A Foundational Treatise on the Simulation of LEO Satellite Communication Links for AI-Driven Adaptive Coding and Modulation

Introduction

The New Space Race: The Imperative for Dynamic Spectrum Management in LEO Constellations

The contemporary space domain is characterized by a paradigm shift, driven by the large-scale deployment of Low Earth Orbit (LEO) satellite constellations. Thousands of satellites are being launched to provide global broadband internet, Earth observation, and other services. This proliferation, while promising unprecedented connectivity, introduces significant technical challenges. Unlike geostationary (GEO) satellites that maintain a fixed position relative to a ground observer, LEO satellites traverse the sky in minutes, creating a highly dynamic communication environment. The geometry of the link—encompassing slant range, elevation angle, and atmospheric path length—changes continuously. This dynamism results in rapid and substantial variations in signal strength and quality. Consequently, the traditional approach of designing communication links for worst-case scenarios, which ensures connectivity at the cost of profound inefficiency, is no longer tenable. In the congested and competitive spectral environment of the new space race, static link design wastes valuable capacity for the majority of a satellite pass. The imperative is clear: to unlock the full potential of LEO constellations, communication systems must be able to adapt to changing channel conditions in near-real-time.

An Overview of Adaptive Coding and Modulation (ACM) as a Key Enabler

Adaptive Coding and Modulation (ACM) emerges as the key enabling technology to address the challenges of dynamic LEO links. ACM is a sophisticated form of link adaptation where the physical layer waveform—specifically the modulation scheme and forward error correction (FEC) code rate—is dynamically adjusted in response to the measured quality of the communication channel.[1] The fundamental principle of ACM is to trade link margin for data throughput. When channel conditions are favorable (e.g., the satellite is at a high elevation angle with clear skies), the system selects a higher-order modulation scheme (like 16-APSK or 32-APSK) and a higher code rate (e.g., 8/9), packing more data bits into each transmitted symbol. Conversely, when the channel degrades (e.g., due to low elevation angle, rain fade, or scintillation), the system switches to a more robust, lower-order scheme (like QPSK) and a lower code rate (e.g., 1/2 or 1/4) to ensure the link remains error-free, albeit at a lower data rate.[3] This intelligent adaptation ensures that the link is always operating at or near its maximum possible capacity for the given conditions, dramatically increasing the overall efficiency and data volume transferred during a pass.[5]

Objectives and Structure of this Treatise

The primary objective of this treatise is to provide a comprehensive, step-by-step guide to building a high-fidelity physical layer simulation of a LEO satellite communication link. This simulation is not an end in itself; it is designed to serve as a foundational environment for the development, testing, and validation of advanced, AI-driven ACM strategies. The report is structured to guide the reader through a logical progression of increasingly complex modeling tasks, from the macro-scale of orbital mechanics to the micro-scale of signal-to-noise ratio calculation and adaptive decision-making.

- **Part I: The Dynamic Geometry of LEO Links** establishes the foundational spatio-temporal model, calculating the precise time-varying geometry between the satellite and a ground station.
- **Part II: Modeling the Propagation Channel** translates this geometry into physical signal impairment, quantifying the losses incurred as the signal traverses the atmosphere.
- **Part III: The Satellite Link Budget as a Computational Graph** integrates all

gains and losses into a comprehensive link budget to compute the dynamic carrier-to-noise ratio, the ultimate measure of link quality.
- **Part IV: AI-Driven Adaptive Coding and Modulation** uses the calculated link quality to implement the ACM logic based on the DVB-S2/S2X standards and provides a complete framework for integrating and training artificial intelligence agents to control the link.

The Python Ecosystem for Satellite Communications

Historically, the simulation of complex systems like satellite links required monolithic, proprietary, and often prohibitively expensive software suites. These tools, while powerful, frequently operated as "black boxes," obscuring the underlying models and limiting extensibility. A significant development in engineering is the rise of a powerful, open-source scientific computing ecosystem, particularly in Python.[6] This treatise leverages this ecosystem to construct a simulation that is not only accurate and robust but also transparent, modular, and infinitely extensible.

The true innovation of the approach detailed herein lies not in any single library but in the synergistic integration of several specialized, research-grade packages. Each library is a domain expert, and by composing them, we create a holistic simulation that is greater than the sum of its parts. skyfield provides world-class astrodynamics calculations [8],

ITU-Rpy offers a rigorous implementation of international standards for radio wave propagation [10], and

pylink-satcom furnishes a generic yet powerful framework for solving the complex, interdependent calculations inherent in a link budget.[12] This component-based philosophy allows for a clear separation of concerns, making the simulation easier to understand, validate, and extend. It represents a modern, flexible, and accessible approach to a classic and challenging engineering problem.

Table 1: Key Python Libraries for LEO Link Simulation

Library Name	PyPI Package Name	Primary Role in Simulation	Key Snippet References

skyfield	skyfield	Orbital mechanics and geometry	8
ITU-Rpy	itur	Atmospheric propagation loss models	10
pylink-satcom	pylink-satcom	Link budget DAG solver and analysis	12
numpy	numpy	Numerical computation backbone	6
pandas	pandas	Data handling and time-series analysis	6

Part I: The Dynamic Geometry of LEO Links

This initial part of the treatise is dedicated to establishing the foundational geometry of the simulation. Before any signal propagation or link budget can be calculated, the precise, time-varying spatial relationship between the LEO satellite and the ground station must be accurately modeled. This involves understanding the principles of orbital mechanics, the data formats used to describe satellite orbits, and the software tools required to compute the essential geometric parameters—slant range, azimuth, and elevation angle—over time.

Chapter 1: Foundations of Orbital Mechanics for Simulation

The SGP4 Propagation Model and the Two-Line Element (TLE) Set

For satellites orbiting the Earth, the de facto standard for orbit propagation is the

Simplified General Perturbations 4 (SGP4) model. SGP4 is an analytical theory that accounts for the primary perturbations on a satellite's orbit, including the Earth's non-spherical shape (oblateness) and atmospheric drag. It provides a balance between computational efficiency and accuracy, making it ideal for tracking the large number of objects in Earth orbit.

The input data for the SGP4 algorithm is a specific text format known as a Two-Line Element (TLE) set.[14] As the name implies, a TLE consists of two lines of 69 characters each, preceded by a title line for the satellite's name. These lines contain a set of mean orbital elements that describe the satellite's motion. Crucially, a TLE represents a snapshot of the satellite's orbital state at a specific moment in time, known as the "epoch".[16] The SGP4 propagator uses this epoch state to predict the satellite's position and velocity at other times, both past and future. However, due to unmodeled forces (like solar radiation pressure, atmospheric density variations, and station-keeping maneuvers), the accuracy of the SGP4 prediction degrades over time. For LEO satellites, a TLE is generally considered valid for only a week or two around its epoch date.[14] Therefore, for any operational simulation, it is essential to use the most recent TLEs available.

Time Systems in Astrodynamics

An often-underestimated complexity in astrodynamics is the precise management of time. Several different time scales are used, and converting between them is critical for accurate calculations. skyfield, the library we will use, expertly manages these conversions.[17] The most relevant time scales for our simulation are:

- **Universal Time (UT1):** Tied to the rotation of the Earth. It is the basis for civil time.
- **Coordinated Universal Time (UTC):** The international standard for timekeeping. It is based on International Atomic Time (TAI) but is periodically adjusted with "leap seconds" to stay synchronized with the Earth's irregular rotation. This is the time scale typically used for scheduling and operations.
- **International Atomic Time (TAI):** A highly precise time scale based on a weighted average of hundreds of atomic clocks worldwide. It does not have leap seconds.
- **Terrestrial Time (TT):** A theoretical time scale used for ephemeris calculations. It is conceptually the time that would be measured by a clock on the Earth's surface

(at the geoid) if the Earth were not rotating.

