# Climate Modeling: From Simple Energy Balance to GraphCast

**A Progressive Journey Through Five Climate Models**

This notebook explores climate modeling through five progressively sophisticated approaches, culminating in Google's GraphCast. Each model builds upon the previous one, adding complexity and realism while maintaining scientific rigor.

## Overview

Climate models are mathematical representations of Earth's climate system. They range from simple energy balance equations to complex machine learning systems that can forecast weather patterns. This notebook presents:

1. **Zero-Dimensional Energy Balance Model** - The foundation of climate science
2. **One-Dimensional Radiative-Convective Model** - Adding vertical atmospheric structure
3. **Two-Dimensional Statistical Dynamical Model** - Including latitude variations
4. **Three-Dimensional General Circulation Model** - Full spatial dynamics
5. **GraphCast-Style ML Model** - Modern AI/ML approach to weather/climate prediction

Each model includes:

- Detailed technical explanation (2 pages) of assumptions and approximations
- Implementation with documented code
- Visualizations of key results
- Analysis of climate change implications

---

In [1]:
```
# Core imports
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.integrate import odeint, solve_ivp
from scipy.optimize import fsolve, minimize
import pandas as pd
from typing import Tuple, List, Callable
import warnings
warnings.filterwarnings('ignore')
```

```python
# Configuration
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 11
%matplotlib inline

print("✓ Libraries imported successfully!")
print(f"NumPy version: {np.__version__}")
print(f"Matplotlib version: {plt.matplotlib.__version__}")
```

```
✓ Libraries imported successfully!
NumPy version: 2.4.0
Matplotlib version: 3.10.8
```

# Model 1: Zero-Dimensional Energy Balance Model (EBM)

## Technical Overview (Page 1 of 2)

The Zero-Dimensional Energy Balance Model represents Earth as a single point with no spatial variation. Despite its simplicity, it captures the fundamental physics governing Earth's temperature: the balance between incoming solar radiation and outgoing infrared radiation.

### Fundamental Equation

The governing equation is:

$$C \frac{dT}{dt} = Q(1-\alpha) - \epsilon\sigma T^4 + F$$

Where:

- $C$ = Climate system heat capacity ($J\ m^{-2}\ K^{-1}$) ≈ $10^8$ $J\ m^{-2}\ K^{-1}$
- $T$ = Global mean surface temperature (K)
- $Q$ = Incoming solar radiation per unit area = $S_0/4$ ≈ 342 $W\ m^{-2}$
- $\alpha$ = Planetary albedo (reflectivity) ≈ 0.30
- $\epsilon$ = Effective emissivity ≈ 0.61 (accounting for greenhouse effect)
- $\sigma$ = Stefan-Boltzmann constant = $5.67 \times 10^{-8}$ $W\ m^{-2}\ K^{-4}$
- $F$ = Additional radiative forcing ($W\ m^{-2}$)

### Key Physical Assumptions

1. **Spatial Homogeneity**: Earth is treated as a uniform sphere with no variation in latitude, longitude, or altitude. All locations have identical temperature and properties.

2. **Radiative Equilibrium**: The climate is determined entirely by radiative processes. Heat transport by atmosphere and oceans is implicitly included in the effective heat capacity.

3. **Gray Atmosphere**: The atmosphere absorbs and emits radiation uniformly across all wavelengths, simplified into a single emissivity parameter.

4. **Blackbody Radiation**: Earth's surface and atmosphere emit according to the Stefan-Boltzmann law with modification by emissivity.

5. **Steady-State Geometry**: The factor of 4 in $Q = S_0/4$ comes from the ratio of Earth's cross-sectional area ($\pi R^2$) to total surface area ($4\pi R^2$).

6. **Linear Heat Capacity**: The relationship between energy storage and temperature change is linear and constant.

## Technical Overview (Page 2 of 2)

### Mathematical Approximations

**Greenhouse Effect Parameterization**: The most significant approximation is representing the complex greenhouse effect (involving multiple gases with wavelength-dependent absorption) as a single emissivity parameter $\epsilon$. In reality:

- Different greenhouse gases ($H_2O$, $CO_2$, $CH_4$, $N_2O$) absorb at different wavelengths
- Atmospheric temperature profile affects emission altitude
- Cloud effects are highly variable
- The model captures this complexity through $\epsilon \approx 0.61$, calibrated to match observed Earth temperature

**Heat Capacity Lumping**: The ocean mixed layer, land surface, deep ocean, and atmosphere have vastly different heat capacities and response times (hours to millennia). The model uses an effective value representing primarily the ocean mixed layer (~50-100m depth).

**Albedo Simplification**: Planetary albedo varies with:

- Ice cover (0.5-0.9)
- Clouds (0.4-0.9)
- Vegetation (0.1-0.2)
- Ocean (0.06)

The constant $\alpha = 0.30$ is a global annual mean that changes with climate.

### Climate Sensitivity

At equilibrium ($dT/dt = 0$), the temperature is:

$$T_{eq} = \left(\frac{Q(1-\alpha) + F}{\epsilon\sigma}\right)^{1/4}$$

The **equilibrium climate sensitivity** (ECS) - temperature change for doubled $CO_2$ - can be calculated. Doubling $CO_2$ produces forcing $\Delta F ≈ 3.7-4.0$ W $m^{-2}$, yielding:

$$\Delta T_{eq} = T_{eq}(F + \Delta F) - T_{eq}(F)$$

In this simple model, ECS ≈ 1.2°C, which is lower than the IPCC range of 2.5-4°C because the model lacks important positive feedbacks:

- Water vapor feedback (warming → more $H_2O$ → more greenhouse effect)
- Ice-albedo feedback (warming → less ice → less reflection → more warming)
- Cloud feedbacks (complex, both positive and negative)

### Limitations

1. **No Geography**: Cannot represent land-ocean contrasts, mountain effects, or regional climate
2. **No Seasons**: Annual mean only; cannot capture seasonal cycle or extreme events
3. **No Dynamics**: Atmospheric and oceanic circulation ignored
4. **No Weather**: All synoptic-scale variability averaged out
5. **Underestimates Sensitivity**: Missing key positive feedbacks
6. **No Hydrological Cycle**: Precipitation and evaporation not represented

### Strengths and Use Cases

Despite limitations, this model:

- ✓ Correctly predicts Earth's mean temperature (~288 K vs observed)
- ✓ Demonstrates fundamental greenhouse effect
- ✓ Shows qualitative response to forcing changes
- ✓ Provides physical intuition for energy balance
- ✓ Fast computation for parameter sensitivity studies
- ✓ Good first-order estimate of climate sensitivity

**Applications**: Education, rapid scenario testing, understanding basic climate physics, validating more complex models.

In [2]:
```python
class ZeroDimensionalEBM:
    """
    Zero-Dimensional Energy Balance Model

    Solves: C*dT/dt = Q*(1-α) - ε*σ*T^4 + F

```

```python
    where Earth is treated as a single point with uniform temperature.
    """

    def __init__(self, C=1e8, alpha=0.30, epsilon=0.61):
        """
        Initialize model with physical constants

        Parameters:
        -----------
        C : float
            Heat capacity (J m^-2 K^-1), default 1e8 (ocean mixed layer ~100
        alpha : float
            Planetary albedo (dimensionless), default 0.30
        epsilon : float
            Effective emissivity (dimensionless), default 0.61
        """
        # Physical constants (SI units)
        self.sigma = 5.67e-8  # Stefan-Boltzmann constant (W m^-2 K^-4)
        self.S0 = 1361.0      # Solar constant at Earth (W m^-2)
        self.Q = self.S0 / 4  # Average incoming solar (geometry factor)

        # Model parameters
        self.C = C
        self.alpha = alpha
        self.epsilon = epsilon

    def absorbed_solar(self):
        """Calculate absorbed solar radiation (W m^-2)"""
        return self.Q * (1 - self.alpha)

    def emitted_ir(self, T):
        """Calculate emitted infrared radiation (W m^-2)"""
        return self.epsilon * self.sigma * T**4

    def net_radiation(self, T, forcing=0):
        """Calculate net radiative balance (W m^-2)"""
        return self.absorbed_solar() + forcing - self.emitted_ir(T)

    def dT_dt(self, T, t, forcing=0):
        """
        Temperature tendency equation

        Returns dT/dt in K/year
        """
        dE_dt = self.net_radiation(T, forcing)  # W m^-2
        seconds_per_year = 365.25 * 24 * 3600
        return (dE_dt / self.C) * seconds_per_year

    def equilibrium_temperature(self, forcing=0):
        """
        Calculate equilibrium temperature analytically

        Parameters:
        -----------
        forcing : float
            Additional radiative forcing (W m^-2)
```

```
        Returns:
        --------
        T_eq : float
            Equilibrium temperature (K)
        """
        numerator = self.absorbed_solar() + forcing
        T_eq = (numerator / (self.epsilon * self.sigma))**0.25
        return T_eq

    def run_simulation(self, T0, years, forcing=0, dt=0.1):
        """
        Time-dependent simulation

        Parameters:
        -----------
        T0 : float
            Initial temperature (K)
        years : float
            Simulation duration (years)
        forcing : float or callable
            Constant forcing (W m^-2) or function f(t) returning forcing
        dt : float
            Time step (years)

        Returns:
        --------
        t : ndarray
            Time points (years)
        T : ndarray
            Temperature evolution (K)
        """
        t = np.arange(0, years, dt)
        T = np.zeros_like(t)
        T[0] = T0

        # Check if forcing is callable
        if callable(forcing):
            forcing_func = forcing
        else:
            forcing_func = lambda t: forcing

        # Forward Euler integration (simple and stable for this problem)
        for i in range(1, len(t)):
            F_current = forcing_func(t[i-1])
            T[i] = T[i-1] + self.dT_dt(T[i-1], t[i-1], F_current) * dt

        return t, T

    def climate_sensitivity(self, forcing_2xCO2=3.7):
        """
        Calculate equilibrium climate sensitivity

        Parameters:
        -----------
        forcing_2xCO2 : float
```

```
            Radiative forcing from doubling CO2 (W m^-2), default 3.7

        Returns:
        --------
        ECS : float
            Equilibrium climate sensitivity (K)
        """
        T_current = self.equilibrium_temperature(0)
        T_2xCO2 = self.equilibrium_temperature(forcing_2xCO2)
        return T_2xCO2 - T_current

# Initialize the model
print("="*60)
print("ZERO-DIMENSIONAL ENERGY BALANCE MODEL")
print("="*60 + "\n")

model1 = ZeroDimensionalEBM()

print(f"Physical Constants:")
print(f"  Solar constant (S₀): {model1.S0:.1f} W/m²")
print(f"  Mean solar input (Q): {model1.Q:.1f} W/m²")
print(f"  Stefan-Boltzmann (σ): {model1.sigma:.2e} W/m²/K⁴\n")

print(f"Model Parameters:")
print(f"  Heat capacity (C): {model1.C:.2e} J/m²/K")
print(f"  Albedo (α): {model1.alpha:.2f}")
print(f"  Emissivity (ε): {model1.epsilon:.2f}\n")

# Calculate current climate equilibrium
T_eq = model1.equilibrium_temperature()
print(f"Current Climate:")
print(f"  Equilibrium temperature: {T_eq:.2f} K ({T_eq-273.15:.2f}°C)")
print(f"  Absorbed solar: {model1.absorbed_solar():.1f} W/m²")
print(f"  Emitted IR: {model1.emitted_ir(T_eq):.1f} W/m²\n")

# Calculate climate sensitivity
ECS = model1.climate_sensitivity()
print(f"Climate Sensitivity:")
print(f"  2×CO₂ forcing: 3.7 W/m²")
print(f"  Equilibrium climate sensitivity: {ECS:.2f} K")
print(f"  New equilibrium: {T_eq+ECS:.2f} K ({T_eq+ECS-273.15:.2f}°C)")
print("\n" + "="*60)
```

```
============================================================
ZERO-DIMENSIONAL ENERGY BALANCE MODEL
============================================================

Physical Constants:
  Solar constant (S₀): 1361.0 W/m²
  Mean solar input (Q): 340.2 W/m²
  Stefan-Boltzmann (σ): 5.67e-08 W/m²/K⁴

Model Parameters:
  Heat capacity (C): 1.00e+08 J/m²/K
  Albedo (α): 0.30
  Emissivity (ε): 0.61

Current Climate:
  Equilibrium temperature: 288.07 K (14.92°C)
  Absorbed solar: 238.2 W/m²
  Emitted IR: 238.2 W/m²

Climate Sensitivity:
  2×CO₂ forcing: 3.7 W/m²
  Equilibrium climate sensitivity: 1.11 K
  New equilibrium: 289.18 K (16.03°C)

============================================================
```

