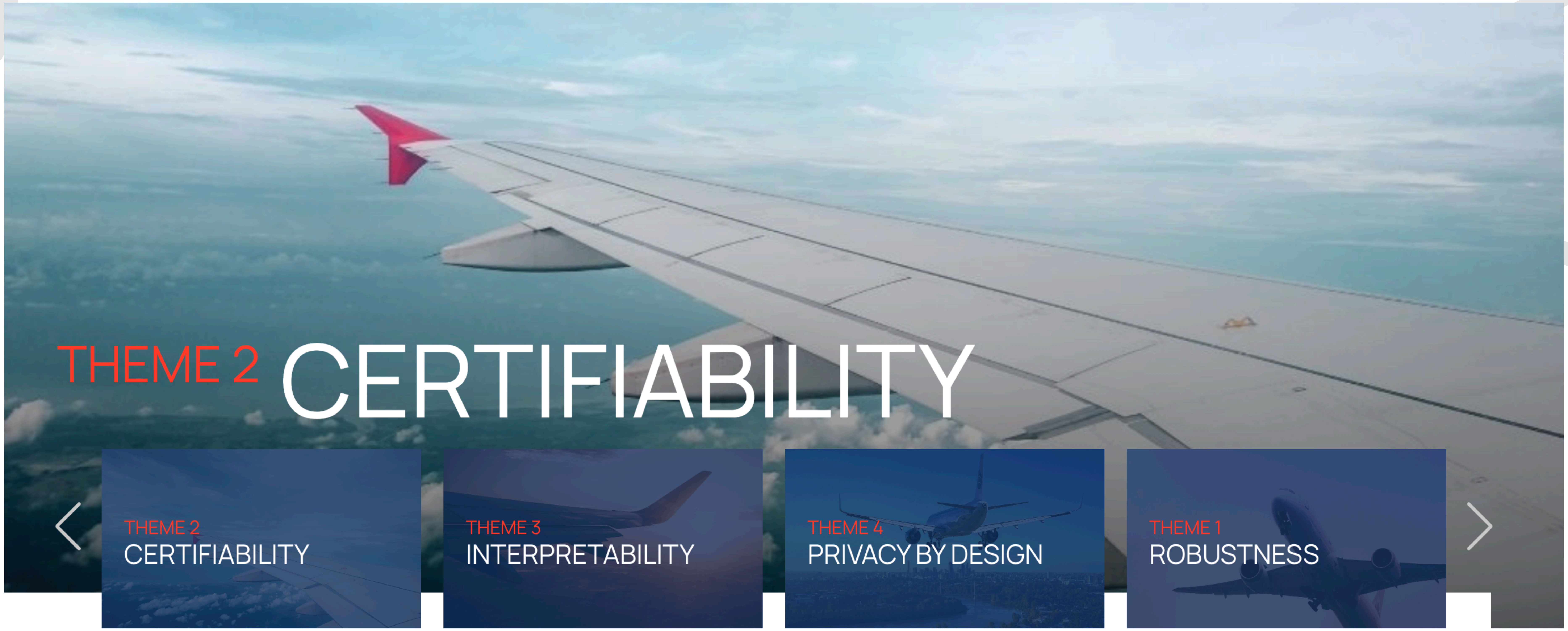
HELLO!

Leveraging OpenInfra and Open Source Gen. AI To Address Climate Change

Armstrong Foundjem

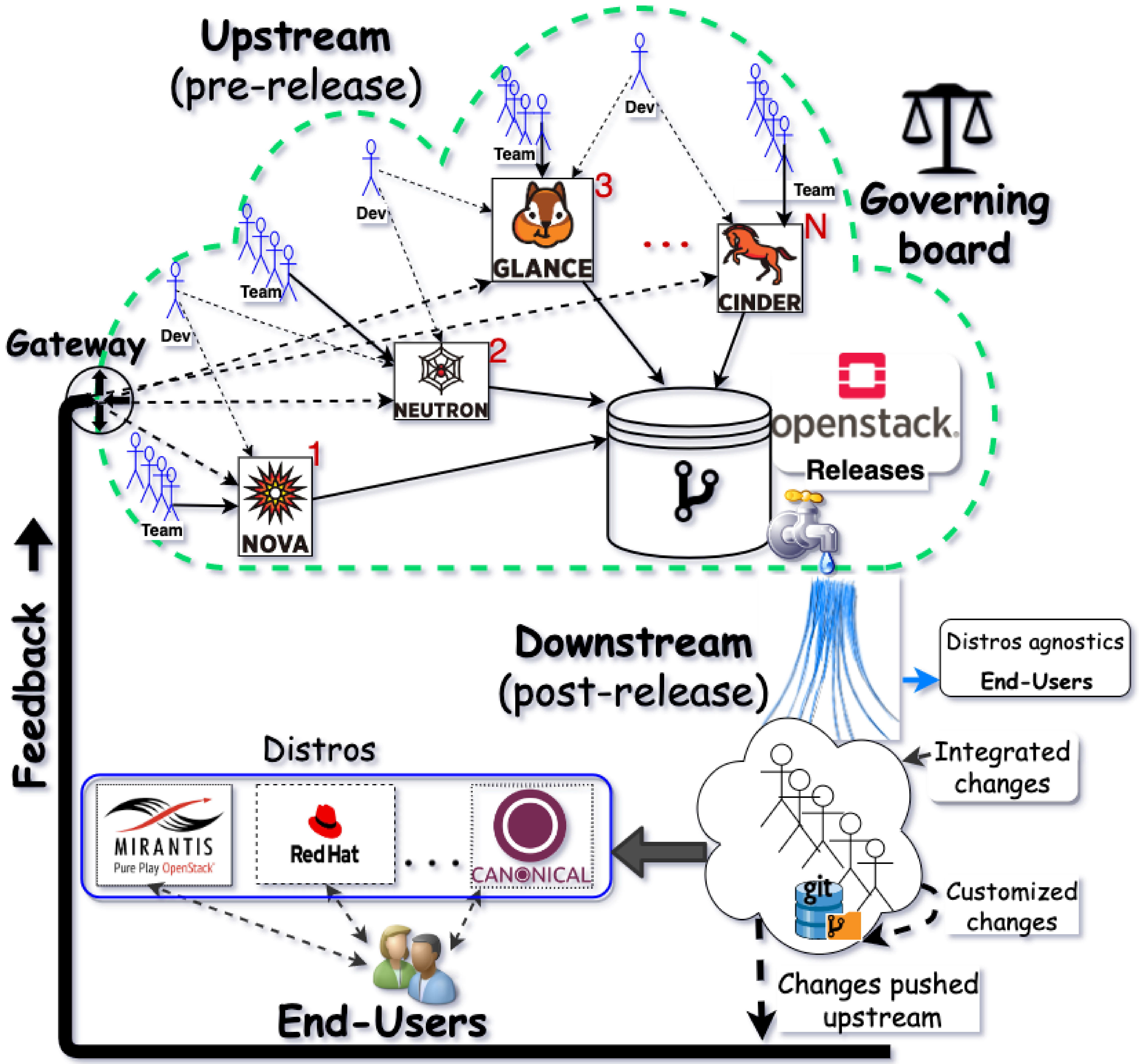
DEPENDABLE & EXPLAINABLE LEARNING — DEEL, Polytechnique Montreal

Certification of Safety-critical systems where failure can result in catastrophic consequences.



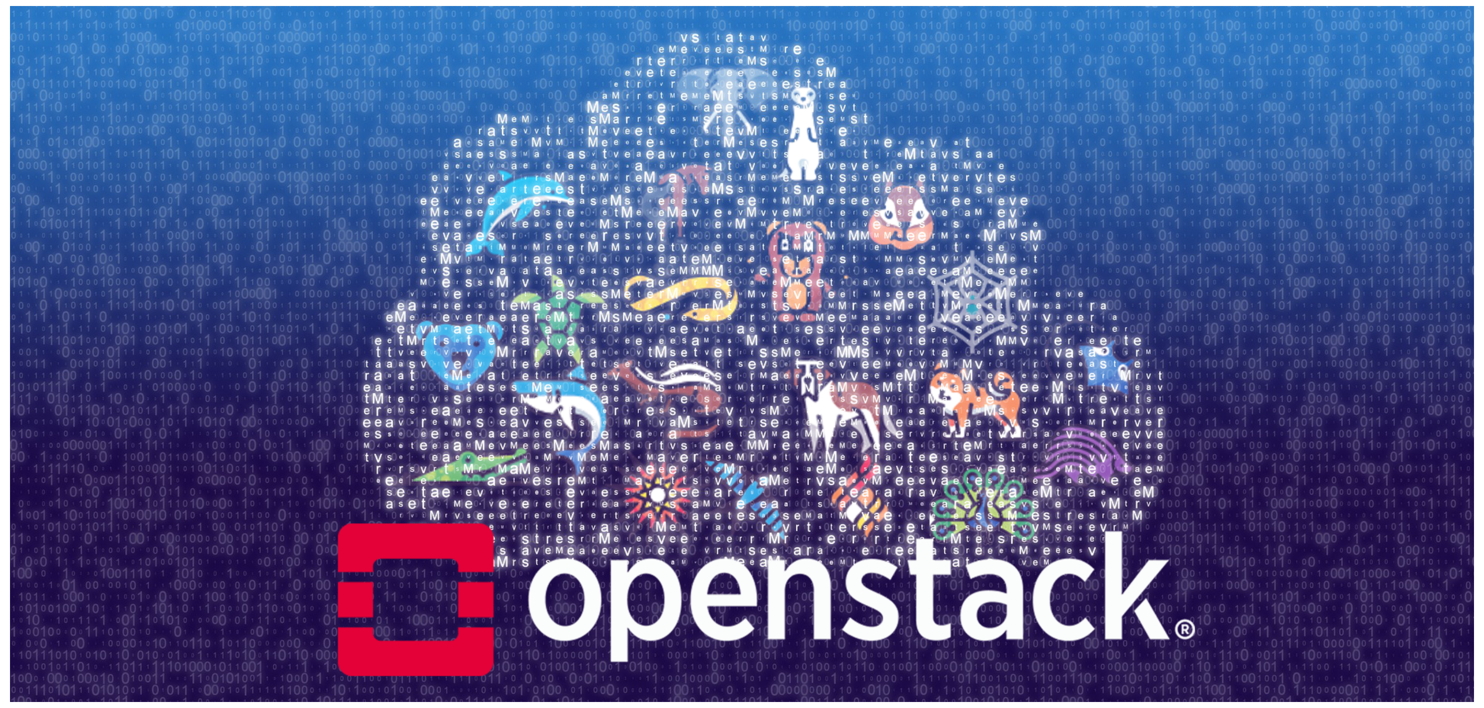
🌿 TOWARDS A GREENER OPENINFRA!