Fortunately, skyfield abstracts away most of this complexity through its Timescale object. By creating a single Timescale instance at the beginning of a script, a user can build Time objects from various formats (e.g., UTC datetimes) and trust that all internal calculations are performed in the correct, consistent reference frame.[17]

Chapter 2: Implementing Satellite and Ground Station Models with skyfield

Acquiring and Loading TLE Data

The most common source for publicly available TLEs is CelesTrak, which maintains curated lists of TLEs for various satellite categories.[16] The

skyfield library provides convenient functions for downloading and caching this data. It is considered best practice to download the data to a local file and reuse it, rather than repeatedly querying the CelesTrak servers for each run of the simulation. This reduces the load on public infrastructure and speeds up the simulation initialization. The load.download() function in skyfield can be used for this purpose, and it can be configured to automatically re-download the file if it becomes older than a specified number of days.[16]

Once a TLE file is available locally, it can be parsed to create a list of EarthSatellite objects. skyfield allows parsing an entire file at once or loading a single satellite from its two TLE lines provided as strings.[14]

Defining Topocentric Observer Locations (Ground Stations)

To calculate the view of a satellite from a specific point on Earth, that point must be defined. This is known as a topocentric location. skyfield uses the World Geodetic System 1984 (WGS84) ellipsoid model of the Earth for these calculations. The wgs84.latlon() method is the standard way to create a topocentric location object, requiring latitude and longitude in degrees, and optionally, elevation in meters.[9] For this treatise, we will use Delhi, India, as our example ground station, with coordinates of approximately 28.61° N latitude and 77.23° E longitude.[19]

Code Implementation: Instantiating Satellite and Ground Station Objects

The following Python code demonstrates the practical steps to set up the simulation's core objects. It initializes the skyfield loader, downloads a TLE file for active satellites from CelesTrak, parses the file to find the International Space Station (ISS), and defines a ground station object for Delhi.

Python

```python
import pandas as pd
from skyfield.api import load, wgs84, N, E

# --- 1. Initialize Skyfield Loader and Timescale ---
# The loader manages downloading and caching of data files.
eph = load('de421.bsp')  # A planetary ephemeris is needed by the timescale
earth = eph['earth']
ts = load.timescale()
print("Skyfield timescale and ephemeris loaded.")

# --- 2. Acquire and Load TLE Data for a Satellite ---
# URL for active satellites from CelesTrak
satellites_url = 'https://celestrak.org/NORAD/elements/gp.php?GROUP=active&FORMAT=tle'
filename = 'active.tle'

# Download the file if it's older than 1 day, otherwise use the cached version.
if not load.exists(filename) or load.days_old(filename) > 1.0:
    print(f"Downloading fresh TLE data to '{filename}'...")
    load.download(satellites_url, filename=filename)
else:
    print(f"Using cached TLE data '{filename}' (less than 1 day old).")
```

```
# Parse the TLE file to get a list of satellite objects
sats = load.tle_file(filename)
satellites_by_name = {sat.name: sat for sat in sats}
print(f"Loaded {len(sats)} satellites.")

# Select a specific satellite, e.g., the ISS
satellite_name = 'ISS (ZARYA)'
try:
    satellite = satellites_by_name[satellite_name]
    print(f"\nSelected satellite: {satellite.name}")
    print(f"NORAD ID: {satellite.model.satnum}")
    print(f"Epoch: {satellite.epoch.utc_jpl()}")
except KeyError:
    print(f"Error: Satellite '{satellite_name}' not found in '{filename}'. Exiting.")
    exit()

# --- 3. Define a Topocentric Ground Station ---
# Using coordinates for Delhi, India [19, 20]
delhi_lat = 28.61 * N
delhi_lon = 77.23 * E
delhi_alt_m = 216.0

ground_station = earth + wgs84.latlon(
    latitude_degrees=delhi_lat,
    longitude_degrees=delhi_lon,
    elevation_m=delhi_alt_m
)
print(f"\nDefined ground station 'Delhi' at:")
print(f"  Latitude: {delhi_lat:.2f}")
print(f"  Longitude: {delhi_lon:.2f}")
print(f"  Elevation: {delhi_alt_m} m")
```

Chapter 3: Calculating Time-Varying Link Geometry

Deriving Slant Range, Azimuth, and Elevation Angles

With the satellite and ground station objects defined, the core geometric calculation can be performed. This involves determining the satellite's position relative to the ground station at a specific time. A novice user of skyfield might be tempted to use the observe() method, as it is commonly shown in examples involving planets.[9] However, for Earth-orbiting satellites, this is both computationally inefficient and physically unnecessary. The

observe() method is designed to account for light-travel time, a significant factor over interplanetary distances. For a LEO satellite at a 500 km altitude, the light-travel time is a mere 1.7 milliseconds. During this time, the satellite moves less than 15 meters. Given that the SGP4 model's intrinsic accuracy is on the order of one kilometer around its epoch [16], correcting for a few meters of movement is a negligible refinement that adds unnecessary computational overhead.

The correct and efficient method, as recommended in the skyfield documentation, is to perform a simple vector subtraction.[16] The expression

satellite - ground_station creates a new vector function. When its .at(time) method is called, skyfield computes the geocentric positions of both the satellite and the ground station and subtracts them, yielding a topocentric position vector pointing from the ground station to the satellite. From this topocentric vector, the key link parameters can be extracted using the .altaz() method, which returns:

- **Altitude:** The angle of the satellite above the local horizon (0° to 90°).
- **Azimuth:** The compass direction to the satellite, measured clockwise from North (0°).
- **Distance:** The straight-line distance, or slant range, to the satellite.

Generating a Time-Series Dataset for a Complete Satellite Pass

To simulate a full communication session, we need to calculate these geometric parameters not just for a single instant, but for a series of time steps throughout a satellite pass. A naive implementation might loop through time, performing one calculation at a time. This is highly inefficient in Python. skyfield is built on numpy and is optimized for vectorized operations. By creating a Time object that contains an array of moments, we can perform the entire geometric calculation for all time steps in a single, highly efficient operation.[17]

The first step is to identify the times when the satellite will be visible. The find_events() method is perfect for this, as it can find the rise, culmination, and set times for a satellite relative to a ground station over a given period.[14] Once a pass is identified, we can create a time array spanning from the rise to the set time and then compute the geometry for the entire pass at once. The results can be stored in a

pandas DataFrame, creating a structured, time-indexed dataset that will serve as the input for the subsequent stages of the simulation.

Code Implementation: A Function to Generate Pass Geometry Data

The following Python function encapsulates this process. It takes a satellite, a ground station, and a time window, finds the next visible pass, and returns a pandas DataFrame containing the time-series of the link geometry for that pass.

Python

```python
def generate_pass_geometry(satellite, ground_station, ts, start_time, end_time):
    """
    Finds the next satellite pass and calculates its geometry.

    Args:
        satellite (skyfield.sgp4lib.EarthSatellite): The satellite object.
        ground_station (skyfield.vectorlib.VectorSum): The ground station object.
        ts (skyfield.timelib.Timescale): The timescale object.
        start_time (skyfield.timelib.Time): The start time for the search window.
        end_time (skyfield.timelib.Time): The end time for the search window.

    Returns:
        pandas.DataFrame: A DataFrame with geometry data for the next pass,
                          or None if no pass is found.
    """
    # Find rise, culmination, and set events for the satellite
    times, events = satellite.find_events(ground_station, start_time, end_time,
altitude_degrees=5.0)
    event_names = 'rise', 'culminate', 'set'

    # Check if a full pass (rise and set) is found
    pass_indices = [i for i, event in enumerate(events) if event_names[event] == 'rise']
    if not pass_indices:
        print("No passes found in the specified window.")
        return None

    # Select the first complete pass
    rise_time = times[pass_indices]
    set_time = times[pass_indices + 2] # Assuming rise, culminate, set sequence

    print(f"\nFound pass from {rise_time.utc_strftime('%Y-%m-%d %H:%M:%S')} UTC to
{set_time.utc_strftime('%Y-%m-%d %H:%M:%S')} UTC")

    # Generate a time array for the duration of the pass with a 10-second step
    pass_duration_days = set_time - rise_time
    num_steps = int(pass_duration_days * 24 * 60 * 6) # 10-second steps
    time_range = ts.linspace(rise_time, set_time, num_steps)

    # Perform vectorized calculation of the geometry
    difference = satellite - ground_station
    topocentric = difference.at(time_range)
    alt, az, dist = topocentric.altaz()

    # Store results in a pandas DataFrame
    df = pd.DataFrame({
        'timestamp_utc': time_range.utc_datetime(),
        'elevation_deg': alt.degrees,
        'azimuth_deg': az.degrees,
        'slant_range_km': dist.km
    })
    df.set_index('timestamp_utc', inplace=True)

    print(f"Generated geometry data for {len(df)} time steps.")
    return df

# --- Example Usage ---
# Define a 24-hour window to search for a pass
now = ts.now()
start_of_pass_search = now
end_of_pass_search = ts.utc(now.utc.year, now.utc.month, now.utc.day + 1)

# Generate the geometry data
pass_geometry_df = generate_pass_geometry(
    satellite, ground_station, ts, start_of_pass_search, end_of_pass_search
)

if pass_geometry_df is not None:
    print("\nSample of generated pass geometry data:")
    print(pass_geometry_df.head())
```

Part II: Modeling the Propagation Channel

Having established the dynamic geometry of the LEO link, the next critical step is to model the physical medium through which the signal propagates. The vacuum of space is nearly lossless, but the Earth's atmosphere is not. As the radio signal travels from the satellite to the ground station, it is attenuated by various atmospheric constituents. This part of the treatise focuses on quantifying these losses, which are a primary driver of the channel variability that ACM seeks to overcome. We will combine the fundamental Free Space Path Loss with detailed atmospheric models based on internationally recognized standards.

