**Green Matrix: A Technical Overview**

**Executive Summary**

Green Matrix is a sophisticated software solution designed to monitor and optimize system performance, with a specialized focus on the efficiency of Artificial Intelligence (AI) models. It provides comprehensive insights and actionable recommendations to improve efficiency, reduce operational costs, and maximize resource utilization.

The platform operates on two primary fronts:

1. An AI-powered core that predicts the performance of specific AI workloads on various hardware configurations, offering recommendations for model optimization, hardware selection, and resource scaling both before and after deployment.
2. A robust, rule-based engine that continuously monitors process-level and virtual machine (VM) metrics, providing insights for general performance tuning, cost reduction, and energy efficiency.

This document provides a detailed technical overview of the Green Matrix system, covering its AI-driven modules, rule-based logic, and underlying architecture.

**PART I: AI-POWERED WORKLOAD OPTIMIZATION**

**This section details the predictive and prescriptive capabilities of Green Matrix designed specifically for AI and Machine Learning workloads.**

**1. Simulate Performance**

This module predicts the performance of AI models on various hardware configurations, enabling users to make informed decisions before deployment. The simulation accounts for both **Inference** and **Training** tasks.

**1.1 Datasets**

**1.1.1 Inference Dataset**

A comprehensive dataset was created by merging two distinct sources: the **MLPerf Inference dataset** and a proprietary **Benchmarking Inference dataset**.

* **MLPerf Dataset:** A collection of benchmark suites from MLCommons, used to fairly evaluate ML systems across tasks like image classification, object detection, and NLP.
* **Benchmarking Dataset:** Created by running a custom script against various hardware using inputs like the MLPerf dataset to maintain integrity.

Model-specific information (Framework, Total Parameters, etc.) was collected via another custom script and merged. Hardware-specific data was also collected from various sources. To enhance the data's effectiveness, additional feature columns were engineered. The final dataset includes:

* **Model:** Name of the AI model.
* **CPU:** Central Processing Unit.
* **GPU:** Graphics Processing Unit.
* **# of GPU:** Quantity of GPUs.
* **Latency (ms):** Time to produce a result.
* **Scenario:** Use-case being benchmarked (e.g., Server, Single Stream).
* **GPU Memory Total - VRAM (GB):** Total video memory.
* **GPU Graphics clock:** GPU core processing speed.
* **GPU Memory clock:** GPU memory speed.
* **GPU SM Cores:** Number of Streaming Multiprocessors.
* **GPU CUDA Cores:** Number of parallel processing units.
* **CPU Total cores (Including Logical cores):** Total CPU cores.
* **CPU Threads per Core:** Simultaneous tasks per CPU core.
* **CPU Base clock (GHz):** Standard CPU operating speed.
* **CPU Max Frequency (GHz):** Maximum CPU speed.
* **L1 Cache:** Size of the fastest CPU memory cache.
* **Framework:** Software library (e.g., PyTorch).
* **Task Type:** Machine learning task (Inference).
* **Total Parameters (Millions):** Number of learnable variables.
* **Model Size (MB):** File size of the model.
* **Architecture type:** Neural network design.
* **Model Type:** General model category.
* **Precision:** Numerical precision (e.g., FP32).
* **Vocabulary Size:** Number of unique tokens.
* **Activation Function:** Neuron output function.
* **GFLOPs (Billions):** Computational complexity.
* **Number of hidden Layers:** Layers between input and output.
* **Number of Attention Layers:** Layers for focusing on relevant input.

**1.1.2 Training Dataset**

Following the same methodology as the inference dataset, this was created by combining the **MLPerf Training** and proprietary **Benchmarking datasets** with model and hardware specifics. The columns are largely the same as the inference dataset, with the addition of:

* **Batch Size:** Number of data samples processed in one iteration.
* **Input Size:** Dimensions of the input data.
* **IsFullTraining:** Flag for training from scratch (1) or fine-tuning (0).

**1.2 Modeling**

An **XGBoost regression model** was determined to be the top performer for this regression task, achieving scores of **93% for Inference** and **92% for Training**. A **Mapie regressor** was implemented on top of the XGBoost model to provide confidence scores for predictions. The data processor and models were saved as pickle files for backend use.

**1.3 Feature Importance:**

Here is the list of top 10 features that were important in predicting the final output and their weightage:

**1.3.1 Inference:**

* CPU Base Clock: 0.64
* Number of Attention Layers: 0.06
* GPU Memory Total - VRAM (GB): 0.05
* CPU: 0.04
* Number of hidden Layers: 0.04
* GPU: 0.02
* # of GPU: 0.01
* Precision: 0.01
* GPU CUDA Cores: 0.005
* CPU Total cores (Including Logical cores): 0.0012

**1.3.2 Training:**

* CPU: 0.68
* Model: 0.13
* Precision: 0.103
* Activation Function: 0.016
* GPU: 0.013
* CPU Base Clock (GHz): 0.012
* CPU Total cores (Including Logical cores): 0.0078
* Batch Size: 0.005
* CPU Max Frequency (GHz): 0.004
* GPU CUDA Cores: 0.001

**1.4 Input and Output**

**1.4.1 Input Parameters**

* **Inference:** Model-specific details (Model, Framework, Total Parameters, etc.) are auto-filled upon model selection but can be overridden. The Scenario defaults to "Single Stream". This is combined with hardware data from the database.
* **Training:** Similar to inference, with the addition of Batch Size, Input Size, and IsFullTraining (defaults to "No").

