
    - **Training Set:** The largest portion of the data, used to train the model's weights.
    - **Validation Set:** A smaller portion used during training to tune hyperparameters and check for overfitting.
    - **Test Set:** A final, held-out portion of the data that the model has never seen. It is used only once at the very end to provide an unbiased evaluation of the final model's performance.

6.2: Normalization and Windowing Explained

The normalization and windowing steps are the most critical transformations in preparing time-series data for LSTMs. A misunderstanding of these concepts can lead to poor model performance or, worse, misleadingly optimistic results.

Normalization with MinMaxScaler

Neural networks learn by adjusting their internal weights based on the magnitude of the error gradient. If input features have vastly different scales (e.g., latency in the tens or hundreds,

while packet loss is between 0 and 1), the features with larger scales will dominate the weight update process, potentially preventing the model from learning from the smaller-scale features.[125]

Normalization solves this by rescaling all features to a consistent range. The MinMaxScaler from scikit-learn is a common choice, scaling data to a default range of .

Crucial Best Practice: Fit Only on Training Data

A common and critical mistake is to apply the scaler to the entire dataset before splitting it into training and test sets. This introduces data leakage, a scenario where information from the future (the validation and test sets) "leaks" into the training process.127 The scaler would learn the minimum and maximum values from the entire dataset, including data the model is not supposed to have seen yet. This gives the model an unfair advantage and results in an evaluation that is not representative of how it would perform on truly unseen data.

The correct procedure is as follows [125]:

1. Split the data into training, validation, and test sets first.
2. Create an instance of the MinMaxScaler.
3. Call the fit_transform() method on the **training data only**. This learns the scaling parameters (min and max) from the training data and transforms it.
4. Call the transform() method on the **validation and test data**. This applies the *same* scaling parameters learned from the training data to these sets, ensuring a consistent transformation without leaking information.

Windowing: The Sliding Window Technique

LSTMs require input data to be in the form of sequences. The **sliding window** technique is the standard method for converting a flat time-series into a dataset of input-output pairs suitable for supervised learning.[130]

The process works as follows:

1. **Define a window_size (or n_steps):** This determines how many past time steps the model will use as input to make a prediction. For example, a window_size of 60 means the model will look at the last 60 seconds of data.
2. **Slide the Window:** A "window" of this size is moved across the time-series data, one time step at a time.
3. **Create Samples:** At each position of the window, the data contained within the window becomes a single input sample (X), and the data point immediately following the window becomes the corresponding output label (y).[132]

For a simple univariate series `` and a window_size of 3, the process would generate the following samples:

- Sample 1: $X = $, y=40
- Sample 2: $X = $, y=50

This transformation converts the original time-series into a dataset with a shape that can be fed into an LSTM, typically a 3D tensor of the format [number_of_samples, window_size,

number_of_features].

Chapter 7: Building and Training the Models

With the data prepared and structured, the next step is to define the neural network architectures and execute the training process. This chapter provides the conceptual framework and key code components for building and training the predictive and anomaly detection models using a modern deep learning library.

7.1: The Training Environment

For building and training deep learning models, it is highly recommended to use a high-level framework that abstracts away the complexities of low-level tensor operations and GPU management. The two dominant frameworks in the industry are **TensorFlow** (with its integrated Keras API) and **PyTorch**.[134]

- **TensorFlow and Keras:** TensorFlow is a comprehensive, end-to-end platform for machine learning. Keras is its official high-level API, designed for fast and easy model prototyping. Keras's emphasis on user-friendliness, simple APIs, and clear error messages makes it an excellent choice for this project, where the goal is to build a functional MVP efficiently.[136]
- **PyTorch:** PyTorch is known for its flexibility, "Pythonic" design, and strong ties to the research community. It offers more granular control over the training process, which is advantageous for complex, research-driven projects but can introduce more boilerplate code for standard applications.[135]

For the purposes of this guide, the code concepts will be presented with a focus on the **Keras API**, as its sequential model-building paradigm is highly intuitive for constructing the required LSTM architectures.

7.2: Code Concepts for Model Building & Training

The process of building and training a model in Keras can be broken down into three main conceptual steps: defining the architecture, compiling the model, and fitting it to the data.

Defining a Sequential Model

The Sequential model in Keras is the simplest way to build a neural network. It allows you to define a model as a linear stack of layers, where you add layers one by one in the order that data will flow through them.[139]

Conceptual Code (Keras):

Python

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, RepeatVector, TimeDistributed