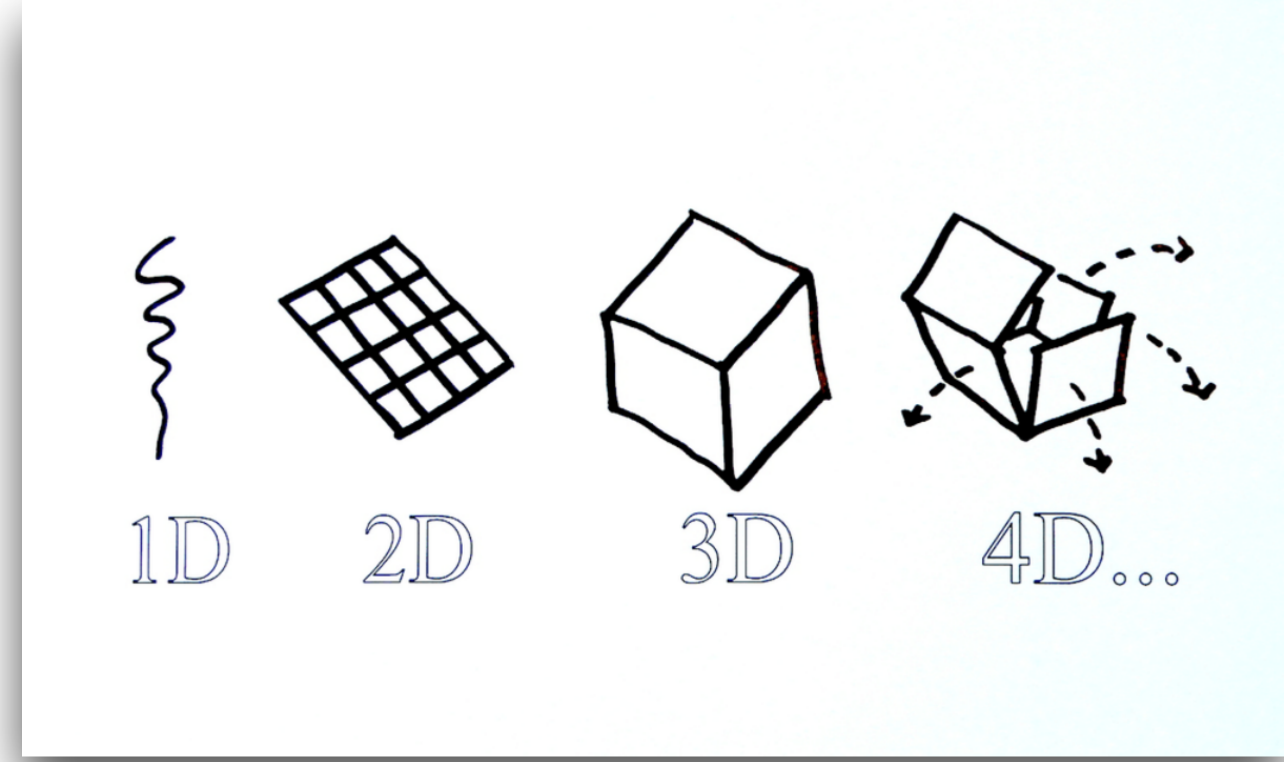
›› Tracking, optimizing, and reducing energy use, we can build sustainable AI systems.



Foundjem, Armstrong, Ellis E. Eghan, and Bram Adams. "**A Grounded Theory of cross-community SECOs: feedback diversity versus synchronization.**" *IEEE Transactions on Software Engineering* 49.10 (2023): 4731-4750.

Driving OpenInfra more sustainable; energy efficiency, resource optimization, and eco-friendly practices.





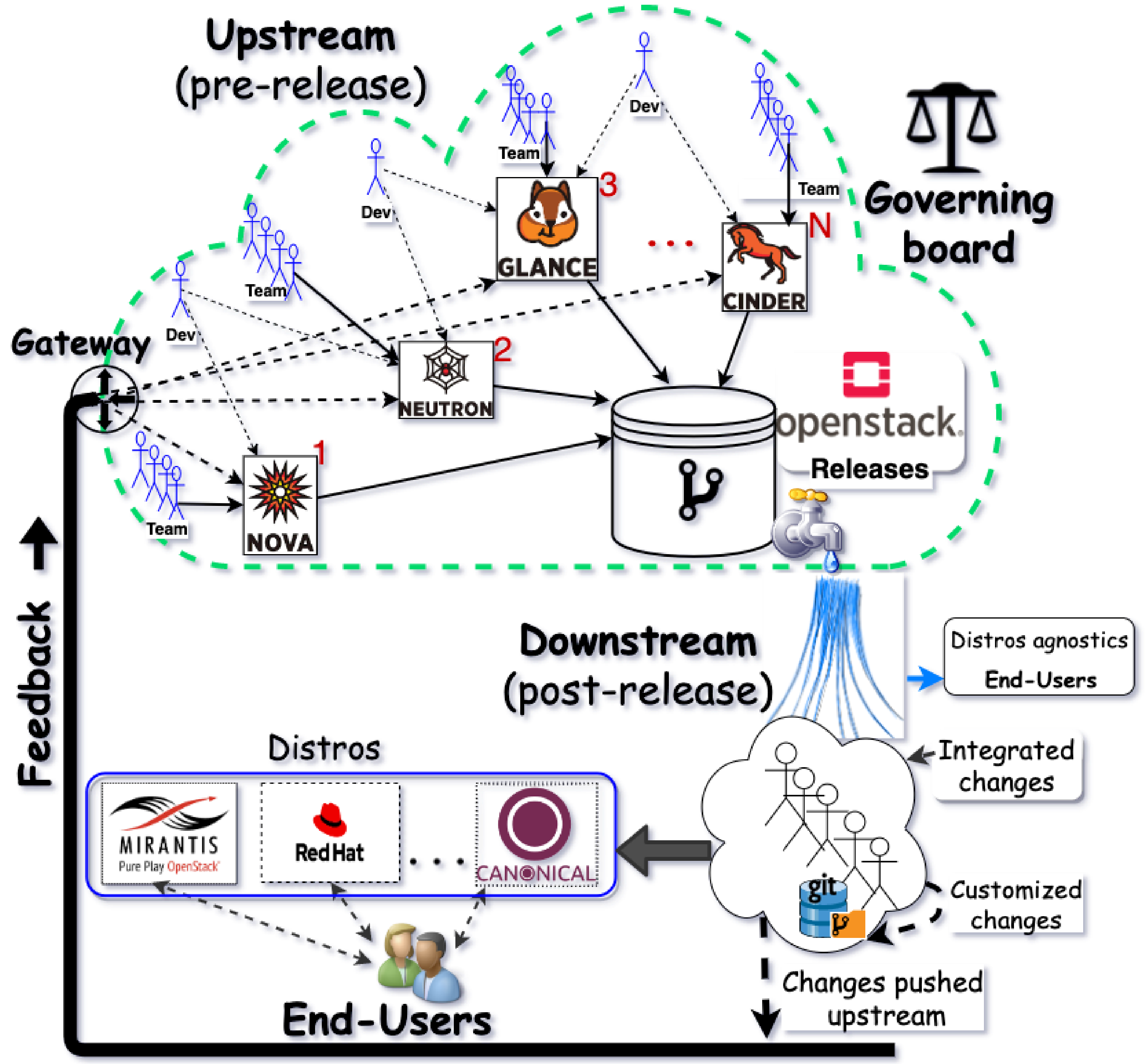
1. **Socio-Technical Dynamics**: Balancing community collaboration with efficient best practices to promote resilience and sustainability.

2. **Cyber Threats**: Protecting against cyber risks to ensure long-term stability and minimize disruptions to the ecosystem. Learn the Techniques, Tactics and Procedures (TTPs) that bad actors are using against your systems

3. **Economic Cooperation**: Encouraging collaboration and resource-sharing to fund and support sustainable development initiatives.

4. **Energy Consumption**: Reducing energy use through optimized coding, energy-efficient infrastructure, and greener hosting solutions.

OpenInfra generates high-volume and varieties of data suitable for integrating Gen. AI into its workflows to address climate change.



CLIMATE CHANGE

requires urgent, sustainable actions that promote responsible resource management, reduce greenhouse gas emissions, and create an eco-friendly software ecosystem that minimizes energy use and supports a climate-resilient digital future.

Foundjem, Armstrong, Ellis E. Eghan, and Bram Adams. "**A Grounded Theory of cross-community SECOs: feedback diversity versus synchronization.**" *IEEE Transactions on Software Engineering* 49.10 (2023): 4731-4750.

›› Chosen the Right Metrics is Essential

CO2eq (Carbon Dioxide Equivalent)

Power Usage Effectiveness $(PUE) = \dfrac{Total\ Facility\ Power}{IT\ Equipment\ Power}$; values [1.1 – 1.4] indicate highly efficient datacenter

KWh quantifies energy. Multiply KWh by local "carbon intensity" factor to estimate total CO_2 emissions.

CO2eq measures the climate impact of greenhouse gas emissions by comparing them to an equivalent amount of CO_2 that would produce the same global warming effect. When you see "gCO_2eq/kWh," it's a carbon intensity factor describing how many grams of CO_2eq are emitted per kWh of electricity generated.

➡ Different regions' energy sources lead to varying carbon intensities (e.g., 700 gCO_2eq/kWh in coal-heavy grids vs. ~100 gCO_2eq/kWh in renewable-focused areas). To determine total CO2eq emissions, multiply total kWh consumed by the regional carbon intensity factor.

$$CO2eq\ (kg) = (Energy\ usage\ (kWh)) \times (Carbon\ intensity\ (kgCO2eq/kWh))$$

>> Integration with OpenStack for Carbon-Aware Operations

OpenStack Telemetry (Ceilometer or Gnocchi) gathers CPU/memory usage, to understand energy patterns.

Power & CO2 Data collected from PDUs (or server IPMI) and stored alongside usage metrics.

AI-Driven Scheduling and Auto-scaling: !

Low-latency or fault-tolerant workload, are scheduled when local carbon intensity is lower.

Non-critical background tasks, are delayed until when the grid mix is greener.

An orchestration script (i.e., Heat templates) is used to auto-scale down idle nodes during high carbon intensity periods.

Why These Metrics Are Important for Sustainability

- **PUE**: Tells you how efficiently your datacenter uses energy beyond just the IT load. A high PUE indicates you should invest in efficient cooling, airflow management, or equipment modernization.

- **kWh**: The total energy usage is the foundation of your carbon footprint. Minimizing kWh (while still meeting workload needs) is key to lowering operational costs and emissions.

- **CO2eq**: Ultimately, the environment is impacted by total greenhouse gas emissions, not just raw power usage. Tracking CO2eq reveals your datacenter’s true environmental impact and shows how shifting workloads to greener hours or locations can lower emissions.

Socio-technical Metrics and Rationales

- **Commits Per Week**: Indicates developer activity; extremes can mean overwork or lack of engagement.
- **Open PRs**: Reflects backlog or review bottlenecks.
- **Code Churn**: Signals rework, potential frustration.
- **Context Switching**: High interrupt-driven tasks cause mental strain.
- **Review Load**: Excessive reviews lead to decision fatigue.
- **Meeting Hours**: Too many disrupt focus time.
- **Communication channels (Email sent/IRC, etc.)**: High communication load can signal stress or misaligned processes.
- **Late Night Work**: Sign of poor work-life boundaries.
- **Weekends Activities**: Indicates consistent overwork.
- **Sentiment Score**: Gauges emotional state; prolonged negativity correlates with burnout risk.

Predicting Burnout in Open-Source communities Based on Socio-Technical Indicators.

Comprehensive view of a developer's **workload**, **work patterns**, **engagement**, and **emotional well-being**:

1. **Workload Management**: Metrics like "Commits Per Week," "Open PRs," and "Review Load" help monitor the distribution and volume of work. If developers are overloaded, it can trigger early intervention.

2. **Cognitive Load**: "Context Switching" and "Meeting Hours" gauge how much mental energy is spent on non-productive tasks. High cognitive load often correlates with burnout.

3. **Work-Life Balance**: "Late Night Work" and "Weekend Activity" track whether developers are balancing their personal and professional lives. Overwork beyond regular hours is one of the leading causes of burnout.

4. **Emotional Well-being**: "Sentiment Score" provides a direct measure of how a developer feels, offering insight into their overall mood and job satisfaction, which are closely tied to burnout.

5. **Predictive Risk**: By combining all these factors into a "Burnout Risk" score, you can proactively identify at-risk developers and make data-driven decisions to prevent burnout before it becomes a major issue.

>> Attributes association with energy efficiency

Socio-technical dynamics influence both **energy consumption** and **cybersecurity**, as healthier collaboration leads to efficient practices and better collective defense against cyber threats.