Chapter 4: The Physics of Signal Attenuation

Free Space Path Loss (FSPL)

The most significant source of signal loss in any satellite link is Free Space Path Loss

(FSPL). This is not a loss in the sense of energy being absorbed, but rather a consequence of the geometric spreading of the electromagnetic wave as it propagates away from the transmitter. As the wavefront expands in a sphere, the power flux density (power per unit area) decreases with the square of the distance.[22] The FSPL formula captures this effect and also includes a dependency on frequency. Higher frequency signals experience greater path loss for the same distance because the effective aperture of a receiving antenna of a given gain is smaller at higher frequencies.[22]

The FSPL in decibels (dB) can be calculated using the following standard formula [23]:

$FSPLdB=20\log10(d)+20\log10(f)+20\log10(c4\pi)$
where:

- d is the slant range (distance) in meters.
- f is the frequency in Hertz.
- c is the speed of light in m/s.

For satellite communication, it is often more convenient to express distance in kilometers and frequency in Gigahertz. The formula then becomes:

$FSPLdB=20\log10(dkm)+20\log10(fGHz)+92.45$
This loss is fundamental and will be calculated at every time step of our simulation using the slant range data generated in Part I.

Mechanisms of Atmospheric Attenuation

While FSPL is the largest loss component, it is the variable atmospheric losses that make the channel dynamic and necessitate adaptive systems. For satellite links operating at frequencies above 1 GHz, several atmospheric phenomena become critical, especially at low elevation angles where the signal path through the atmosphere is longest.[25] The primary mechanisms are:

- **Gaseous Absorption:** Certain molecules in the atmosphere, primarily oxygen (O2) and water vapor (H2O), have resonant frequencies at which they absorb radio wave energy. This absorption is highly frequency-dependent, with significant peaks around 22 GHz for water vapor and in a broad band around 60 GHz for oxygen.[26] This attenuation is always present but varies with temperature, pressure, and humidity.

- **Hydrometeors (Rain, Clouds, Fog):** Water droplets and ice crystals in the atmosphere can absorb and scatter radio signals. Rain is the most significant of these, with attenuation increasing dramatically with both rain rate and signal frequency.[28] Cloud and fog attenuation is generally less severe than rain but can be a significant factor, especially for systems with low link margins.[25] This is a highly variable and geographically dependent loss component.
- **Tropospheric Scintillation:** Turbulence in the troposphere causes rapid fluctuations in the refractive index of the air. These fluctuations can cause the signal to rapidly fade and enhance, an effect known as scintillation. This effect is most pronounced at low elevation angles, high frequencies, and for small-aperture antennas.[13]

Chapter 5: A Pythonic Approach to ITU-R Propagation Models

Introduction to the ITU-Rpy Library

Modeling these complex atmospheric effects from first principles is a formidable task. Fortunately, the International Telecommunication Union Radiocommunication Sector (ITU-R) publishes a series of recommendations that provide standardized, empirically validated models for predicting these losses. The ITU-Rpy library is a Python implementation of these recommendations, providing a powerful and accessible tool for radio propagation analysis.[10] A key feature of

ITU-Rpy is its use of numpy for vectorized calculations, allowing for the efficient computation of attenuation over large datasets of points or times, which aligns perfectly with our time-series simulation approach.[10]

Modeling Key Attenuation Components

The primary function for our simulation is itur.atmospheric_attenuation_slant_path. This function acts as a high-level wrapper that computes the total slant path attenuation by invoking the models from several underlying ITU-R recommendations [10]:

- **Gaseous Attenuation:** Modeled according to **ITU-R P.676**. This model calculates the specific attenuation due to oxygen and water vapor based on frequency, temperature, pressure, and water vapor density.[26]
- **Rain Attenuation:** Modeled according to **ITU-R P.618**, which itself relies on methods from **ITU-R P.838** to determine the specific attenuation from a given rain rate.[13] The model requires statistical rain rate data for the ground station's location, which
  ITU-Rpy can estimate using built-in maps from **ITU-R P.837** if not provided by the user.[10]
- **Cloud and Fog Attenuation:** Modeled according to **ITU-R P.840**. This model calculates attenuation based on the total columnar content of liquid water, which can also be estimated from internal maps.
- **Tropospheric Scintillation:** Modeled as part of the overall **ITU-R P.618** procedure. This model calculates the standard deviation of signal fluctuations based on factors like frequency, elevation angle, antenna diameter, and local climate data.[13]

A powerful feature of the itur.atmospheric_attenuation_slant_path function is the return_contributions=True flag. When set, the function returns not only the total attenuation but also the individual contributions from each of these sources, allowing for detailed analysis of the channel behavior.[10]

Code Implementation: A Module for Calculating Total Atmospheric Loss

The following Python function takes the geometric data from Part I, along with link and site parameters, and uses ITU-Rpy to compute a time-series of atmospheric losses.

Python

```
import itur
import numpy as np
```

```
def calculate_atmospheric_losses(geometry_df, ground_station_coords, link_params):
    """
    Calculates time-varying atmospheric losses for a satellite pass.

    Args:
        geometry_df (pandas.DataFrame): DataFrame with elevation and slant range.
        ground_station_coords (dict): Dict with 'lat', 'lon', 'alt_m'.
        link_params (dict): Dict with 'freq_GHz', 'ant_diam_m', 'rain_p'.

    Returns:
        pandas.DataFrame: A DataFrame with atmospheric loss components.
    """
    print("\nCalculating atmospheric losses...")

    # Extract necessary arrays from the input DataFrame and parameters
    lat = ground_station_coords['lat']
    lon = ground_station_coords['lon']
    el = geometry_df['elevation_deg'].values
    f = link_params['freq_GHz']
    D = link_params['ant_diam_m']
    p = link_params['rain_p'] # Percentage of time attenuation is exceeded

    # Use itur.atmospheric_attenuation_slant_path to compute all losses
    # The function is vectorized, so we can pass the entire elevation array
    Ag, Ac, Ar, As, A_total = itur.atmospheric_attenuation_slant_path(
        lat=lat,
        lon=lon,
        f=f,
        el=el,
        p=p,
        D=D,
        return_contributions=True
    )

    # Create a new DataFrame to hold the loss data
    loss_df = pd.DataFrame({
        'gaseous_loss_dB': Ag.value,
        'cloud_loss_dB': Ac.value,
        'rain_fade_dB': Ar.value,
        'scintillation_fade_dB': As.value,
        'total_atmospheric_loss_dB': A_total.value
    }, index=geometry_df.index)

    print(f"Atmospheric loss calculation complete.")
    return loss_df

# --- Example Usage ---
# Define link and ground station parameters for the loss model
ground_station_parameters = {
    'lat': delhi_lat.value,
    'lon': delhi_lon.value,
    'alt_m': delhi_alt_m
}

link_parameters = {
    'freq_GHz': 20.0,      # Ku-band downlink example
    'ant_diam_m': 1.2,     # VSAT antenna diameter
    'rain_p': 0.01         # Link availability target of 99.99% for rain fade
}

# Check if we have a valid pass geometry DataFrame from Part I
if 'pass_geometry_df' in locals() and pass_geometry_df is not None:
    # Calculate the atmospheric losses
    atmospheric_loss_df = calculate_atmospheric_losses(
        pass_geometry_df, ground_station_parameters, link_parameters
    )

    # Combine the geometry and loss data into a single DataFrame
    simulation_df = pass_geometry_df.join(atmospheric_loss_df)

    print("\nSample of combined simulation data with atmospheric losses:")
    print(simulation_df.head())
```

Chapter 6: Quantifying Total Path Loss

Combining FSPL and Atmospheric Losses

With the two major loss components calculated, we can now determine the total path loss for each time step of the satellite pass. Since both FSPL and atmospheric attenuation are expressed in decibels (dB), the total path loss is simply their sum.

LTotal(dB)=FSPL(dB)+ATotal_Atmospheric(dB)
This total loss value represents the reduction in signal power from the satellite's transmitting antenna to the input of the ground station's receiving antenna.

Analysis of Loss Contributions

By plotting the calculated loss components against the satellite's elevation angle, we can gain valuable insight into the link's behavior. A typical plot would show that FSPL varies relatively little during a pass, changing only as a function of the slant range. In contrast, the atmospheric losses, particularly rain and scintillation, would show a dramatic increase at low elevation angles. This is because the signal path length through the dense lower atmosphere is much longer when the satellite is near the horizon.

