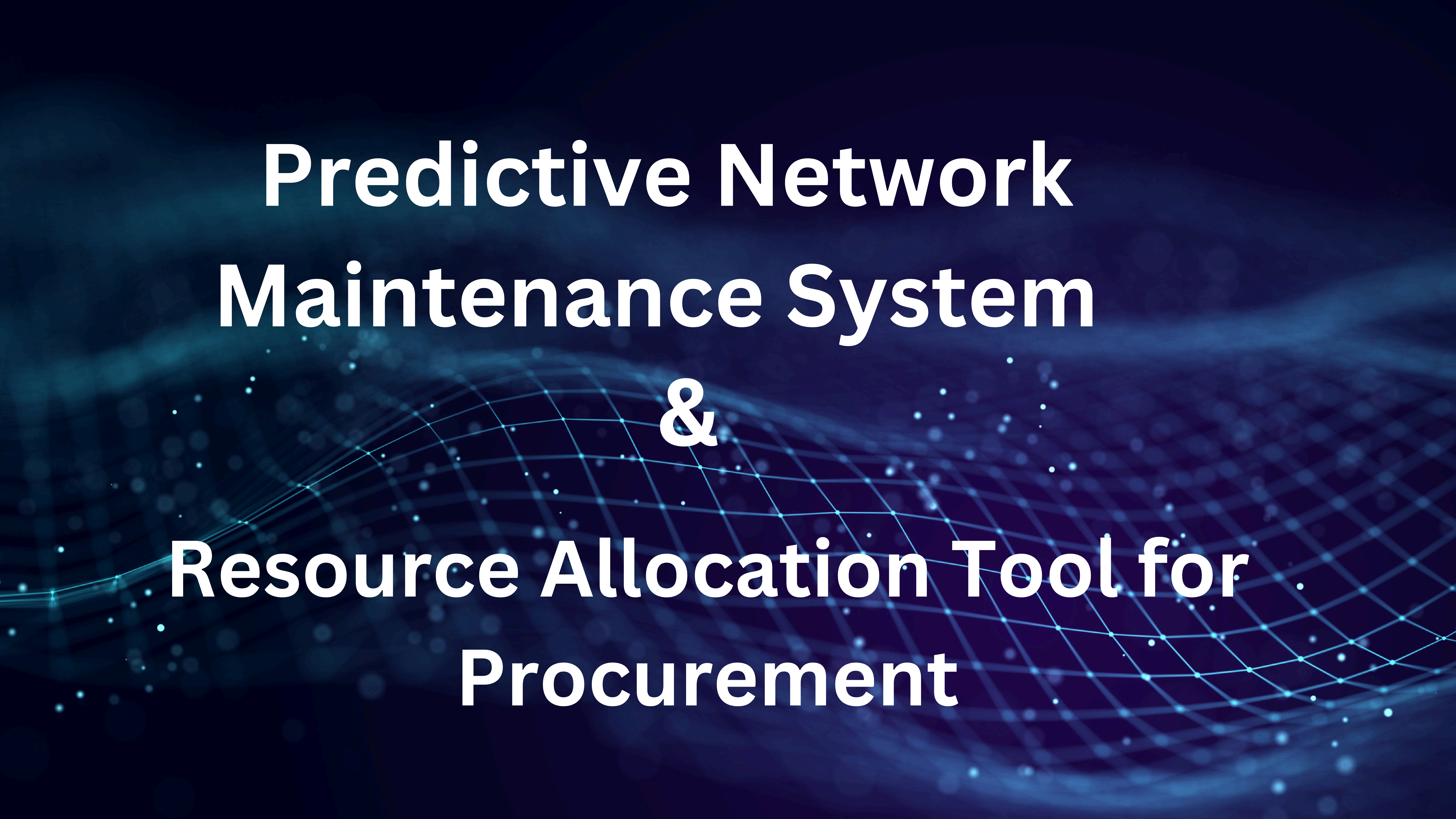


Predictive Network Maintenance System

&

Resource Allocation Tool for Procurement

Efficient Network Management System

01

AI Monitoring & Prediction

- Develop AI algorithms to monitor network performance.
- Predict potential outages and hardware failures.