# Define the LSTM Autoencoder architecture
timesteps = 60
features = 8

model = Sequential(name="LSTM_Autoencoder")
model.add(LSTM(128, activation='relu', input_shape=(timesteps, features)))
model.add(RepeatVector(timesteps)) # Repeats the encoded vector for the decoder
model.add(LSTM(128, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(features))) # Output layer for each timestep

model.summary()
```

The Training Loop: compile() and fit()

After defining the model's architecture, you configure the learning process using the compile() method and then start the training using the fit() method. These two methods encapsulate the core components of the training loop.

- **Optimizer:** The optimizer is the algorithm that adjusts the model's internal weights to minimize the loss function. The **Adam** (Adaptive Moment Estimation) optimizer is the recommended default choice.[142] Adam is an efficient and robust algorithm that combines the benefits of two other popular optimizers: Momentum (which helps accelerate gradients in the correct direction) and RMSprop (which adapts the learning rate for each weight). This adaptive learning rate capability makes it well-suited for a wide range of problems with minimal hyperparameter tuning.[144]
- **Loss Function:** The loss function quantifies how far the model's prediction is from the true label. The goal of training is to minimize this value. For the regression-style tasks in this project (reconstructing the input sequence for the autoencoder and predicting a continuous value for the predictive model), **Mean Squared Error (MSE)** is the standard and appropriate choice.[146] MSE calculates the average of the squared differences between the predicted values and the actual values. By squaring the error, it heavily penalizes larger mistakes, pushing the model to make more accurate predictions and be particularly sensitive to outliers.[147]

- **Epochs:** An **epoch** represents one complete pass of the entire training dataset through the model.[149] A model is not trained in a single pass; it needs to see the data multiple times to learn the underlying patterns effectively. The
  fit() method is typically run for a specified number of epochs. While more epochs can lead to a better model, training for too many epochs carries the risk of **overfitting**, where the model starts to memorize the training data, including its noise, and loses its ability to generalize to new, unseen data.[152]

Conceptual Code (Keras):

Python

```
# 1. Compile the model with the optimizer and loss function
model.compile(optimizer='adam', loss='mean_squared_error')

# 2. Fit the model to the training data for a set number of epochs
# X_train and y_train are the preprocessed, windowed data
# For the autoencoder, y_train would be the same as X_train
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))
```

Chapter 8: The Remediation Engine & MVP Application

The final chapter focuses on integrating the two trained AI models into a functional application. This involves creating a decision-making "Remediation Engine" that interprets the models' outputs and translates them into actionable recommendations. We will then outline how to build a simple but effective Graphical User Interface (GUI) using the Streamlit library to present this information to a user in real-time.

8.1: The Logic of the AI Co-Pilot

At each time step, our system will generate two distinct outputs from the AI models:

1. A **path recommendation** from the supervised predictive LSTM model (e.g., 'switch to wired').
2. An **anomaly score** (the reconstruction error) from the unsupervised LSTM Autoencoder.

The Remediation Engine, or "AI Co-Pilot," is the logic that synthesizes these two outputs into a single, coherent action. The guiding principle for this logic is that system stability and integrity are paramount. A critical, unforeseen failure must always take precedence over a routine performance optimization.

This creates a clear hierarchy in the decision-making process. The unsupervised anomaly detection model acts as a "safety net" or a guardrail. If it detects a severe anomaly, its alert must override any recommendation from the predictive model. This is a crucial design choice for building a trustworthy and robust operational AI system. For instance, consider a scenario where a latent software bug causes the satellite link's KPIs to report deceptively perfect, albeit non-sensical, values. The predictive model, trained on normal patterns, might see these "perfect" values and recommend switching traffic *to* the failing satellite link. However, the *pattern* of these KPIs would be bizarre and unlike anything the autoencoder was trained on, resulting in a very high reconstruction error. The remediation engine's logic would catch this high anomaly score first, issue a critical alert, and prevent the disastrous switch, thereby demonstrating the necessity of prioritizing the anomaly signal.
The decision logic can be implemented with a simple but effective conditional block.
Pseudo-code for the Remediation Engine:

Python

```
# Define the anomaly threshold, determined during model validation
# This is the reconstruction error value above which we consider the state anomalous
ANOMALY_THRESHOLD = 0.85

def run_remediation_engine(current_kpi_window, current_active_path):
    """
    Analyzes network state using AI models and determines the appropriate action.
    """

    # 1. Get outputs from both AI models
    anomaly_score = lstm_autoencoder.predict(current_kpi_window)
    predicted_optimal_path = predictive_lstm.predict(current_kpi_window)

    # 2. Implement the prioritized decision logic
    if anomaly_score > ANOMALY_THRESHOLD:
        # HIGHEST PRIORITY: A critical, unforeseen failure is detected.
        action_message = "CRITICAL ANOMALY DETECTED! Network state is abnormal. Manual intervention may be required."
        alert_level = "ERROR"
        # In a production system, this could trigger automated failover or page an engineer.

    elif predicted_optimal_path!= current_active_path:
        # SECOND PRIORITY: The predictive model recommends a proactive switch for optimization.
        action_message = f"Proactive steering recommended: Switch from '{current_active_path}' to '{predicted_optimal_path}' to avoid predictable degradation."
```

```
    alert_level = "INFO"
    # Logic to execute the network path switch would be called here.

else:
    # LOWEST PRIORITY: Network is stable and operating on the optimal path.
    action_message = "Network stable. Current path is optimal. No action required."
    alert_level = "SUCCESS"