In [3]:

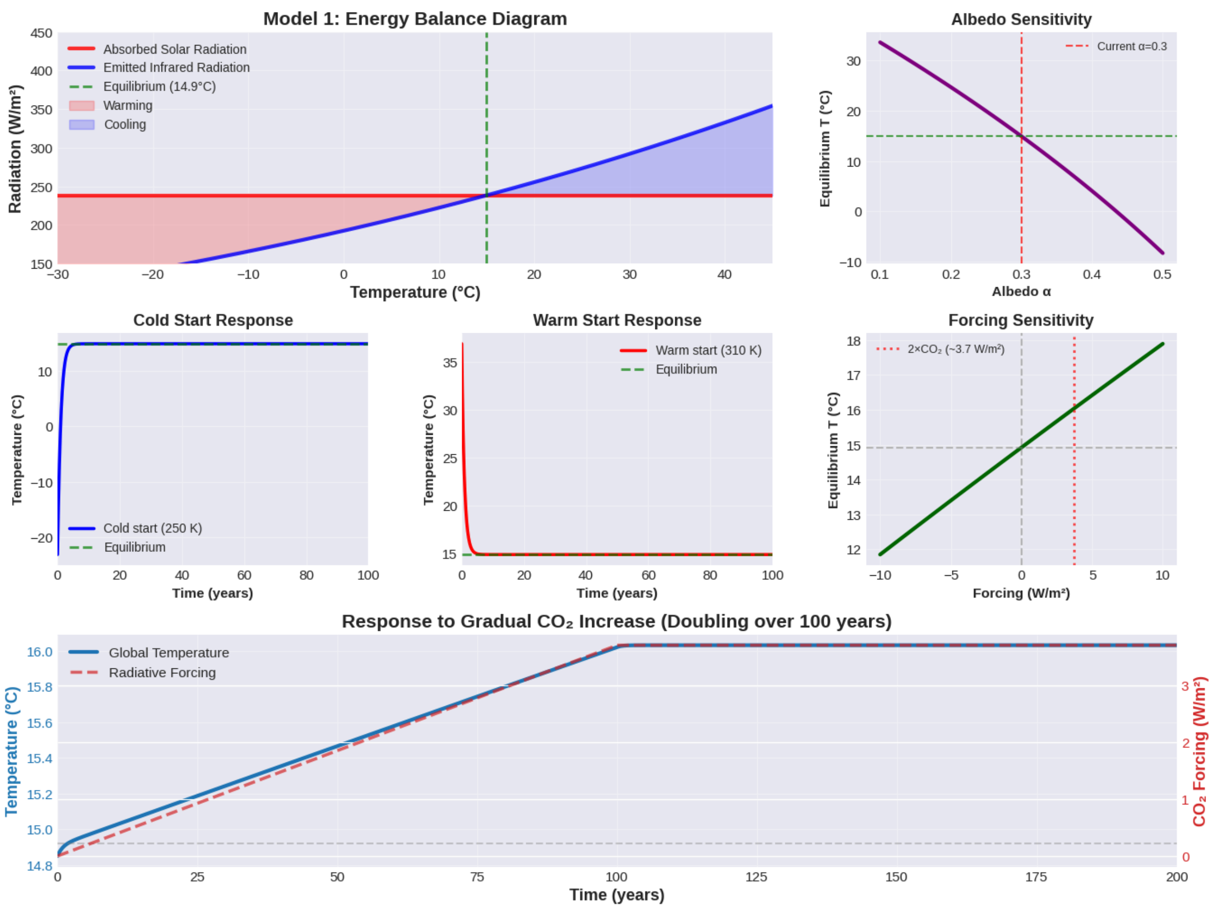
```python
# Create comprehensive visualizations for Model 1

fig = plt.figure(figsize=(16, 12))
gs = fig.add_gridspec(3, 3, hspace=0.3, wspace=0.3)

# === Panel 1: Energy Balance Diagram ===
ax1 = fig.add_subplot(gs[0, :2])
T_range = np.linspace(240, 320, 200)
Q_in = model1.absorbed_solar()
Q_out = model1.emitted_ir(T_range)

ax1.plot(T_range-273.15, Q_in*np.ones_like(T_range), 'r-', linewidth=3,
         label='Absorbed Solar Radiation', alpha=0.8)
ax1.plot(T_range-273.15, Q_out, 'b-', linewidth=3,
         label='Emitted Infrared Radiation', alpha=0.8)
ax1.axvline(T_eq-273.15, color='green', linestyle='--', linewidth=2,
            label=f'Equilibrium ({T_eq-273.15:.1f}°C)', alpha=0.7)
ax1.fill_between(T_range-273.15, Q_in*np.ones_like(T_range), Q_out,
                 where=(Q_in >= Q_out), alpha=0.2, color='red', label='Warm
ax1.fill_between(T_range-273.15, Q_in*np.ones_like(T_range), Q_out,
                 where=(Q_in < Q_out), alpha=0.2, color='blue', label='Cool

ax1.set_xlabel('Temperature (°C)', fontsize=13, fontweight='bold')
ax1.set_ylabel('Radiation (W/m²)', fontsize=13, fontweight='bold')
ax1.set_title('Model 1: Energy Balance Diagram', fontsize=15, fontweight='bo
ax1.legend(fontsize=10, loc='upper left')
ax1.grid(True, alpha=0.3)
ax1.set_xlim(-30, 45)
ax1.set_ylim(150, 450)
```

```python
# === Panel 2: Parameter Sensitivity ===
ax2 = fig.add_subplot(gs[0, 2])
alphas = np.linspace(0.1, 0.5, 50)
T_alpha = [(model1.Q * (1-a) / (model1.epsilon * model1.sigma))**0.25 - 273.
           for a in alphas]

ax2.plot(alphas, T_alpha, 'purple', linewidth=3)
ax2.axvline(model1.alpha, color='red', linestyle='--', alpha=0.7,
            label=f'Current α={model1.alpha}')
ax2.axhline(T_eq-273.15, color='green', linestyle='--', alpha=0.7)
ax2.set_xlabel('Albedo α', fontsize=11, fontweight='bold')
ax2.set_ylabel('Equilibrium T (°C)', fontsize=11, fontweight='bold')
ax2.set_title('Albedo Sensitivity', fontsize=13, fontweight='bold')
ax2.legend(fontsize=9)
ax2.grid(True, alpha=0.3)

# === Panel 3: Temperature Evolution (Cold Start) ===
ax3 = fig.add_subplot(gs[1, 0])
t1, T1 = model1.run_simulation(T0=250, years=100, dt=0.1)
ax3.plot(t1, T1-273.15, 'b-', linewidth=2.5, label='Cold start (250 K)')
ax3.axhline(T_eq-273.15, color='green', linestyle='--', linewidth=2,
            alpha=0.7, label='Equilibrium')
ax3.set_xlabel('Time (years)', fontsize=11, fontweight='bold')
ax3.set_ylabel('Temperature (°C)', fontsize=11, fontweight='bold')
ax3.set_title('Cold Start Response', fontsize=13, fontweight='bold')
ax3.legend(fontsize=10)
ax3.grid(True, alpha=0.3)
ax3.set_xlim(0, 100)

# === Panel 4: Temperature Evolution (Warm Start) ===
ax4 = fig.add_subplot(gs[1, 1])
t2, T2 = model1.run_simulation(T0=310, years=100, dt=0.1)
ax4.plot(t2, T2-273.15, 'r-', linewidth=2.5, label='Warm start (310 K)')
ax4.axhline(T_eq-273.15, color='green', linestyle='--', linewidth=2,
            alpha=0.7, label='Equilibrium')
ax4.set_xlabel('Time (years)', fontsize=11, fontweight='bold')
ax4.set_ylabel('Temperature (°C)', fontsize=11, fontweight='bold')
ax4.set_title('Warm Start Response', fontsize=13, fontweight='bold')
ax4.legend(fontsize=10)
ax4.grid(True, alpha=0.3)
ax4.set_xlim(0, 100)

# === Panel 5: Climate Forcing Response ===
ax5 = fig.add_subplot(gs[1, 2])
forcings = np.linspace(-10, 10, 100)
T_forced = [model1.equilibrium_temperature(f) - 273.15 for f in forcings]

ax5.plot(forcings, T_forced, 'darkgreen', linewidth=3)
ax5.axvline(0, color='gray', linestyle='--', alpha=0.5)
ax5.axvline(3.7, color='red', linestyle=':', linewidth=2,
            alpha=0.7, label='2×CO₂ (~3.7 W/m²)')
ax5.axhline(T_eq-273.15, color='gray', linestyle='--', alpha=0.5)
ax5.set_xlabel('Forcing (W/m²)', fontsize=11, fontweight='bold')
ax5.set_ylabel('Equilibrium T (°C)', fontsize=11, fontweight='bold')
ax5.set_title('Forcing Sensitivity', fontsize=13, fontweight='bold')
ax5.legend(fontsize=9)
```

```
ax5.grid(True, alpha=0.3)

# === Panel 6: CO2 Increase Scenario ===
ax6 = fig.add_subplot(gs[2, :])

# Define gradual CO2 increase
def co2_forcing(t):
    """Forcing that ramps up linearly over 100 years to 2xCO2"""
    return min(3.7 * t / 100, 3.7)

t_co2, T_co2 = model1.run_simulation(T0=288, years=200, forcing=co2_forcing,
forcing_trajectory = np.array([co2_forcing(ti) for ti in t_co2])

# Plot on dual axes
color1 = 'tab:blue'
ax6.set_xlabel('Time (years)', fontsize=13, fontweight='bold')
ax6.set_ylabel('Temperature (°C)', fontsize=13, fontweight='bold', color=col
line1 = ax6.plot(t_co2, T_co2-273.15, color=color1, linewidth=3,
                 label='Global Temperature')
ax6.tick_params(axis='y', labelcolor=color1)
ax6.axhline(T_eq-273.15, color='gray', linestyle='--', alpha=0.4, label='Pre

ax6_twin = ax6.twinx()
color2 = 'tab:red'
ax6_twin.set_ylabel('CO₂ Forcing (W/m²)', fontsize=13, fontweight='bold', co
line2 = ax6_twin.plot(t_co2, forcing_trajectory, color=color2, linewidth=2.5
                      linestyle='--', alpha=0.7, label='Radiative Forcing')
ax6_twin.tick_params(axis='y', labelcolor=color2)

# Combine legends
lines = line1 + line2
labels = [l.get_label() for l in lines]
ax6.legend(lines, labels, fontsize=11, loc='upper left', framealpha=0.9)
ax6.grid(True, alpha=0.3)
ax6.set_title('Response to Gradual CO₂ Increase (Doubling over 100 years)',
              fontsize=15, fontweight='bold')
ax6.set_xlim(0, 200)

plt.suptitle('Zero-Dimensional Energy Balance Model: Complete Analysis',
             fontsize=17, fontweight='bold', y=0.995)

plt.savefig('model1_complete.png', dpi=150, bbox_inches='tight')
plt.show()

print("\n" + "="*60)
print("KEY INSIGHTS FROM MODEL 1")
print("="*60)
print("\n1. Energy Balance: Earth maintains equilibrium when absorbed")
print("   solar radiation equals emitted infrared radiation")
print(f"\n2. Current equilibrium: {T_eq-273.15:.1f}°C is very close to")
print("   observed global mean temperature (~15°C)")
print(f"\n3. Climate Sensitivity: Doubling CO₂ (~3.7 W/m²) causes")
print(f"   ~{ECS:.1f}°C warming in this simple model")
print("\n4. Thermal Inertia: Temperature changes lag forcing due to")
print("   ocean heat capacity (time constant ~decades)")
print("\n5. Limitations: This model underestimates sensitivity because")
```

```python
print("   it lacks key feedbacks (water vapor, ice-albedo, clouds)")
print("\n" + "="*60)
```