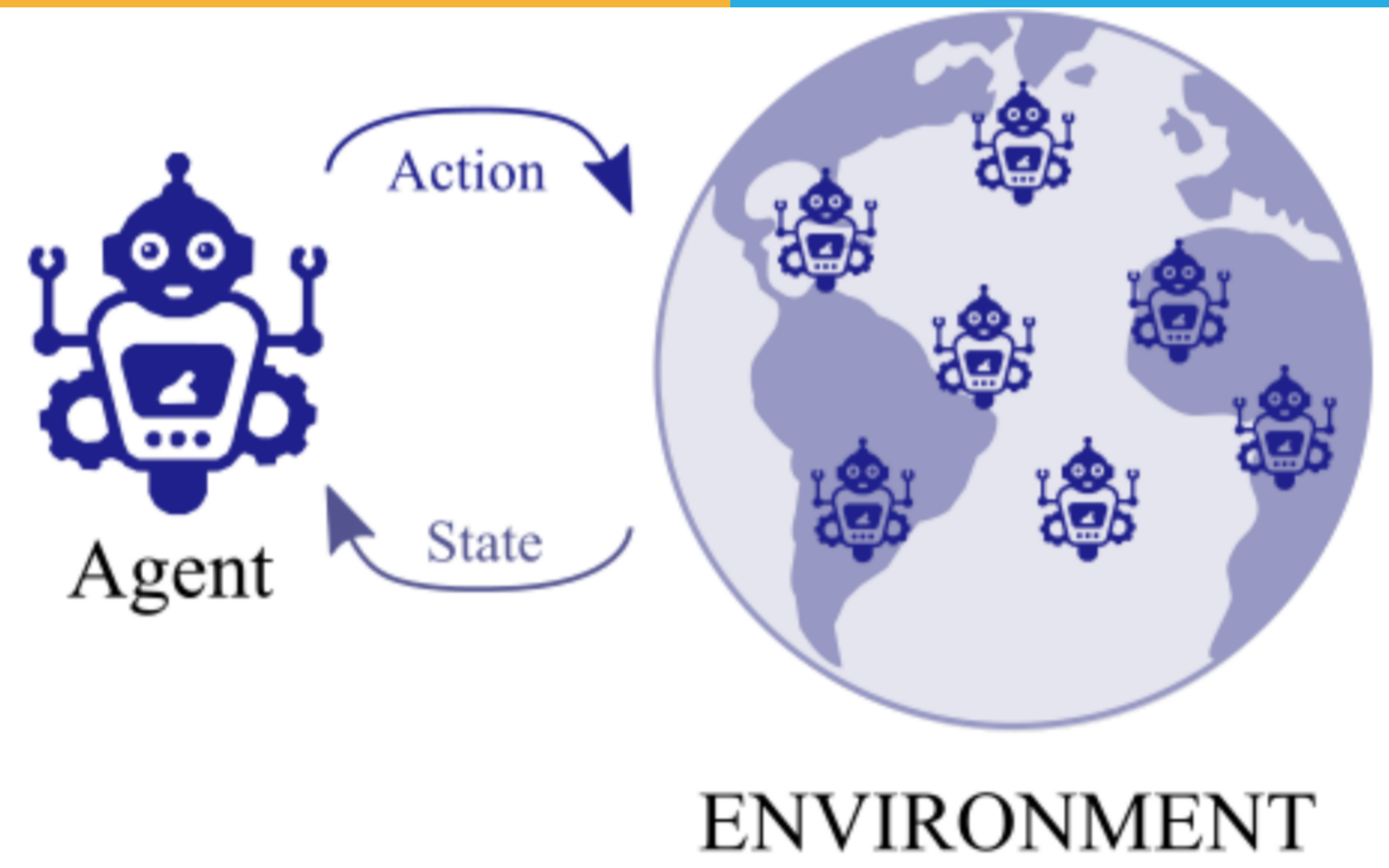
Cyber threats impact **economic cooperation** by potentially disrupting financial investments, while effective cybersecurity practices ensure that resources are used wisely and without waste.

Economic cooperation can fund **energy-efficient** infrastructure and incentivize contributors to adopt greener practices, fostering sustainable energy usage within the ecosystem.

E.g., Socio-Technical Dynamics and Energy Consumption

Contributor Well-being & Productivity:
Healthy socio-technical dynamics (such as clear communication, fair workload distribution, and community engagement) is **associated with higher productivity**, which can reduce unnecessary resource consumption, like redundant computations or excessive server usage (build), thus, promoting greener practices

>> HOW TO ADDRESS CARBON FOOTPRINTS IN OPEN SOURCE?



Agent-Based Generative Models (Multi-Agent Systems)

- **Use Cases**:
  - **Simulation of Developer and System Interactions**: Agent-based generative models simulate the behavior of multiple entities (e.g., developers, systems, tasks) interacting with each other. These models can help predict how developers' work habits impact system performance and energy consumption in a collaborative open-source ecosystem.
  - **Energy Optimization**: Multi-agent systems can autonomously adjust resource allocation, system configurations, and workload distribution by simulating and optimizing developer behavior and system load in a generative manner.

- **Reinforcement learning** algorithms (e.g., **PPO**) shows optimal performance, where agents learn to optimize their actions based on the environment (system performance, energy consumption) and interactions with other agents (developers) in a non intrusive manner to collect real-time data.

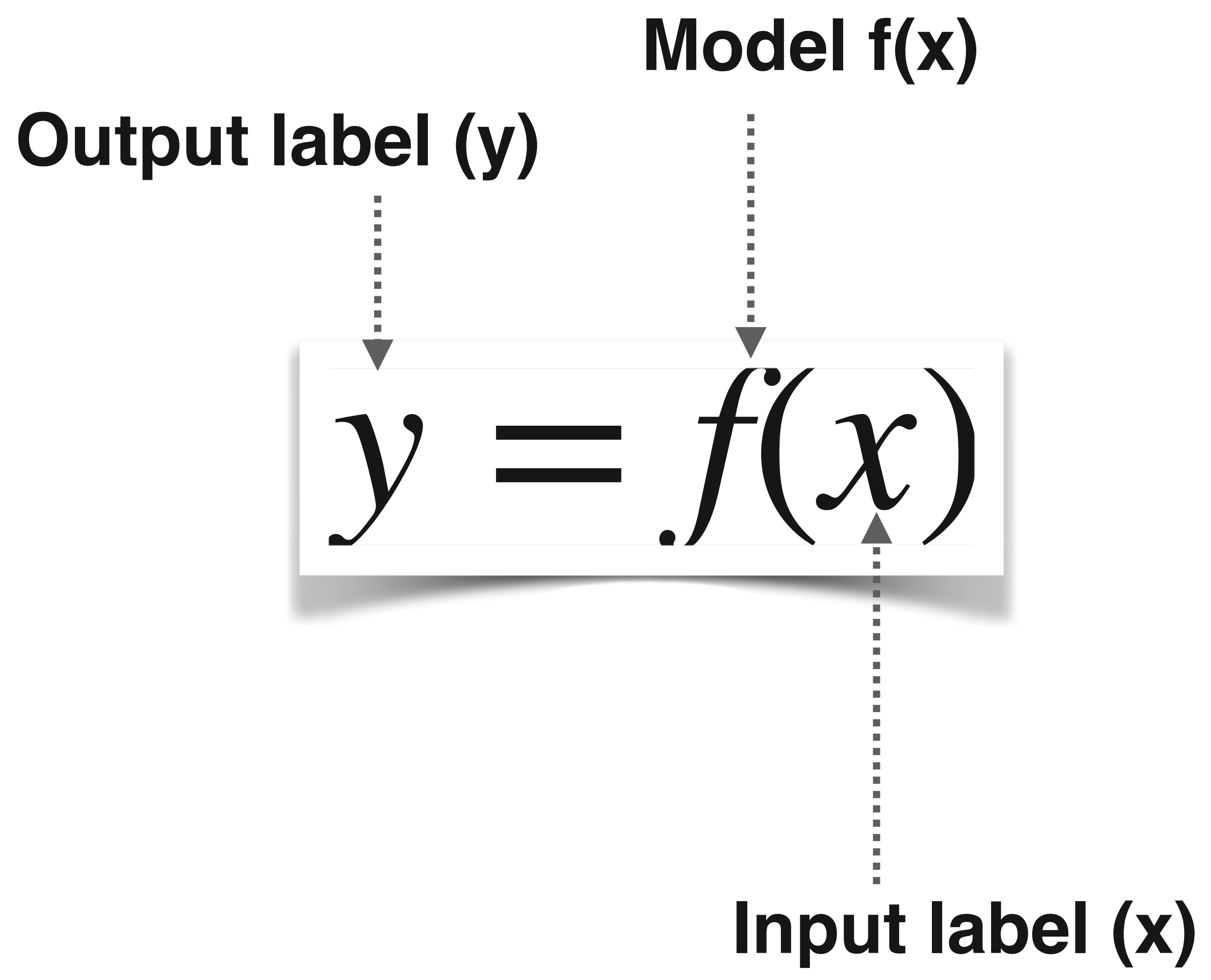
What do humans learn and what is AI?

Human Age	Cognitive Development Milestones
0–6 months	Recognizing faces, tracking objects, early memory formation.
6–12 months	Object permanence, early problem-solving, understanding cause-effect relationships.
12–24 months	First words, imitation of actions, simple problem-solving.
2–3 years	Explosion in language, understanding of categories, beginning of reasoning skills.
3–5 years	Symbolic thinking, early logical reasoning, basic numeracy, social intelligence development.

$f := Discriminative \mid Generative \mid Traditional$

What is a Model? $AI \subset AGI$

Type of Algorithm	Generative AI 🚀	Discriminative AI 🎯	Reinforcement Learning (RL) 🔄	Traditional Methods 📜
Goal	Learn the joint distribution $P(X,Y)$	Learn the conditional probability $P(Y \mid X)$	Learn the policy $\pi(s)$ to maximize cumulative reward	Use explicit rules and heuristics
Examples	GANs, VAEs, HMMs, Naive Bayes	Logistic Regression, SVMs, Decision Trees	Q-Learning, PPO, DQN	Regex, Decision Trees, Rule-based systems
Use Case	Code generation, test case generation	Bug detection, image classification	Robotics, Game AI, task scheduling	Static analysis, bug detection
Mathematical Representation	$P(Y \mid X) = \frac{P(X,Y)}{P(X)}$	$P(Y \mid X) = \frac{1}{1+e^{-(wX+b)}}$	$Q(s,a) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$	$f(X) = \sum_{i=1}^{n} c_i r_i(X)$



Discriminative if y = Probability, Number, or Class

Probabilistic

Generative: if y = text, image, audio, video, code, etc.

Deterministic

Traditional:

Rule: if $X_1 > 0.5$ **then** $Y = 1,$ **else** $Y = 0$

$$f(X) = \begin{cases} 1, & \text{if code violates rule} \\ 0, & \text{otherwise} \end{cases}$$

Regex$(X) = \{x \mid x$ **matches the rule**$\}$

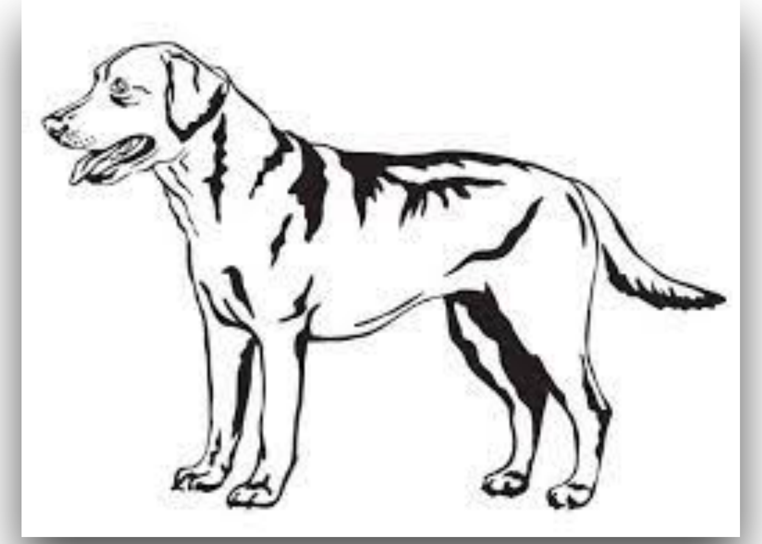
Generative Models Generate New Data Instances Within Similar Distribution, Discriminative Models Discriminate Between Different Cases, and Traditional Techniques Rely on Rule-Based Systems for Classification