02

Transparent Procurement Optimization

- Build an ML-powered tool for cost-effective equipment procurement.
- Ensure quality and sustainability in recommendations.

03

Energy Optimization for Off-Grid Schools

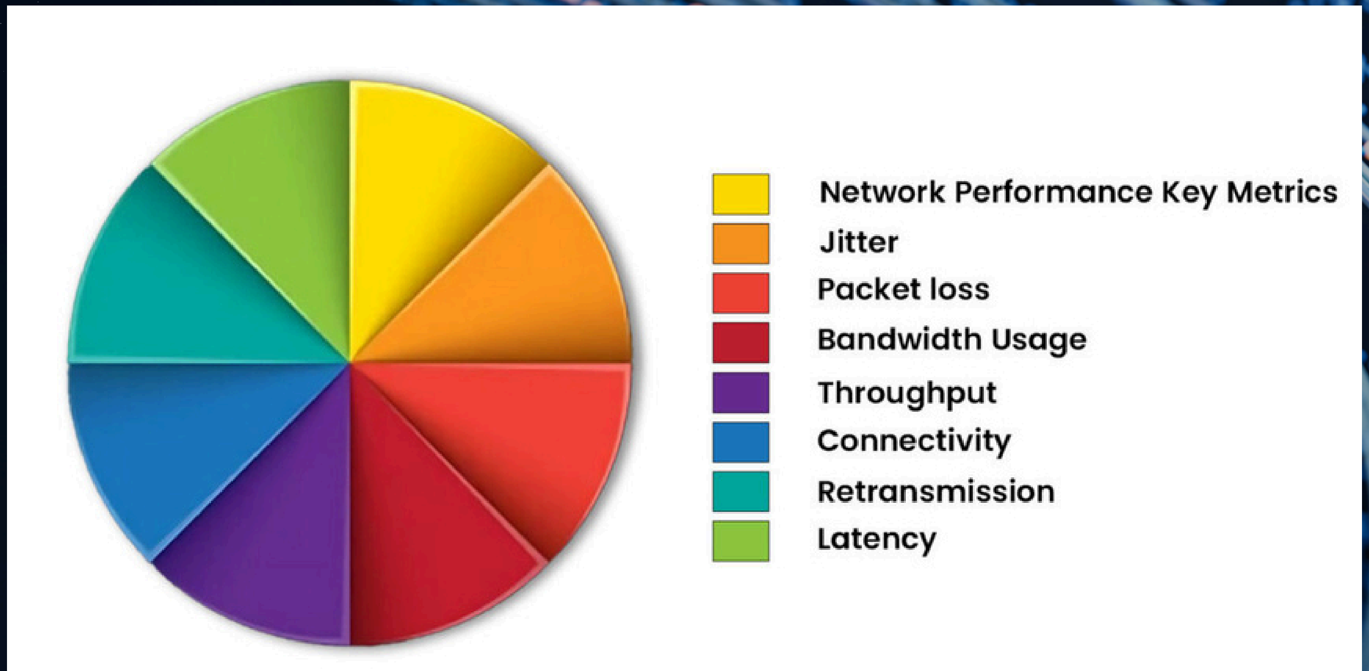
- Optimize power consumption for network infrastructure.
- Leverage renewable energy sources and smart load balancing.