# 3. Return the decision for the GUI to display
return action_message, alert_level
```

8.2: Building the Streamlit GUI

For the MVP, we need a simple way to visualize the network's status and the AI Co-Pilot's decisions. **Streamlit** is an open-source Python framework designed for this exact purpose. It allows developers to create and share interactive web applications for machine learning and data science projects with remarkably little code.[153] Instead of requiring knowledge of front-end web development (HTML, CSS, JavaScript), you can build a functional dashboard using simple Python commands.

The GUI for our AI Network Stabilization Tool will consist of three main sections, each built with a specific Streamlit component:

1. **Real-Time KPI Charts:** To visualize the incoming network data, we will use st.line_chart. This function can take a Pandas DataFrame and quickly render an interactive line chart, making it perfect for plotting the time-series of Latency, Jitter, Packet Loss, and SNR for both the wired and satellite links.[153]
2. **Current Status Metrics:** To display the most important "at-a-glance" information, we will use st.metric. This component displays a single key metric in a large, bold font, with an optional delta indicator. We will use it to clearly show the **Active Path** (e.g., "Wired") and its current status (e.g., "Optimal" or "Degraded").[157]
3. **AI Co-Pilot Alerts:** To display the output from our Remediation Engine, we will use Streamlit's color-coded callout boxes. These functions are ideal for conveying the severity of a message:
    - st.error(): Displays a message in a red box. This will be used for the ERROR alert_level when a critical anomaly is detected.[159]
    - st.info(): Displays a message in a blue box. This is perfect for the INFO alert_level, informing the user of a proactive steering recommendation.
    - st.success(): Displays a message in a green box, used for the SUCCESS alert_level to confirm that the network is stable and operating optimally.

By combining these simple components, a developer can create a powerful and intuitive real-time monitoring dashboard that not only shows what the network is doing but also

explains what the AI is thinking and recommending.

Works cited

1. Hybrid connectivity satellite technology - AST Networks, accessed on August 1, 2025, https://ast-networks.com/insights/blog/what-does-hybrid-connectivity-mean-in-the-world-of-satellite-technology/
2. Hybrid Network Solutions for Modern Connectivity - SynchroNet, accessed on August 1, 2025, https://synchronet.net/hybrid-network/
3. Fiber Optic Cable Buying Guide | Eaton, accessed on August 1, 2025, https://tripplite.eaton.com/products/fiber-optic-cable-buying-guide
4. Fiber Optics vs Ethernet: Understanding the Key Differences, accessed on August 1, 2025, https://www.truecable.com/blogs/cable-academy/fiber-optics-vs-ethernet-understanding-the-key-differences
5. What are Two Characteristics of Fiber Optic Cable: #1 Best Insights, accessed on August 1, 2025, https://accutechcom.com/what-are-two-characteristics-of-fiber-optic-cable/
6. Wired vs. Wireless Networks (2.5.1) | CIE A-Level Computer Science ..., accessed on August 1, 2025, https://www.tutorchase.com/notes/cie-a-level/computer-science/2-5-1-wired-vs--wireless-networks
7. Satellite vs Fiber Internet: The 2025 Latency & Bandwidth Showdown, accessed on August 1, 2025, https://ts2.tech/en/satellite-vs-fiber-internet-the-2025-latency-bandwidth-showdown/
8. Top 6 Advantages and Disadvantages of Fiber Optic Cable in 2025, accessed on August 1, 2025, https://www.truecable.com/blogs/cable-academy/advantages-and-disadvantages-of-fiber-optic-cable
9. Satellite Internet vs Cable: Key Differences to Consider | Beambox, accessed on August 1, 2025, https://beambox.com/townsquare/satellite-internet-vs-cable
10. Technology - Starlink, accessed on August 1, 2025, https://www.starlink.com/technology
11. Starlink satellites: Facts, tracking and impact on astronomy - Space, accessed on August 1, 2025, https://www.space.com/spacex-starlink-satellites.html
12. Starlink - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Starlink
13. A Novel Feeder Link Handover Strategy for Backhaul in LEO Satellite Networks - PMC, accessed on August 1, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10301820/
14. Starlink Internet Review: Low Satellites, High Pricing - CNET, accessed on August 1, 2025, https://www.cnet.com/home/internet/starlink-internet-review/
15. A transport protocol's view of Starlink | Hacker News, accessed on August 1,

2025, https://news.ycombinator.com/item?id=42284758
16. A Transport Protocol's View of Starlink | blabs - APNIC Labs, accessed on August 1, 2025, https://labs.apnic.net/index.php/2024/05/16/a-transport-protocols-view-of-starlink/