**1.4.2 Output Parameters**

The model's predictions and intervals are fed to a custom function to generate the UI table. This function calculates:

* **Cost:** Based on power consumption and latency.
* **Memory Fit:** Determines if the model can run on the available memory.
* **Confidence Score:** Derived from the Mapie regressor's intervals.
* **Predicted Latency:** The model's latency prediction.

**Memory Fit Formulas:**

* **Inference:**
  + VRAM Needed = Total Parameters \* Bytes per Parameter + 20% Buffer (for dynamic overhead)
  + RAM = VRAM \* 1.5
* **Training:**
  + Parameter Memory = Total Parameters \* Bytes per Parameter
  + Gradient Memory = Total Parameters \* Bytes per Parameter
  + Optimizer Memory = Total Parameters \* Bytes per Parameter \* 2
  + Total Memory = (Parameter + Gradient + Optimizer) / 1e9
  + Activation Memory = (Batch Size + Input Length + Hidden Dimension + Number of layers \* 10) / 1e9
  + Framework Overhead = 1.5
  + VRAM = Total Memory + Activation Memory + Framework Overhead
  + RAM = VRAM \* 1.5

A rounding function is used to provide memory values in terms of 2^n.

**Bytes Per Parameter Mapping:** "FP32": 4, "FLOAT32": 4, "FP16": 2, "FLOAT16": 2, "BF16": 2, "BFLOAT16": 2, "INT8": 1

**2. Pre-Deployment Recommendations**

**2.1 Bare Metal Recommendations**

* **Input:** Same as the "Simulate Performance" module.
* **Output:** The top three hardware configurations based on cost-effectiveness.

**2.2 VM Level Recommendations**

* **Input:** Model-specific information (auto-filled and overridable).
* **Output:** The minimum VRAM required for a VM slice, calculated using the memory fit formulas.

**3. Post-Deployment Recommendations**

**3.1 Bare Metal**

* **Dataset:** A public resource utilization dataset (CPU, RAM, GPU, VRAM utilization) was used. A "Recommendations" column was engineered with classes: Upgrade, Downgrade, Maintain.
* **Modeling:** A classifier model was trained on this dataset, achieving an accuracy of **99%**.
* **Input:** Utilization metrics (CPU, GPU, RAM, VRAM, Network Bandwidth, DISK IOPS), model specifications, and the current hardware.
* **Output:** A rule-based system uses the model's output to rank available hardware and recommend an upgrade or downgrade relative to the current hardware.

**3.2 VM Level**

* **Dataset:** A public resource utilization dataset was used, with an engineered "Recommendations" column: Scale Out, Scale In, Maintain.
* **Modeling:** A classifier model trained on this dataset, achieving an accuracy of **100%**.
* **Input:** Utilization metrics and model specifications. Current VM VRAM is captured via a script.
* **Output:** The system first computes memory fit to check if vertical scaling is needed. If so, a new VM instance with the recommended VRAM is suggested. Based on the AI model's output (Scale In or Scale Out), horizontal scaling with the exact VRAM is recommended. The script also captures available VRAM on the bare metal for more efficient recommendations.

**4. Model Optimization Recommendations**

* **Dataset:** A dataset was created by combining model specifications with custom output columns: "Recommended method" and "Recommended Precision type."
  + **Recommended Methods:**
    - **Quantization-Aware Training (QAT):** Simulates quantization effects during training to adapt the model to reduced precision and maintain higher accuracy.
    - **Post-Training Static Quantization (PTSQ):** A simpler, faster method that converts a trained model to a lower precision format without retraining.
  + **Recommended Precision:** Ranging from FP16 to INT4.
* **Modeling:** A classification model was built with a score of **99%**.
* **Input:** Model specifications, auto-filled upon model selection.
* **Output:** Recommendations for both the optimization method (QAT/PTSQ) and the precision type to be used when creating the AI model.

**PART II: RULE-BASED SYSTEM & RESOURCE MONITORING**

Green Matrix implements an advanced rule-based recommendation system that analyzes system performance, resource utilization, and cost patterns using percentage-based thresholds, peak usage analysis, and frequency pattern detection. Key enhancements include:

* Dynamic Percentage-Based Thresholds: Scales with actual hardware capacity instead of fixed values.
* Peak Usage Analysis: Analyzes peak consumption patterns over time periods.
* Frequency Analysis: Considers how often thresholds are crossed, not just averages.
* Intelligent Capacity Estimation: Automatically estimates system capacity from usage patterns.
* Enhanced Cost Analysis: More accurate savings calculations based on usage patterns.