Discriminative techniques

Classifier →

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Dog →



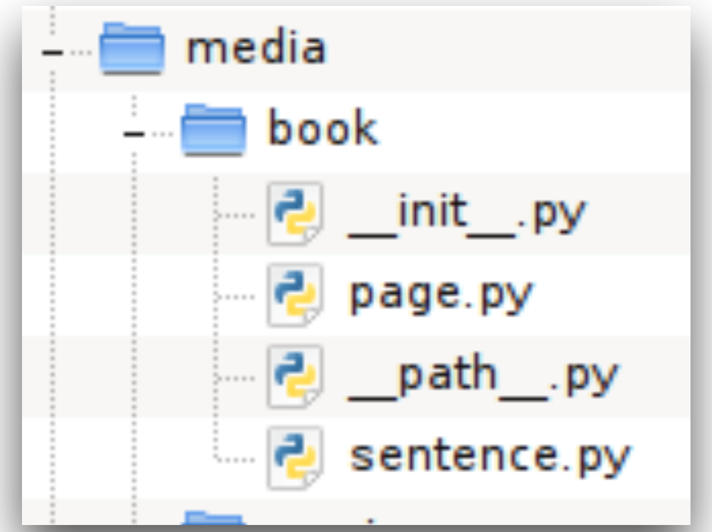
Generative techniques

Generator →

$$P(x,y) = P(x|y)P(y)$$

Cat →

Traditional techniques



Rule base →

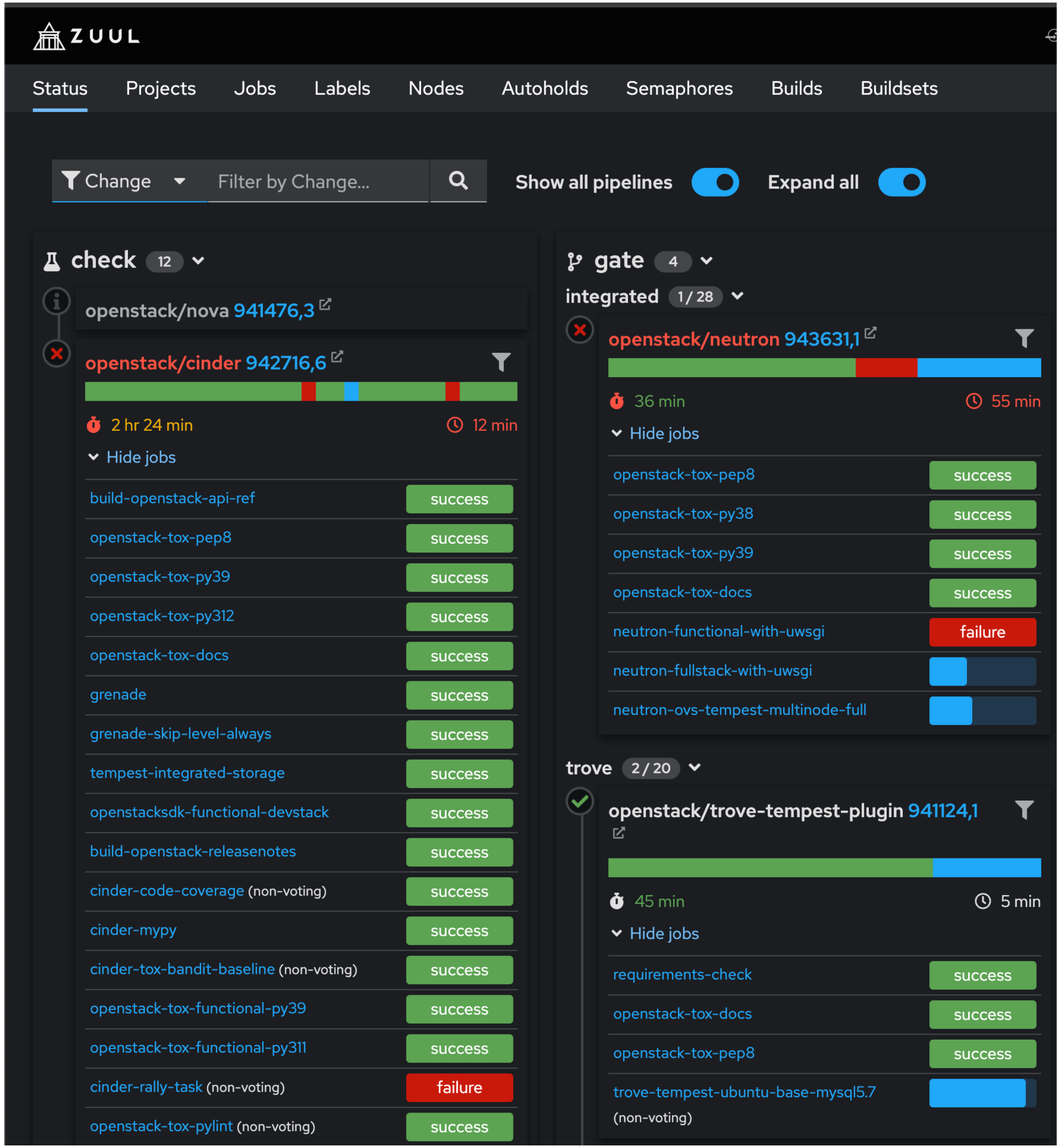
```
grep -Rn --include=\*.py 'def [a-zA-Z_][a-zA-Z0-9_]*' /path/to/codebase
```

↓

```
src/utils.py:23:def process_data():
src/main.py:45:def run_analysis():
```

This finds **all Python function definitions** in the codebase.

>> Observing the natural environment



Training multi-agents AI for specific tasks in the environment

>>

```python
# ============================
# 🤖 2. MULTI-AGENT RL ENVIRONMENT
# ============================
class MultiAgentTaskEnv(gym.Env):
    def __init__(self, num_agents=3):
        super(MultiAgentTaskEnv, self).__init__()
        self.num_agents = num_agents
        self.observation_space = spaces.Box(low=0, high=1, shape=(4,), dtype=np.float32)
        self.action_space = spaces.Discrete(3)  # Low, Medium, High Priority
        self.task_index = 0
        self.max_tasks = len(df_tasks)

    def reset(self, seed=None, options=None):
        self.task_index = 0
        return self._get_obs(), {}

    def step(self, action):
        if self.task_index >= self.max_tasks:
            return self.reset()

        task = df_tasks.iloc[self.task_index]
        optimal_priority = min(2, int(task["Priority Score"] // (np.max(task["Priority Score"]) / 3)))
        reward = -abs(action - optimal_priority)  # Reward inversely proportional to difference

        self.task_index += 1
        done = self.task_index >= self.max_tasks

        return self._get_obs(), reward, done, False, {}

    def _get_obs(self):
        if self.task_index >= self.max_tasks:
            ret (variable) df_tasks: DataFrame
        task = df_tasks.iloc[self.task_index]
        return np.array([task["Task Complexity"], task["Developer Skill"], task["Estimated Time"], task["Cost"]])

    def render(self, mode='human'):
        pass  # Add visualization if needed

def make_env():
    return MultiAgentTaskEnv()

env = SubprocVecEnv([make_env for _ in range(3)])  # Multi-agent environment

# ============================
# 🚀 3. TRAIN MULTI-AGENT RL MODEL
# ============================
rl_model = PPO("MlpPolicy", env, verbose=1)
rl_model.learn(total_timesteps=100000)
print("✅ Multi-Agent RL Training Completed.")
```

```python
# ============================
# 🤖 2. RL ENVIRONMENT
# ============================
class TaskSchedulingEnv(gym.Env):
    def __init__(self):
        super(TaskSchedulingEnv, self).__init__()
        self.observation_space = spaces.Box(low=0, high=1, shape=(4,), dtype=np.float32)
        self.action_space = spaces.Discrete(3)  # Low, Medium, High Priority
        self.task_index = 0
        self.max_tasks = len(df_tasks)

    def reset(self, seed=None, options=None):
        self.task_index = 0
        return self._get_obs(), {}

    def step(self, action):
        if self.task_index >= self.max_tasks:
            return self.reset()

        task = df_tasks.iloc[self.task_index]
        ideal_priority = min(int(task["Priority Score"] * 3), 2)
        reward = 1 - abs(action - ideal_priority)
        reward -= 0.1 * task["Estimated Time"]
        reward += 0.2 * task["Developer Skill"]
        reward -= 0.05 * task["Cost"]

        self.task_index += 1
        done = self.task_index >= self.max_tasks
        return self._get_obs(), reward, done, False, {}

    def _get_obs(self):
        if self.task_index >= self.max_tasks:
            return np.zeros(4)
        task = df_tasks.iloc[self.task_index]
        return np.array([task["Task Complexity"], task["Developer Skill"], task["Estimated Time"], task["Cost"]])

# Create environment
env = Monitor(TaskSchedulingEnv())
```

```python
# Import necessary libraries
import torch
from transformers import Trainer, TrainingArguments, AutoModelForSequenceClassification, AutoTokenizer,
DefaultDataCollator
import optuna
from huggingface_hub import login
from torch.utils.data import Dataset # Import the Dataset class

# Log into Hugging Face (Replace with your actual token)
login(token="hf_iLV[redacted]")

# Use a lightweight model instead of LLaMA-2
model_name = "distilbert-base-uncased"  # Fast & lightweight (~66M parameters)

# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2) # Change model loading here and
define num_labels

# Define a custom dataset class
class OpenStackLogDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
```

```python
LOG_URL = "https://f6e43f00df489c814215-87141d8cdbf10530595552debffaf82b.ssl.cf2.rackcdn.com/opendev-prod-hourly/
opendev.org/opendev/system-config/master/infra-prod-bootstrap-bridge/d96e828/job-output.json"

def extract_zuul_data(log_url):
    response = requests.get(log_url)

    if response.status_code == 200:
        log_data = response.json()

        jobs = []

        # Iterate over each job entry in the JSON
        for job in log_data:
            branch = job.get("branch", "unknown")
            phase = job.get("phase", "unknown")
            playbook = job.get("playbook", "unknown")

            for play_entry in job.get("plays", []):
                play_name = play_entry["play"].get("name", "unknown")
                start_time = play_entry["play"]["duration"].get("start", "unknown")
                end_time = play_entry["play"]["duration"].get("end", "unknown")

                # Extract tasks under each playbook execution
                for task in play_entry.get("tasks", []):
                    for host, task_data in task["hosts"].items():
                        action = task_data.get("action", "unknown")
                        os_version = task_data.get("ansible_facts", {}).get("ansible_distribution_version", "unknown")
                        arch = task_data.get("ansible_facts", {}).get("ansible_architecture", "unknown")
                        timestamp = task_data.get("ansible_facts", {}).get("ansible_date_time", {}).get("epoch",
                        "unknown")
```

```
Step    Training Loss
[codecarbon INFO @ 03:44:12] Energy consumed for RAM : 0.000950 kWh. RAM Power : 4.7530388832092285 W
[codecarbon INFO @ 03:44:12] Energy consumed for all CPUs : 0.008500 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 03:44:12] 0.009451 kWh of electricity used since the beginning.
[codecarbon INFO @ 03:44:12] 0.001821 g.CO2eq/s mean an estimation of 57.41670484802263 kg.CO2eq/year
[codecarbon INFO @ 03:44:30] Energy consumed for RAM : 0.000974 kWh. RAM Power : 4.7530388832092285 W
[codecarbon INFO @ 03:44:30] Energy consumed for all CPUs : 0.008712 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 03:44:30] 0.009686 kWh of electricity used since the beginning.
[codecarbon WARNING @ 03:44:33] Another instance of codecarbon is already running. Exiting.
[I 2025-03-01 03:44:33,116] A new study created in memory with name: no-name-2dfd7d4c-aa4e-4854-bea6-1689b18c75e2
<ipython-input-13-b9481bf54b49>:76: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/
v3.0.0. Use suggest_float(..., log=True) instead.
  lr = trial.suggest_loguniform('lr', 1e-5, 1e-3)
[I 2025-03-01 03:44:33,227] Trial 0 finished with value: -0.0009584832536018757 and parameters: {'lr': 9.041516746398125e-05, 'batch_size': 4}. Best is trial 0 with value: -0.0009584832536018757.
[I 2025-03-01 03:44:33,230] Trial 1 finished with value: -0.0029085656688640706 and parameters: {'lr': 7.09143433113593e-05, 'batch_size': 2}. Best is trial 0 with value: -0.0009584832536018757.
[I 2025-03-01 03:44:33,235] Trial 2 finished with value: -0.007719088486409932 and parameters: {'lr': 2.2809115135900683e-05, 'batch_size': 2}. Best is trial 0 with value: -0.0009584832536018757.
[I 2025-03-01 03:44:33,241] Trial 3 finished with value: -0.002920492164658346 and parameters: {'lr': 7.079507835341654e-05, 'batch_size': 2}. Best is trial 0 with value: -0.0009584832536018757.
[I 2025-03-01 03:44:33,244] Trial 4 finished with value: -0.01944419104272195 and parameters: {'lr': 0.0002944419104272195, 'batch_size': 2}. Best is trial 0 with value: -0.0009584832536018757.
Best Hyperparameters: {'lr': 9.041516746398125e-05, 'batch_size': 4}
```

