Catatan Kuliah
Dasar Sistem Tenaga Listrik
Minggu 13: Tren dan Inovasi Masa Depan Sistem Tenaga
Menuju Grid Cerdas dan Berkelanjutan

Universitas Bengkulu
Fakultas Teknik - Teknik Elektro

---

1 Pendahuluan

Revolusi industri 4.0 dan komitmen global terhadap Net Zero Emissions (NZE) 2060 membawa transformasi fundamental dalam sistem tenaga listrik dunia. Indonesia, dengan posisi strategis di jantung Asia Tenggara dan komitmen kuat terhadap energi terbarukan, menghadapi tantangan dan peluang unik dalam mengembangkan sistem tenaga masa depan [**iea_indonesia_2023**].

Permatasan dan Peluang Bengkulu:
Provinsi Bengkulu, dengan karakteristik geografis coastal, potensi energi terbarukan yang signifikan (3,475 MW solar potential [**potensi_bengkulu_2023**]), dan sebagai bagian integral dari sistem JAMALI, memiliki peran krusial dalam achieve target renewable energy 23% pada 2025 dan NZE 2060 [**ruptl_pln_2021**].

Delapan Pilar Transformasi Sistem Tenaga Modern:

1. **Smart Grid**: Grid pintas dengan observabilitas dan kontrol real-time
2. **Integrasi Energi Terbarukan**: Pengelolaan intermitensi pada skala besar
3. **Energy Storage Systems (ESS)**: Penyimpanan energi untuk fleksibilitas grid
4. **Building Energy Management System (BEMS)**: Efisiensi energi gedung
5. **Electric Vehicles dan V2G**: Transpor elektrifikasi dan penyimpanan terdistribusi
6. **HVDC Systems**: Transmisi jarak jauh dan interkoneksi regional
7. **Cybersecurity**: Perlindungan infrastruktur energi kritis
8. **Regulatory Framework**: Kerangka hukum dan regulasi modern

2 Smart Grid: Backbone Sistem Tenaga Masa Depan

2.1 Definisi dan Arsitektur Smart Grid

Smart Grid didefinisikan sebagai jaringan listrik yang mengintegrasikan infrastruktur fisik dengan teknologi digital, komputasi, dan komunikasi untuk menghasilkan operasi yang lebih andal, efisien, aman, dan rendah karbon [**smart_grid_us_dept_2024**].

Definisi Matematis Smart Grid:
Smart Grid dapat dimodelkan sebagai sistem hibrida energi-informasi:

$$S_G = \{\mathcal{P}, \mathcal{C}, \mathcal{I}, \mathcal{M}\} \tag{1}$$

Dimana:

- $\mathcal{P}$ = Conjunto física (generators, transmission lines, consumers)
- $\mathcal{C}$ = Conjunto komunikacional (AMI, PMU, sensor networks)
- $\mathcal{I}$ = Conjunto informasi (data analytics, AI/ML algorithms)
- $\mathcal{M}$ = Conjunto manajemen (optimal control, demand response)

2.2 Advanced Metering Infrastructure (AMI)

AMI merupakan tulang punggung observabilitas sisi pelanggan dalam Smart Grid, consisting of smart meters, communication networks, dan data management systems.

Arquitectura AMI:

$$AMI = \{MT, CN, DMS, MDMS\} = \begin{cases} MT & \text{Smart Meter (1-5 tahun lifetime, 1\% accuracy)} \\ CN & \text{Communication Network (GPRS, RF Mesh, PLC)} \\ DMS & \text{Data Management System} \\ MDMS & \text{Meter Data Management System} \end{cases} \tag{2}$$

Karakteristik Teknis AMI Global:

Tabel 1: AMI Deployment Statistics 2024

Region	Penetrasi (%)	Jumlah Meter (juta)	Tahun Deployment
Amerika Serikat	72%	110.2	2020-2024
Uni Eropa	85%	180.5	2019-2024
China	95%	450.8	2018-2024
Indonesia	8.5%	2.1	2022-2024
Bengkulu	12.3%	0.15	2023-2024

Benefits AMI untuk Bengkulu:

1. **Remote Connect/Disconnect**: Operasi 24/7 tanpa field visit
2. **Consumption Analytics**: Behavioral analysis untuk demand response
3. **Outage Management**: Deteksi otomatis gangguan dan lokasinya

4. **Loss Reduction**: Identifikasi non-technical losses
5. **Time-of-Use Pricing**: Dynamic pricing untuk load shifting