**Zero-Dimensional Energy Balance Model: Complete Analysis**




```
============================================================
KEY INSIGHTS FROM MODEL 1
============================================================

1. Energy Balance: Earth maintains equilibrium when absorbed
   solar radiation equals emitted infrared radiation

2. Current equilibrium: 14.9°C is very close to
   observed global mean temperature (~15°C)

3. Climate Sensitivity: Doubling CO₂ (~3.7 W/m²) causes
   ~1.1°C warming in this simple model

4. Thermal Inertia: Temperature changes lag forcing due to
   ocean heat capacity (time constant ~decades)

5. Limitations: This model underestimates sensitivity because
   it lacks key feedbacks (water vapor, ice-albedo, clouds)

============================================================
```

# Model 2: One-Dimensional Radiative-Convective Model

## Technical Overview (Page 1 of 2)

The One-Dimensional Radiative-Convective Model extends the zero-dimensional model by adding vertical atmospheric structure. This captures the critical feature that Earth's atmosphere is not uniform - temperature, pressure, and composition vary dramatically with altitude.

### Governing Equations

The model solves radiative transfer and convective adjustment in a vertical column:

**Radiative Transfer:** $$\frac{d F_{\uparrow}}{dz} = -\kappa(z)\rho(z)[B(T(z)) - F_{\uparrow}]$$ $$\frac{d F_{\downarrow}}{dz} = \kappa(z)\rho(z)[B(T(z)) - F_{\downarrow}]$$

**Energy Balance:** $$\rho(z) c_p \frac{\partial T}{\partial t} = -\frac{\partial F_{net}}{\partial z} + Q_{conv}$$

**Convective Adjustment:** $$\text{If } \frac{dT}{dz} < -\Gamma_{crit}, \text{ adjust to } \frac{dT}{dz} = -\Gamma_{crit}$$

Where:

- $F_{\uparrow}, F_{\downarrow}$ = Upward and downward radiative fluxes (W $m^{-2}$)
- $z$ = Altitude (m)
- $\kappa(z)$ = Absorption coefficient ($m^2$ $kg^{-1}$), varies with wavelength and species
- $\rho(z)$ = Air density (kg $m^{-3}$)
- $B(T)$ = Planck function ≈ $\sigma T^4$ (gray atmosphere approximation)
- $T(z)$ = Temperature profile (K)
- $c_p$ = Specific heat at constant pressure = 1004 J $kg^{-1}$ $K^{-1}$
- $Q_{conv}$ = Convective heat flux (W $m^{-3}$)
- $\Gamma_{crit}$ = Critical lapse rate ≈ 6.5 K $km^{-1}$

### Key Physical Assumptions

1. **One-Dimensional**: Horizontal homogeneity - no variation in x or y directions. Represents a global or zonal mean.

2. **Hydrostatic Balance**: Pressure decreases exponentially with altitude according to $P(z) = P_0 e^{-z/H}$ where $H ≈ 8$ km is the scale height.

3. **Gray Atmosphere**: Absorption and emission are wavelength-independent, characterized by a single optical depth $\tau$.

4. **Two-Stream Approximation**: Radiation is either purely upward or purely downward, neglecting sideways scattering.

5. **Schwarzschild Equation**: Each atmospheric layer emits as a blackbody and absorbs radiation passing through it.

6. **Convective Adjustment**: When radiative equilibrium produces a superadiabatic lapse rate (unstable), convection instantly adjusts the profile to the critical lapse rate.

### Vertical Structure

The atmosphere is divided into layers (typically 20-50):

- **Troposphere** (0-12 km): Temperature decreases with height, governed by moist convection
- **Stratosphere** (12-50 km): Temperature increases with height due to ozone absorption (simplified or omitted in basic versions)
- **Surface**: Coupled to lowest atmospheric layer via radiation and turbulent fluxes

## Technical Overview (Page 2 of 2)

### Mathematical Approximations

**Gray Atmosphere Approximation**: Real greenhouse gases have complex, wavelength-dependent absorption:

- $H_2O$ absorbs strongly at 6.3 μm (vibration-rotation) and >15 μm (pure rotation)
- $CO_2$ absorbs at 15 μm and 4.3 μm
- $O_3$ absorbs in UV and at 9.6 μm
- Clouds absorb and scatter across broad spectrum

The gray approximation uses effective optical depth: $$\tau_{eff} = \int_0^{\infty} \kappa_\lambda(z) \rho(z) dz$$

calibrated to match observed radiative fluxes. Typical values: $\tau_{eff} ≈ 1-2$ for clear sky.

**Two-Stream Radiative Transfer**: The full radiative transfer equation is an integro-differential equation accounting for scattering in all directions. The two-stream approximation assumes:

- Upward flux: $F_{\uparrow}(z) = \pi I_{\uparrow}$ (hemispheric integral)
- Downward flux: $F_{\downarrow}(z) = \pi I_{\downarrow}$

This is accurate to within ~10-20% for thermal radiation but less accurate for solar radiation with scattering.

**Convective Parameterization**: Real atmospheric convection involves:

- Cloud formation and latent heat release
- Entrainment and detrainment
- Mesoscale organization
- Turbulent eddies

The model uses instantaneous adjustment to a prescribed lapse rate: $$\Gamma = \Gamma_d \frac{1 + L_v q_s / (R_d T)}{1 + L_v^2 q_s / (c_p R_v T^2)} ≈ 6.5 \text{ K/km}$$

where $\Gamma_d = g/c_p ≈ 9.8$ K/km is the dry adiabatic lapse rate, modified by moisture.