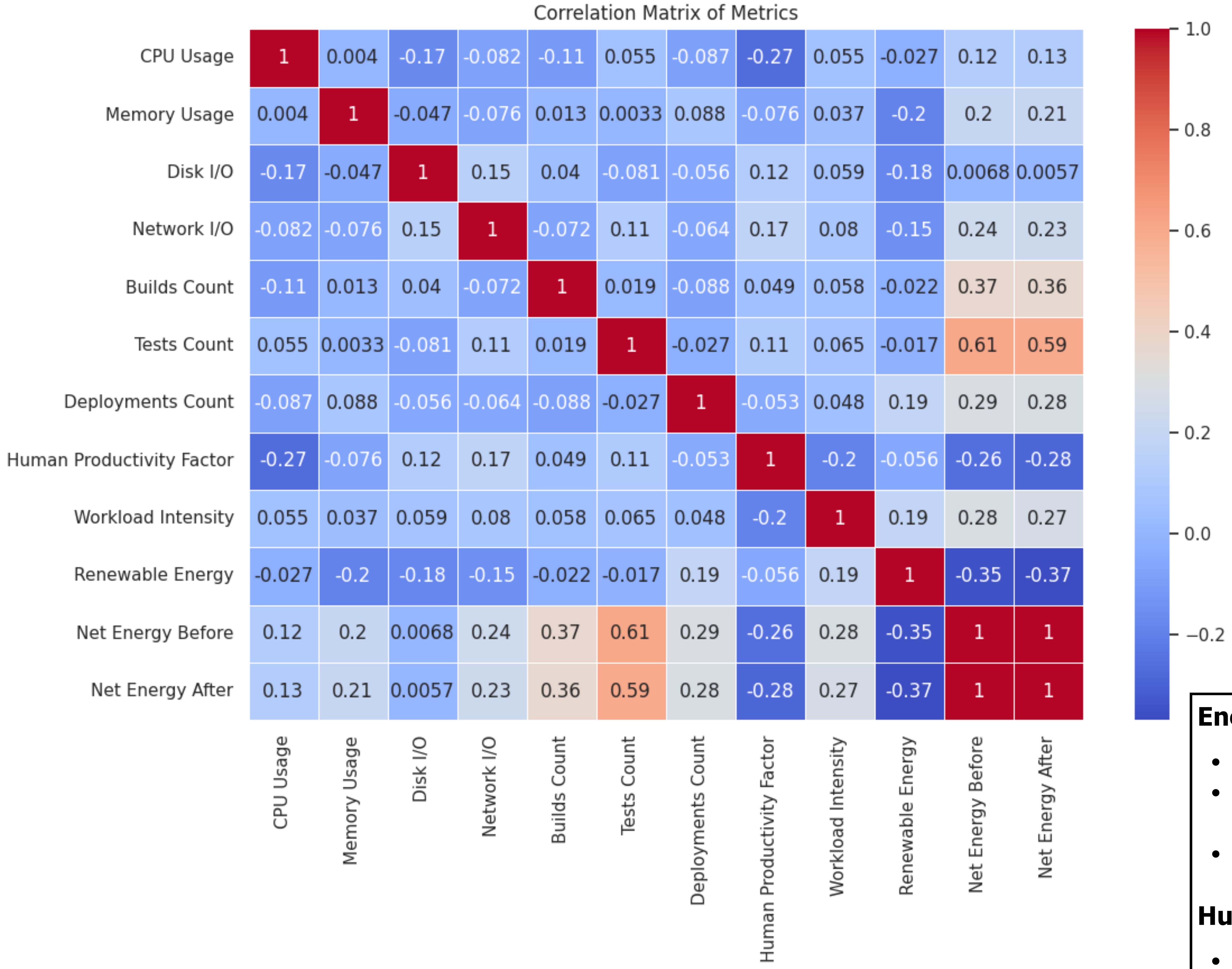
Training and Fine-tuning our models

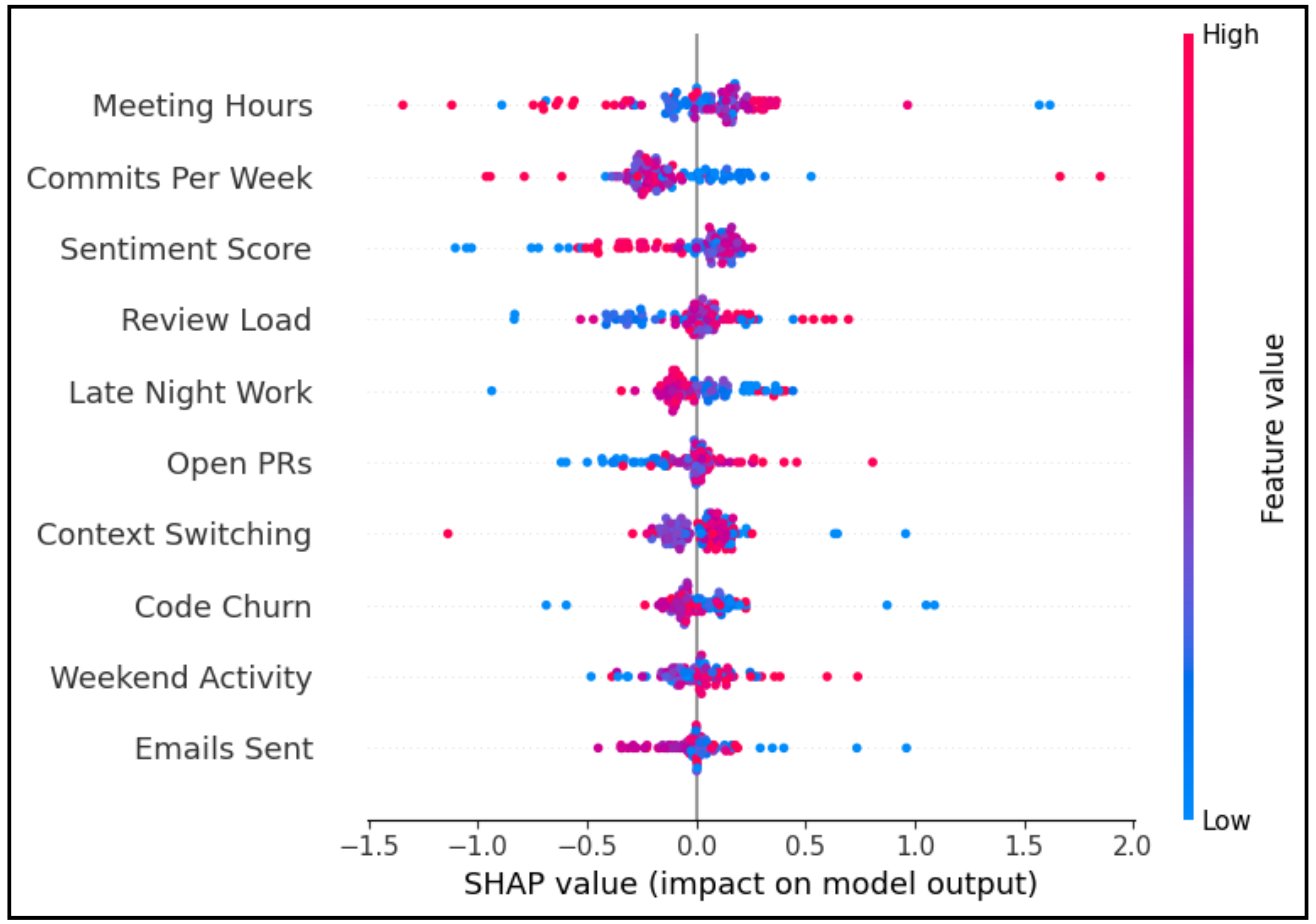
>>

```python
# ============================
# 📅 4. LLM-POWERED PLANNING ASSISTANT
# ============================
def generate_planning_assistant(task_description):
    prompt = f"""
    You are an AI-driven software planning assistant. Given the following task description:
    {task_description}
    Suggest an optimal developer allocation, priority level, and estimated effort for completion.
    Format your response in JSON format.
    """
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt}]
    )

    return response.choices[0].message.content
# Example Task
software_task = "Develop a cloud-native CI/CD pipeline for OpenStack contributions."
planning_output = generate_planning_assistant(software_task)
print("📌 GPT-4 Planning Assistant Output:", planning_output)

# ============================
# 🔬 5. OPTIMIZE RL POLICY USING OPTUNA
# ============================
def objective(trial):
    n_steps = trial.suggest_int("n_steps", 512, 4096)
    learning_rate = trial.suggest_float("lr", 1e-5, 1e-2, log=True)  # Fixed deprecation warning
    gamma = trial.suggest_float("gamma", 0.8, 0.99)  # Fixed deprecation warning

    model = PPO("MlpPolicy", env, n_steps=n_steps, learning_rate=learning_rate, gamma=gamma, verbose=0)
    model.learn(total_timesteps=50000)
    return -np.mean(model.predict(env.reset()[0])[0])  # Maximize reward

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)

print("🎯 Best RL Hyperparameters:", study.best_params)
```


```
✅ Multi-Agent RL Training Completed.
📌 GPT-4 Planning Assistant Output: {
  "task": {
    "description": "Develop a cloud-native CI/CD pipeline for OpenStack contributions",
    "components": [
      {
        "component": "Cloud-native architecture design and planning",
        "developers_allocated": 2,
        "skill_required": "high",
        "estimated_effort_in_days": 10
      },
      {
        "component": "OpenStack integration with cloud-native environment",
        "developers_allocated": 3,
        "skill_required": "high",
        "estimated_effort_in_days": 12
      },
      {
        "component": "CI/CD pipeline design",
        "developers_allocated": 2,
        "skill_required": "high",
        "estimated_effort_in_days": 8
      },
      {
        "component": "Pipeline testing and fine-tuning",
        "developers_allocated": 2,
        "skill_required": "medium",
        "estimated_effort_in_days": 6
      },
      {
        "component": "Documentation",
        "developers_allocated": 1,
        "skill_required": "medium",
        "estimated_effort_in_days": 3
      }
    ],
    "total_developers_allocated": 10,
    "total_estimated_effort_in_days": 39,
    "priority": "high"
  }
```