04

Intelligent Bandwidth Management

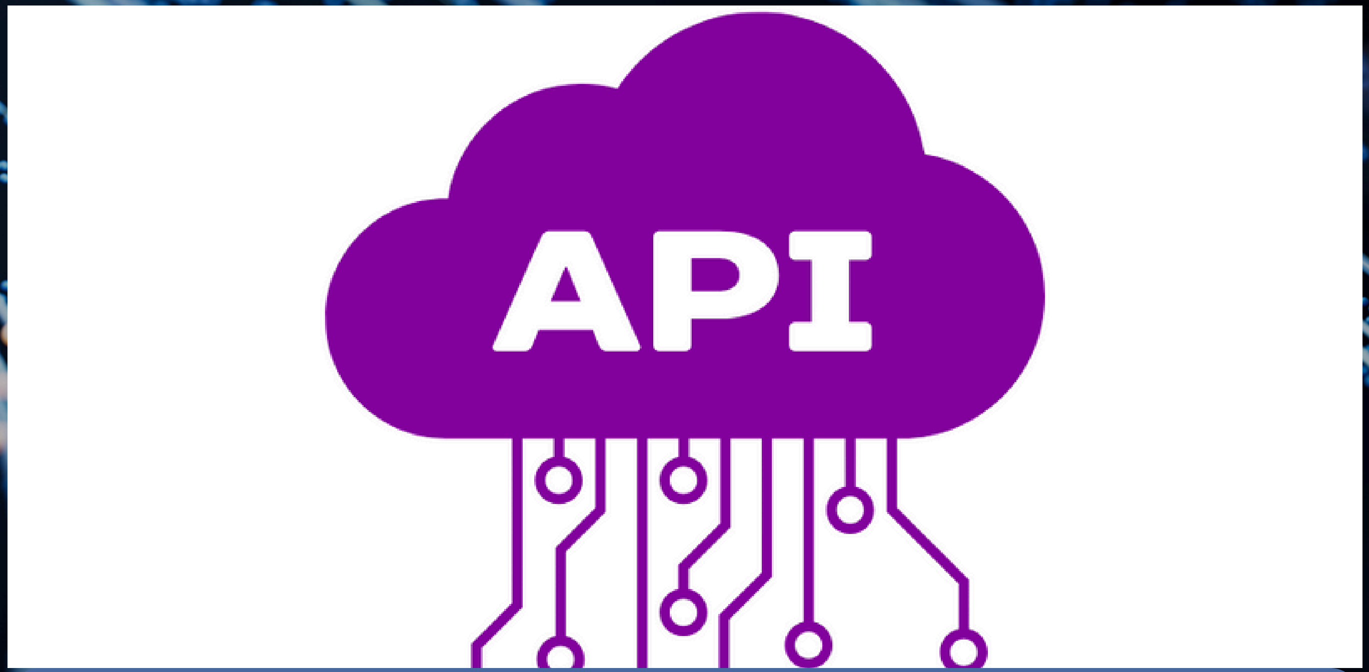
Implement systems for efficient bandwidth allocation and usage.

ARCHITECTURE OVERVIEW

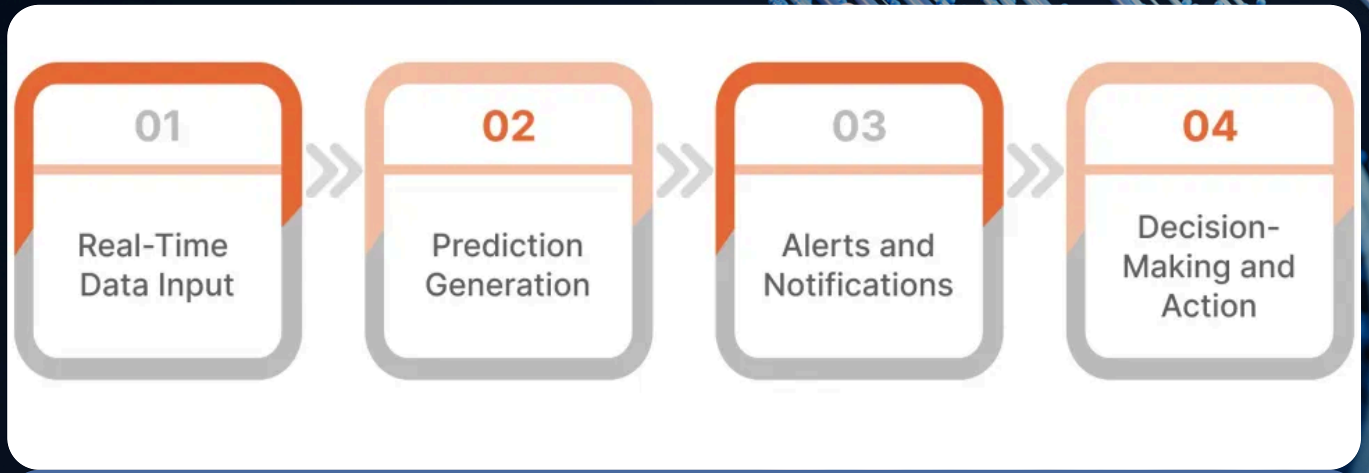


Data Types:

Network performance logs, bandwidth usage statistics, hardware life cycle data, and energy metrics. Ingestion.