This analysis visually and quantitatively reinforces the need for ACM. A link designed with enough power to overcome the extreme losses at 5° elevation would be massively over-provisioned when the satellite is at its culmination (highest point), wasting an enormous amount of potential capacity. The simulation data we are generating provides the precise, time-varying profile of this challenge, which the ACM system will be designed to solve.

Python

```python
# --- Code to calculate FSPL and Total Path Loss ---

def calculate_fspl(slant_range_km, freq_GHz):
    """Calculates Free Space Path Loss in dB."""
    return 20 * np.log10(slant_range_km) + 20 * np.log10(freq_GHz) + 92.45
```

```
if 'simulation_df' in locals() and simulation_df is not None:
    # Calculate FSPL for each time step
    simulation_df = calculate_fspl(
        simulation_df['slant_range_km'],
        link_parameters['freq_GHz']
    )

    # Calculate the total path loss
    simulation_df = simulation_df + simulation_df

    print("\nSample of simulation data with total path loss:")
    print(simulation_df].head())

    # --- Plotting for analysis (requires matplotlib) ---
    try:
        import matplotlib.pyplot as plt

        fig, ax1 = plt.subplots(figsize=(12, 7))

        ax1.set_xlabel('Time during pass (UTC)')
        ax1.set_ylabel('Loss (dB)', color='tab:red')
        ax1.plot(simulation_df.index, simulation_df, label='Total Path Loss', color='tab:red')
        ax1.plot(simulation_df.index, simulation_df, label='FSPL', color='tab:orange',
linestyle='--')
        ax1.tick_params(axis='y', labelcolor='tab:red')
        ax1.grid(True)

        ax2 = ax1.twinx()
        ax2.set_ylabel('Elevation (degrees)', color='tab:blue')
        ax2.plot(simulation_df.index, simulation_df['elevation_deg'], label='Elevation Angle',
color='tab:blue')
        ax2.tick_params(axis='y', labelcolor='tab:blue')

        fig.suptitle(f'Link Loss Profile for {satellite.name} Pass over Delhi')
        fig.legend(loc='upper right', bbox_to_anchor=(0.9, 0.9))
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
        # To display the plot, uncomment the following line:
        # plt.show()
        print("\nPlot generated showing loss profile vs. elevation.")

    except ImportError:
        print("\nMatplotlib not installed. Skipping plot generation.")
```

Part III: The Satellite Link Budget as a Computational Graph

This part represents the central integration point of the simulation. Here, we combine the geometric parameters from Part I and the propagation losses from Part II with the performance characteristics of the satellite and ground station hardware. The goal is to construct a complete link budget, a systematic accounting of all gains and losses, to calculate the final, critical metric of link quality: the Carrier-to-Noise-Density Ratio (C/N0). We will eschew traditional spreadsheet-based methods in favor of a more robust, programmatic approach using a computational graph.

Chapter 7: Core Principles of the Satellite Link Budget

The Link Budget Equation

The link budget is the cornerstone of communication system design. It allows an engineer to predict the performance of a link before it is built. The ultimate goal is to calculate the ratio of the received carrier power (C) to the noise power spectral density (N0). This ratio, C/N0, is expressed in dB-Hz and is a fundamental measure of signal quality, independent of the final signal bandwidth. The master equation for a satellite downlink is [33]:

$$ \left(\frac{C}{N_0}\right)_{dB-Hz} = EIRP_{dBW} - L_{Total_{dB}} + \left(\frac{G}{T}\right)_{dB/K} - k_{dBW/K/Hz} $$

Key Parameters Explained

- **EIRP (Equivalent Isotropically Radiated Power):** This represents the effective power transmitted from the satellite in the direction of the receiver. It is the sum (in dB) of the transmitter's output power and the gain of its transmitting antenna.[24] For a given satellite, this is typically a fixed value, though it may vary slightly across the satellite's coverage area.
- **System Noise Temperature (Tsys):** Noise is an unavoidable random signal that contaminates the desired carrier signal. The total noise in a receiving system is quantified by its equivalent noise temperature, Tsys, measured in Kelvin (K). A lower noise temperature signifies a better-performing, "quieter" receiver. Tsys is the sum of noise contributions from all components in the receive chain.[34] The two primary contributors are:
  - **Antenna Noise Temperature (Tant):** The noise captured by the antenna from its surroundings. This includes cosmic background radiation, thermal noise from the Earth's surface seen by the antenna's sidelobes, and, critically, noise radiated by the atmosphere itself. Atmospheric attenuation and noise are two sides of the same coin; an attenuating medium like rain not only weakens the signal but also radiates thermal noise at its own physical temperature.[29] Therefore,
    Tant increases significantly during a rain fade.
  - **Receiver Noise Temperature (Trx):** The noise generated internally by the electronic components of the receiver, primarily the Low-Noise Amplifier (LNA) or Low-Noise Block downconverter (LNB) that is the first active component after the antenna.[38]
- **Receiver Figure of Merit (G/T):** This ratio, pronounced "G over T," is the single most important figure of merit for a receiving ground station. It is the ratio of the receive antenna's gain (G) to the total system noise temperature (Tsys).[39] A higher G/T indicates a more sensitive station, capable of receiving weaker signals. It is typically expressed in dB/K.[35]
- **Boltzmann's Constant (k):** A fundamental constant of physics ($1.38\times10{-}23$ Joules/K) that links temperature to energy. In its logarithmic form, it is a constant value of -228.6 dBW/K/Hz, which is used to convert the noise temperature into noise power spectral density.[34]

Chapter 8: Constructing a Link Budget with pylink-satcom

The Directed Acyclic Graph (DAG) Paradigm

While link budgets are often calculated in spreadsheets, this approach becomes cumbersome and error-prone for dynamic simulations with many interdependent variables. A more powerful and robust paradigm is the Directed Acyclic Graph (DAG). In a DAG, each variable in the calculation is a "node," and the dependencies between them are "edges." For example, the C/N0 node depends on the EIRP, Total_Path_Loss, and G/T nodes.

The pylink-satcom library is a Python package designed specifically for this purpose. It provides a framework for defining a link budget (or any similar complex calculation) as a DAG.[12] This has several advantages:

- **Clarity:** The dependencies are explicitly defined in code, making the model easy to understand and audit.
- **Reusability:** A model can be defined once and reused for many different scenarios simply by changing the input parameters.
- **Caching:** The library automatically caches the results of calculations, so if an input value doesn't change, its dependent nodes are not recomputed, leading to significant efficiency gains.
- **Solving:** It includes advanced features like solvers for finding an input value that achieves a desired output, and mechanisms for handling cyclical dependencies, which we will leverage in Part IV.[12]

Defining Static and Calculated Nodes

In pylink-satcom, a model is built by defining two types of nodes:

- **Static Nodes:** These are the fixed input parameters of the simulation. They are simple key-value pairs provided when the model is initialized. Examples include the satellite's transmit power, the ground station's antenna diameter, and the LNB's noise figure.

- **Calculated Nodes:** These are Python functions (or class methods) that compute their value based on other nodes in the model. For example, we will define a calculated node for fspl_dB that takes the slant_range_km and freq_GHz nodes as input. The DAG framework automatically manages resolving these dependencies.

Code Implementation: Building a Reusable Link Budget Model

We will now create a Python class that encapsulates our entire link budget. This class inherits from pylink.core.DAGModel and defines all the necessary nodes as methods. This approach creates a clean, modular, and self-contained structure for our link budget calculations.