>> Feature-Impact Analysis from Correlation to SHAP





Energy Efficiency

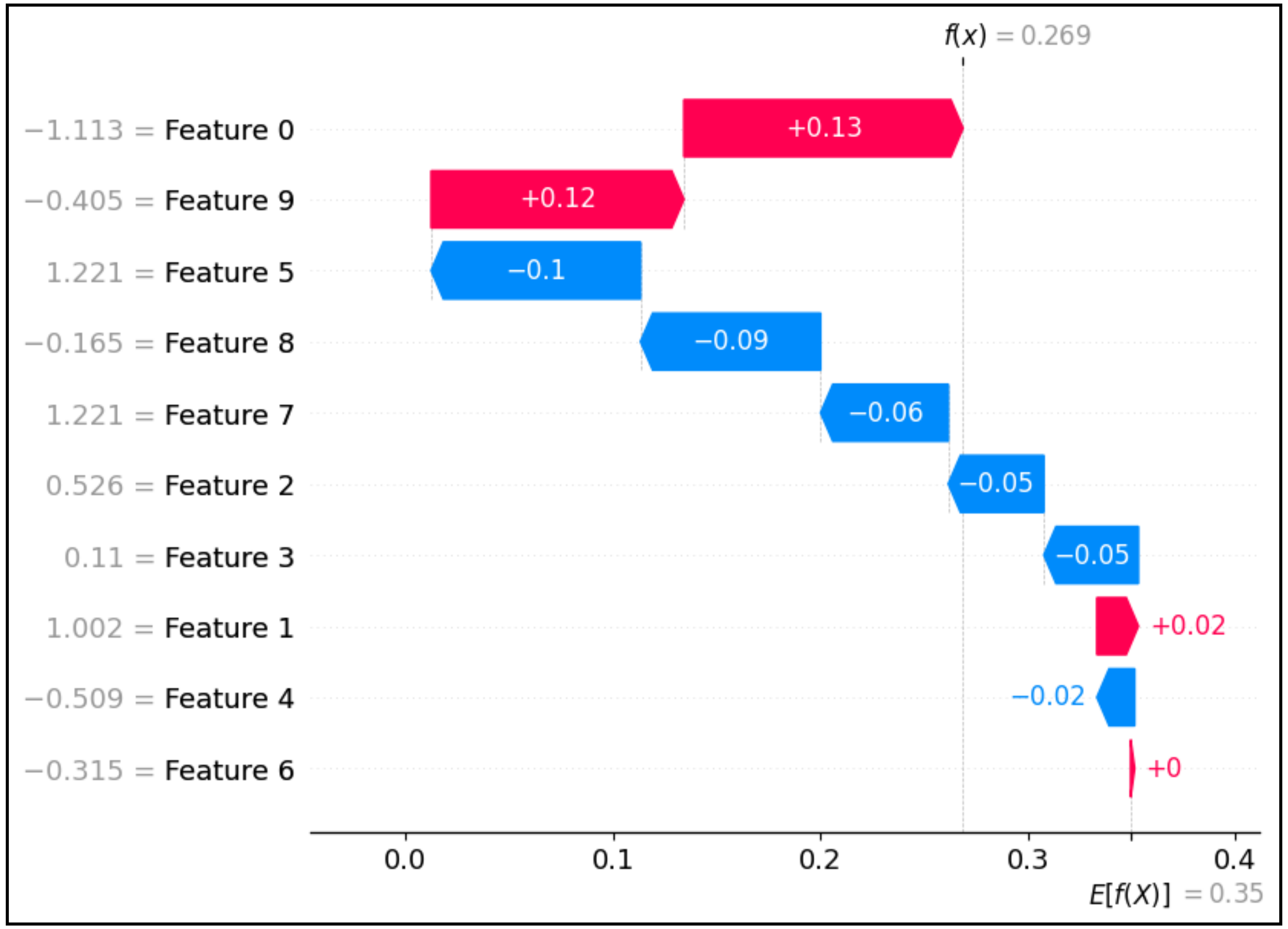
- More **tests, builds, and deployments** significantly **increase energy consumption**.
- Higher **CPU and memory usage** also **slightly contribute to higher energy consumption**.
- Using **renewable energy strongly reduces net energy consumption**.

Human Productivity

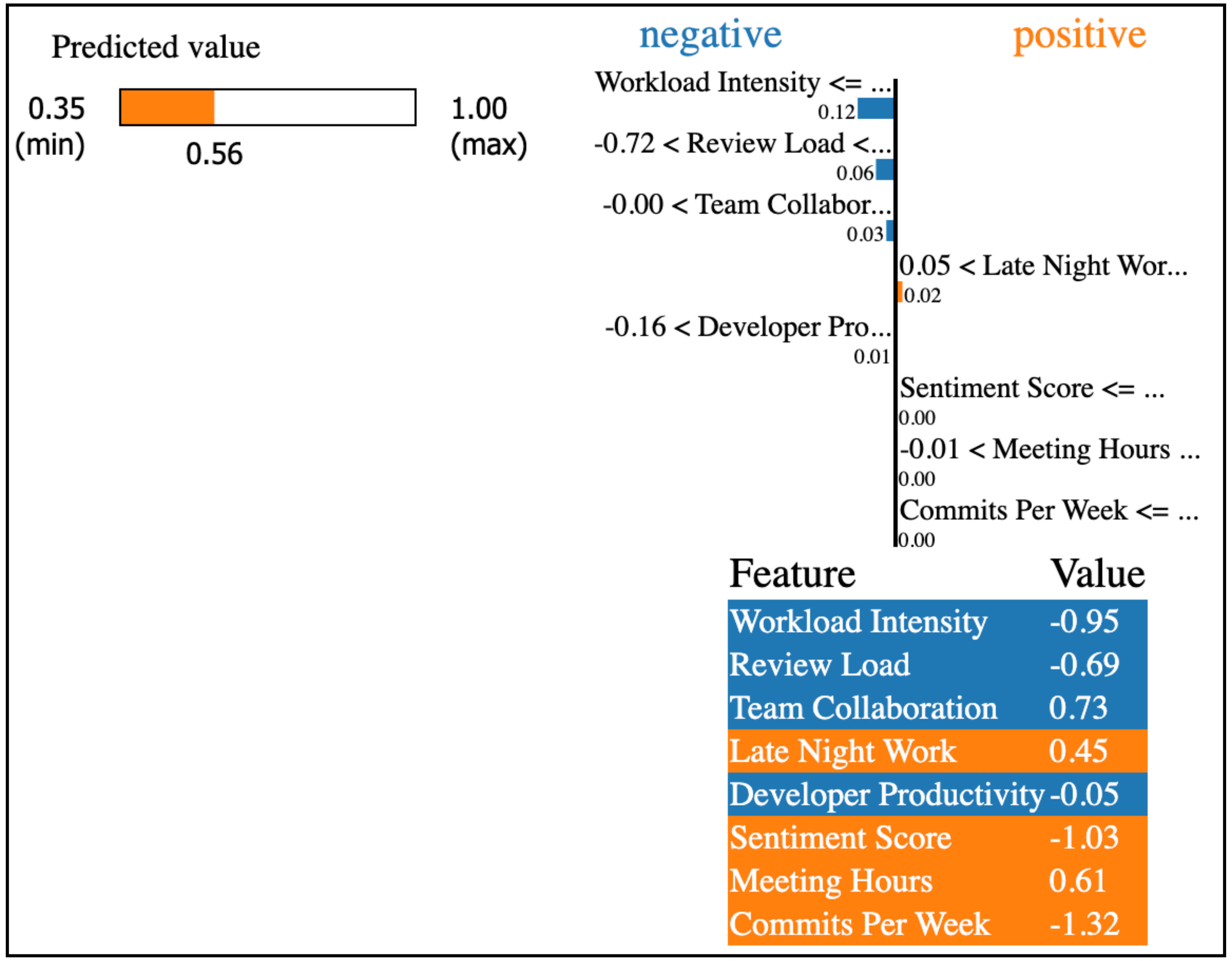
- **Heavy workloads negatively impact human productivity** (-0.27 correlation).
- **Efficient network usage improves productivity** (0.17 correlation).
- **Increased deployments & builds do not necessarily boost productivity**.
-

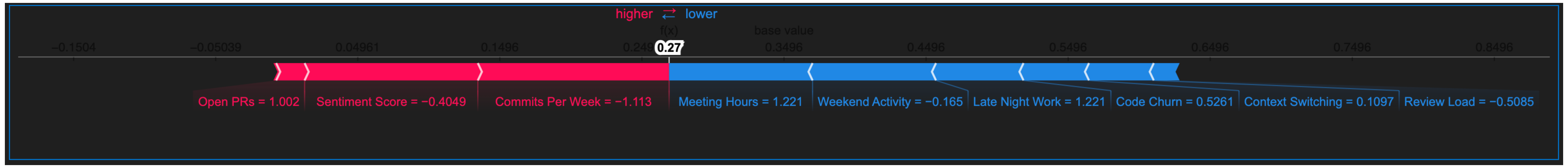
>> Feature-Impact Analysis from Correlation to SHAP and LIME

Shapley Additive Explanations



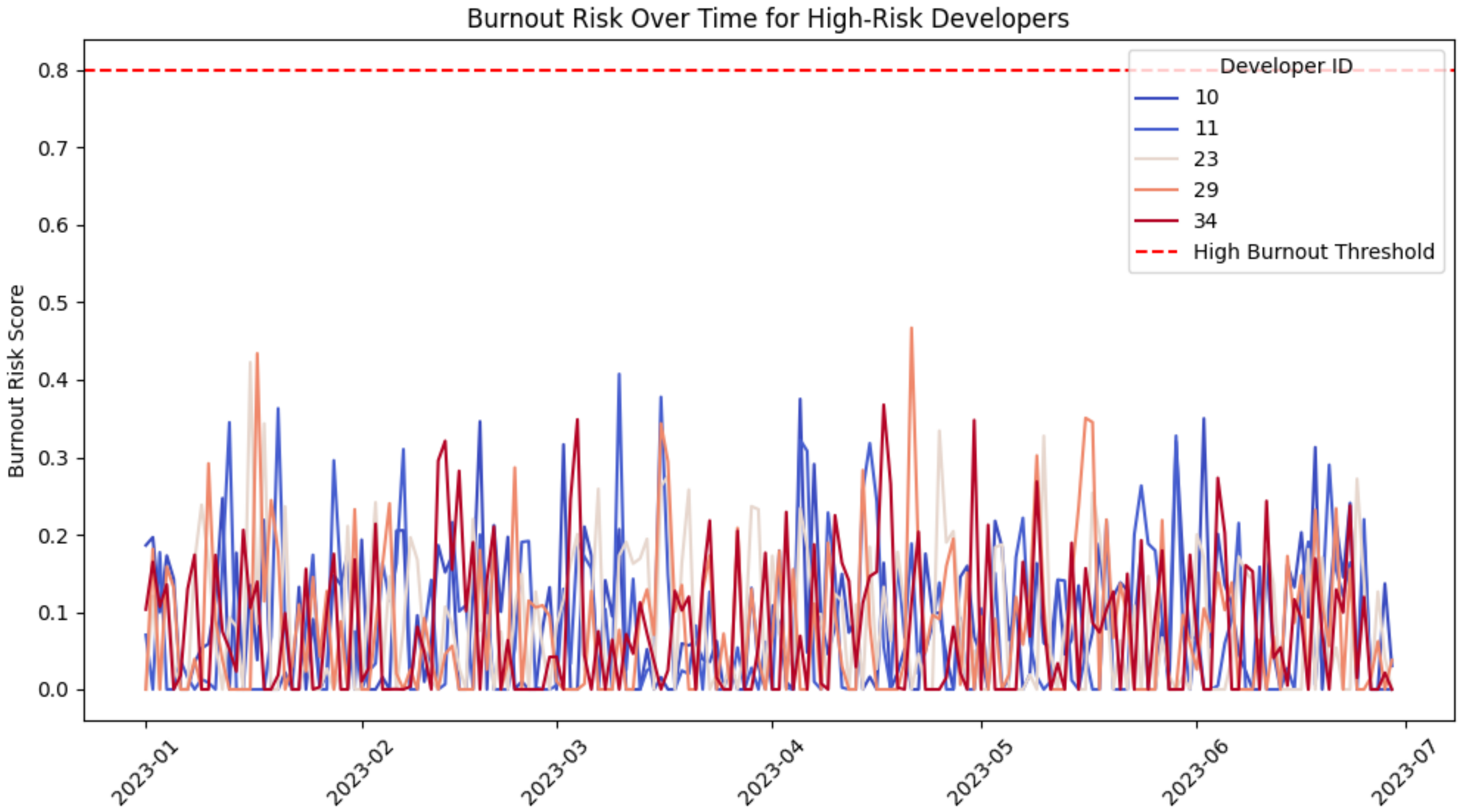
Local Interpretable Model-Agnostic Explanations