17. Rain fade - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Rain_fade
18. On Starlink - Geoff Huston, accessed on August 1, 2025, https://www.potaroo.net/presentations/2024-04-12-starlink.pdf
19. Top 12 KPIs To Know & Use: Key Performance Indicators Explained - Splunk, accessed on August 1, 2025, https://www.splunk.com/en_us/blog/learn/kpis-key-performance-indicators.html
20. What are Network KPIs, Why Should You Care? 16 Metrics/KPIs to Chase - WhatsUp Gold, accessed on August 1, 2025, https://www.whatsupgold.com/blog/what-are-network-kpis-why-should-you-care-16-metrics-kpis-to-chase
21. Real-Time Latency: Rethinking Remote Networks - Telesat, accessed on August 1, 2025, https://www.telesat.com/resources/real-time-latency-rethinking-remote-networks/
22. What is Satellite Jitter and should we care about it? - NE-ONE By Calnex - Itrinegy, accessed on August 1, 2025, https://itrinegy.com/satellite-communications-blog-part-2/
23. The Top 5 Network Monitoring KPIs - Statseeker - Techniche, accessed on August 1, 2025, https://technichegroup.com/determining-kpis-for-network-monitoring/
24. KPI Metrics in Network Dashboard and Application Flow Map, accessed on August 1, 2025, https://docs.appdynamics.com/appd/24.x/latest/en/infrastructure-visibility/network-visibility/network-visibility-metrics/kpi-metrics-in-network-dashboard-and-application-flow-map
25. Signal-to-noise ratio - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Signal-to-noise_ratio
26. Mastering Signal to Noise Ratio in Satellite Communications, accessed on August 1, 2025, https://www.numberanalytics.com/blog/mastering-signal-to-noise-ratio-in-satellite-communications
27. What is Signal to Noise Ratio and How to calculate it? | Advanced PCB Design Blog, accessed on August 1, 2025, https://resources.pcb.cadence.com/blog/2020-what-is-signal-to-noise-ratio-and-how-to-calculate-it
28. A Survey of Rain Fade Models for Earth–Space Telecommunication ..., accessed on August 1, 2025, https://www.mdpi.com/2072-4292/13/10/1965
29. An Effective Handover Strategy for Fast-moving Devices in LEO ..., accessed on August 1, 2025, https://icoin.org/media?key=site/icoin2025/abs/A-7-2.pdf
30. Handover scenario in LEO satellite system. | Download Scientific ..., accessed on

August 1, 2025, https://www.researchgate.net/figure/Handover-scenario-in-LEO-satellite-system_fig1_352219117
31. Handover Schemes in Satellite Networks: State-of-the-Art and Future Research Directions - OU School of Computer Science, accessed on August 1, 2025, https://cs.ou.edu/~netlab/Pub/SURV_HANDOFF_SN-ComSurveys.pdf
32. Starlink for Gaming: Latency, Stability & Real-World Performance - Dishy Central, accessed on August 1, 2025, https://dishycentral.com/starlink-for-gaming
33. The Ultimate Guide to Internet Congestion Control | Compira Labs, accessed on August 1, 2025, https://www.compiralabs.com/ultimate-guide-congestion-control
34. How Peak Hours Affect Your Internet Speed, accessed on August 1, 2025, https://www.compareinternet.com/blog/how-peak-hours-affect-your-internet-speed/
35. How Does Time of Day Affect Internet Speed? - Viasat Satellite Internet, accessed on August 1, 2025, https://www.rsinc.com/how-does-time-of-day-affect-internet-speed.php
36. How Does Time of Day Affect Internet Speed? - BandwidthPlace, accessed on August 1, 2025, https://www.bandwidthplace.com/article/how-does-time-of-day-affect-internet-speed
37. What KPIs Do Network Analysts Use? - InetSoft, accessed on August 1, 2025, https://www.inetsoft.com/info/what-kpis-network-analysts-use/
38. Network performance and visibility: The art of KPIs - Singtel, accessed on August 1, 2025, https://www.singtel.com/business/articles/network-performance-and-visibility-the-art-of-kpis
39. Top Network Configuration Errors | GREYCORTEX, accessed on August 1, 2025, https://www.greycortex.com/blog/top-network-configuration-errors-and-how-fix-them
40. Top 20 Network Configuration Errors – CISCONET Training Solutions, accessed on August 1, 2025, https://www.cisconetsolutions.com/top-20-most-common-network-configuration-errors/
41. 15 Common Network Problems & How To Solve Them - SADOS, accessed on August 1, 2025, https://sados.com/blog/15-network-problem-solutions/
42. 8 Common Network Issues & How to Address Them - NinjaOne, accessed on August 1, 2025, https://www.ninjaone.com/blog/common-network-issues/
43. Starlink Acknowledges Software Failure Behind Outage of Satellite ..., accessed on August 1, 2025, https://www.cnet.com/news-live/starlink-acknowledges-software-failure-behind-outage-of-satellite-internet-service/
44. 10 historical software bugs with extreme consequences - Pingdom, accessed on August 1, 2025, https://www.pingdom.com/blog/10-historical-software-bugs-with-extreme-consequences/