Table 2: Simulation Input Parameters and Configuration

Category	Parameter	Symbol / Key	Value	Unit
Satellite	TLE Source URL	tle_url	https://...	-
	Satellite Name in TLE	sat_name	ISS (ZARYA)	-
	Operating Frequency	freq_GHz	20.0	GHz
	Satellite EIRP	sat_eirp_dBW	50.0	dBW
Ground Station	Location Name	gs_name	Delhi	-
	Latitude	gs_lat_deg	28.61	degrees
	Longitude	gs_lon_deg	77.23	degrees
	Altitude	gs_alt_m	216	meters
	Antenna Diameter	gs_ant_diam_m	1.2	meters

	Antenna Efficiency	gs_ant_eff	0.65	-
	LNB Noise Temperature	gs_lnb_noise_temp_K	75	K
	Feeder/Other Losses	gs_other_losses_dB	0.5	dB
Channel	Rain Model Availability	rain_p	0.01	%
	DVB-S2 Roll-off Factor	rolloff_factor	0.20	-
Simulation	Start Time	sim_start	Now	UTC
	Duration	sim_duration_hrs	24	hours
	Time Step	sim_step_s	10	seconds

Python

```python
from pylink import DAGModel, Enum
import numpy as np

# Define an Enum for our node names to avoid magic strings
class LEO_Nodes(Enum):
    # Inputs from geometry/channel simulation
    elevation_deg = ()
    slant_range_km = ()
    atmospheric_loss_dB = ()
    rain_fade_dB = ()

    # Static inputs (from config)
    freq_GHz = ()
    sat_eirp_dBW = ()
    gs_ant_diam_m = ()
    gs_ant_eff = ()
    gs_lnb_noise_temp_K = ()
    gs_other_losses_dB = ()

    # Calculated nodes
    fspl_dB = ()
    total_path_loss_dB = ()
    gs_ant_gain_dBi = ()
    ant_noise_temp_K = ()
    rx_noise_temp_K = ()
    sys_noise_temp_K = ()
    gs_gt_dBK = ()
    boltzmann_dB = ()
    cn0_dBHz = ()

class LeoLinkBudget(DAGModel):
    def __init__(self, **kwargs):
        # The 'tribute' dict maps node enums to the methods that calculate them
        self.tribute = {
            LEO_Nodes.fspl_dB: self._fspl_dB,
            LEO_Nodes.total_path_loss_dB: self._total_path_loss_dB,
            LEO_Nodes.gs_ant_gain_dBi: self._gs_ant_gain_dBi,
            LEO_Nodes.ant_noise_temp_K: self._ant_noise_temp_K,
            LEO_Nodes.rx_noise_temp_K: self._rx_noise_temp_K,
            LEO_Nodes.sys_noise_temp_K: self._sys_noise_temp_K,
            LEO_Nodes.gs_gt_dBK: self._gs_gt_dBK,
            LEO_Nodes.cn0_dBHz: self._cn0_dBHz,
        }
        # Initialize the DAGModel with static values
        super().__init__(**kwargs)

    # --- Calculated Node Implementations ---

    def _fspl_dB(self, model):
        # Implements FSPL formula
        return 20 * np.log10(model.slant_range_km) + 20 * np.log10(model.freq_GHz) +
92.45
```

```
def _total_path_loss_dB(self, model):
    # Total loss is FSPL + atmospheric losses
    return model.fspl_dB + model.atmospheric_loss_dB

def _gs_ant_gain_dBi(self, model):
    # Standard formula for parabolic antenna gain [24]
    c = 299792458.0  # Speed of light in m/s
    wavelength = c / (model.freq_GHz * 1e9)
    area = np.pi * (model.gs_ant_diam_m / 2)**2
    effective_area = area * model.gs_ant_eff
    gain_linear = (4 * np.pi * effective_area) / (wavelength**2)
    return 10 * np.log10(gain_linear)

def _ant_noise_temp_K(self, model):
    # Simplified model for antenna noise temperature.
    # A full model would be more complex, accounting for spillover, etc.
    # This model assumes clear-sky noise + noise from rain attenuation. [29]
    T_sky_clear = 5 + (90 - model.elevation_deg) * 0.5  # Simple empirical model
    T_physical_rain = 275  # Effective physical temperature of rain [29]
    loss_linear = 10**(model.rain_fade_dB / 10)

    # Noise contribution from rain: T_rain = T_physical * (1 - 1/L)
    T_rain = T_physical_rain * (1 - 1/loss_linear)
    return T_sky_clear + T_rain

def _rx_noise_temp_K(self, model):
    # Noise from the receiver electronics (LNB + other losses)
    T_other_physical = 290 # Ambient temperature for passive components
    loss_linear = 10**(model.gs_other_losses_dB / 10)
    # Noise from other losses: T_other = T_physical * (L - 1)
    T_other = T_other_physical * (loss_linear - 1)
    return model.gs_lnb_noise_temp_K + T_other

def _sys_noise_temp_K(self, model):
    # Total system noise temperature referred to the LNB input [35, 38]
    return model.ant_noise_temp_K + model.rx_noise_temp_K

def _gs_gt_dBK(self, model):
        # G/T calculation [39]
        return model.gs_ant_gain_dBi - 10 * np.log10(model.sys_noise_temp_K)

    def _cn0_dBHz(self, model):
        # The final link budget equation
        k_boltzmann_dB = -228.6  # [34]
        return model.sat_eirp_dBW - model.total_path_loss_dB + model.gs_gt_dBK -
k_boltzmann_dB
```

Chapter 9: Simulating End-to-End Link Performance

Integrating Models into the pylink-satcom Framework

The final step is to orchestrate the entire simulation. The time-series data generated in Parts I and II, which captures the dynamic aspects of the link, will be fed into the pylink-satcom link budget model. The DAGModel is designed to be updated with new values for its static nodes. In our case, the "static" nodes for slant_range_km, elevation_deg, etc., will be updated at each time step of the simulation.

Executing the Time-Step Simulation

The main simulation loop will iterate through each row of the simulation_df DataFrame created earlier. For each time step, it will:

1. Extract the dynamic values (elevation, slant range, atmospheric losses).
2. Update the LeoLinkBudget model instance with these new values using the model.update() method.
3. Request the final cn0_dBHz value from the model. pylink-satcom will automatically re-calculate only the necessary nodes in the DAG.
4. Store the resulting C/N0 back into the DataFrame.

This process efficiently calculates the end-to-end link performance for the entire satellite pass, producing a rich dataset ready for the ACM logic in Part IV.

Code Implementation: The Main Simulation Loop

The following script brings all the pieces together, executing the simulation and populating our DataFrame with the final C/N0 values.

Python

```python
# --- Example Usage of the LeoLinkBudget Model ---

# Define the static parameters for the link budget model
static_params = {
    LEO_Nodes.freq_GHz: link_parameters['freq_GHz'],
    LEO_Nodes.sat_eirp_dBW: 50.0, # Example EIRP
    LEO_Nodes.gs_ant_diam_m: link_parameters['ant_diam_m'],
    LEO_Nodes.gs_ant_eff: 0.65,
    LEO_Nodes.gs_lnb_noise_temp_K: 75.0,
    LEO_Nodes.gs_other_losses_dB: 0.5,
}

# Instantiate the link budget model
link_budget_model = LeoLinkBudget(**static_params)

# --- Main Simulation Loop ---
if 'simulation_df' in locals() and simulation_df is not None:
    print("\nExecuting time-step simulation to calculate C/N0...")

    # Prepare lists to store the results
    cn0_results =
    gt_results =

    # Iterate through each time step in our simulation DataFrame
```

```
 for index, row in simulation_df.iterrows():
     # Define the dynamic inputs for this time step
     dynamic_inputs = {
         LEO_Nodes.elevation_deg: row['elevation_deg'],
         LEO_Nodes.slant_range_km: row['slant_range_km'],
         LEO_Nodes.atmospheric_loss_dB: row,
         LEO_Nodes.rain_fade_dB: row
     }

     # Update the model with the dynamic inputs for the current time step
     link_budget_model.update(dynamic_inputs)

     # Calculate and store the C/N0 for this time step
     cn0_results.append(link_budget_model.cn0_dBHz)
     gt_results.append(link_budget_model.gs_gt_dBK)

 # Add the results to our main DataFrame
 simulation_df = gt_results
 simulation_df = cn0_results

 print("Simulation complete.")
 print("\nFinal simulation data sample with C/N0:")
 print(simulation_df].head())
```

Part IV: AI-Driven Adaptive Coding and Modulation

In this final part, we leverage the high-fidelity simulation results to implement the core logic of an Adaptive Coding and Modulation system. We will translate the calculated C/N0 into a decision on which Modulation and Coding (MODCOD) scheme to use, with the goal of maximizing data throughput at every moment. This section will first introduce the relevant DVB-S2 and DVB-S2X standards, then implement a classical ACM decision engine, and finally, lay out a complete framework for replacing this classical engine with a predictive AI agent.