Kode Python untuk Simulasi AMI Data Analysis:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

class AMIDataAnalyzer:
    def __init__(self):
        self.sampling_interval = 15  # minutes
        self.accuracy = 0.01  # 1%

    def generate_consumption_profile(self, days=30):
        """Generate realistic consumption profile for Bengkulu"""
        time_points = int(days * 24 * 60 / self.sampling_interval)

        # Base load + daily pattern + random variation
        base_load = np.ones(time_points) * 500  # Wh

        # Daily pattern (Bengkulu typical)
        daily_pattern = np.zeros(time_points)
        for i in range(time_points):
            hour = (i * self.sampling_interval) % 1440 // 60
            if 18 <= hour <= 22:  # Peak evening hours
                daily_pattern[i] = 1.8
            elif 12 <= hour <= 14:  # Lunch peak
                daily_pattern[i] = 1.3
            elif 0 <= hour <= 6:  # Night minimum
                daily_pattern[i] = 0.6
            else:  # Regular hours
                daily_pattern[i] = 1.0

        # Add solar contribution (for rooftop PV)
        solar_contribution = np.zeros(time_points)
        for i in range(time_points):
            hour = (i * self.sampling_interval) % 1440 // 60
            if 6 <= hour <= 18:
                solar_contribution[i] = 300 * np.sin(np.pi * (hour - 6) / 12)

        # Combine and add noise
        consumption = (base_load * daily_pattern + solar_contribution +
                       np.random.normal(0, 50, time_points))

        # Ensure non-negative
        consumption = np.maximum(consumption, 0)
```

```
        return consumption

    def analyze_load_pattern(self, consumption_data):
        """Analyze consumption patterns for demand response"""
        peak_threshold = np.percentile(consumption_data, 85)
        peak_hours = np.where(consumption_data > peak_threshold)[0]

        # Calculate metrics
        daily_peak = np.max(consumption_data)
        daily_average = np.mean(consumption_data)
        load_factor = daily_average / daily_peak

        return {
            'peak_threshold': peak_threshold,
            'peak_hours_percentage': len(peak_hours) / len(consumption_data) * 100,
            'load_factor': load_factor,
            'daily_peak': daily_peak,
            'daily_average': daily_average
        }

# Example usage for Bengkulu residential customer
ami_analyzer = AMIDataAnalyzer()
consumption = ami_analyzer.generate_consumption_profile(days=30)
metrics = ami_analyzer.analyze_load_pattern(consumption)