Tools: APIs for real-time data streaming from devices and external APIs for procurement pricing. AI and ML Model Layer.



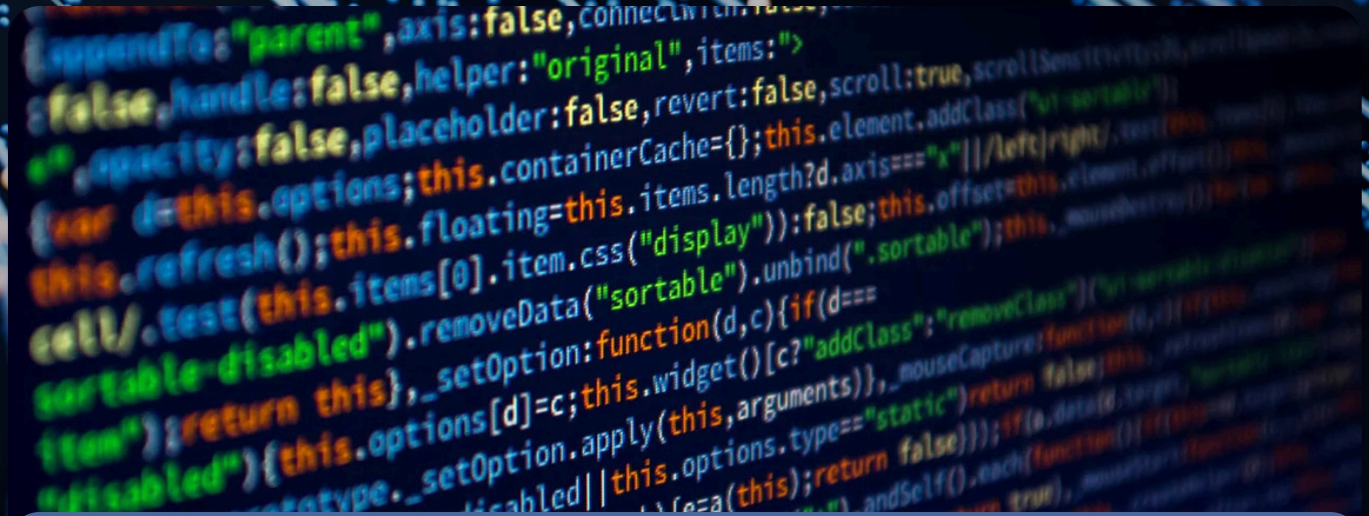
01 Real-Time Data Input

02 Prediction Generation

03 Alerts and Notifications

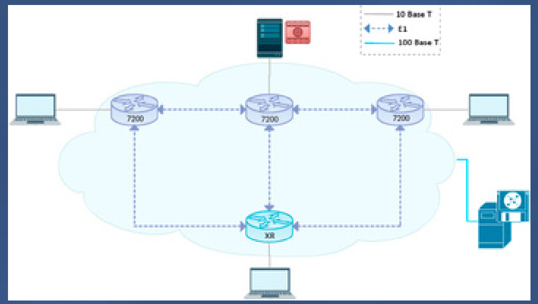
04 Decision-Making and Action

Predictive Maintenance Model: Uses historical and real-time performance data to predict failures, optimize repairs, and suggest preemptive actions.



Algorithims:

- Utilize anomaly detection, time-series forecasting, and supervised learning.
- Predict failures to enhance network reliability.



Intelligent Bandwidth Allocation Model

-Prioritizes educational content over non-essential services.
-Classifies network traffic using deep learning techniques.

