AIM-BASE: AI INTEGRATED MODEL TO PREDICT THE ENERGY CONSUMPTION OF EXTRA-TERRESTRIAL COLONIES

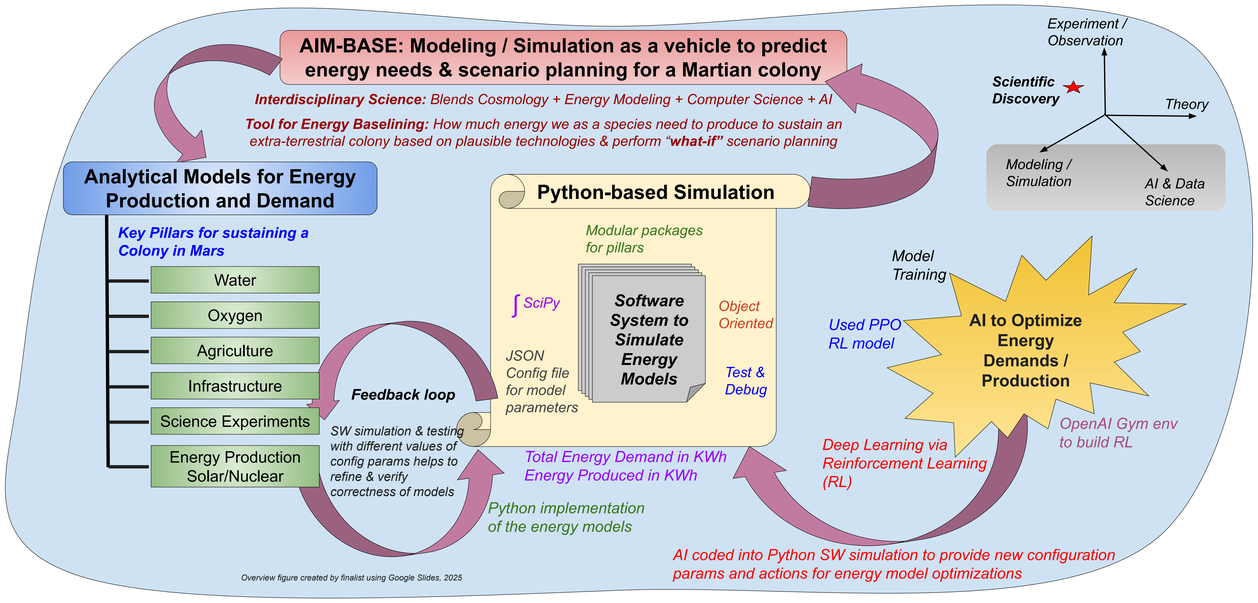
Grand Challenge Problem Facing Humanity

Context: Creating and sustaining extra-terrestrial colonies is the next frontier of human exploration.

Motivation: Before we embark on that, we need to fundamentally understand the ***baseline energy requirements*** for a human space colony.
- ❖ How can we predict and optimize energy requirements?
- ❖ Provide policy makers / mission planners a *what-if* scenario tool

Approach: AIM-BASE integrates advanced analytical modeling, Python based computer simulations, and AI reinforcement learning driven optimizations to develop efficient & adaptable resource management systems for a Mars colony.

Scientific Merit:
- ❖ Models the dynamic energy demands and constraints of Mars' harsh environment and varying conditions.
- ❖ Focus on understanding the energy needs of life sustaining critical systems & techniques.
- ❖ Explores innovative base designs and technologies, emphasizing energy-efficient solutions for oxygen generation, water production, and thermal regulation of the habitat.
- ❖ Its comprehensive simulations empower users to conduct "what-if" scenario planning, optimize infrastructure, and ***evaluate energy mixes*** for long-term viability.



Python Software Systems for Simulation of Models

Modular Python Packages for Key Pillars

```
AIM-BASE/ (~ 2500 lines of Python code)
|___EnergyModeling.py (main)
|___Config.json (> 300 parameters for models)
|___ColonyPillars/ (packages to import)
      |___WaterModels.py
      |___OxygenModels.py
      |___EnergyProductionModels.py
      |___InfraModels.py
      |___AgriModels.py
      |___SciExptsModels.py
|___AI-Suggestions/ (packages to import)
      |___AIWater.py
      |___AIOxygen.py
      |___AIinfra.py
      |___AIenergyproduction.py
      |___AIsciexpts.py
```

Scientific Python: Quad Integral method for Habitat Temp. Control

```
def calculate_thermal_energy(params):
    E_thermal_total, _ = quad(
        lambda a: (
            calculate_heat_flux_habitat(params) +
            calculate_greenhouse_temperature_energy(params)
        ), 0, A * delta_t)          # A is area of habitat
    return E_thermal_total  # delta_t is duration of heating
```

Creating a function on the fly.