print(f"Load Factor: {metrics['load_factor']:.3f}")
print(f"Peak Hours Percentage: {metrics['peak_hours_percentage']:.1f}%")
```

2.3 Phasor Measurement Units (PMU)

PMU merupakan perangkat critical untuk observabilitas dinamis sistem tenaga, providing synchronized phasor measurements dengan akurasi tinggi dan sampling rate hingga 120 samples/second.

Fundamental PMU Equations:

Phasor representation dari sinusoidal signal:

$$x(t) = A\cos(\omega t + \phi) = \Re\{\underline{X}e^{j\omega t}\} \tag{3}$$

Dimana $\underline{X} = Ae^{j\phi}$ adalah complex phasor.

PMU measurement dengan GPS synchronization:

$$\underline{X}_{PMU}(t) = \frac{\sqrt{2}}{T}\int_{t-T}^{t} x(\tau)e^{-j\omega_0\tau}d\tau \tag{4}$$

Dimana $\omega_0 = 2\pi \cdot 50$ rad/s untuk sistem 50 Hz.

PMU Deployment Statistics:

Tabel 2: PMU Global Deployment 2024

Country/Region	PMU Units	Sampling Rate	Accuracy	Application
Amerika Serikat	1,847	30-120 Hz	±0.1%	WAMS
China	2,156	50-100 Hz	±0.05%	State estimation
India	487	25-50 Hz	±0.1%	Grid monitoring
Indonesia (JAMALI)	24	50 Hz	±0.1%	Pilot deployment
Bengkulu Region	3	50 Hz	±0.1%	Research pilot

PMU Applications untuk Sistem Bengkulu:

1. **State Estimation**: Real-time system state monitoring
2. **Voltage Stability Assessment**: PV and QV curve monitoring
3. **Angle Stability Analysis**: Generator angle difference tracking
4. **Islanding Detection**: Automatic detection of grid separation
5. **Frequency Disturbance Analysis**: Rate of change of frequency (ROCOF)

2.4 Self-Healing Grid

Self-healing grid merupakan kemampuan sistem untuk mendeteksi, mengisolasi, dan memulihkan gangguan secara otomatis tanpa intervensi manual.

Self-Healing Algorithm Framework:

$$SHG = \{D, I, R, O\} \quad \text{where} \begin{cases} D = \{F_{detection}, F_{localization}\} \\ I = \{F_{isolation}, F_{containment}\} \\ R = \{F_{reconfiguration}, F_{restoration}\} \\ O = \{F_{optimization}, F_{learning}\} \end{cases} \tag{5}$$

Three-Phase Self-Healing Process:

1. **Fault Detection** ($t = 0 - 0.1$ s):

$$P_{fault} = |V_{diff}| > \delta_{th} \quad \text{or} \quad I_{fault} > I_{th} \tag{6}$$

2. **Fault Isolation** ($t = 0.1 - 0.3$ s):

$$\Omega_{isolated} = \arg\min_{i} \{d(fault, node_i) | node_i \in \Omega_{feeders}\} \tag{7}$$

3. **Service Restoration** ($t = 0.3 - 2.0$ s):

$$\Omega_{restored} = \{nodes | path \neq \emptyset \text{ and } capacity > demand\} \tag{8}$$

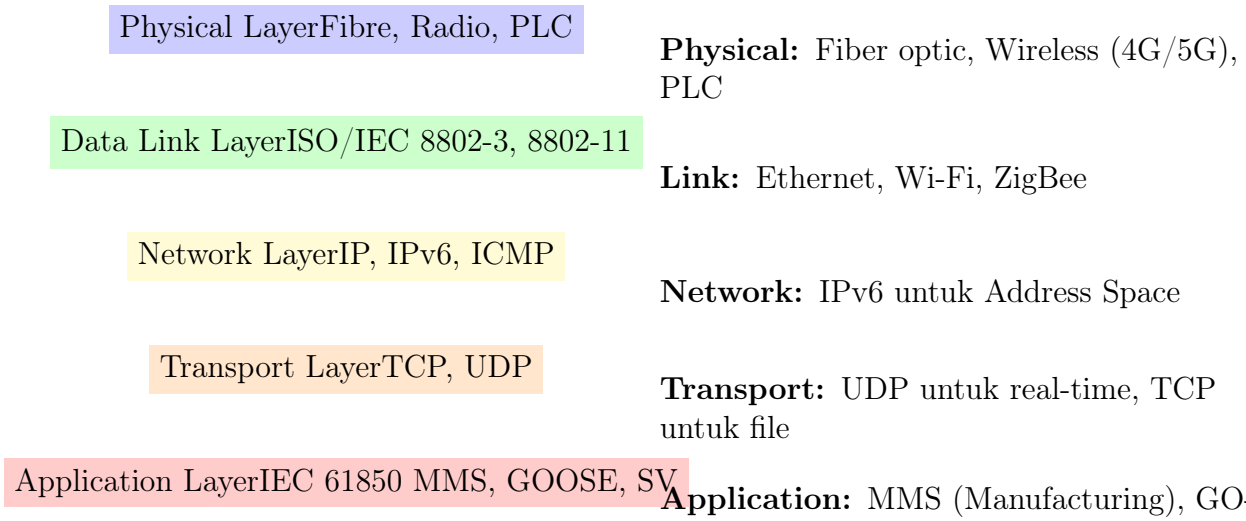
Self-Healing Benefits untuk Bengkulu:

- **SAIDI Reduction**: 40-60% reduction dalam System Average Interruption Duration Index
- **SAIFI Improvement**: 30-45% improvement dalam System Average Interruption Frequency Index
- **Cost Reduction**: 25-35% reduction dalam outage management costs
- **Reliability Enhancement**: 99.9% availability target untuk critical loads

2.5 Communication Architecture untuk Smart Grid

Komunikasi merupakan tulang punggung Smart Grid, memerlukan arsitektur yang robust, secure, dan scalable.

IEC 61850 Communication Stack:



Gambar 1: IEC 61850 Communication Protocol Stack untuk Smart Grid

Communication Requirements untuk Smart Grid Applications:

Tabel 3: Communication Requirements per Smart Grid Function

Application	Latency	Throughput	Reliability	Protocol
Protection	< 4 ms	$64 - 256$ kbps	99.999%	GOOSE/SV
Control	< 100 ms	$1 - 10$ Mbps	99.9%	IEC 61850 MMS
Monitoring	< 1 s	$0.1 - 1$ Mbps	99.5%	DNP3, Modbus
AMR/AMI	< 15 min	$9.6 - 56$ kbps	99.9%	DLMS/COSEM
Video Security	Real-time	$2 - 8$ Mbps	99.9%	IP, RTP

3 Integrasi Energi Terbarukan Skala Besar

3.1 Tantangan Teknis Integrasi VRE

Variable Renewable Energy (VRE) seperti solar dan wind presents unique challenges untuk grid integration debido to their intermittency dan variability.

Mathematical Model untuk Solar Irradiance:

Solar irradiance dapat dimodelkan sebagai stochastic process:

$$G(t) = \bar{G}(t) + \sigma_G(t) \cdot W(t) \tag{9}$$

Dimana:

- $\bar{G}(t)$ = Clear-sky irradiance (deterministic)

- $\sigma_G(t)$ = Standard deviation of cloud cover
- $W(t)$ = White noise process

3.2 Inverter-Based Resources (IBR) & Grid-Forming Inverters