Chapter 10: The DVB-S2 & DVB-S2X Standards and MODCOD Performance

Overview of DVB-S2 and DVB-S2X

The Digital Video Broadcasting - Satellite - Second Generation (DVB-S2) is a highly efficient and flexible digital satellite transmission standard.[41] Its successor, DVB-S2X (Extensions), further enhances performance and adds new capabilities.[43] The remarkable performance of these standards, which operates close to the theoretical Shannon limit, is primarily due to two features:

1. **Powerful Forward Error Correction (FEC):** DVB-S2/S2X uses a concatenation of a modern Low-Density Parity-Check (LDPC) code as the inner code and a Bose-Chaudhuri-Hocquenghem (BCH) code as the outer code. This combination provides extremely robust error correction capabilities.[3]
2. **A Wide Range of MODCODs:** The standards define a large set of combinations of modulation schemes (QPSK, 8PSK, 16APSK, 32APSK, and even higher in S2X) and LDPC code rates (from as low as 1/4 to as high as 9/10).[1] This provides a fine-grained ladder of options, allowing the system to trade robustness for spectral efficiency with high precision. DVB-S2X extends this ladder even further, adding more efficient constellations and very-low-SNR (VL-SNR) modes for challenging link conditions.[45]

Spectral Efficiency and Required Es/N0

Each MODCOD is characterized by two key performance metrics that govern the ACM decision [5]:

- **Spectral Efficiency (η):** Measured in information bits per symbol, this indicates how efficiently the MODCOD uses the available bandwidth. For example, QPSK (2 bits/symbol) with a 2/3 code rate has a spectral efficiency of approximately $2 \times 2/3 = 1.33$ bits/symbol.
- **Required Energy-per-Symbol to Noise-Density Ratio (Es/N0):** This is the

minimum signal-to-noise ratio, measured at the symbol level, required to achieve a target performance, known as Quasi-Error-Free (QEF) operation. For DVB-S2, QEF is typically defined as a Packet Error Rate (PER) of less than 10−7.[4] This value represents the performance threshold for each MODCOD.

The ACM system's goal is to select the MODCOD with the highest possible spectral efficiency whose required Es/N0 is met or exceeded by the available Es/N0 on the link.

Table 3: DVB-S2 MODCOD Performance Characteristics

This table, derived from the ETSI EN 302 307-1 standard, provides the essential lookup data for a DVB-S2 ACM implementation. It lists the ideal Es/N0 thresholds for QEF operation over an AWGN channel with normal FECFRAME length (64,800 bits).[5]

Mode	Spectral efficiency (bits/symbol)	Ideal Es/No (dB) for FECFRAME length = 64 800
QPSK 1/4	0.490	-2.35
QPSK 1/3	0.656	-1.24
QPSK 2/5	0.789	-0.30
QPSK 1/2	0.989	1.00
QPSK 3/5	1.188	2.23
QPSK 2/3	1.322	3.10
QPSK 3/4	1.487	4.03
QPSK 4/5	1.587	4.68
QPSK 5/6	1.655	5.18

QPSK 8/9	1.766	6.20
QPSK 9/10	1.789	6.42
8PSK 3/5	1.780	5.50
8PSK 2/3	1.981	6.62
8PSK 3/4	2.228	7.91
8PSK 5/6	2.479	9.35
8PSK 8/9	2.646	10.69
8PSK 9/10	2.679	10.98
16APSK 2/3	2.637	8.97
16APSK 3/4	2.967	10.21
16APSK 4/5	3.166	11.03
16APSK 5/6	3.300	11.61
16APSK 8/9	3.523	12.89
16APSK 9/10	3.567	13.13
32APSK 3/4	3.703	12.73
32APSK 4/5	3.952	13.64
32APSK 5/6	4.120	14.28
32APSK 8/9	4.398	15.69
32APSK 9/10	4.453	16.05

Table 4: DVB-S2X MODCOD Performance Characteristics (Selected)

This table provides a selection of the more extensive MODCOD options available in DVB-S2X, derived from ETSI EN 302 307-2. It includes new, highly efficient linear and non-linear constellations and very-low-SNR modes.[49] For brevity, a representative subset is shown. A full implementation would use the complete tables from the standard.

Canonical MODCOD name	Spectral efficiency (bits/symbol)	Ideal Es/N0 (AWGN Linear Channel)
VL-SNR Modes		
QPSK 2/9	0.435	-2.85
QPSK 13/45	0.568	-2.03
Normal Modes		
QPSK 9/20	0.889	0.22
QPSK 11/20	1.089	1.45
8PSK 23/36	1.896	6.12
8PSK 13/18	2.145	7.49
16APSK 26/45	2.282	7.51
16APSK 7/9	3.077	10.65
32APSK 2/3-L	3.292	11.10
32APSK 7/9	3.841	13.05

64APSK 4/5	4.735	15.87
64APSK 5/6	4.937	16.55
128APSK 3/4	5.163	17.73
256APSK 32/45	5.593	18.59
256APSK 3/4	5.901	19.57

Chapter 11: Implementing the Adaptive Loop

From C/N0 to Link Margin

The link budget simulation provides the available C/N0. To compare this with the MODCOD table thresholds, it must be converted to the available Es/N0. This conversion depends on the symbol rate (Rs) of the transmission:

$$\left(\frac{E_s}{N_0}\right)_{dB} = \left(\frac{C}{N_0}\right)_{dB-Hz} - 10\log_{10}(R_s)$$

The symbol rate itself depends on the desired information data rate (Rinfo) and the spectral efficiency (η) of the chosen MODCOD:

$$R_s = \frac{R_{info}}{\eta}$$

This reveals a cyclical dependency: to choose a MODCOD, we need the available Es/N0, which depends on the symbol rate, which in turn depends on the spectral efficiency of the very MODCOD we are trying to choose.[2]

This is a classic problem in ACM system design. A simple approach is to assume a fixed symbol rate, but this is suboptimal. A more robust method, and one that is elegantly handled by the pylink-satcom framework, is to iterate through the possibilities. The set of all available MODCODs is a finite, discrete set. We can test each one, calculate the resulting required Es/N0, and check if the link can support it. The pylink-satcom library's ability to temporarily override node values and resolve the graph makes it perfectly suited for this kind of iterative solving.[12]

Once a MODCOD is selected, the link margin can be calculated as:

MargindB=(N0Es)available−(N0Es)required

The ACM Decision Engine

The logic for a classical (non-predictive) ACM decision engine is as follows:

1. For a given time step, take the available C/N0 from the link budget.
2. Iterate through the list of all available MODCODs, ordered from highest spectral efficiency to lowest.
3. For each candidate MODCOD:
   a. Assume a target information data rate or a fixed symbol rate to break the dependency cycle for this check. Let's assume a fixed symbol rate for simplicity in this example logic.
   b. Calculate the available Es/N0.
   c. Retrieve the required Es/N0 for the candidate MODCOD from the lookup table.
   d. Add a system implementation margin (e.g., 1.5 dB) to the required Es/N0. This accounts for real-world hardware imperfections not captured in the ideal thresholds.
   e. If the available Es/N0 is greater than the margin-adjusted required Es/N0, this MODCOD is a valid choice.
4. Select the first valid MODCOD found in the iteration (which will be the one with the highest spectral efficiency) as the optimal choice for that time step.
5. Calculate the achievable data rate based on the selected MODCOD's spectral efficiency and the system's symbol rate.

Code Implementation: The ACM Function

The following code implements this decision logic. It takes the simulation DataFrame and the MODCOD performance tables as input and adds new columns for the selected MODCOD and the resulting data rate.

Python

```
# Load MODCOD tables into pandas DataFrames
dvbs2_modcods = pd.DataFrame().sort_values(by='eta',
ascending=False).reset_index(drop=True)

# For a full implementation, the DVB-S2X table would also be loaded and concatenated.

def acm_decision_engine(cn0_available_dB, symbol_rate_Hz, modcod_table,
implementation_margin_dB=1.5):
    """
   Selects the best MODCOD based on available C/N0.

   Args:
      cn0_available_dB (float): The available C/N0 from the link budget.
      symbol_rate_Hz (float): The system's symbol rate.
      modcod_table (pandas.DataFrame): The table of MODCOD performance.
      implementation_margin_dB (float): A margin to add to the required Es/N0.

   Returns:
      tuple: (selected_modcod_name, achievable_data_rate_Mbps)
   """
    # Calculate available Es/N0
    esn0_available_dB = cn0_available_dB - 10 * np.log10(symbol_rate_Hz)

    # Iterate through MODCODs from highest efficiency to lowest
    for index, modcod in modcod_table.iterrows():
        required_esn0_with_margin = modcod + implementation_margin_dB

        if esn0_available_dB >= required_esn0_with_margin:
            # This is the highest-efficiency MODCOD that closes the link
            achievable_rate_bps = symbol_rate_Hz * modcod['eta']
            return modcod['name'], achievable_rate_bps / 1e6 # return in Mbps

    # If no MODCOD can close the link, return link down
    return "Link Down", 0.0

# --- Run ACM simulation ---
if 'simulation_df' in locals() and simulation_df is not None:
```

```
print("\nRunning ACM simulation...")

# Assume a fixed symbol rate for this example, e.g., 25 Msps
symbol_rate = 25e6

results = simulation_df.apply(
    acm_decision_engine,
    args=(symbol_rate, dvbs2_modcods)
)

# Unpack results into new columns
simulation_df['selected_modcod'], simulation_df['data_rate_Mbps'] = zip(*results)

print("ACM simulation complete.")
print("\nFinal simulation data sample with ACM decisions:")
print(simulation_df].head())
```

Chapter 12: A Framework for AI-Based Link Adaptation

Conceptualizing ACM as a Learning Problem

The classical ACM engine is purely reactive. It measures the current channel state and selects a MODCOD. However, there is inherent latency in any real-world system: the time taken to measure the channel at the receiver, send this information back to the transmitter (often via a return channel), process the feedback, and switch the MODCOD.[2] For a fast-moving LEO satellite, the channel conditions may have already changed by the time the adaptation occurs.