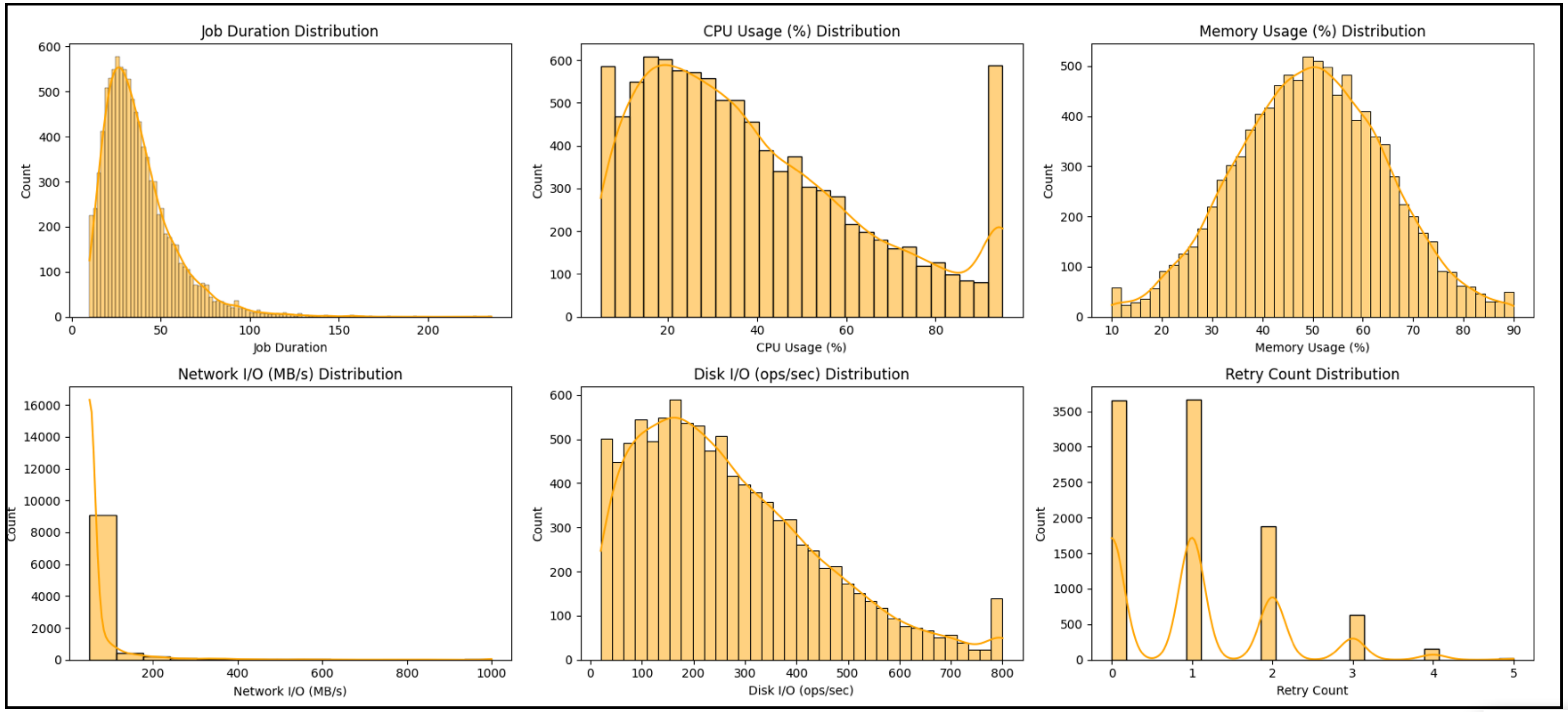


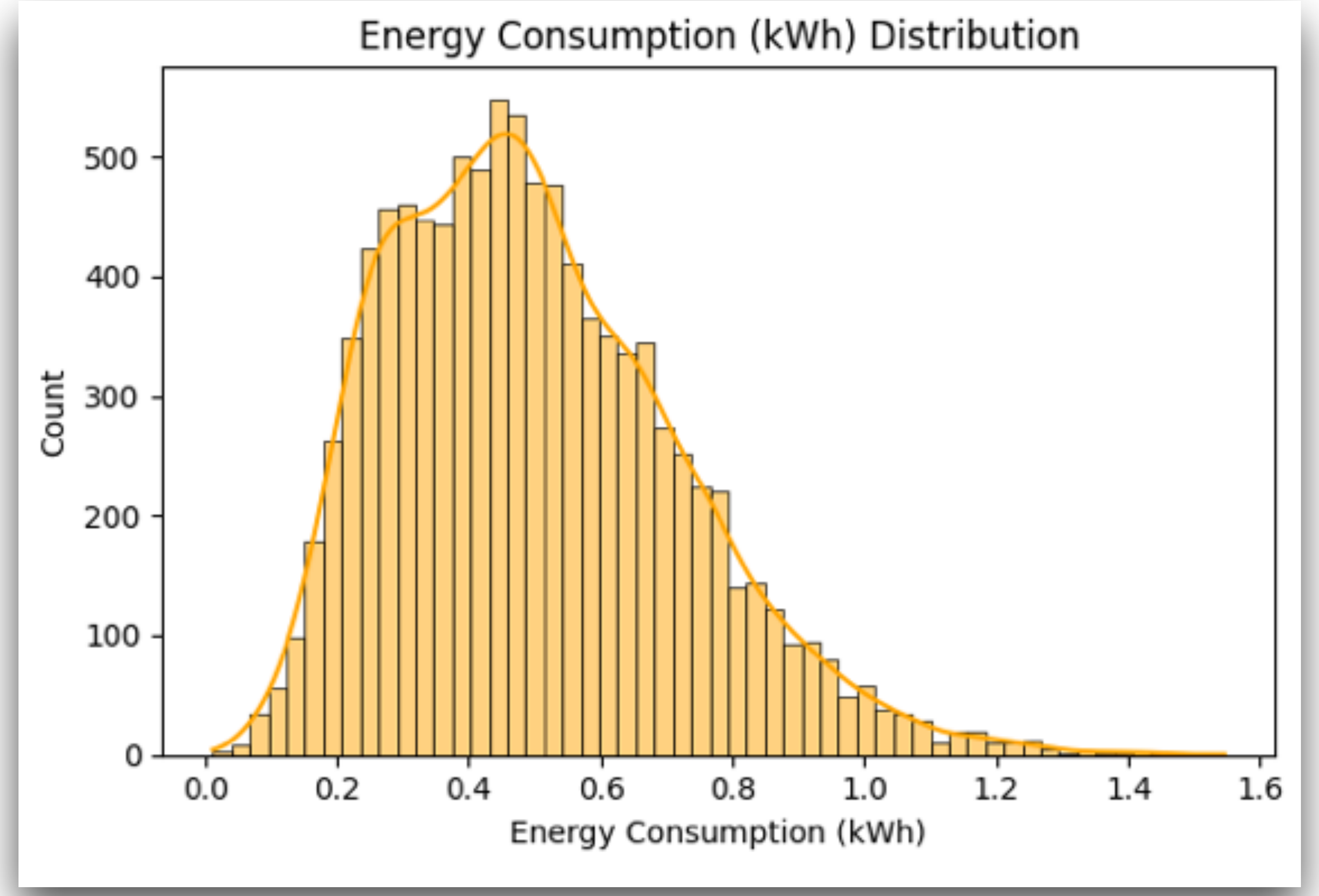
Predicting Burnout in Open-Source communities Based on Socio-Technical Indicators.