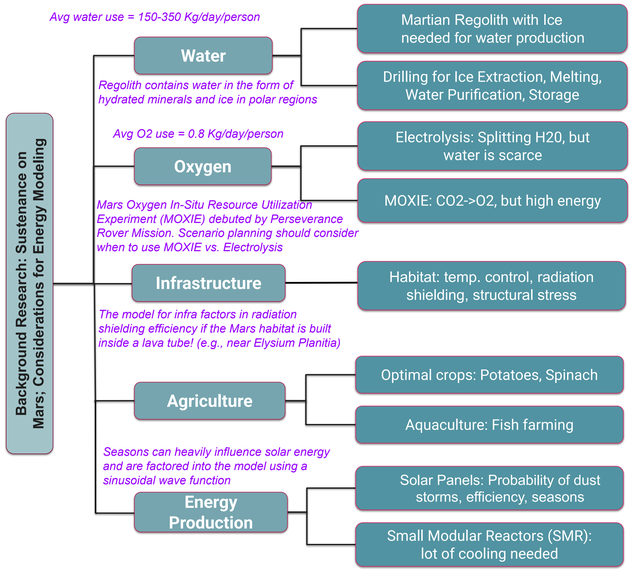
JSON Config Parameter Example (> 300 params for all models)

```
"P_drill": 1.0,  // Power used for drilling (in kW)
"M_CO2": 1000,  // Mass of CO2 processed by MOXIE (kg)
"M_water_electrolysis": 500,  // Mass of water consumed in electrolysis (kg)
"P_filtration": 0.05,  // Power required for filtration (in kW)
"P_SMR": [100000000000, 180000000, 10500000],  // Power outputs of the SMRs (in kW)
```

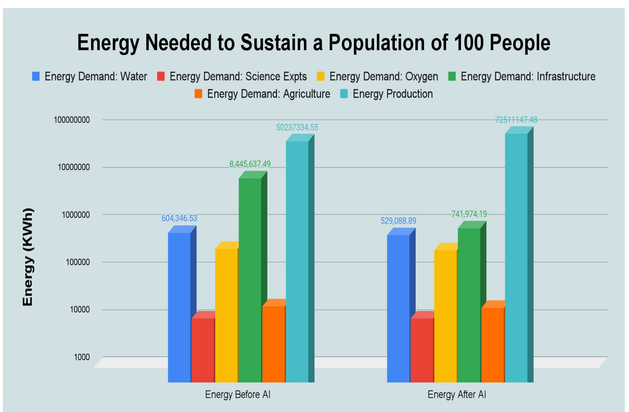
Python Code for Moxie vs. Electrolysis use Scenario in Oxygen Energy Demand

```
def transition_moxie_electrolysis(params):
    W_low = params['W_low']  # water level for low reserves
    E_high = params['E_high']  # surplus energy for high energy
    E_moderate = params['E_moderate']  # surplus energy for moderate
    T_shift_t_prev_o2 = params['T_shift_t_prev_o2']  # Transition factor
    delta_t = params['delta_t']  # Time step duration
    kappa_o2 = params['kappa_o2_transition']  # Transition rate
    if calculate_water_available(params) < W_low and
calculate_energy_surplus(params) > E_high:
        print("Using MOXIE. Low water levels and energy surplus")
        return 1  # Full MOXIE
    elif params['W_available'] > W_moderate and params['E_surplus'] <
E_moderate:
        print("Using Electrolysis for O2")
        return 0  # Full electrolysis
    else:
        print(" Using both MOXIE and Electrolysis for O2")
        return T_shift_t_prev_o2 + (delta_t * kappa_o2)
def calculate_moxie_energy(params, T_shift_t):
    M_CO2 = params['M_CO2']  # Mass of CO2 processed
    kWh_per_kg_O2 = params['kWh_per_kg_O2']
    E_moxie = M_CO2 * kWh_per_kg_O2 * T_shift_t /
params['eta_moxie']
    return E_moxie
```

Technologies / Artefacts / Considerations for a life sustaining colony & used in analytical modeling



Results: Evaluation of Energy Models with its Parameter Space, Python Simulation SW & AI Optimizations



Graph created by finalist using Google Sheets, 2025

Oxygen Production Energy Demand Models & What-if Scenarios Analysis

Energy for Moxie O2 production

$$E_{\text{moxie}} = \frac{M_{\text{CO2}} \cdot \text{kWh}_{\text{per\_kg\_O2}} \cdot T_{\text{shift}}(t)}{\eta_{\text{moxie}}}$$

Factors in the mass of CO2 processed (M_CO2), the energy needed per Kg of Oxygen produced (KWh/Kg_O2), a time-dependent transition factor (T_shift(t)), and the efficiency of the system.