45. What Is Unusual Software Bug? | NinjaOne, accessed on August 1, 2025, https://www.ninjaone.com/it-hub/endpoint-security/what-is-unusual-software-bug/
46. 7 Common Types of Software Bugs or Defects | BrowserStack, accessed on August 1, 2025, https://www.browserstack.com/guide/types-of-software-bugs
47. How to Resolve Network Failures: It's Not Down But It's Slow - Obkio, accessed on August 1, 2025, https://obkio.com/blog/network-failures/
48. Packet Loss Explained - Causes and Best Solutions | IR, accessed on August 1, 2025, https://www.ir.com/guides/what-is-network-packet-loss
49. Software Error Incident Categorizations in Aerospace | Journal of ..., accessed on August 1, 2025, https://arc.aiaa.org/doi/10.2514/1.I011240
50. Time Series Data Analysis | InfluxData, accessed on August 1, 2025, https://www.influxdata.com/what-is-time-series-data/
51. Time series analysis: a gentle introduction - Quix, accessed on August 1, 2025, https://quix.io/blog/time-series-analysis
52. TSAGen: Synthetic Time Series Generation for KPI ... - Tongqing Zhou, accessed on August 1, 2025, https://tongqingzhou-nudt.github.io/pubs/2021tnsm_2.pdf
53. Understanding Time Series Metrics in Observability - Edge Delta, accessed on August 1, 2025, https://edgedelta.com/company/blog/what-are-time-series-metrics
54. Signal processing - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Signal_processing
55. Time Series Analysis: Steps, Types, and Examples - MATLAB ..., accessed on August 1, 2025, https://www.mathworks.com/discovery/time-series-analysis.html
56. Digital Signal Processing in Machine Learning | Teradata, accessed on August 1, 2025, https://www.teradata.com/insights/ai-and-machine-learning/digital-signal-processing-machine-learning
57. Extraction of Features for Time Series Classification Using Noise Injection - MDPI, accessed on August 1, 2025, https://www.mdpi.com/1424-8220/24/19/6402
58. How to Handle Noise in Your Time Series Data | by Witsarut ..., accessed on August 1, 2025, https://medium.com/@row3no6/how-to-handle-noise-in-your-time-series-data-5979ca2ceb5f
59. What are the advantages and disadvantages to the various ..., accessed on August 1, 2025, https://www.adinstruments.com/support/knowledge-base/what-are-advantages-and-disadvantages-various-smoothing-functions-available
60. Savitzky–Golay filter - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter
61. Study of smoothing filters – Savitzky-Golay filters | Bart Wronski, accessed on August 1, 2025, https://bartwronski.com/2021/11/03/study-of-smoothing-filters-savitzky-golay-filters/
62. Savitzky-Golay Filter, accessed on August 1, 2025,

http://www.statistics4u.info/fundstat_eng/cc_filter_savgolay.html
63. Introduction to the Savitzky-Golay Filter: A Comprehensive Guide (Using Python) - Medium, accessed on August 1, 2025, https://medium.com/pythoneers/introduction-to-the-savitzky-golay-filter-a-comprehensive-guide-using-python-b2dd07a8e2ce
64. What is a Savitzky Golay filter and how can I use to to remove noise from my signal? Is it better than adjacent averaging? | ResearchGate, accessed on August 1, 2025, https://www.researchgate.net/post/What_is_a_Savitzky_Golay_filter_and_how_can_I_use_to_to_remove_noise_from_my_signal_Is_it_better_than_adjacent_averaging
65. 3.5 The Fourier Transform | A Very Short Course on Time Series ..., accessed on August 1, 2025, https://bookdown.org/rdpeng/timeseriesbook/the-fourier-transform.html
66. Modeling Toolkit For Time Series Analysis — AstroML Interactive Book, accessed on August 1, 2025, https://www.astroml.org/astroML-notebooks/chapter10/astroml_chapter10_Modeling_Toolkit_for_Time_Series_Analysis.html
67. An Interactive Guide To The Fourier Transform - BetterExplained, accessed on August 1, 2025, https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/
68. A time series technique Fourier all seasons - Lawrence R. De Geest, accessed on August 1, 2025, https://lrdegeest.github.io/blog/fourier-series
69. fourier transform - Identify random repetitive patterns - Signal Processing Stack Exchange, accessed on August 1, 2025, https://dsp.stackexchange.com/questions/52680/identify-random-repetitive-patterns
70. Periodicity: Detecting Rhythms in Data - Let's Data Science, accessed on August 1, 2025, https://letsdatascience.com/periodicity-detecting-rhythms-in-data/
71. A fully automated periodicity detection in time series, accessed on August 1, 2025, https://project.inria.fr/aaltd19/files/2019/08/AALTD_19_Boussard.pdf
72. Supervised learning - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Supervised_learning
73. Supervised and Unsupervised learning - GeeksforGeeks, accessed on August 1, 2025, https://www.geeksforgeeks.org/machine-learning/supervised-unsupervised-learning/