BURNOUT
in Software
Development

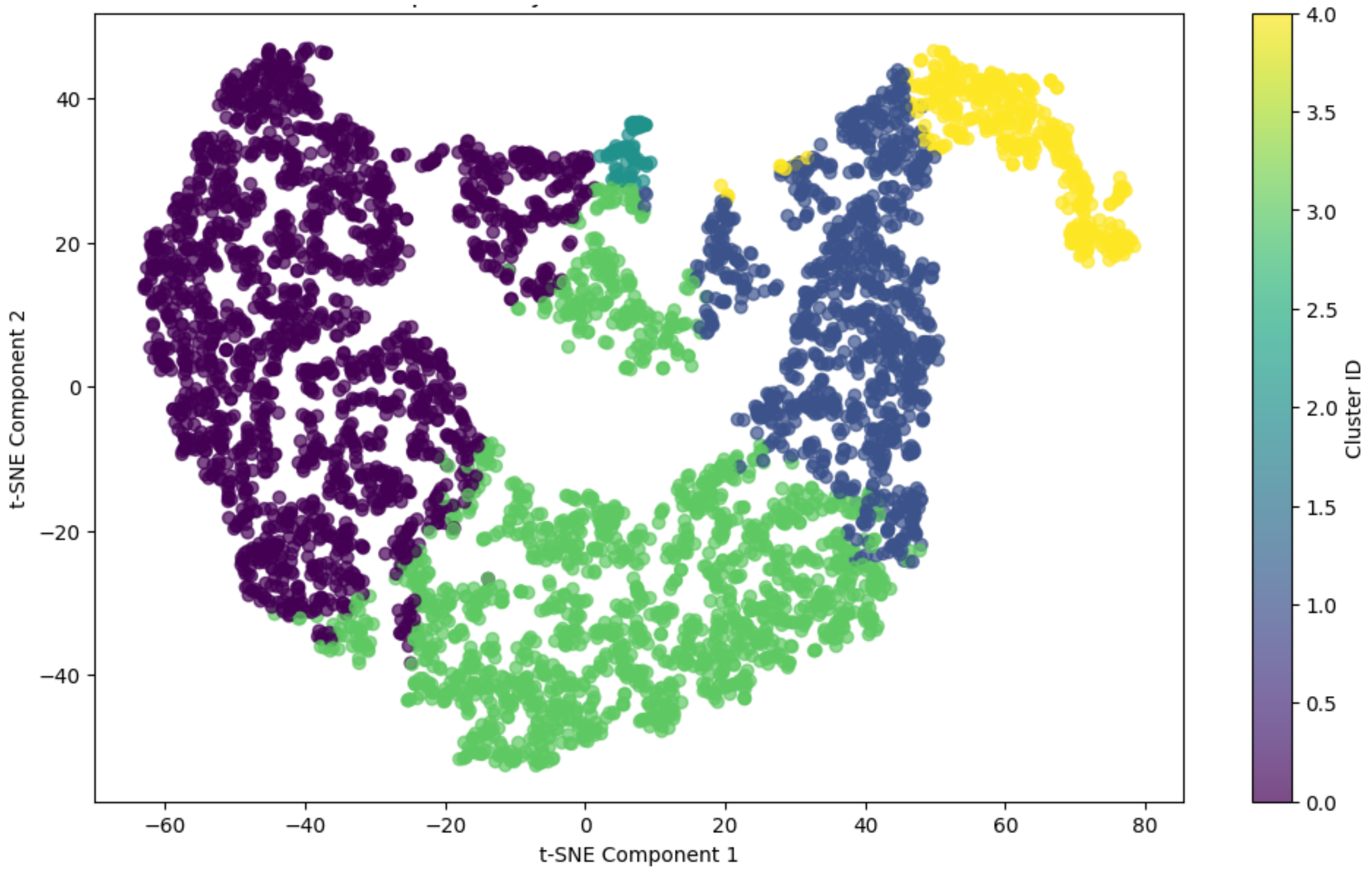


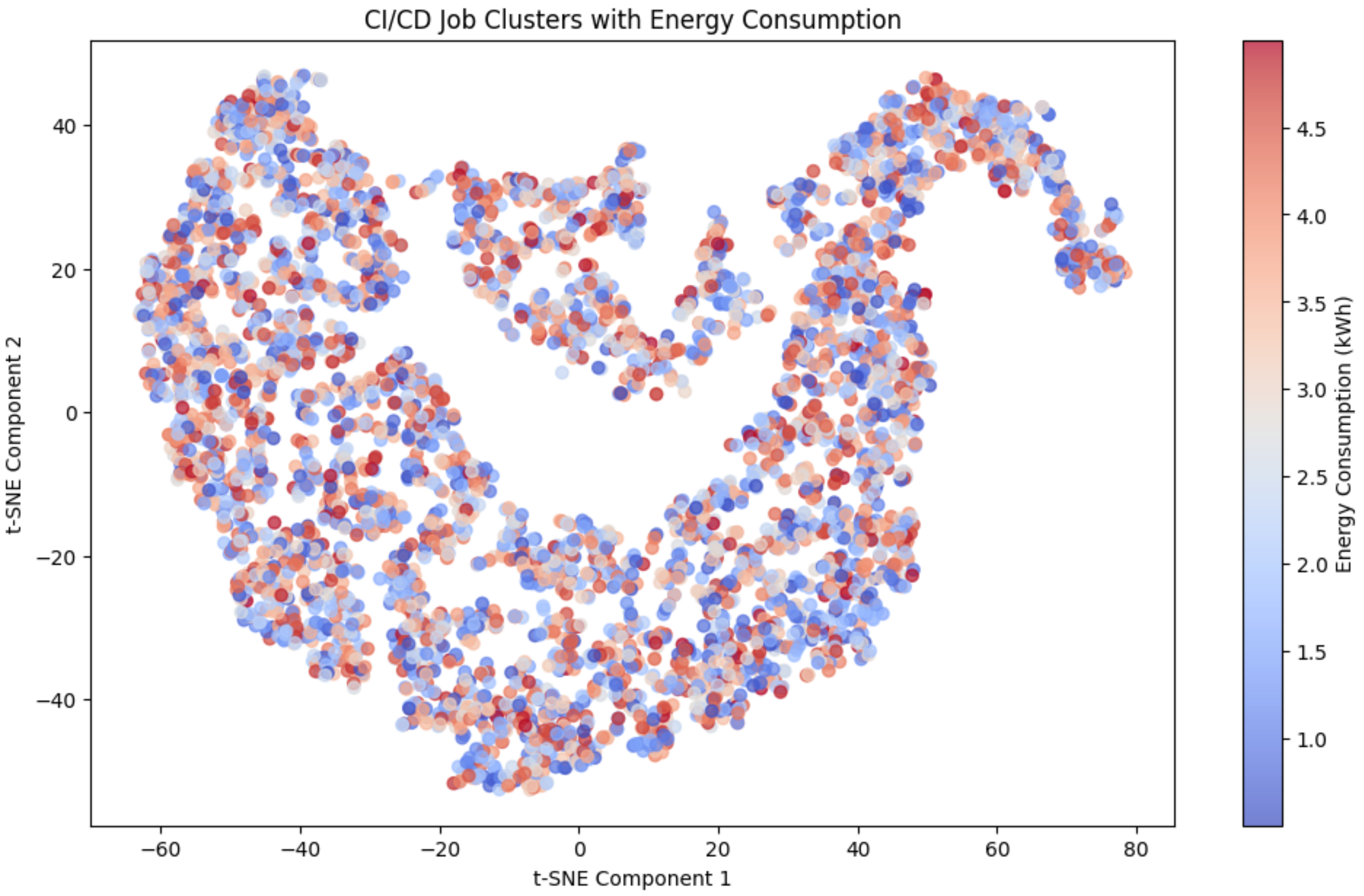
>> The underline representation of features and relationships to energy consumption





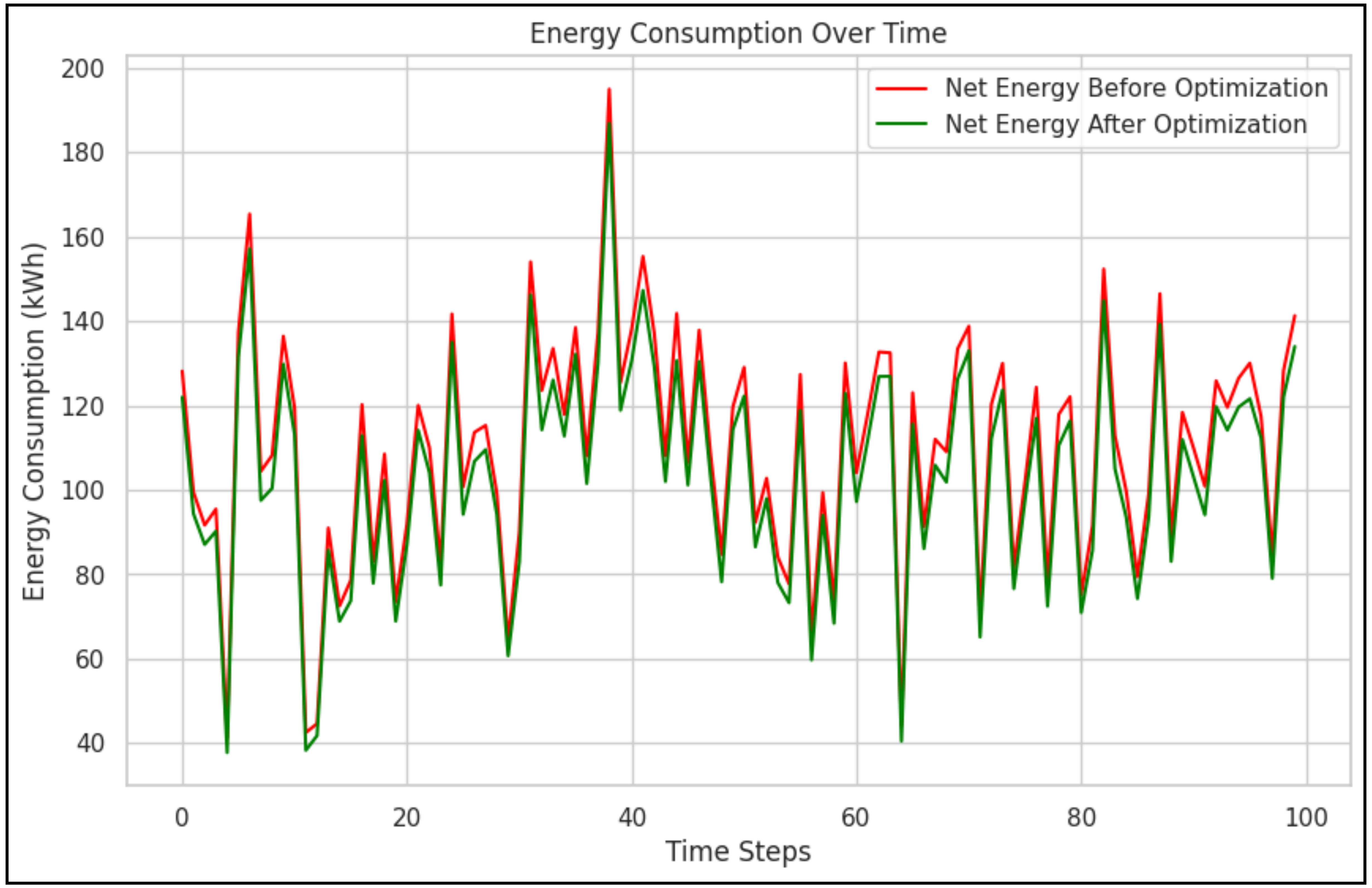
>> Dimensionality reduction to latent space

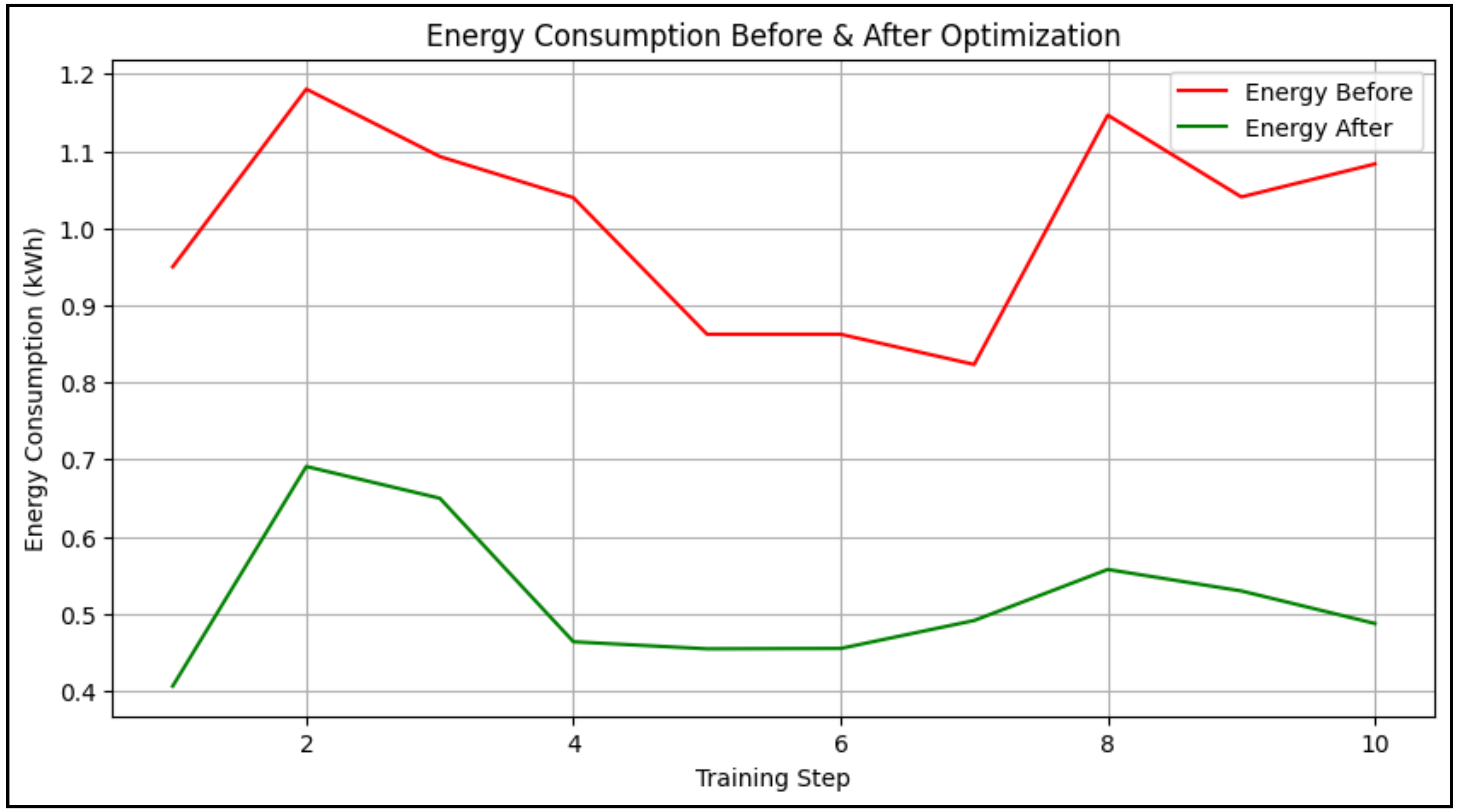




1. **Graph Representation of CI/CD Jobs**:
   - Jobs are treated as **nodes** in a **directed graph (DiGraph)**.
   - Nodes are connected by **retry dependencies** (if a job failed and retried).
   - Each node is assigned **features**, such as:
     - **Job duration**
     - **CPU usage**
     - **Memory usage**
     - **Network & Disk I/O**
     - **Retry counts**
     - **Energy consumption (kWh)**
2. **Graph Neural Network (GNN) Training**:
   - A **Graph Convolutional Network (GCN)** and **Graph Attention Network (GAT)** are used to learn **node embeddings**.
   - The model is trained to **predict CI/CD failures** based on job characteristics.
3. **Dimensionality Reduction using t-SNE**:
   - The **high-dimensional embeddings** from the GNN are projected into **2D space** using **t-SNE**.
   - This helps in **visualizing job clusters** and **identifying patterns in energy consumption**.

Energy optimization





Question?

OpenInfra SUMMIT > VANCOUVER '23