This is where AI, and specifically predictive modeling, offers a transformative advantage. Instead of reacting to the present, an AI agent can learn the complex, time-varying dynamics of the LEO link and *predict* the channel state a short time into the future. By adapting to the predicted state, the system can compensate for the feedback latency, maintaining a more consistently optimal link. The simulation we have built is the perfect "gym" for training such an agent.

Defining the Environment for Reinforcement Learning (RL)

To frame ACM as a learning problem, we can use the paradigm of Reinforcement Learning (RL). In RL, an "agent" learns to make optimal decisions by interacting with an "environment" and receiving "rewards" for its actions. Our simulation provides all the necessary components:

- **Environment:** The entire simulation framework constructed in Parts I-III. It takes an action (a chosen MODCOD) and a time step, and returns the next state and a reward.
- **State Space:** The set of observations the agent uses to make a decision. A well-designed state would include not just the current link parameters but also their recent history, to capture the dynamics. For example, the state at time t could be a vector containing:
    - [elev(t),elev(t−1),...,elev(t−N)]
    - [cn0(t),cn0(t−1),...,cn0(t−N)]
    - [rain_fade(t),rain_fade(t−1),...]
    - Current selected MODCOD
- **Action Space:** The discrete set of all available MODCODs from the DVB-S2/S2X tables that the agent can choose from.
- **Reward Function:** This is a critical design choice that guides the agent's learning. The simplest reward function is simply the achieved data rate at each time step. A more sophisticated reward function might also include:
    - A large negative penalty for choosing a MODCOD that results in a "Link Down" state (negative margin).
    - A small negative penalty for switching MODCODs too frequently, to encourage stability.

Architectural Considerations

A common architecture for an RL agent in this context is a Deep Q-Network (DQN) or a similar policy-based method like Proximal Policy Optimization (PPO). The agent would typically be a neural network that takes the state vector as input and outputs a value for each possible action (each MODCOD). The agent then selects the action with the highest predicted value. During training, the agent explores different actions, observes the rewards from the simulation environment, and updates the weights of its neural network to improve its decision-making policy.

Code Implementation: AI Agent Interface

To facilitate the integration of an AI model, we can define a simple interface class. A user could then create their own agent (e.g., using TensorFlow or PyTorch) that conforms to this interface and plug it directly into the main simulation loop.

Python

```python
class AIBasedACMAgent:
    """
    A stub class representing the interface for an AI-driven ACM agent.
    """
    def __init__(self, action_space):
        """
        Initializes the agent.

        Args:
            action_space (list): A list of all possible MODCOD names.
        """
        self.action_space = action_space
        # In a real implementation, the neural network model would be initialized here.
        print("AI Agent Initialized.")

    def select_action(self, state):
        """
        Selects a MODCOD based on the current state.

        Args:
            state (dict or np.array): The current state vector of the environment.
        Returns:
            str: The name of the selected MODCOD.
        """
        # In a real implementation, this method would:
        # 1. Preprocess the state vector.
        # 2. Feed it into the neural network.
        # 3. Get the output Q-values or policy.
        # 4. Select an action (e.g., the one with the highest value).

        # For this stub, we'll just return a default robust MODCOD.
        return "QPSK 1/2"

    def train(self, state, action, reward, next_state, done):
        """
        Trains the agent on a single transition from the environment.

        Args:
            state: The state before the action.
            action: The action taken.
            reward: The reward received.
            next_state: The state after the action.
            done (bool): Whether the episode (pass) has ended.
        """
        # This method would implement the learning algorithm (e.g., updating
        # the network weights based on the Bellman equation).
        pass

# The main simulation loop would be modified to call this agent instead of
# the classical decision engine. For each step, it would construct the state,
# call agent.select_action(), apply that action to the link, calculate the
# reward (data rate), and then call agent.train().
```

Conclusion and Future Work

Summary of the Simulation Framework's Capabilities

This treatise has detailed the design and implementation of a comprehensive, high-fidelity simulation framework for LEO satellite communication links. By systematically integrating specialized, open-source Python libraries, we have constructed a tool that progresses logically from first principles of orbital mechanics to the complex, dynamic calculations of a complete link budget and the implementation of an adaptive control system. The framework is capable of:

1. **Accurate Geometric Modeling:** Using skyfield to propagate satellite orbits from TLE data and calculate the precise, time-varying geometry (slant range, elevation) for any ground station.
2. **Standard-Compliant Channel Modeling:** Using ITU-Rpy to quantify time-varying atmospheric losses, including gaseous absorption, rain fade, cloud attenuation, and scintillation, according to established ITU-R recommendations.
3. **Robust Link Budget Calculation:** Using pylink-satcom to implement the link budget as a Directed Acyclic Graph, providing a clear, efficient, and extensible method for calculating the dynamic C/N0.
4. **Adaptive System Emulation:** Implementing a classical ACM decision engine based on the DVB-S2/S2X standards to select the optimal MODCOD and maximize data throughput at each point in a satellite pass.
5. **AI-Ready Architecture:** Providing a complete, well-defined environment and interface for the development and training of advanced, AI-driven predictive control agents.

The resulting simulation is not a black box but a transparent, modular, and powerful tool for research, development, and education in the field of satellite communications.

Avenues for Extension

The framework presented here is foundational and can be extended in numerous directions to explore more complex scenarios and research questions. Potential avenues for future work include:

- **Modeling the Uplink and Inter-Satellite Links:** The current model focuses on the downlink. A natural extension would be to model the uplink from the ground station to the satellite and the links between satellites in a constellation, each with its own unique link budget challenges.

- **Incorporating Interference:** The current simulation assumes a noise-limited environment. A significant extension would be to model interference, both from other satellites in the same or adjacent orbits (inter-system interference) and from terrestrial sources, to calculate the Carrier-to-Noise-plus-Interference Ratio (CNIR).
- **Advanced Antenna and Beam-Hopping Models:** The model uses simplified antenna patterns. It could be extended to incorporate detailed, measured antenna patterns and to simulate advanced techniques like beam-hopping, where a single satellite beam rapidly serves multiple ground locations, requiring even more sophisticated resource management.[44]
- **Full AI Agent Implementation and Training:** The most direct next step is to implement a complete Reinforcement Learning agent using the provided interface and train it within the simulation environment. This would involve exploring different network architectures, state representations, and reward functions to develop an agent that can outperform the classical reactive ACM.
- **Validation Against Real-World Data:** The ultimate validation of any simulation is comparison with reality. The framework could be used to model a link for a satellite that has a publicly available beacon signal. By comparing the simulated C/N0 and atmospheric attenuation with measurements from a real ground station receiver and radiometer, the model's fidelity can be rigorously assessed and refined.[50]

Works cited

1. DVB-S2 Experiment Over NASA's Space Network, accessed on August 2, 2025, https://ntrs.nasa.gov/api/citations/20170006587/downloads/20170006587.pdf
2. Channel SNR and MODCOD threshold | Download Scientific Diagram - ResearchGate, accessed on August 2, 2025, https://www.researchgate.net/figure/Channel-SNR-and-MODCOD-threshold_fig1_333516659
3. Migration from DVB-S to DVB-S2 and Related Efficiencies - Comtech EF Data, accessed on August 2, 2025, https://www.comtechefdata.com/files/articles_papers/WP-Migration_DVB-S_to_DVB-S2.pdf
4. REPORT ITU-R BO.2101* Digital satellite broadcasting system (television, sound and data) with flexible configuration, accessed on August 2, 2025, https://www.itu.int/dms_pub/itu-r/opb/rep/r-rep-bo.2101-2007-pdf-e.pdf
5. EN 302 307-1 - V1.4.1 - Digital Video Broadcasting (DVB) - ETSI, accessed on August 2, 2025, https://www.etsi.org/deliver/etsi_en/302300_302399/30230701/01.04.01_60/en_30230701v010401p.pdf
6. The Python Standard Library — Python 3.13.5 documentation, accessed on