74. Why Recurrent Neural Networks (RNNs) Dominate Sequential Data Analysis - Shelf.io, accessed on August 1, 2025, https://shelf.io/blog/recurrent-neural-networks/
75. Sequential Data — and the Neural Network Conundrum! | by Aashish Chaubey | Analytics Vidhya | Medium, accessed on August 1, 2025, https://medium.com/analytics-vidhya/sequential-data-and-the-neural-network-conundrum-b2c005f8f865

76. What is RNN? - Recurrent Neural Networks Explained - AWS, accessed on August 1, 2025, https://aws.amazon.com/what-is/recurrent-neural-network/
77. Recurrent neural network - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Recurrent_neural_network
78. What is Recurrent Neural Networks (RNN)? - Analytics Vidhya, accessed on August 1, 2025, https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/
79. Recurrent Neural Network - NVIDIA Developer, accessed on August 1, 2025, https://developer.nvidia.com/discover/recurrent-neural-network
80. What Are Recurrent Neural Networks (RNNs)? | Built In, accessed on August 1, 2025, https://builtin.com/data-science/recurrent-neural-networks-and-lstm
81. Recurrent neural networks (RNNs) for sequential data | AI and Art Class Notes | Fiveable, accessed on August 1, 2025, https://library.fiveable.me/art-and-artificial-intelligence/unit-5/recurrent-neural-networks-rnns-sequential-data/study-guide/2mS9X7nSbEgzZBSm
82. What is a Recurrent Neural Network (RNN)? - IBM, accessed on August 1, 2025, https://www.ibm.com/think/topics/recurrent-neural-networks
83. Introduction to Recurrent Neural Networks - GeeksforGeeks, accessed on August 1, 2025, https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/
84. How to Solve the Vanishing Gradient Problem in RNNs: Why Ghajini? - Medium, accessed on August 1, 2025, https://medium.com/@digitaldadababu/how-to-solve-the-vanishing-gradient-problem-in-rnns-why-ghajini-d36fc29dccd3
85. Recurrent Neural Networks (RNN) - The Vanishing Gradient Problem, accessed on August 1, 2025, https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem
86. Vanishing Gradient Problem in Deep Learning: Explained - DigitalOcean, accessed on August 1, 2025, https://www.digitalocean.com/community/tutorials/vanishing-gradient-problem
87. Vanishing gradient problem - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Vanishing_gradient_problem
88. Long Short-Term Memory (LSTM) | wb-tutorials – Weights & Biases - Wandb, accessed on August 1, 2025, https://wandb.ai/wandb_fc/wb-tutorials/reports/Tutorial-Long-Short-Term-Memory-LSTM---Vmlldzo0NTM5ODI0
89. Introduction to Long Short-Term Memory(LSTM) | Simplilearn, accessed on August 1, 2025, https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm
90. What is LSTM - Long Short Term Memory? - GeeksforGeeks, accessed on August 1, 2025, https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-lon

g-short-term-memory/
91. Understanding LSTM and its diagrams | by Shi Yan - ML Review, accessed on August 1, 2025, https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714
92. Long Short Term Memory Networks | Architecture Of LSTM - Analytics Vidhya, accessed on August 1, 2025, https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/
93. Applications of Long Short-Term Memory (LSTM) Networks in Polymeric Sciences: A Review, accessed on August 1, 2025, https://www.mdpi.com/2073-4360/16/18/2607
94. accessed on January 1, 1970, httpshttps://www.mdpi.com/2073-4360/16/18/2607
95. Long short-term memory - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Long_short-term_memory
96. 9.2. Long Short-Term Memory (LSTM) — Dive into Deep Learning ..., accessed on August 1, 2025, https://classic.d2l.ai/chapter_recurrent-modern/lstm.html
97. Understanding the Core of LSTM: Forget Gate, Input Gate ... - Medium, accessed on August 1, 2025, https://medium.com/@shishiradhikari444/understanding-the-core-of-lstm-forget-gate-input-gate-candidate-memory-and-output-gate-850a4e8dae69
98. Long Short Term Memory Networks Explanation - GeeksforGeeks, accessed on August 1, 2025, https://www.geeksforgeeks.org/machine-learning/long-short-term-memory-networks-explanation/
99. What is LSTM? Introduction to Long Short-Term Memory, accessed on August 1, 2025, https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/
100. What's the use of output gate of LSTM? - Stack Overflow, accessed on August 1, 2025, https://stackoverflow.com/questions/42712256/whats-the-use-of-output-gate-of-lstm
101. The Math Behind LSTM | Towards Data Science, accessed on August 1, 2025, https://towardsdatascience.com/the-math-behind-lstm-9069b835289d/
102. 10.10 LSTM - CEDAR, accessed on August 1, 2025, https://cedar.buffalo.edu/~srihari/CSE676/10.10%20LSTM.pdf