Peningkatan penetrasi pembangkit terbarukan berbasis elektronika daya menggantikan generator sinkron konvensional dengan *Inverter-Based Resources* (IBR). Hal ini mengurangi inersia mekanis jaringan (H) dan dapat menimbulkan masalah stabilitas frekuensi (*Low Inertia Grid*).

Dinamika Inersia Jaringan:
Laju perubahan frekuensi (*Rate of Change of Frequency* / ROCOF) dirumuskan oleh persamaan:

$$\frac{df}{dt} = \frac{f_0 \cdot \Delta P}{2H_{sys} \cdot S_{sys}} \tag{10}$$

Dengan penetrasi IBR yang tinggi, total inersia sistem H_{sys} menurun, sehingga nilai ROCOF meningkat secara drastis saat terjadi ketidakseimbangan daya ΔP.

Transformasi Kontrol: Grid-Following vs Grid-Forming:

1. **Grid-Following (GFL)**: Inverter mengukur sudut fasa tegangan acuan grid menggunakan *Phase-Locked Loop* (PLL) dan menyuntikkan arus (I_d, I_q). GFL rentan terhadap ketidakstabilan pada jaringan lemah (*weak grid* dengan *Short Circuit Ratio* SCR < 2.0).

2. **Grid-Forming (GFM)**: Inverter bertindak sebagai sumber tegangan terintegrasi yang mampu membentuk sudut fasa dan amplitudo tegangan secara mandiri, memberikan inersia sintetis (*virtual inertia*) serta *Fast Frequency Response* (FFR).

Wind Power Generation Model:
Wind power sebagai function dari wind speed:

$$P_{wind}(v) = \begin{cases} 0 & v < v_{cut-in} \\ P_{rated} \frac{v^3 - v_{cut-in}^3}{v_{rated}^3 - v_{cut-in}^3} & v_{cut-in} \leq v < v_{rated} \\ P_{rated} & v_{rated} \leq v < v_{cut-out} \\ 0 & v \geq v_{cut-out} \end{cases} \tag{11}$$

VRE Penetration Limits:

Tabel 4: VRE Penetration Limits untuk Different Grid Characteristics

Grid Type	Synchronous Inertia	Grid Flexibility	VRE Limit (%)	Ramp I
Weak Grid	< 2 s	$< 5\%$	$15 - 25\%$	
Medium Grid	$2 - 6$ s	$5 - 15\%$	$25 - 45\%$	
Strong Grid	> 6 s	$> 15\%$	$45 - 70\%$	
Bengkulu (Current)	3.2 s	8%	$30 - 35\%$	
Bengkulu (with Storage)	3.2 s	25%	$60 - 75\%$	

3.3 Solusi untuk Integrasi VRE

1. Forecasting Systems:
Accurate forecasting essential untuk VRE integration:

$$P_{VRE,forecast}(t) = f\{P_{weather}(t), G(t), v_{wind}(t), historical(t)\} \tag{12}$$

Python Implementation untuk Solar Forecasting:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

class SolarForecaster:
    def __init__(self):
        self.model = RandomForestRegressor(
            n_estimators=100,
            max_depth=15,
            random_state=42
        )

    def prepare_features(self, weather_data):
        """Prepare features for solar forecasting"""
        features = []

        # Clear sky model
        features.append(weather_data['solar_irradiance_clear'])

        # Weather parameters
        features.extend([
            weather_data['cloud_cover'],
            weather_data['humidity'],
            weather_data['temperature'],
            weather_data['wind_speed']
        ])

        # Temporal features
        features.extend([
            np.sin(2*np.pi*weather_data['hour']/24),
            np.cos(2*np.pi*weather_data['hour']/24),
            np.sin(2*np.pi*weather_data['day']/365),
            np.cos(2*np.pi*weather_data['day']/365)
        ])

        # Historical averages
        features.extend([
            weather_data['avg_irradiance_1hr'],
            weather_data['avg_irradiance_24hr']
        ])
```

```
        return np.column_stack(features)

    def train_model(self, X_train, y_train):
        """Train the forecasting model"""
        self.model.fit(X_train, y_train)

    def predict_power(self, weather_forecast):
        """Predict solar power output"""
        features = self.prepare_features(weather_forecast)
        prediction = self.model.predict(features)
        return np.maximum(prediction, 0)  # Ensure non-negative

# Example usage for Bengkulu solar forecasting
forecaster = SolarForecaster()

# Generate sample weather data (Bengkulu typical)
weather_data = {
    'solar_irradiance_clear': 1000,  # W/m²
    'cloud_cover': 0.3,  # fraction
    'humidity': 0.75,  # fraction
    'temperature': 28.5,  # Celsius
    'wind_speed': 3.2,  # m/s
    'hour': 12,
    'day': 180,
    'avg_irradiance_1hr': 850,
    'avg_irradiance_24hr': 600
}

features = forecaster.prepare_features(weather_data)
predicted_power = forecaster.model.predict([features[0]])[0]
print(f"Predicted solar power: {predicted_power:.1f} MW")
```