August 2, 2025, https://docs.python.org/3/library/index.html
7. PyPI · The Python Package Index, accessed on August 2, 2025, https://pypi.org/
8. skyfield - PyPI, accessed on August 2, 2025, https://pypi.org/project/skyfield/
9. Skyfield — documentation - Rhodes Mill, accessed on August 2, 2025, https://rhodesmill.org/skyfield/
10. itur package — ITU-Rpy 0.4.0 documentation, accessed on August 2, 2025, https://itu-rpy.readthedocs.io/en/latest/apidoc/itur.html
11. ITU-Rpy documentation — ITU-Rpy 0.4.0 documentation, accessed on August 2, 2025, https://itu-rpy.readthedocs.io/
12. pylink-satcom·PyPI, accessed on August 2, 2025, https://pypi.org/project/pylink-satcom/
13. Recommendation ITU-R P.618 — ITU-Rpy 0.4.0 documentation - Read the Docs, accessed on August 2, 2025, https://itu-rpy.readthedocs.io/en/latest/apidoc/itu618.html
14. API Reference — Earth Satellites — Skyfield documentation - Rhodes Mill, accessed on August 2, 2025, https://rhodesmill.org/skyfield/api-satellites.html
15. Satellite analysis for Sat with no TLE · skyfielders python-skyfield · Discussion #759 - GitHub, accessed on August 2, 2025, https://github.com/skyfielders/python-skyfield/discussions/759
16. Earth Satellites — Skyfield documentation - Rhodes Mill, accessed on August 2, 2025, https://rhodesmill.org/skyfield/earth-satellites.html
17. Dates and Time — Skyfield documentation - Rhodes Mill, accessed on August 2, 2025, https://rhodesmill.org/skyfield/time.html
18. API Reference — Skyfield documentation - Rhodes Mill, accessed on August 2, 2025, https://rhodesmill.org/skyfield/api.html
19. Delhi - Wikipedia, accessed on August 2, 2025, https://en.wikipedia.org/wiki/Delhi
20. Map of New Delhi, India Latitude, Longitude, Altitude/ Elevation - Climatemps.com, accessed on August 2, 2025, https://www.climate.top/india/new-delhi/map/
21. Table of Contents — Skyfield documentation - Rhodes Mill, accessed on August 2, 2025, https://rhodesmill.org/skyfield/toc.html
22. Free Space Path Loss: Details & Calculator - Electronics Notes, accessed on August 2, 2025, https://www.electronics-notes.com/articles/antennas-propagation/propagation-overview/free-space-path-loss.php
23. Understanding Free Space Path Loss - Study CCNP, accessed on August 2, 2025, https://study-ccnp.com/understanding-free-space-path-loss/
24. Link Budget: Comunicaciones Satelitales | PDF | Antenna (Radio) | Decibel - Scribd, accessed on August 2, 2025, https://www.scribd.com/presentation/46729924/Link-Budget
25. Earth-Space Propagation Losses - MATLAB & Simulink - MathWorks, accessed on August 2, 2025, https://www.mathworks.com/help/satcom/gs/p618-channel-modeling.html
26. Attenuation by atmospheric gases - ITU, accessed on August 2, 2025, https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.676-11-201609-I!!PDF-E.pdf

27. RECOMMENDATION ITU-R P.676-9 - Attenuation by atmospheric gases, accessed on August 2, 2025, https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.676-9-201202-S!!PDF-E.pdf
28. [PDF] RECOMMENDATION ITU-R P.618-8-Propagation data and prediction methods required for the design of Earth-space telecommunication systems | Semantic Scholar, accessed on August 2, 2025, https://www.semanticscholar.org/paper/RECOMMENDATION-ITU-R-P.618-8-Propagation-data-and/81a307a044eb0895ccdc32276a47246190ecd9a8
29. Antenna Noise Temperature for Low Earth Orbiting Satellite Ground Stations at L and S Band ( ) - ThinkMind, accessed on August 2, 2025, https://www.thinkmind.org/articles/spacomm_2011_1_10_30007.pdf
30. cloud-attenuation · GitHub Topics, accessed on August 2, 2025, https://github.com/topics/cloud-attenuation?l=python
31. P.676-3 - Attenuation by atmospheric gases - ITU, accessed on August 2, 2025, https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.676-3-199708-S!!PDF-E.pdf
32. Recommendation ITU-R P.838 — ITU-Rpy 0.4.0 documentation - Read the Docs, accessed on August 2, 2025, https://itu-rpy.readthedocs.io/en/latest/apidoc/itu838.html
33. Link Budgets - Signal-to-Noise Ratio, accessed on August 2, 2025, https://www.waves.utoronto.ca/prof/svhum/ece422/notes/22-linkbudget.pdf
34. 6.7 Link Budget – A Guide to CubeSat Mission and Bus Design - UH Pressbooks, accessed on August 2, 2025, https://pressbooks-dev.oer.hawaii.edu/epet302/chapter/9-6-link-budget/
35. Antenna gain-to-noise-temperature - Wikipedia, accessed on August 2, 2025, https://en.wikipedia.org/wiki/Antenna_gain-to-noise-temperature
36. Modeling Antenna Noise Temperature Due to Rain Clouds at Microwave and Millimeter-Wave Frequencies - ResearchGate, accessed on August 2, 2025, https://www.researchgate.net/publication/3018573_Modeling_Antenna_Noise_Temperature_Due_to_Rain_Clouds_at_Microwave_and_Millimeter-Wave_Frequencies
37. Lecture 7: Antenna Noise Temperature and System Signal-to-Noise Ratio, accessed on August 2, 2025, https://www.ece.mcmaster.ca/faculty/nikolova/antenna_dload/current_lectures/L07_Noise.pdf
38. Noise temperature, Noise Figure (NF) and noise factor (f) - SatSig.net, accessed on August 2, 2025, https://www.satsig.net/noise.htm
39. www.satnow.com, accessed on August 2, 2025, https://www.satnow.com/community/what-do-you-mean-by-g-t-ratio-in-satellite-communication#:~:text=The%20G%2FT%20ratio%20is%20calculated%20by%20dividing%20the,Kelvin%20(dB%2FK).
40. What is Antenna G/T Ratio in Satellite Communication? - SatNow, accessed on August 2, 2025, https://www.satnow.com/community/what-do-you-mean-by-g-t-ratio-in-satellite-communication
41. NASA Near Earth Network (NEN) DVB-S2 Demonstration Testing for Enhancing

Data Rates for CubeSat/SmallSat, accessed on August 2, 2025, https://ntrs.nasa.gov/api/citations/20190028945/downloads/20190028945.pdf
42. Digital satellite broadcasting system with flexible configuration (television, sound and data) - ITU, accessed on August 2, 2025, https://www.itu.int/dms_pubrec/itu-r/rec/bo/R-REC-BO.1784-1-201612-I!!PDF-E.pdf
43. DVB-S2X Technology - Rohde & Schwarz, accessed on August 2, 2025, https://www.rohde-schwarz.com/us/technologies/satellite-broadcast/dvb-s2x/dvb-s2x-technology/dvb-s2x-technology_230588.html
44. DS/ETSI EN 302 307-2:2021 - Digital Video Broadcasting (DVB) - ANSI Webstore, accessed on August 2, 2025, https://webstore.ansi.org/standards/ds/dsetsien3023072021
45. EN 302 307-2 - V1.2.1 - Digital Video Broadcasting (DVB) - ETSI, accessed on August 2, 2025, https://www.etsi.org/deliver/etsi_en/302300_302399/30230702/01.02.01_20/en_30230702v010201a.pdf
46. End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames - MATLAB & - MathWorks, accessed on August 2, 2025, https://www.mathworks.com/help/satcom/ug/end-to-end-dvbs2x-simulation-with-rf-impairments-and-corrections-for-vlsnr-frames.html
47. S2 Extensions (DVB-S2X), accessed on August 2, 2025, https://dvb.org/wp-content/uploads/2020/02/A171-2_DVB-S2X_Implementation-Guidelines_Draft-TR-102-376-2_v121_Apr-2020.pdf
48. VL-SNR Implementation For Mobile Broadband Applications - SatixFy, accessed on August 2, 2025, https://www.satixfy.com/vlnsr-implementation-for-mobile-2/
49. ETSI EN 302 307-2 V1.2.1 (2020-08), accessed on August 2, 2025, https://www.etsi.org/deliver/etsi_en/302300_302399/30230702/01.02.01_60/en_30230702v010201p.pdf
50. Radiometric Measurements of Slant Path Attenuation in the V/W Bands - DTIC, accessed on August 2, 2025, https://apps.dtic.mil/sti/tr/pdf/ADA610427.pdf
51. Millimeter-wave antenna noise temperature due to rain clouds: Theoretical model and statistical prediction | Request PDF - ResearchGate, accessed on August 2, 2025, https://www.researchgate.net/publication/224502627_Millimeter-wave_antenna_noise_temperature_due_to_rain_clouds_Theoretical_model_and_statistical_prediction