Processing and Decision Engine

-Combines outputs from all ML models to generate actionable recommendations.
-Executes priority-based actions, e.g., throttling bandwidth or triggering maintenance alerts.
-Provides a centralized platform for monitoring and managing recommendations and actions

Data Collection

-Provide market data to inform procurement decisions.
-Track network and energy performance in real-time.

Data Processing

- Analyze collected data to predict outages.
- Optimize purchasing decisions.
- Manage and reduce energy consumption.

Decision-Making

-Intelligently allocates bandwidth for efficient usage.

-Recommends optimal procurement strategies.

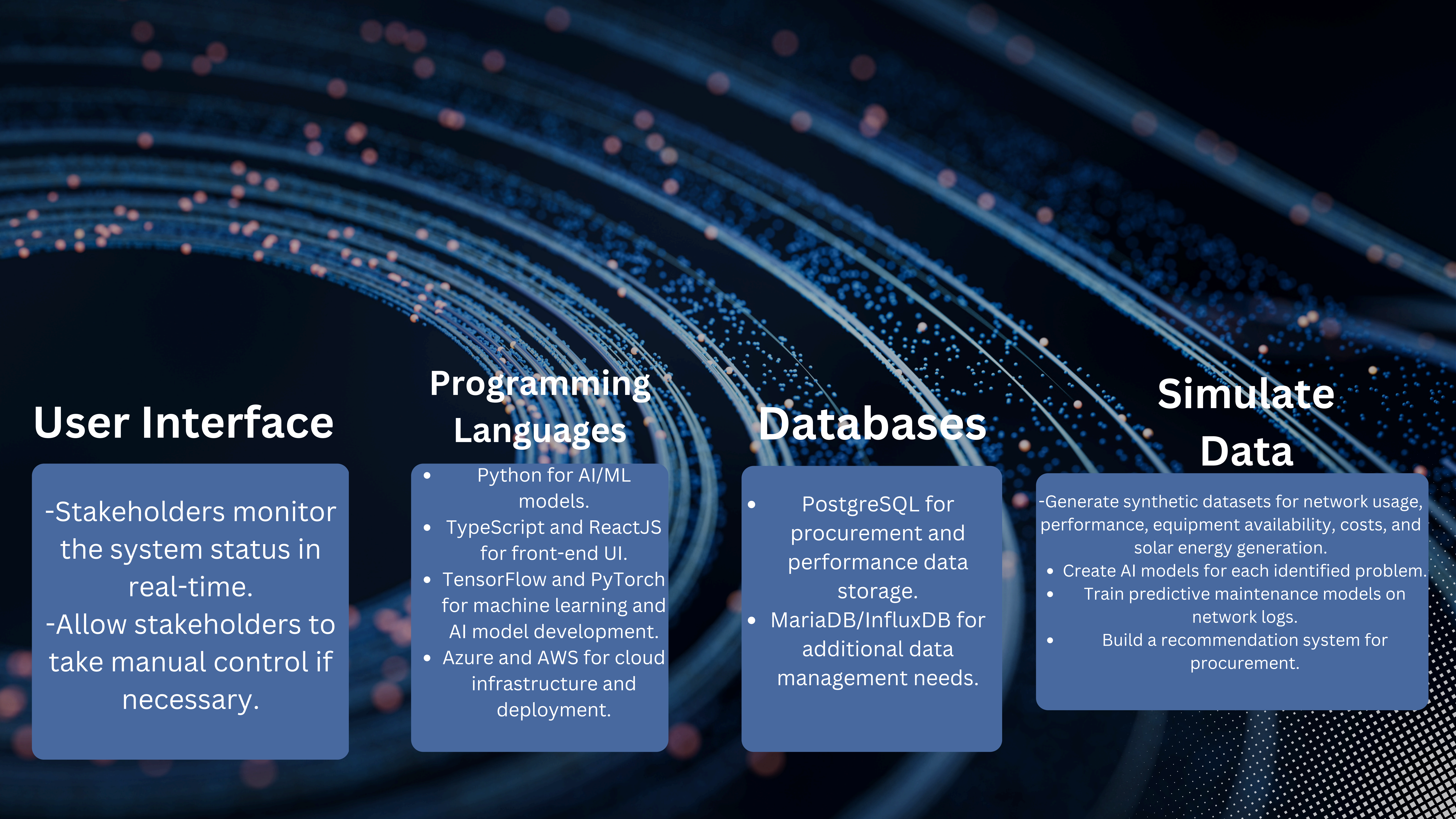
-Engine prioritizes critical maintenance tasks.