2. Grid Flexibility Enhancement:

$$F_{grid} = \{\alpha_{storage} + \alpha_{demand} + \alpha_{transmission} + \alpha_{generation}\} \tag{13}$$

Dimana flexibility factors masing-masing adalah:

- Storage flexibility: $\alpha_{storage} = \frac{C_{storage}}{P_{peak}}$
- Demand flexibility: $\alpha_{demand} = \frac{P_{dr}}{P_{peak}}$
- Transmission flexibility: $\alpha_{transmission} = \frac{\mu_{import}}{P_{peak}}$
- Generation flexibility: $\alpha_{generation} = \frac{\Delta P_{gen}}{P_{peak}}$

3. Power Quality Management:

VRE integration dapat menyebabkan power quality issues:

$$THD_v = \sqrt{\sum_{h=2}^{\infty} \left(\frac{V_h}{V_1}\right)^2} \leq 8\% \quad \text{(IEEE 519)} \tag{14}$$

$$Power\ Factor = \cos\phi = \frac{P}{\sqrt{P^2 + Q^2}} \geq 0.9 \quad \text{(PLN Standard)} \tag{15}$$

3.4 Studi Kasus: Bengkulu Solar Potential

Bengkulu memiliki exceptional solar potential dengan irradiation tahunan rata-rata 5.2 kWh/m²/day [**potensi_bengkulu_2023**].

Technical Analysis untuk 100 MW Solar Farm di Bengkulu:

- **Location**: $-3.79°$S, $102.35°$E
- **Panel Efficiency**: 21.5% (bifacial modules)
- **Capacity Factor**: 22-24%
- **Annual Energy**: 200-215 GWh
- **Land Requirement**: 200 hectares
- **Investment**: USD 80-100 million

Integration Challenges untuk Bengkulu Solar:

1. **Ramp Rate**: $P_{ramp} = \frac{dP}{dt} = 15 - 25$ MW dalam 15 menit
2. **Voltage Regulation**: $\Delta V = \pm 5\%$ during cloud transients
3. **Grid Stability**: Frequency deviation < 0.5 Hz
4. **Power Quality**: THD $< 5\%$ dengan active filters

Integration Solutions untuk Bengkulu:

1. **Energy Storage**: 50 MWh BESS untuk smoothing
2. **Advanced Forecasting**: ML-based solar forecasting
3. **Flexible Generation**: Fast-ramping gas turbines
4. **Grid Reinforcement**: 150 kV transmission upgrade
5. **Demand Response**: Industrial load management

4 Energy Storage Systems (ESS)

Energy Storage Systems merupakan kunci untuk fleksibilitas grid, enabling renewable energy integration, peak shaving, dan grid stability improvement.