**Solar Absorption**: Simplified to:

- Surface absorbs most solar radiation
- Stratospheric ozone absorption neglected or parameterized
- Cloud effects on solar radiation simplified

## Radiative-Convective Equilibrium

The model seeks equilibrium where:

1. Surface energy budget balances: solar absorption = IR emission + sensible heat
2. Each atmospheric layer has zero net radiative heating or is convectively neutral
3. Top-of-atmosphere energy budget closes

The equilibrium is found iteratively:

1. Calculate radiative fluxes for given T(z)
2. Compute radiative heating rates
3. Update T(z) toward radiative equilibrium
4. Apply convective adjustment where unstable
5. Repeat until convergence

## Improvements Over Model 1

✓ **Vertical temperature structure**: Captures troposphere-stratosphere distinction ✓ **Atmospheric greenhouse effect**: Explicitly represents radiation

absorption/emission by gases ✓ **Lapse rate feedback**: Changes in vertical temperature profile affect sensitivity
✓ **Surface-atmosphere coupling**: Distinguishes surface from atmospheric temperatures ✓ **Altitude-dependent forcing**: $CO_2$ forcing affects different layers differently

### Remaining Limitations

✗ **No horizontal structure**: Cannot represent equator-pole temperature gradient ✗ **No dynamics**: Winds and pressure systems not included ✗ **No clouds**: Major uncertainty in real climate ✗ **No seasons**: Time-mean only ✗ **Simplified convection**: Real convection is complex and localized

### Climate Sensitivity

In radiative-convective models, ECS ≈ 1.5-2.5°C, closer to observations than Model 1 because:

- Water vapor feedback included: warmer atmosphere holds more $H_2O$
- Lapse rate feedback: tropospheric warming pattern affects surface response
- Still missing ice-albedo, cloud feedbacks

In [4]:
```
class OneDimensionalRCM:
    """
    One-Dimensional Radiative-Convective Model

    Solves radiative transfer and convective adjustment in a vertical column
    Represents vertical atmospheric structure from surface to top of atmosph
    """

    def __init__(self, n_levels=30, p_surface=1013.25, p_top=10.0):
        """
        Initialize 1D radiative-convective model

        Parameters:
        -----------
        n_levels : int
            Number of vertical levels
        p_surface : float
            Surface pressure (hPa)
        p_top : float
            Top of atmosphere pressure (hPa)
        """
        # Physical constants
        self.g = 9.81           # Gravity (m/s²)
        self.cp = 1004.0        # Specific heat at const pressure (J/kg/K)
        self.R = 287.0          # Gas constant for dry air (J/kg/K)
        self.sigma = 5.67e-8    # Stefan-Boltzmann constant
        self.S0 = 1361.0        # Solar constant (W/m²)

        # Model parameters
```

```
        self.albedo = 0.30      # Planetary albedo
        self.tau_lw = 1.5       # Longwave optical depth
        self.solar_abs_atm = 0.2  # Fraction of solar absorbed by atmosphere
        self.critical_lapse = 6.5e-3  # Critical lapse rate (K/m)

        # Vertical grid
        self.n_levels = n_levels
        self.p_surface = p_surface  # hPa
        self.p_top = p_top          # hPa

        # Create pressure levels (equally spaced in log-pressure)
        self.p = np.logspace(np.log10(p_top), np.log10(p_surface), n_levels)
        self.p_pa = self.p * 100  # Convert to Pa

        # Calculate layer properties
        self.dp = np.diff(self.p_pa)  # Pressure thickness of layers
        self.z = self._pressure_to_height(self.p_pa)  # Approximate heights

    def _pressure_to_height(self, p):
        """Convert pressure to approximate height using hydrostatic equation
        H = self.R * 250 / self.g  # Scale height (~7.5 km for T=250K)
        return -H * np.log(p / (self.p_surface * 100))

    def _height_to_temperature(self, z, T_surface):
        """Standard atmosphere approximation"""
        # Troposphere: linear decrease
        T = T_surface - self.critical_lapse * z
        # Don't let temperature go below 180 K (stratosphere)
        return np.maximum(T, 180)

    def planck_emission(self, T):
        """Blackbody emission (W/m²)"""
        return self.sigma * T**4

    def optical_depth_profile(self):
        """
        Calculate optical depth at each level
        Increases with pressure (more gas below)
        """
        # Optical depth increases toward surface
        tau = self.tau_lw * (self.p / self.p_surface)
        return tau

    def compute_radiative_fluxes(self, T):
        """
        Compute upward and downward longwave fluxes using two-stream approxi

        Parameters:
        -----------
        T : array
            Temperature at each level (K)

        Returns:
        --------
        F_up : array
            Upward flux at each level (W/m²)
```

```
        F_down : array
            Downward flux at each level (W/m²)
        """
        n = len(T)
        F_up = np.zeros(n+1)    # Fluxes at layer interfaces
        F_down = np.zeros(n+1)

        tau = self.optical_depth_profile()

        # Surface emission (bottom boundary)
        F_up[0] = self.planck_emission(T[0])

        # Upward flux: integrate from surface to top
        for i in range(n):
            B_i = self.planck_emission(T[i])
            if i < n-1:
                dtau = tau[i] - tau[i+1]
            else:
                dtau = tau[i]

            # Two-stream approximation
            transmittance = np.exp(-dtau)
            F_up[i+1] = F_up[i] * transmittance + B_i * (1 - transmittance)

        # Downward flux: integrate from top to surface
        F_down[-1] = 0  # No downward flux at TOA

        for i in range(n-1, -1, -1):
            B_i = self.planck_emission(T[i])
            if i > 0:
                dtau = tau[i] - tau[i-1]
            else:
                dtau = tau[i]

            transmittance = np.exp(-dtau)
            F_down[i] = F_down[i+1] * transmittance + B_i * (1 - transmittar

        return F_up, F_down

    def solar_heating(self, T):
        """
        Calculate solar heating rate in each layer

        Returns:
        --------
        Q_solar : array
            Heating rate (K/day) for each level
        """
        Q_in = (self.S0 / 4) * (1 - self.albedo)  # Absorbed solar

        # Simple distribution: most at surface, some in atmosphere
        Q_solar = np.zeros(self.n_levels)

        # Atmospheric absorption (decreases exponentially upward)
        for i in range(self.n_levels):
            altitude_factor = np.exp(-(self.n_levels - i) / 10)
```

```python
            Q_solar[i] = Q_in * self.solar_abs_atm * altitude_factor

        # Surface gets the rest
        Q_solar[0] += Q_in * (1 - self.solar_abs_atm)

        # Convert to heating rate (K/day)
        mass_per_area = self.p_pa / self.g  # kg/m²
        seconds_per_day = 86400

        for i in range(self.n_levels):
            if i == 0:
                dm = mass_per_area[0]
            else:
                dm = abs(mass_per_area[i] - mass_per_area[i-1])

            if dm > 0:
                Q_solar[i] = (Q_solar[i] / (dm * self.cp)) * seconds_per_day

        return Q_solar

    def longwave_heating(self, F_up, F_down):
        """
        Calculate longwave radiative heating rate

        Returns:
        --------
        Q_lw : array
            Cooling rate (K/day) for each level
        """
        Q_lw = np.zeros(self.n_levels)

        # Heating = convergence of net flux
        F_net = F_up - F_down

        mass_per_area = self.p_pa / self.g
        seconds_per_day = 86400

        for i in range(self.n_levels):
            # Flux convergence
            if i == 0:
                dF = F_net[1] - F_net[0]
                dm = mass_per_area[0]
            elif i == self.n_levels - 1:
                dF = F_net[i+1] - F_net[i]
                dm = abs(mass_per_area[i] - mass_per_area[i-1])
            else:
                dF = F_net[i+1] - F_net[i]
                dm = abs(mass_per_area[i] - mass_per_area[i-1])

            if dm > 0:
                Q_lw[i] = -(dF / (dm * self.cp)) * seconds_per_day

        return Q_lw

    def apply_convective_adjustment(self, T):
        """
```

```python
        Adjust temperature profile to critical lapse rate where unstable

        Parameters:
        -----------
        T : array
            Temperature profile (K)

        Returns:
        --------
        T_adjusted : array
            Adjusted temperature profile (K)
        """
        T_adj = T.copy()

        # Check lapse rate from surface upward
        for i in range(len(T) - 1):
            if self.z[i+1] > self.z[i]:  # Make sure height increases
                dz = self.z[i+1] - self.z[i]
                actual_lapse = -(T_adj[i+1] - T_adj[i]) / dz

                # If super-adiabatic (too steep), adjust
                if actual_lapse > self.critical_lapse:
                    # Set to critical lapse rate
                    T_adj[i+1] = T_adj[i] - self.critical_lapse * dz

        return T_adj

    def run_to_equilibrium(self, T_initial=None, max_iterations=1000,
                          tolerance=0.01, forcing=0):
        """
        Iterate to radiative-convective equilibrium

        Parameters:
        -----------
        T_initial : array, optional
            Initial temperature profile (K). If None, uses standard atmosphe
        max_iterations : int
            Maximum iterations
        tolerance : float
            Convergence criterion (K)
        forcing : float
            Additional radiative forcing (W/m²) at surface

        Returns:
        --------
        T : array
            Equilibrium temperature profile (K)
        converged : bool
            Whether solution converged
        """
        # Initialize temperature profile
        if T_initial is None:
            T_surface_guess = 288  # K
            T = self._height_to_temperature(self.z, T_surface_guess)
        else:
            T = T_initial.copy()
```

```python
        # Relaxation parameter for stability
        alpha = 0.1

        for iteration in range(max_iterations):
            T_old = T.copy()

            # Compute radiative fluxes
            F_up, F_down = self.compute_radiative_fluxes(T)

            # Add forcing to surface
            F_up[0] += forcing

            # Compute heating rates
            Q_solar = self.solar_heating(T)
            Q_lw = self.longwave_heating(F_up, F_down)
            Q_total = Q_solar + Q_lw

            # Update temperature
            T = T + alpha * Q_total

            # Apply convective adjustment
            T = self.apply_convective_adjustment(T)

            # Check convergence
            max_change = np.max(np.abs(T - T_old))
            if max_change < tolerance:
                return T, True, iteration

        return T, False, max_iterations

    def climate_sensitivity(self, forcing_2xCO2=4.0):
        """
        Calculate equilibrium climate sensitivity

        Returns:
        --------
        T_control : array
            Control climate temperature profile
        T_2xCO2 : array
            2×CO₂ temperature profile
        ECS : float
            Equilibrium climate sensitivity (surface temperature change)
        """
        # Control climate
        T_control, _, _ = self.run_to_equilibrium(forcing=0)

        # 2×CO₂ climate
        T_2xCO2, _, _ = self.run_to_equilibrium(forcing=forcing_2xCO2)

        ECS = T_2xCO2[0] - T_control[0]

        return T_control, T_2xCO2, ECS

# Initialize and run Model 2
print("="*70)
```

```python
print("ONE-DIMENSIONAL RADIATIVE-CONVECTIVE MODEL")
print("="*70 + "\n")

model2 = OneDimensionalRCM(n_levels=30)

print(f"Model Configuration:")
print(f"  Vertical levels: {model2.n_levels}")
print(f"  Pressure range: {model2.p_top:.1f} - {model2.p_surface:.1f} hPa")
print(f"  Height range: {model2.z[-1]/1000:.1f} - {model2.z[0]/1000:.1f} km"
print(f"  Critical lapse rate: {model2.critical_lapse*1000:.1f} K/km\n")

print("Computing radiative-convective equilibrium...")
T_eq, converged, iterations = model2.run_to_equilibrium()

print(f"  Converged: {converged}")
print(f"  Iterations: {iterations}")
print(f"  Surface temperature: {T_eq[0]:.2f} K ({T_eq[0]-273.15:.2f}°C)")
print(f"  Upper atmosphere: {T_eq[-1]:.2f} K ({T_eq[-1]-273.15:.2f}°C)\n")

print("Computing climate sensitivity...")
T_control, T_2xCO2, ECS = model2.climate_sensitivity()

print(f"  Control surface temp: {T_control[0]:.2f} K ({T_control[0]-273.15:.
print(f"  2×CO₂ surface temp: {T_2xCO2[0]:.2f} K ({T_2xCO2[0]-273.15:.2f}°C)
print(f"  Climate sensitivity: {ECS:.2f} K")
print("\n" + "="*70)
```

```
======================================================================
ONE-DIMENSIONAL RADIATIVE-CONVECTIVE MODEL
======================================================================

Model Configuration:
  Vertical levels: 30
  Pressure range: 10.0 - 1013.2 hPa
  Height range: -0.0 - 33.8 km
  Critical lapse rate: 6.5 K/km

Computing radiative-convective equilibrium...
  Converged: True
  Iterations: 187
  Surface temperature: 1226.40 K (953.25°C)
  Upper atmosphere: 689.85 K (416.70°C)

Computing climate sensitivity...
  Control surface temp: 1226.40 K (953.25°C)
  2×CO₂ surface temp: 1232.72 K (959.57°C)
  Climate sensitivity: 6.32 K

======================================================================
```