Energy for Electrolysis O2 production

$$E_{\text{electrolysis}} = \frac{M_{\text{water\_electrolysis}} \cdot \text{kWh}_{\text{per\_kg\_O2}} \cdot (1 - T_{\text{shift}}(t))}{\eta_{\text{electrolysis}}}$$

The mass of water times energy per kg of O2 times percent of O2 from electrolysis divided by the efficiency of the system.

Transition Between MOXIE and electrolysis

$$T_{\text{shift}} = \begin{cases} 1 & \text{if } W_{\text{available}} < W_{\text{low}} \text{ and } E_{\text{surplus}} > E_{\text{high}} \\ 0 & \text{if } W_{\text{available}} > W_{\text{moderate}} \text{ and } E_{\text{surplus}} < E_{\text{moderate}} \\ T_{\text{shift, prev}} + (\delta_t \cdot \kappa_{o2}) & \text{otherwise} \end{cases}$$

The model determines the transition factor T_shift for Oxygen production. It outputs 1 for full MOXIE usage when water is low and energy is abundant, 0 for full electrolysis when water is sufficient and energy is limited, and adjusts incrementally otherwise.

Solar/Nuclear Energy Production Models & What-if Scenario Analysis

Solar Energy production

$$P_{\text{solar\_t}} = P_{\text{peak}} \cdot f_{\text{season}} \cdot (1 - P_{\text{duststorm}} \cdot \delta_{\text{dust}}) \cdot \eta_{\text{dustremoval}}$$

The solar energy production P_solar_t is calculated as the product of the solar peak energy, seasonal adjustment factor, and the efficiency of dust removal, adjusted for the reduction caused by dust storms. This model accounts for environmental conditions such as dust storm probability and seasonal variations.

Nuclear Energy production

$$P_{\text{nuclear}} = \sum_{i=1}^{n} P_{\text{SMR},i} \cdot \eta_{\text{SMR},i}$$

Nuclear energy is the sum of each reactor's power output multiplied by its efficiency.

Transition between Solar and Nuclear

$$T_{\text{solar},t} = \begin{cases} 1, & \text{if } P_{\text{duststorm},t} < \phi_{\text{threshold}} \text{ and } f_{\text{season},t} > f_{\min} \\ 0, & \text{if } P_{\text{duststorm},t} > \phi_{\text{threshold}} \text{ or } f_{\text{season},t} \le f_{\min} \\ T_{\text{solar},t,\text{prev}} + \kappa \cdot \frac{P_{\text{battery},t}}{P_{\text{total demand}}} \cdot f_{\text{season},t}, & \text{otherwise} \end{cases}$$

The equation decides solar energy usage: 1 for full solar, 0 for no solar and full nuclear, or a mix of two based on battery power and demand.

Seasonal Amplification factor model of solar energy

$$f_{\text{season}} = 1 + 0.2 \cdot \sin\left(\frac{2\pi \cdot (\text{t}_{\text{season}} - T_{\max\ \text{sunlight}})}{687}\right)$$

The seasonal adjustment factor is calculated as 1 plus 0.2 times the sine of a term representing sunlight variation over a Martian year.

Water Production Energy Demand Models & What-if Scenario Analysis

Energy for Regolith (Ice) Drilling

$$E_{\text{drill}} = \sum_{j=1}^{k} \int_0^d \left[\frac{P_{\text{drill}} \cdot \rho_0 \cdot \ln\left(1 + \delta_0 \cdot e^{\ k_2 \cdot t_{\text{drilling}}}\right)}{\eta_0 \cdot e^{-(\alpha x + \beta x + \gamma_{\text{wear}})}} \cdot \left(1 + P_0 \cdot e^{-0.2x} \cdot (1 + \sin(\omega x))\right) \cdot \left(R_0 + \beta x^{1.5}\right) \cdot h(x) \cdot f_T \cdot (1 - \phi(x)) \right] dx$$

The total drilling energy is determined by summing contributions from multiple drills, accounting for depth, power, resistance, efficiency decay, material properties, and environmental modifiers. Key factors include dynamic and cyclical power adjustments, material porosity, and corrections for depth and wear.