Action Execution

○ Throttles non-educational bandwidth.

○ Triggers maintenance tickets when needed.

○ Switches to backup power for energy management.



User Interface

-Stakeholders monitor the system status in real-time.

-Allow stakeholders to take manual control if necessary.

Programming Languages

- Python for AI/ML models.
- TypeScript and ReactJS for front-end UI.
- TensorFlow and PyTorch for machine learning and AI model development.
- Azure and AWS for cloud infrastructure and deployment.

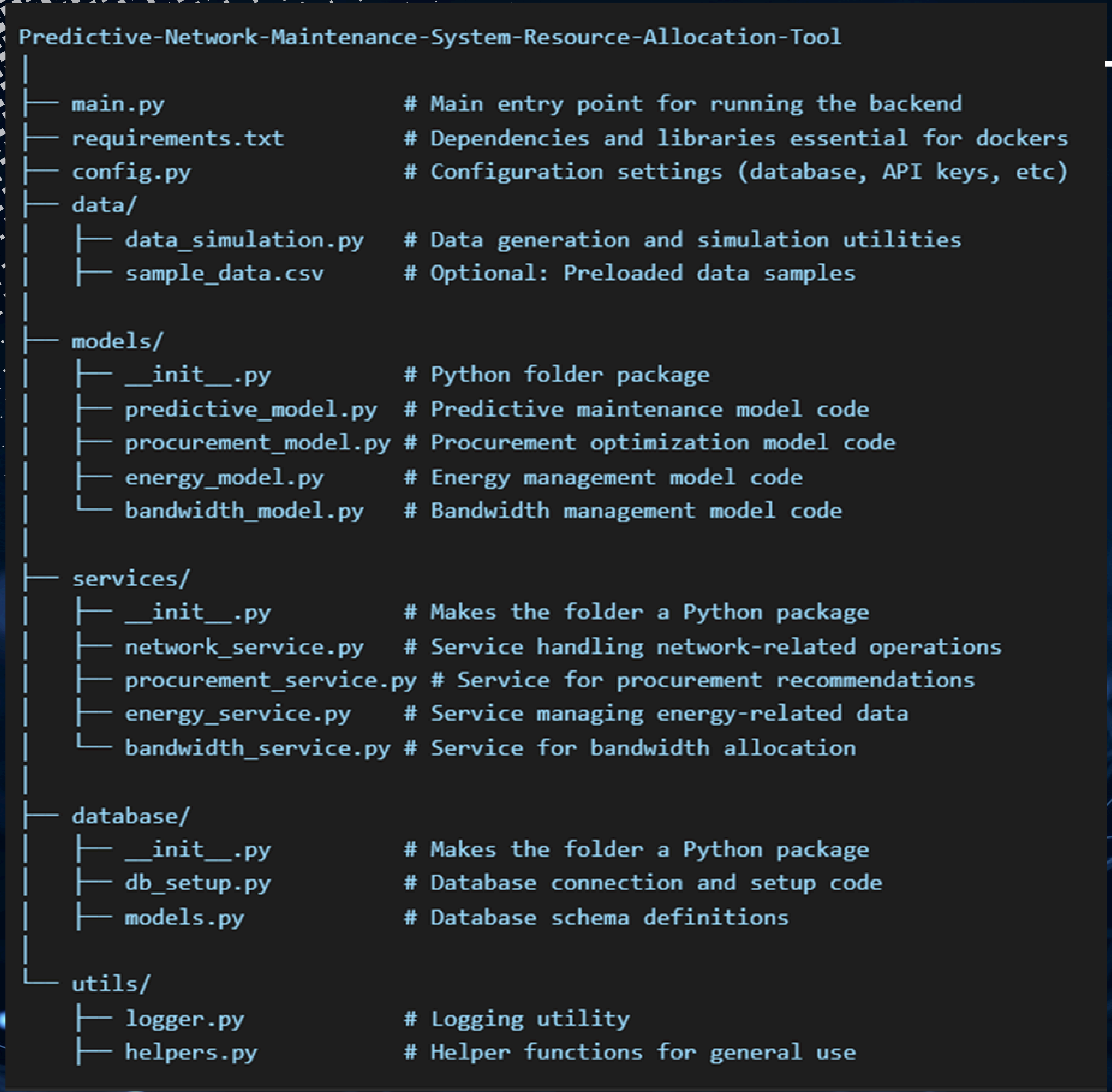
Databases

- PostgreSQL for procurement and performance data storage.
- MariaDB/InfluxDB for additional data management needs.

Simulate Data

-Generate synthetic datasets for network usage, performance, equipment availability, costs, and solar energy generation.

- Create AI models for each identified problem.
- Train predictive maintenance models on network logs.
- Build a recommendation system for procurement.



```
Predictive-Network-Maintenance-System-Resource-Allocation-Tool
│
├── main.py                  # Main entry point for running the backend
├── requirements.txt         # Dependencies and libraries essential for dockers
├── config.py                # Configuration settings (database, API keys, etc)
├── data/
│   ├── data_simulation.py   # Data generation and simulation utilities
│   ├── sample_data.csv      # Optional: Preloaded data samples
│
├── models/
│   ├── __init__.py          # Python folder package
│   ├── predictive_model.py  # Predictive maintenance model code
│   ├── procurement_model.py # Procurement optimization model code
│   ├── energy_model.py      # Energy management model code
│   └── bandwidth_model.py   # Bandwidth management model code
│
├── services/
│   ├── __init__.py          # Makes the folder a Python package
│   ├── network_service.py   # Service handling network-related operations
│   ├── procurement_service.py # Service for procurement recommendations
│   ├── energy_service.py    # Service managing energy-related data
│   └── bandwidth_service.py # Service for bandwidth allocation
│
├── database/
│   ├── __init__.py          # Makes the folder a Python package
│   ├── db_setup.py          # Database connection and setup code
│   ├── models.py            # Database schema definitions
│
└── utils/
    ├── logger.py            # Logging utility