103. Anomaly Detection for Time Series Data: Techniques and Models, accessed on August 1, 2025, https://victoriametrics.com/blog/victoriametrics-anomaly-detection-handbook-chapter-3/
104. A simple method for unsupervised anomaly detection: An application to Web time series data | PLOS One, accessed on August 1, 2025, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0262463
105. Unsupervised Deep Anomaly Detection for Industrial Multivariate Time Series Data - MDPI, accessed on August 1, 2025,

https://www.mdpi.com/2076-3417/14/2/774
106. [D] Struggling with Autoencoder-Based Anomaly Detection for Fraud Detection – Need Guidance : r/MachineLearning - Reddit, accessed on August 1, 2025, https://www.reddit.com/r/MachineLearning/comments/1gl92zm/d_struggling_with_autoencoderbased_anomaly/
107. Autoencoder - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Autoencoder
108. Autoencoders in Machine Learning - GeeksforGeeks, accessed on August 1, 2025, https://www.geeksforgeeks.org/machine-learning/auto-encoders/
109. Auto-Encoder: What Is It? And What Is It Used For? (Part 1 ..., accessed on August 1, 2025, https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726/
110. What Is an Autoencoder? | IBM, accessed on August 1, 2025, https://www.ibm.com/think/topics/autoencoder
111. Complete Guide to Anomaly Detection with ... - Analytics Vidhya, accessed on August 1, 2025, https://www.analyticsvidhya.com/blog/2022/01/complete-guide-to-anomaly-detection-with-autoencoders-using-tensorflow/
112. Anomaly Detection Using Autoencoders: A Deep Dive into Fraud ..., accessed on August 1, 2025, https://medium.com/@stacymacbrains/anomaly-detection-using-autoencoders-a-deep-dive-into-fraud-detection-9f59bcb5ab32
113. Anomaly Detection Using Autoencoder Reconstruction upon Industrial Motors - PMC, accessed on August 1, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9103022/
114. Autoencoder reconstruction error threshold - Cross Validated - Stats Stackexchange, accessed on August 1, 2025, https://stats.stackexchange.com/questions/427448/autoencoder-reconstruction-error-threshold
115. LSTM-Autoencoder Based Anomaly Detection Using Vibration Data ..., accessed on August 1, 2025, https://www.mdpi.com/1424-8220/24/9/2833
116. Anomaly Detection in Time Series Data using LSTM Autoencoders ..., accessed on August 1, 2025, https://medium.com/@zhonghong9998/anomaly-detection-in-time-series-data-using-lstm-autoencoders-51fd14946fa3
117. A Gentle Introduction to LSTM Autoencoders ..., accessed on August 1, 2025, https://machinelearningmastery.com/lstm-autoencoders/
118. Filter Time Series data using NumPy and SciPy Signal processing, accessed on August 1, 2025, https://www.w3resource.com/python-exercises/numpy/filter-time-series-data-using-numpy-and-scipy-signal-processing.php
119. Generating Time Series Data for Python Analysis - Index.dev, accessed on August 1, 2025, https://www.index.dev/blog/generate-time-series-data-python

120. Time Series Data with NumPy - KDnuggets, accessed on August 1, 2025, https://www.kdnuggets.com/time-series-data-with-numpy
121. Time Series Anomaly Detection with PyCaret | by PySquad | Medium, accessed on August 1, 2025, https://medium.com/@pysquad/time-series-anomaly-detection-with-pycaret-e1cf6fda5216
122. tutorials/12-anomaly-detection/README.md at main · colonyos ..., accessed on August 1, 2025, https://github.com/colonyos/tutorials/blob/main/12-anomaly-detection/README.md
123. Preptimize: Automation of Time Series Data Preprocessing and ..., accessed on August 1, 2025, https://www.mdpi.com/1999-4893/17/8/332
124. Evaluating Preprocessing Strategies for Time Series Prediction Using Deep Learning Architectures - AAAI, accessed on August 1, 2025, https://cdn.aaai.org/ocs/15475/15475-68721-1-PB.pdf
125. How to Normalize and Standardize Time Series Data in Python - Machine Learning Mastery, accessed on August 1, 2025, https://machinelearningmastery.com/normalize-standardize-time-series-data-python/
126. 7.3. Preprocessing data — scikit-learn 1.7.1 documentation, accessed on August 1, 2025, https://scikit-learn.org/stable/modules/preprocessing.html
127. Understanding the Implications of Scaling Test Data Using the Same ..., accessed on August 1, 2025, https://www.kaggle.com/discussions/questions-and-answers/415136
128. ML Series: Day 47 — Scaling and Normalization | by Ebrahim Mousavi | Medium, accessed on August 1, 2025, https://medium.com/@ebimsv/ml-series-day-47-scaling-and-normalization-073e6a10fa7b
129. How to Apply Min-Max Scaler on Time Series Data (Using Python ..., accessed on August 1, 2025, https://medium.com/@mohcenelmakkaoui/how-to-apply-min-max-scaler-on-time-series-data-using-64363eef0690