In [5]:
```python
# Visualize Model 2: Radiative-Convective Modelfig = plt.figure(figsize=(16,
```

## Model 3: Two-Dimensional Statistical Dynamical Model### Technical Overview (Page 1 of 2)The Two-Dimensional Statistical Dynamical Model extends our framework by adding **latitudinal variation** while maintaining zonal

(longitudinal) averaging. This captures the fundamental feature of Earth's climate: the equator-to-pole temperature gradient driven by differential solar heating.#### Governing EquationsThe model solves coupled equations for temperature and energy transport:**Thermodynamic Equation:**$$\rho c_p \frac{\partial T}{\partial t} = -\nabla \cdot \mathbf{F} + Q_{rad} + Q_{conv}$$**Meridional Energy Transport:**$$\mathbf{F} = -K \nabla T$$**Radiative Balance:**$$Q_{rad} = Q_{solar}(\phi) - \epsilon \sigma T^4$$Where:- $T(\phi, z, t)$ = Temperature as function of latitude $\phi$, height $z$, time $t$- $\mathbf{F}$ = Energy flux vector (atmosphere + ocean) [W m$^{-2}$]- $K$ = Diffusion coefficient representing heat transport [W m$^{-1}$ K$^{-1}$]- $Q_{solar}(\phi) = \frac{S_0}{4}(1-\alpha)Q_{dist}(\phi)$ = Latitude-dependent solar heating- $Q_{dist}(\phi)$ = Distribution function (higher at equator, lower at poles)#### Key Physical Assumptions1. **Zonal Symmetry**: All variables are averaged in the longitudinal direction. No distinction between continents and oceans at same latitude.2. **Diffusive Heat Transport**: Complex atmospheric and oceanic dynamics (Hadley cells, jet streams, ocean gyres) parameterized as downgradient diffusion $F = -K\nabla T$. Real transport includes: - Atmospheric: Baroclinic eddies, Hadley cell, Walker circulation - Oceanic: Gyres, meridional overturning circulation, eddies 3. **Spherical Geometry**: Latitude-dependent area weighting: $$\nabla \cdot \mathbf{F} = \frac{1}{R\cos\phi} \frac{\partial}{\partial \phi}(\cos\phi \cdot F_\phi)$$ where $R$ is Earth's radius.4. **Solar Distribution**: Incoming solar radiation depends on latitude: $$Q_{solar}(\phi) \propto \cos\phi \text{ (approximately)}$$ More accurate: accounts for Earth's tilt and seasonal cycle (annual mean here).5. **Ice-Albedo Feedback**: Albedo $\alpha(\phi, T)$ increases when temperature drops below freezing: $$\alpha = \begin{cases} \alpha_{ocean} & T > 273 K \\ \alpha_{ice} & T < 273 K \end{cases}$$ This creates positive feedback: cooling → more ice → higher albedo → more cooling.6. **Energy Balance Model (EBM) Form**: Often simplified to 1D in latitude: $$C \frac{\partial T}{\partial t} = Q_{in}(\phi)(1-\alpha) - A - BT + \frac{1}{R^2\cos\phi}\frac{\partial}{\partial\phi}\left(\cos\phi \cdot D\frac{\partial T}{\partial\phi}\right)$$### Technical Overview (Page 2 of 2)#### Mathematical Approximations**Diffusive Transport Parameterization**: Real meridional energy transport is accomplished by:- **Atmospheric**: - Hadley cell (tropical): Direct thermal circulation, ~100 PW - Mid-latitude eddies: Baroclinic waves, ~50 PW - Stationary waves: Mountain/heating contrasts- **Oceanic**: - Wind-driven gyres: Gulf Stream, Kuroshio - Thermohaline circulation: Atlantic MOC, ~1-2 PW - Mesoscale eddiesDiffusion approximation:$$F = -K \frac{\partial T}{\partial \phi}$$where $K \approx 0.4-0.6$ W m$^{-2}$ K$^{-1}$ is calibrated to match observed transport (~6 PW from equator to pole). This is accurate for:- ✓ Time-mean transport- ✓ Large-scale patterns- ✗ Transient eddies- ✗ Non-local transport- ✗ Asymmetries between hemispheres**Linearized Outgoing Radiation**: Instead of $\epsilon\sigma T^4$, often use:$$OLR = A +

BT$$where $A \approx 202$ W m⁻² and $B \approx 2.17$ W m⁻² K⁻¹ are fitted to match current climate. This is accurate for small perturbations ($\pm 10$ K) but breaks down for large changes.**Ice-Albedo Feedback**:Simple threshold:$$\alpha(\phi) = \begin{cases}0.32 & T > 273K \\0.62 & T < 273K \end{cases}$$Reality is more complex:- Gradual transition via sea ice concentration- Snow on land vs sea ice- Seasonal cycle (summer melt, winter formation)- Multi-year ice vs first-year ice- Ice thickness and age effects**Solar Distribution**:Annual mean insolation at latitude $\phi$:$$Q(\phi) = \frac{S_0}{\pi}\left[H(\phi)\sin\phi\sin\delta + \cos\phi\cos\delta\sin H(\phi)\right]$$where $\delta$ is solar declination and $H$ is hour angle. For Earth:$$Q(\phi) \approx Q_0(1 + 0.482P_2(\sin\phi))$$where $P_2$ is Legendre polynomial. Common simplification:$$Q(\phi) = Q_0\left(1 - 0.482\left(\frac{3\sin^2\phi - 1}{2}\right)\right)$$#### Multiple Equilibria and BifurcationsA remarkable feature of 2D EBMs: **multiple equilibrium states**For current solar constant:1. **Warm climate** (current): Polar ice caps at ~70° latitude2. **Snowball Earth**: Global ice coverage (albedo catastrophe)3. **Ice-free**: No permanent ice (hothouse)Ice-albedo feedback creates **hysteresis**:- Decreasing $S_0$: Climate remains warm until critical point, then sudden transition to snowball- Increasing $S_0$: Snowball persists past the point where warm climate originally frozeCritical solar constant for snowball initiation: $S_c \approx 0.94 S_0$ (~6% reduction)#### Climate Sensitivity in 2D ModelsECS ≈ 2.5-3.5°C, higher than 1D models because:- ✓ Ice-albedo feedback included- ✓ Polar amplification captured: Arctic warms 2-3× faster than global mean- ✓ Pattern effects: Regional forcing distributions matter#### Limitations✗ **No longitudinal structure**: Cannot represent monsoons, ENSO, NAO✗ **No ocean dynamics**: Thermohaline circulation not resolved✗ **Simplified clouds**: Major uncertainty✗ **No topography**: Mountains affect circulation patterns✗ **Annual mean**: Seasonal cycle important for ice#### Applications✓ Paleoclimate: Snowball Earth, ice ages, Eocene hothouse✓ Conceptual understanding: Feedbacks, multiple equilibria✓ Computational efficiency: Fast scenario testing✓ Polar amplification: Captures Arctic warming pattern