    ├── helpers.py           # Helper functions for general use
```

-Contains configuration settings for the system, such as API keys, database details, and other global variables.

-Lists the required libraries and dependencies for development and deployment environments.

-Contains system configuration settings like API keys, database details, and other global variables.

-Provides tools for generating and simulating test data.

-Includes sample data for testing or demonstration purposes.

-Implements algorithms to predict and prevent network failures.

-Optimizes procurement processes and resource allocation for cost efficiency.

-Manages energy efficiency and consumption models for sustainable operations.

-Handles bandwidth management and allocation algorithms for efficient usage.

- Tracks application events, errors, and aids in debugging.
- Provides general utility functions used throughout the system for efficiency.

-Oversees and manages network operations and processes.

-Provides recommendations for procurement based on AI model predictions.

-Manages energy-related operations and integrates energy models.

-Allocates and optimizes bandwidth resources for efficient performance.

-Contains code for establishing database connections and initializing tables.

-Defines database schemas and their relationships for structured data storage.

ENHANCEMENTS AND SUGGESTIONS

- Add a README.md at the root level to guide users on setting up and using the tool.
- Ensure all scripts and functions include detailed docstrings.
- Create a tests/ directory to house unit and integration tests.
- Utilize a testing framework such as Pytest or Unittest.
- Maintain a CHANGELOG.md file to log updates and new features.
- Use an .env file to manage sensitive configurations (e.g., API keys, database credentials) and load them with libraries like python-decouple or dotenv.

Containerization

- Add a Docker file and docker-compose.yml for seamless containerized deployment.

Logging Improvements

- Enhance logger.py to include multiple log levels (e.g., DEBUG, INFO, WARNING, ERROR).
- Implement log rotation to archive old logs automatically.

-Directory structure ensures modularity, scalability, and maintainability for the system.
-Suggested enhancements improve usability, robustness, and deployment readiness.