130. What is a sliding window approach in time series forecasting? - Milvus, accessed on August 1, 2025, https://milvus.io/ai-quick-reference/what-is-a-sliding-window-approach-in-time-series-forecasting
131. Sliding Window Technique — reduce the complexity of your ..., accessed on August 1, 2025, https://medium.com/@data-overload/sliding-window-technique-reduce-the-complexity-of-your-algorithm-5badb2cf432f
132. Develop LSTM Models for Time Series Forecasting - Kaggle, accessed on August 1, 2025, https://www.kaggle.com/code/ritesh7355/develop-lstm-models-for-time-series-forecasting
133. How to Develop LSTM Models for Time Series Forecasting ..., accessed on

August 1, 2025, https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
134. Keras vs TensorFlow vs PyTorch: Key Differences 2025 - Carmatec, accessed on August 1, 2025, https://www.carmatec.com/blog/keras-vs-tensorflow-vs-pytorch-key-differences/
135. Keras vs PyTorch in 2025: The Comparison | DistantJob - Remote ..., accessed on August 1, 2025, https://distantjob.com/blog/keras-vs-pytorch/
136. Pytorch Vs Tensorflow Vs Keras: The Differences You Should Know, accessed on August 1, 2025, https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article
137. Keras: Deep Learning for humans, accessed on August 1, 2025, https://keras.io/
138. PyTorch vs TensorFlow: Comparative Guide of AI Frameworks 2025 - OpenCV, accessed on August 1, 2025, https://opencv.org/blog/pytorch-vs-tensorflow/
139. The Sequential model | TensorFlow Core, accessed on August 1, 2025, https://www.tensorflow.org/guide/keras/sequential_model
140. The Sequential model - TensorFlow for R, accessed on August 1, 2025, https://tensorflow.rstudio.com/guides/keras/sequential_model
141. 3 ways to create a Keras model with TensorFlow 2.0 (Sequential ..., accessed on August 1, 2025, https://pyimagesearch.com/2019/10/28/3-ways-to-create-a-keras-model-with-tensorflow-2-0-sequential-functional-and-model-subclassing/
142. Mastering the Adam Optimizer: Unlocking Superior Deep Learning ..., accessed on August 1, 2025, https://www.lunartech.ai/blog/mastering-the-adam-optimizer-unlocking-superior-deep-learning-performance
143. Complete Guide to the Adam Optimization Algorithm | Built In, accessed on August 1, 2025, https://builtin.com/machine-learning/adam-optimization
144. What is Adam Optimizer? - Analytics Vidhya, accessed on August 1, 2025, https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/
145. What is Adam Optimizer? - GeeksforGeeks, accessed on August 1, 2025, https://www.geeksforgeeks.org/deep-learning/adam-optimizer/
146. Mean squared error - Wikipedia, accessed on August 1, 2025, https://en.wikipedia.org/wiki/Mean_squared_error
147. What is Loss Function? | IBM, accessed on August 1, 2025, https://www.ibm.com/think/topics/loss-function
148. Linear regression: Loss | Machine Learning | Google for Developers, accessed on August 1, 2025, https://developers.google.com/machine-learning/crash-course/linear-regression/loss
149. www.ultralytics.com, accessed on August 1, 2025, https://www.ultralytics.com/glossary/epoch#:~:text=In%20machine%20learning%20(ML)%2C,seeing%20examples%20from%20the%20data.

150. What Is an Epoch in Machine Learning? | Coursera, accessed on August 1, 2025, https://www.coursera.org/articles/epoch-in-machine-learning
151. What is Epoch in Machine Learning | Deepchecks, accessed on August 1, 2025, https://www.deepchecks.com/glossary/epoch-in-machine-learning/
152. Epoch in Machine Learning | Understanding the Core of Model ..., accessed on August 1, 2025, https://medium.com/@saiwadotai/epoch-in-machine-learning-understanding-the-core-of-model-training-bfd64bbd5604
153. Basic concepts of Streamlit, accessed on August 1, 2025, https://docs.streamlit.io/get-started/fundamentals/main-concepts
154. Streamlit • A faster way to build and share data apps, accessed on August 1, 2025, https://streamlit.io/
155. Building a Real-Time Dashboard with Streamlit and Kafka - Dev3lop, accessed on August 1, 2025, https://dev3lop.com/building-a-real-time-dashboard-with-streamlit-and-kafka/
156. st.line_chart - Streamlit Docs, accessed on August 1, 2025, https://docs.streamlit.io/develop/api-reference/charts/st.line_chart
157. st.metric - Streamlit Docs, accessed on August 1, 2025, https://docs.streamlit.io/develop/api-reference/data/st.metric
158. How to display metrics in streamlit - ProjectPro, accessed on August 1, 2025, https://www.projectpro.io/recipes/display-metrics-streamlit
159. Handling Server Errors in Streamlit: Capturing and Displaying Error ..., accessed on August 1, 2025, https://discuss.streamlit.io/t/handling-server-errors-in-streamlit-capturing-and-displaying-error-messages-to-users/45142
160. st.error - Streamlit Docs, accessed on August 1, 2025, https://docs.streamlit.io/develop/api-reference/status/st.error