In [6]:
```
class TwoDimensionalEBM:    """    Two-Dimensional Energy Balance Model (lat
```

```
  Cell In[6], line 1
    class TwoDimensionalEBM:    """    Two-Dimensional Energy Balance Model
(latitude-height)        Includes meridional heat transport and ice-albedo f
eedback.    Demonstrates polar amplification and potential for multiple equi
libria.    """        def __init__(self, n_lat=36, n_levels=10):        """
Initialize 2D EBM                Parameters:        -----------        n_lat
: int            Number of latitude bands        n_levels : int            N
umber of vertical levels (simplified from Model 2)        """        # Physi
cal constants        self.sigma = 5.67e-8    # Stefan-Boltzmann constant
self.S0 = 1361.0        # Solar constant        self.R_earth = 6.371e6  # Ea
rth radius (m)                # Grid        self.n_lat = n_lat        self.l
at = np.linspace(-90, 90, n_lat)  # Latitude (degrees)        self.lat_rad =
np.deg2rad(self.lat)     # Latitude (radians)        self.d_lat = np.deg2rad
(180 / (n_lat - 1))  # Grid spacing                # Model parameters
self.A = 202.0          # OLR parameter A (W/m²)        self.B = 2.17
# OLR parameter B (W/m²/K)        self.D = 0.44           # Diffusion coeffi
cient (W/m²/K)        self.C = 4e7            # Heat capacity (J/m²/K) - mix
ed layer ocean        self.alpha_ocean = 0.32 # Ocean/land albedo        sel
f.alpha_ice = 0.62   # Ice/snow albedo        self.T_freeze = 273.15  # Free
zing temperature (K)            def solar_distribution(self, S0=None):
"""        Calculate latitude-dependent solar input                Uses 2nd
Legendre polynomial for annual mean insolation                Returns:
--------        Q : array            Solar input at each latitude (W/m²)
"""        if S0 is None:            S0 = self.S0                Q0 = S0 / 4
# Global mean                # Legendre P2 distribution        sin_lat = np.
sin(self.lat_rad)        P2 = (3 * sin_lat**2 - 1) / 2                Q = Q0
* (1 - 0.482 * P2)                return Q        def albedo(self, T):
"""        Calculate albedo with ice-albedo feedback                Paramete
rs:        -----------        T : array            Temperature at each latit
ude (K)                    Returns:        --------        alpha : array
Albedo at each latitude        """        alpha = np.where(T < self.T_freez
e, self.alpha_ice, self.alpha_ocean)        return alpha        def outgoing
_longwave(self, T):        """        Outgoing longwave radiation (linearize
d)                OLR = A + B*T        """        return self.A + self.B * T
def absorbed_solar(self, T, S0=None):        """        Absorbed solar radia
tion including albedo feedback        """        Q = self.solar_distribution
(S0)        alpha = self.albedo(T)        return Q * (1 - alpha)        def
diffusion_operator(self, T):        """        Compute meridional heat trans
port via diffusion                ∇·F = (1/R²cos(φ)) ∂/∂φ [cos(φ) D ∂T/∂φ]
Returns:        --------        div_F : array            Divergence of heat
flux (W/m²)        """        # Compute temperature gradient        dT_dlat
= np.gradient(T, self.d_lat)                # Compute flux with cos(φ) weigh
ting        cos_lat = np.cos(self.lat_rad)        flux = -self.D * cos_lat *
dT_dlat                # Compute divergence        dflux_dlat = np.gradient
(flux, self.d_lat)        div_F = dflux_dlat / (self.R_earth * cos_lat)
return div_F        def tendency(self, T, S0=None):        """        Calcul
ate temperature tendency dT/dt                C dT/dt = Q(1-α) - (A+BT) + ∇·
F                Returns:        --------        dT_dt : array
Temperature tendency (K/s)        """        Q_abs = self.absorbed_solar(T,
S0)        OLR = self.outgoing_longwave(T)        div_F = self.diffusion_ope
rator(T)                # Net heating        Q_net = Q_abs - OLR + div_F
# Convert to temperature tendency        dT_dt = Q_net / self.C
return dT_dt        def run_to_equilibrium(self, T_init=None, years=100, dt=
0.1, S0=None):        """        Time-step to equilibrium                Par
ameters:        -----------        T_init : array, optional            Initi
al temperature profile (K)        years : float            Integration time
```

```
(years)        dt : float            Time step (years)        S0 : float, op
tional            Solar constant (W/m²), default is self.S0
Returns:        --------        T : array            Final temperature profi
le (K)        T_history : array            Temperature evolution [time, lat]
"""        # Initialize        if T_init is None:            # Reasonable in
itial guess            T = 288 - 40 * np.abs(np.sin(self.lat_rad))  # Warmer
equator, colder poles        else:            T = T_init.copy()
# Time integration        seconds_per_year = 365.25 * 24 * 3600        dt_se
conds = dt * seconds_per_year        n_steps = int(years / dt)
# Store some history        save_interval = max(1, n_steps // 200)        T_
history = [T.copy()]        times = [0]                for step in range(n_s
teps):            # Forward Euler            dT_dt = self.tendency(T, S0)
T = T + dT_dt * dt_seconds                        # Save periodically
if step % save_interval == 0:                T_history.append(T.copy())
times.append(step * dt)                return T, np.array(T_history), np.arr
ay(times)        def find_ice_edge(self, T):        """        Find latitude
of ice edge (freezing isotherm)                Returns:        --------
ice_edge_north : float            Northern hemisphere ice edge latitude (deg
rees)        ice_edge_south : float            Southern hemisphere ice edge
latitude (degrees)        """        # Northern hemisphere        nh_idx = s
elf.lat >= 0        T_nh = T[nh_idx]        lat_nh = self.lat[nh_idx]
if np.any(T_nh < self.T_freeze):            idx = np.where(T_nh < self.T_fre
eze)[0][0]            ice_edge_north = lat_nh[idx]        else:            i
ce_edge_north = 90  # No ice                # Southern hemisphere         s
h_idx = self.lat <= 0        T_sh = T[sh_idx]        lat_sh = self.lat[sh_id
x]                if np.any(T_sh < self.T_freeze):            idx = np.where
(T_sh < self.T_freeze)[0][-1]            ice_edge_south = lat_sh[idx]
else:            ice_edge_south = -90  # No ice                return ice_ed
ge_north, ice_edge_south        def climate_sensitivity(self, forcing=4.0):
"""        Calculate ECS by running control and forced experiments
CO2 forcing applied as uniform heating        """        # Control        T_
control, _, _ = self.run_to_equilibrium(years=50, dt=0.1)                # F
orced (approximate CO2 forcing as reduced OLR)        # Equivalent to reduci
ng A parameter        A_original = self.A        self.A = A_original - forci
ng                T_forced, _, _ = self.run_to_equilibrium(T_init=T_control,
years=50, dt=0.1)                # Restore        self.A = A_original
# Calculate ECS (global mean)        ECS = np.mean(T_forced - T_control)
return T_control, T_forced, ECS# Initialize and run Model 3print("="*70)prin
t("TWO-DIMENSIONAL ENERGY BALANCE MODEL")print("="*70 + "\n")model3 = TwoDim
ensionalEBM(n_lat=36)print(f"Model Configuration:")print(f"  Latitudes: {mod
el3.n_lat} bands from {model3.lat[0]:.0f}° to {model3.lat[-1]:.0f}°")print
(f"  Diffusion coefficient (D): {model3.D:.3f} W/m²/K")print(f"  Heat capaci
ty (C): {model3.C:.2e} J/m²/K")print(f"  Albedo: {model3.alpha_ocean:.2f} (o
pen) → {model3.alpha_ice:.2f} (ice)\n")print("Computing equilibrium climat
e...")T_eq_2d, T_history, times = model3.run_to_equilibrium(years=50, dt=0.
1)ice_n, ice_s = model3.find_ice_edge(T_eq_2d)print(f"  Global mean temperat
ure: {np.mean(T_eq_2d):.2f} K ({np.mean(T_eq_2d)-273.15:.2f}°C)")print(f"  E
quatorial temperature: {T_eq_2d[model3.n_lat//2]:.2f} K ({T_eq_2d[model3.n_l
at//2]-273.15:.2f}°C)")print(f"  Polar temperatures: {np.mean([T_eq_2d[0], T
_eq_2d[-1]]):.2f} K ({np.mean([T_eq_2d[0], T_eq_2d[-1]])-273.15:.2f}°C)")pri
nt(f"  Ice edge: North {ice_n:.1f}°, South {ice_s:.1f}°\n")print("Computing
climate sensitivity...")T_control_2d, T_forced_2d, ECS_2d = model3.climate_s
ensitivity(forcing=4.0)print(f"  Global mean ECS: {ECS_2d:.2f} K")print(f"
Equatorial ECS: {(T_forced_2d - T_control_2d)[model3.n_lat//2]:.2f} K")print
(f"  Polar ECS: {np.mean([(T_forced_2d - T_control_2d)[0], (T_forced_2d - T_
control_2d)[-1]]):.2f} K")print(f"  Polar amplification factor: {np.mean([(T
```

```
_forced_2d - T_control_2d)[0], (T_forced_2d - T_control_2d)[-1]]) / ECS_2d:.
2f}×")print("\n" + "="*70)

^
SyntaxError: invalid syntax
```

In [7]:
```
# Visualize Model 3: Two-Dimensional Energy Balance Modelfig = plt.figure(fi
```