Energy for Ice melting

$$E_{\text{melt}} = M_{\text{total water}} \cdot \left(\int_{T_{\text{Initial}}}^{T(x)} c_{\text{ice}}(T)\, dT + L_{\text{ice}} \right)$$

Calculates the energy required to melt ice (E_melt) using the total mass of water (M_total_water), the integral of the specific heat capacity of ice (c_ice(T)) over the temperature range (T_initial to T(x)), and the latent heat of fusion of ice (L_ice).

Energy for water purification

$$E_{\text{purification}} = \int_0^V \left[P_{\text{filtration}} \cdot (1 + \phi_{\text{clog}} \cdot (1 - e^{-\lambda t})) + \frac{M_{\text{water}} \cdot L_{\text{vaporization}}}{\eta_{\text{distillation}}} + (P_{\text{UV}} + P_{\text{chemical}}) \right] dV$$

Calculates the total energy for water purification by integrating filtration, distillation, and additional treatment energy over the total volume. It accounts for clogging, vaporization efficiency, and UV/chemical treatments.

Energy Model for water demand

$$W_{\text{produced}} + W_{\text{stored}} \geq W_{\text{demand}}$$

The variables describe water availability and usage: W_produced represents daily generated or extracted water, W_stored accounts for reserves in storage systems, and W_demand is the total daily water required.

AI to Optimize Models' Energy Production and Demand

AI Code Snippet: Highlights heart of RL. ***observe→act→learn→improve***

1 Environment Setup: Martian colony's energy model systems where AI learns.
env = EnergyProductionEnv(params, P_total_demand) *# Setup custom environment with OpenAI Gym for the energy production pillar model with its params; Configure high/low values for env's state, captured by its key params, thereby creating an observation space.*

2 Define the RL PPO Model: The brain that makes decisions to improve energy efficiency.
from stable_baselines3 import PPO
model = PPO("MlpPolicy", env) *# PPO algorithm for decision-making*

3 Training Process: The AI tests different strategies to learn what works best.
model.learn(total_timesteps=50000) *# AI learns by trial & error to maximize energy output*

Actions & Rewards: AI tries an action, gets feedback (reward), and adjusts.

4 Making Decisions (Policy Inference)
obs = env.reset() # resets state parameters in the observation object
action, _ = model.predict(obs) *# AI picks the best action based on what it has learned*

5 Feedback Loop (Reward System)
obs, reward, done, info = env.feedbackLoop(action) *# AI receives feedback (reward) to improve future decisions. Continually updates params in observation space to realize action*

```
def feedbackLoop(self, action):
    # Apply actions to modify params dynamically
    if action == 0:  # Enhance battery capacity by 20%
        self.params['E_stored'] = min(self.params['E_stored'] * 1.2, 1000000)
    elif action == 1:  # Increase solar panel efficiency by 10%
        self.params['eta_panel'] = min(self.params['eta_panel'] * 1.1, 1.0)
    reward = updated_energy_production - self.initial_energy_production
```

Example AI suggestions explored across Martian Energy Model Environments
- **AI Water:** Implement precision irrigation schedules and adopt drip irrigation systems to improve water distribution efficiency and reduce waste by up to 30%. **Energy Saved: 245 KWh** *(from the AI implementation with params)*
- **AI Oxygen:** Implement adaptive MOXIE CO2 intake schedules based on colony activity cycles, reducing overprocessing and saving up to 15% energy. **Energy Saved: 39632 KWh**
- **AI Agri:** Improve nutrient delivery efficiency with advanced micro-irrigation, increasing efficiency by 10%. **Energy Saved: 630 KWh**
- **AI Infra:** Optimize internal habitat temperature settings to decrease heating/cooling energy by 2K. **Energy Saved: 26 KWh**

Why AI?
Efficient & powerful way to optimize a dynamic & complex environment by monitoring usage
Tweak both energy supply & demand models

Optimizing What?
Config Parameters used by the energy models to represent physical processes (e.g., drilling, solar energy production)
Achieve desired outcomes (e.g., enhance lighting efficiency in the habitat)

How is AI Applied?
Reinforcement Learning (RL): Energy models' environments captured as "States" (of key params); Desired outcomes as "Actions" to optimize energy; & "Rewards" represent incentives / disincentives as the RL algorithm sweeps through param space