1. **Host Process Analysis Rules**

* Data Collection: Metrics are collected every 2 seconds over a default 7-day period. The system focuses on processes with comprehensive resource data, enabling real-time pattern detection.
* **CPU Utilization Analysis:**
* High CPU Usage: >80% sustained CPU utilization triggers performance recommendations.
* Low CPU Usage: <20% sustained CPU utilization triggers cost optimization recommendations.
* Trigger Condition: Must exceed thresholds for >30% of the monitoring period.
* **Memory Usage Analysis:**
* Percentage-Based Thresholds are used, based on total system memory capacity.
* High Memory Pressure: >80% of total memory with >30% frequency of threshold crossing.
* Low Memory Usage: <20% of total memory for >70% of the monitoring period.
* Peak Analysis is used to analyze peak memory consumption patterns.
* **Power Consumption Analysis:**
* Dynamic Power Thresholds are based on estimated system power capacity.
* High Power Usage: >70% of estimated capacity or >85% with >25% frequency.
* Low Power Usage: <20% of capacity for >60% of the monitoring period.
* Capacity Estimation: Uses 95th percentile of observed power + 20% buffer.
* Recommendation Categories: Critical (>85% capacity with high frequency), High Priority (>80% sustained), Medium Priority (<20% sustained), and Low Priority (pattern-based).

1. **VM Instance Analysis Rules**

* **CPU Utilization Analysis:**
* High CPU Usage: >80% sustained utilization → Scale up CPU resources.
* Low CPU Usage: <20% sustained utilization → Downsize CPU allocation (30-50% cost reduction).
* Frequency Threshold: Must persist for >30% of the analysis period.
* **Memory Usage Analysis:**
* Critical Memory Pressure: >80% of VM memory AND >30% frequency of crossing. Action: Immediate memory increase (Peak × 1.2 + buffer).
* High Memory Usage: >80% average utilization. Action: Consider increasing memory (Total × 1.3).
* Memory Over-Allocation: <20% usage for >70% of the period. Action: Reduce allocation to max (peak\_memory × 1.2, 2GB minimum).
* **Power Consumption Analysis:**
* Critical Power: >85% of estimated capacity with >25% frequency. Action: Immediate power management review.
* High Power Usage: >70% of estimated capacity sustained. Action: Implement power management policies.
* Power Over-Provisioning: <20% capacity for >60% of period. Action: Recommend infrastructure consolidation.
* **GPU Utilization Analysis:**
* High GPU Usage: >80% sustained → Scale GPU resources or optimize distribution.
* GPU Under-Utilization: <20% with GPU allocated → Remove/share for significant cost savings.

1. **Cost and Energy Efficiency Optimization**

**9.1. Cost Optimization Rules**

* Migration Threshold: A minimum of 15% cost reduction is required to trigger migration recommendations.
* Priority Levels: High (>30% savings), Medium (15-30% savings), Low (<15% savings).
* Cost Calculation:
* Daily energy (kWh) = (avg\_power\_watts × 24) / 1000
* Monthly projection (kWh) = daily\_energy\_kwh × 30
* Migration Considerations: Network latency, data transfer costs, compliance, and service availability are evaluated.

**9.2. Energy Efficiency Rules**

* Dynamic Threshold System: Adapts to hardware capacity using intelligent estimation (95th\_percentile\_power × 1.2, minimum 150W).
* Critical Power Conditions: >85% capacity AND >25% frequency of high power events. Action: Immediate thermal and power supply review.
* High Power Usage: >70% of estimated capacity sustained. Action: Advanced power management, renewable energy scheduling.
* Power Under-Utilization: <20% capacity for >60% of the monitoring period. Opportunity for infrastructure consolidation.
* Energy Calculations: kWh = (power\_watts × time\_hours) / 1000. CO2 impact is calculated as total\_energy\_kwh × 0.4kg.

**PART III: SYSTEM ARCHITECTURE & IMPLEMENTATION**

1. **Data Sources and Configuration**

* Data Sources:
* Host Metrics (PostgreSQL): host\_process\_metrics table for CPU, memory, GPU, and power data.
* VM Metrics (TimescaleDB): vm\_process\_metrics table for per-VM trends and patterns.
* Cost Models (PostgreSQL): cost\_models table with regional electricity pricing.
* Configuration (config.ini):  
  cost\_per\_kwh = 0.12  
  currency\_symbol = $  
  current\_region = US
* Engine Thresholds:
* min\_savings\_threshold: 0.15 (15%)
* default\_time\_range\_days: 7
* default\_projection\_days: 30
* memory\_high\_threshold\_percent: 80
* memory\_low\_threshold\_percent: 20
* power\_high\_threshold\_percent: 70
* power\_critical\_threshold\_percent: 85
* frequency\_threshold\_high: 30
* frequency\_threshold\_low: 70

1. **API Endpoints**

* GET /api/recommendations/host-processes: For host-level process recommendations.
* GET /api/recommendations/vm/{vm\_name}: For specific VM resource right-sizing.
* GET /api/recommendations/cost-optimization: For regional cost comparisons.
* GET /api/recommendations/top-energy-processes: To identify highest energy consumers.