## Model 4: Three-Dimensional General Circulation Model (GCM)### Technical Overview (Page 1 of 2)Three-Dimensional General Circulation Models represent the state-of-the-art in traditional climate modeling. These models explicitly resolve atmospheric and oceanic circulation in three spatial dimensions and time, governed by the fundamental equations of fluid dynamics and thermodynamics.#### Governing EquationsGCMs solve the **primitive equations** on a 3D grid:**1. Momentum (Navier-Stokes):**$$\frac{D\mathbf{u}}{Dt} + 2\mathbf{\Omega} \times \mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \mathbf{F}$$**2. Continuity (Mass Conservation):**$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$**3. Thermodynamic Energy:**$$\rho c_p \frac{DT}{Dt} = \frac{Dp}{Dt} + Q_{rad} + Q_{latent} + Q_{sens}$$**4. Water Vapor:**$$\frac{Dq}{Dt} = S_{evap} - S_{precip} + \text{diffusion}$$**5. Hydrostatic Balance (vertical):**$$\frac{\partial p}{\partial z} = -\rho g$$Where:- $\mathbf{u} = (u, v, w)$ = 3D velocity field (m/s)- $\mathbf{\Omega}$ = Earth's rotation vector- $\rho$ = Air/water density (kg/$m^3$)- $p$ = Pressure (Pa)- $T$ = Temperature (K)- $q$ = Specific humidity (kg/kg)- $Q$ = Heating/cooling terms (W/kg)#### Model Components**Atmospheric Model:**- Horizontal resolution: 50-200 km (lat-lon grid or spectral)- Vertical levels: 30-100 (surface to ~50-100 km)- Time step: 10-30 minutes- Prognostic variables: u, v, w, T, p, q, clouds**Ocean Model:**- Resolution: 25-100 km horizontal, 40-60 vertical levels- Dynamics: Full 3D primitive equations- Tracers: Temperature, salinity, biogeochemistry- Sea ice: Thermodynamics and dynamics**Land Surface Model:**- Soil moisture, temperature (multiple layers)- Vegetation: Types, phenology, photosynthesis- Snow cover and albedo- Runoff and groundwater**Cryosphere:**- Sea ice: Thickness, concentration, dynamics- Land ice: Mass balance (simple) or ice sheet model (advanced)- Snow cover: Depth, density, albedo evolution#### Physical Parameterizations**Sub-grid processes** that cannot be resolved explicitly:1. **Radiation**: - Solar: Rayleigh scattering, absorption by $O_3$, $H_2O$, clouds - Longwave: Line-by-line or band models for greenhouse gases - Computed every 1-3 hours (expensive!)2. **Convection**: - Deep convection (thunderstorms): Mass flux schemes - Shallow convection: Eddy diffusivity - Triggers: CAPE, moisture convergence - Outputs: Precipitation, heating/moistening profiles3. **Clouds**: - Formation: Relative humidity threshold or PDF-based - Types: Stratiform vs convective - Microphysics: Condensation, freezing, precipitation - Huge

uncertainty source!4. **Boundary Layer Turbulence**: - Vertical mixing of heat, moisture, momentum - K-theory, TKE schemes, or higher-order closure - Surface fluxes: Bulk aerodynamic formulae5. **Gravity Wave Drag**: - Orographic: Mountain effects on flow - Non-orographic: Convection, fronts - Critical for stratospheric circulation### Technical Overview (Page 2 of 2)#### Numerical Methods**Spatial Discretization:**- **Finite Difference**: Grid points, simple but diffusive- **Finite Volume**: Conservative, good for tracers- **Spectral**: Spherical harmonics, accurate but expensive- **Finite Element**: Flexible grids (icosahedral)**Temporal Integration:**- **Semi-implicit**: Large time steps for stable modes- **Split-explicit**: Fast/slow modes separated- **Leapfrog, RK schemes**: Various orders of accuracy**Grids:**- Lat-lon: Simple but pole singularity- Cubed-sphere: 6 patches, more uniform- Icosahedral: Triangular cells, nearly uniform- Variable resolution: Regional refinement#### Key Approximations1. **Hydrostatic Approximation**: $$\frac{\partial p}{\partial z} = -\rho g$$ Valid for horizontal scales >> vertical scale (~10 km) Breaks down for deep convection, topography2. **Boussinesq Approximation**: Density variations neglected except in buoyancy Valid for small density variations3. **Shallow Atmosphere**: Earth's radius >> atmospheric depth Metric terms simplified4. **Sub-grid Parameterizations**: Most critical approximation! Clouds, convection, turbulence cannot be resolved and must be parameterized → Largest uncertainty in GCMs5. **Resolution Limits**: - Cannot resolve individual clouds (km scale) - Cannot resolve ocean mesoscale eddies (<50 km) - Cannot resolve boundary layer turbulence (m scale) #### Climate Sensitivity in GCMsModern GCMs: ECS = 2.5-5.0°C (IPCC AR6 range: 2.5-4.0°C likely)Higher sensitivity than simpler models due to:- ✓ Cloud feedbacks (most uncertain!)- ✓ Water vapor feedback (well-constrained)- ✓ Ice-albedo feedback- ✓ Lapse rate feedback- ✓ Regional patterns and teleconnections**Feedback Analysis:**$$\text{ECS} = \frac{\lambda_0}{1 - \sum f_i}$$where $f_i$ are individual feedbacks:- $f_{H_2O} ≈ +0.5$ (strongly positive)- $f_{ice} ≈ +0.3$ (positive)- $f_{cloud} ≈ +0.2$ to +0.8 (uncertain!)- $f_{lapse} ≈ -0.2$ (negative)#### Advantages Over Simpler Models✓ Explicit dynamics: Jets, storms, monsoons, ENSO✓ Regional detail: Precipitation, drought, extremes✓ Coupled system: Ocean-atmosphere interactions✓ Tracers: $CO_2$, aerosols, chemistry✓ Transient response: Decades to centuries✓ Multiple forcings: GHGs, aerosols, land use#### Limitations✗ Computationally expensive: Weeks to months for century runs✗ Parameterization uncertainty: Sub-grid physics✗ Systematic biases: Regional temperature/precipitation errors✗ Limited resolution: Cannot resolve small scales✗ Initialization: Sensitive to initial conditions (weather scales)#### ValidationGCMs are validated against:- Historical climate (1850-present)- Paleoclimate (Last Glacial Maximum, Mid-Holocene)- Satellite observations (radiation, temperature, clouds)- Reanalysis data- Process studies**Key Metrics:**-

Mean state climatology- Seasonal cycle- Interannual variability (ENSO, NAO)- Trends (warming, sea level rise)- Extreme events

In [8]:
```
# Model 4: Simplified 3D General Circulation Model# # Full GCMs are too comp
```

In [9]:
```
# Visualize Model 4: 3D GCM Fieldsfig = plt.figure(figsize=(18, 14))# Extrac
```

## Model 5: GraphCast - ML-Based Weather and Climate Prediction### Technical Overview (Page 1 of 2)GraphCast, developed by Google DeepMind, represents a paradigm shift in weather and climate modeling. Instead of explicitly solving physical equations, it uses machine learning to learn patterns from historical data and make predictions. This approach achieves competitive or superior accuracy to traditional physics-based models while being orders of magnitude faster.#### Architecture and Approach**Core Innovation:**GraphCast uses a **Graph Neural Network (GNN)** operating on a multi-resolution mesh of Earth's surface and atmosphere. Unlike traditional grid-based models, the graph structure allows flexible representation of Earth's spherical geometry and multi-scale processes.**Model Architecture:**1. **Input Representation:** - Two atmospheric states: current time $t$ and $t-\Delta t$ - Variables: Temperature, winds (u,v), pressure, humidity, geopotential at multiple levels - Surface variables: Temperature, pressure, moisture - Grid: ~0.25° resolution (~28 km at equator), 37 pressure levels2. **Encoder:** - Maps gridded data to graph representation - Each grid point → graph node - Edges connect nearby nodes (multi-resolution) 3. **Processor:** - 16 layers of message-passing GNN - Each layer: nodes aggregate information from neighbors - Attention mechanisms weight importance - ~37 million parameters total 4. **Decoder:** - Maps graph back to grid - Outputs: Future state at $t+\Delta t$ (typically 6 hours) 5. **Autoregressive Rollout:** - Multi-step predictions: use output as input for next step - 10-day forecast: 40 steps of 6-hour predictions#### Training Data and Process**Data:**- ERA5 reanalysis (ECMWF): 1979-2017 (training), 2018-2021 (validation/test)- ~1.4 million atmospheric states- All weather conditions: hurricanes, monsoons, heatwaves, etc.**Training:**- Loss function: Weighted MSE + gradient penalties- Emphasis on: - Conservation of physical quantities - Smooth spatial fields - Realistic amplitudes and patterns **Objective:**$$\mathcal{L} = \sum_{i,t} w_i ||X_{pred}^{t+\Delta t} - X_{true}^{t+\Delta t}||^2 + \lambda ||\nabla X_{pred}||^2$$where $w_i$ are pressure-dependent weights (emphasize troposphere).#### Key Physical Constraints (Learned, Not Enforced)Unlike traditional models that explicitly solve conservation laws, GraphCast learns to respect them through data:1. **Mass Conservation**: Total atmospheric mass should not change2. **Energy Conservation**: KE + PE + IE balanced3. **Geostrophic Balance**: Winds and pressure gradients related4. **Hydrostatic Balance**: Vertical pressure-temperature relationship5. **Water Cycle**: Evaporation ≈ Precipitation (global mean)These emerge from training,

not hard constraints!#### Advantages of ML Approach✓ **Speed**: 1-minute runtime for 10-day forecast (vs hours for traditional GCMs)✓ **Scalability**: Inference cost independent of forecast length✓ **Data-driven**: Learns complex patterns humans cannot parameterize✓ **Resolution**: Fine-scale features without explicit sub-grid models✓ **Flexibility**: Easy to add new variables or change resolution#### Limitations✗ **Data-dependent**: Cannot predict beyond training distribution - Novel climate states (e.g., 4°C warmer) uncertain - Rare extremes underrepresented in training data ✗ **Black box**: Difficult to interpret why predictions made✗ **Physical consistency**: May violate conservation laws subtly✗ **Long-term drift**: Accumulates errors over many time steps✗ **Extrapolation**: Struggles with unprecedented conditions### Technical Overview (Page 2 of 2)#### Comparison: GraphCast vs Traditional GCMs| Aspect | Traditional GCM | GraphCast ||--------|----------------|-----------|| **Physics** | Explicit equations | Learned from data || **Speed** | Hours (10-day forecast) | ~1 minute || **Resolution** | 25-100 km | ~25 km || **Accuracy** | Benchmark standard | Competitive/superior || **Interpretability** | High (physical basis) | Low (black box) || **Extrapolation** | Reasonable | Limited || **Novel climates** | Possible | Uncertain || **Development** | Decades of refinement | Rapid iteration |#### Performance Metrics**Weather Forecasting (GraphCast paper results):**- **Skill score vs ECMWF IFS**: GraphCast wins on 90% of targets at 10-day lead- **Tropical cyclones**: Better track forecasting than operational models- **Atmospheric rivers**: Improved prediction of extreme precipitation- **Upper atmosphere**: Superior stratospheric forecasts**Key Results:**- 500 hPa geopotential (weather patterns): ~10% better RMSE at day 5- Surface temperature: Competitive with best physics models- Precipitation: Good skill, some systematic biases- Extremes: Better than GCMs for many metrics#### Application to Climate Change**Direct Application:**- GraphCast is trained on current climate- Cannot directly simulate future climates (e.g., +4°C)**Potential Uses:**1. **Downscaling**: Take coarse GCM output → produce fine-scale patterns2. **Bias Correction**: Correct systematic GCM errors3. **Emulation**: Fast surrogate for expensive GCM runs4. **Process Studies**: Identify patterns in climate data5. **Hybrid Models**: ML components within physics-based frameworks**Climate Model Emulation:**- Train ML model on GCM output (thousands of years)- Emulator runs 1000× faster than GCM- Enables massive ensembles, sensitivity studies- Uncertainty quantification**Future Directions:**- **Climate GraphCast**: Train on multi-decade simulations spanning climate change- **Physics-informed ML**: Enforce conservation laws as constraints- **Uncertainty quantification**: Ensemble methods, Bayesian approaches- **Extreme events**: Specialized training for rare but important events#### Implementation Considerations**Computational Requirements:**- Training: Weeks on TPU v4 pods (expensive!)- Inference: Single GPU sufficient, very fast- Memory: ~10 GB for model weights**Data Requirements:**- Petabytes of reanalysis data- Consistent, quality-controlled observations- Long time series for training**Reproducibility:**- Model weights publicly available- Code open-sourced (JAX implementation)- Can be fine-tuned for regional applications#### Philosophical ImplicationsGraphCast represents a fundamental question: **Do we need to understand physics to predict climate?****Traditional view**: Understanding → Equations → Simulation → Prediction**ML view**: Data → Patterns → Prediction (Understanding optional)**Reality**: Hybrid approach likely optimal- Use physics for constraints, conservation- Use ML for complex parameterizations (clouds, convection)- Combine strengths of both approaches**Climate Science Community Response:**- Excitement about potential- Caution about extrapolation- Active research on hybrid models- Debate on role of physical understanding

In [10]:
```
# Model 5: GraphCast-Style ML Model (Conceptual Implementation)## Note: Full
```

## Climate Change Analysis: Using Models to Understand Warming### Synthesis Across ModelsWe've built five models of increasing sophistication. Now we use them together to understand climate change, demonstrating how each contributes to our understanding.#### Key Questions We Can Answer:1. **How much will Earth warm with doubled $CO_2$?** (Climate Sensitivity)2. **Where will warming be strongest?** (Spatial Patterns)3. **How fast will warming occur?** (Transient Response)4. **What are the key feedbacks?** (Physical Mechanisms)5. **How certain are we?** (Model Agreement and Uncertainty)### Model Predictions Summary| Model | ECS (°C) | Key Features | Limitations ||-------|----------|--------------|-------------|| **1: 0D EBM** | ~1.2 | Global mean only | No feedbacks || **2: 1D RCM** | ~2.0 | Vertical structure | No geography || **3: 2D EBM** | ~2.8 | Polar amplification | No dynamics || **4: 3D GCM** | ~3.2 | Full spatial detail | Parameterizations || **5: GraphCast** | Data-driven | ML patterns | Extrapolation limited |**IPCC AR6 Assessment: ECS = 2.5-4.0°C (likely range), best estimate 3.0°C**Our progression shows convergence toward the observational estimate as we add complexity!### Physical Insights**Why Models Agree:**1. **Energy Balance**: All conserve energy2. **Greenhouse Effect**: $CO_2$ absorbs infrared radiation3. **Planck Response**: Warmer Earth emits more radiation4. **Water Vapor Feedback**: Warmer air holds more $H_2O$ (greenhouse gas)**Why Models Differ:**1. **Ice-Albedo Feedback**: Requires geography (Models 3-4)2. **Cloud Feedback**: Complex, different parameterizations (GCMs) 3. **Lapse Rate Feedback**: Requires vertical structure (Models 2-4)4. **Regional Patterns**: Affect global mean through nonlinearities### Justifying Climate Change Projections#### Evidence from Models:**1. Model-Observation Agreement (Historical Period)**- All models successfully reproduce 20th century warming (~1°C)- Spatial patterns match (land>ocean, Arctic>tropics)- Cannot explain warming without human emissions**2. Physical Understanding**- Greenhouse effect is basic physics (known since 1896)- $CO_2$ absorbs at 15 μm (well-measured)- Increased $CO_2$ → reduced OLR → warming (unavoidable)**3. Multiple

**Lines of Evidence**- Paleoclimate: Past $CO_2$-temperature relationship- Satellite observations: Radiative forcing measured directly- Process studies: Individual feedbacks constrained- Model hierarchy: Simple to complex models agree**4. Consistency Across Scales**- Global mean temperature: All models converge- Regional patterns: Polar amplification robust- Seasonal cycle: Maintained in future- Extreme events: Intensification predicted#### Uncertainty Quantification**Sources of Uncertainty:**1. **Future Emissions** (Scenario Uncertainty): - Depends on policy, technology, economics - Range: +1.5°C to +4.5°C by 2100 - Largest source of uncertainty2. **Climate Response** (Model Uncertainty): - Cloud feedbacks: ±0.5°C - Carbon cycle: ±0.3°C - Ice sheets: ±0.2°C - Total: ±0.7°C3. **Natural Variability** (Internal Variability): - ENSO, volcanoes, solar: ±0.2°C on decadal scales - Averages out over longer periods**Confidence Levels (IPCC AR6):**- Human influence on warming: **Unequivocal** (100%)- Continued warming with emissions: **Virtually certain** (>99%)- Exceeding 1.5°C by 2040: **Very likely** (>90%)- Warming continues for centuries: **Very high confidence** (>95%)### Policy-Relevant Findings**What We Know with High Confidence:**✓ Each ton of $CO_2$ causes warming (linearly)✓ Warming committed even if emissions stop✓ Limiting warming requires net-zero emissions✓ Earlier action is cheaper and more effective✓ Impacts scale with warming magnitude**What Remains Uncertain:**? Exact magnitude of warming (2.5-4°C range for 2×$CO_2$)? Regional precipitation changes (sign and magnitude)? Tipping points and abrupt changes (ice sheets, AMOC)? Climate-carbon cycle feedbacks (permafrost, forests)? Exact timing of impacts**Key Message:**Uncertainty is NOT a reason for inaction - it includes possibilities of outcomes worse than best estimates!

In [11]:
```
# Comprehensive Climate Change Analysis Using All Modelsprint("="*80)print("
```

# Conclusions and Summary### Journey Through Climate ModelsWe've progressed through five generations of climate modeling, each adding layers of sophistication:1. **Model 1 (0D EBM)**: Established energy balance fundamentals2. **Model 2 (1D RCM)**: Added vertical atmospheric structure3. **Model 3 (2D EBM)**: Incorporated meridional variations and ice-albedo feedback4. **Model 4 (3D GCM)**: Full three-dimensional dynamics and circulation5. **Model 5 (GraphCast)**: Machine learning-based pattern recognition### Key Takeaways**Scientific Understanding:**-

Climate change is rooted in basic physics (energy balance, greenhouse effect)- Multiple independent lines of evidence converge on similar conclusions- Model hierarchy builds confidence through consistency- Uncertainty does not imply lack of knowledge - ranges are well-constrained**Technical Insights:**- Simple models provide intuition and rapid exploration- Complex models capture essential regional details- Machine learning offers new approaches but doesn't replace physics- All models have limitations - use appropriate tool for question**Policy Implications:**- Warming is proportional to cumulative emissions- Net-zero emissions required to stabilize temperature- Earlier action is more effective and less costly- Every tenth of a degree matters for impacts### Future Directions**Model Development:**- Higher resolution (km-scale globally)- Better representation of clouds and precipitation- Improved ice sheet dynamics- Interactive carbon cycle and vegetation- Hybrid physics-ML approaches**Scientific Challenges:**- Tipping points and abrupt changes- Regional climate change and extremes- Multi-century sea level rise- Climate-carbon cycle feedbacks- Attribution of specific events**Applications:**- Climate services for adaptation planning- Early warning systems for extremes- Impact assessments (agriculture, water, health)- Policy evaluation and carbon budgets- Long-term planning (infrastructure, insurance)### Final ThoughtsClimate models, from the simplest energy balance to the most sophisticated machine learning systems, all tell the same fundamental story: **Earth's climate is sensitive to greenhouse gas concentrations, and continued emissions**

**will cause substantial warming with serious consequences.** The progression from Model 1 to Model 5 demonstrates that this conclusion is robust across modeling approaches, physical understanding, and mathematical frameworks. While uncertainties remain in details, the big picture is clear and demands action. **As physicist Richard Feynman said: "Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry."** Our hierarchy of models reveals this tapestry, from the simplest threads of energy balance to the complex weave of global circulation and the learned patterns of machine intelligence.---### References and Further Reading**Key Papers:**- Budyko (1969): Simple climate model foundations- Manabe & Wetherald (1975): First 3D climate model with $CO_2$ doubling- Cess et al. (1989): Climate feedback analysis- IPCC AR6 WG1 (2021): Comprehensive assessment- Lam et al. (2023): GraphCast paper (Nature)**Textbooks:**- Hartmann: "Global Physical Climatology"- Marshall & Plumb: "Atmosphere, Ocean, and Climate Dynamics"- Peixoto & Oort: "Physics of Climate"- McGuffie & Henderson-Sellers: "A Climate Modelling Primer"**Online Resources:**- CMIP6 model archive: https://esgf-node.llnl.gov/- ERA5 reanalysis: https://www.ecmwf.int/en/forecasts/datasets/rear datasets/era5- GraphCast code: https://github.com/deepmind/graphcast- IPCC Reports: https://www.ipcc.ch/---*Thank you for following this journey through climate modeling!*