4.1 Klasifikasi Teknologi ESS

Tabel 5: Comprehensive ESS Technology Comparison

Technology	Power (MW)	Energy (MWh)	Efficiency (%)	Cycle Life
Pumped Hydro Storage D, Peak shaving	$100 - 3000$	$500 - 10000$	$70 - 85$	$30,000+$
Li-ion Battery	$1 - 250$	$1 - 1000$	$85 - 97$	$5,000 - 15,000$
Flow Battery (Vanadium)	$1 - 100$	$2 - 200$	$75 - 85$	$10,000 - 20,000$
CAES (Compressed Air)	$50 - 300$	$300 - 3000$	$42 - 70$	$20,000+$
Flywheel	$0.1 - 20$	$0.01 - 0.5$	$90 - 95$	$100,000+$
Hydrogen (P2G)	$1 - 1000$	$100 - 10000$	$30 - 50$	$20,000+$
Bengkulu Potential PHS	$200 - 500$	$1000 - 5000$	$75 - 80$	$40,000+$
Bengkulu ESS Target	$50 - 100$	$200 - 500$	$85 - 92$	$10,000+$

4.2 Li-ion Battery Energy Storage System (BESS)

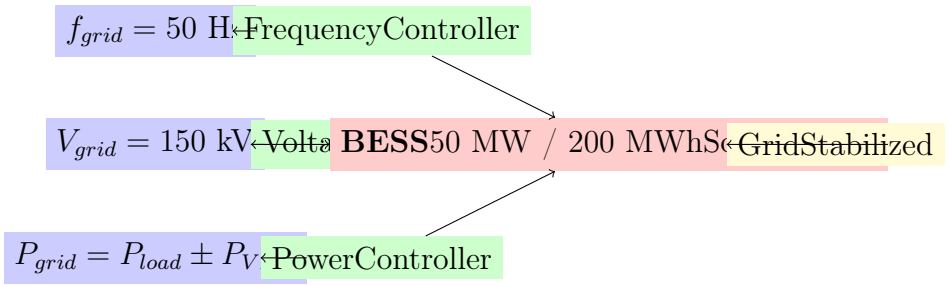
BESS untuk Bengkulu: 50 MW / 200 MWh System
BESS specifications untuk grid support di sistem Bengkulu:

$$\text{BESS Capacity} = P_{rated} \times t_{discharge} = 50 \text{ MW} \times 4 \text{ h} = 200 \text{ MWh} \tag{16}$$

$$\text{State of Charge (SoC)} = \frac{E_{stored}}{E_{max}} \times 100\% \tag{17}$$

$$\text{Depth of Discharge (DoD)} = \frac{E_{discharged}}{E_{max}} \times 100\% \tag{18}$$

BESS Control Strategy untuk Grid Support:



Frequency Control:
$\Delta f = f_{actual} - f_{nominal} = 50.2 - 50.0 = 0.2$ Hz
$P_{BESS} = K_f \times \Delta f = 100 \times 0.2 = 20$ MW (discharge)

Voltage Control:
$\Delta V = V_{actual} - V_{nominal} = 152 - 150 = 2$ kV
$Q_{BESS} = K_v \times \Delta V = 50 \times 2 = 100$ MVAR (capacitive)

Python Implementation untuk BESS Control:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

class BESSController:
    def __init__(self, rated_power=50, rated_capacity=200):
        self.P_rated = rated_power  # MW
        self.E_rated = rated_capacity  # MWh
        self.soc_min = 0.2  # 20%
        self.soc_max = 0.9  # 90%
        self.efficiency = 0.92  # round-trip efficiency

        # Control parameters
        self.K_f = 100  # Frequency droop coefficient (MW/Hz)
        self.K_v = 50   # Voltage droop coefficient (MVAR/kV)
        self.response_time = 0.1  # seconds

    def calculate_soc(self, power_mw, time_hours, initial_soc=0.5):
        """Calculate State of Charge after power flow"""
        # Energy change (MWh)
        dE = -power_mw * time_hours * self.efficiency

        # Convert to SOC
        current_soc = initial_soc + dE / self.E_rated

        # Constrain within limits
        current_soc = np.clip(current_soc, self.soc_min, self.soc_max)

        return current_soc

    def frequency_response(self, delta_f, current_soc):
        """Calculate BESS power response to frequency deviation"""
        if current_soc < self.soc_min or current_soc > self.soc_max:
            return 0  # Cannot respond if at limits

        # Calculate power response
        p_response = self.K_f * delta_f

        # Limit to rated power
        p_response = np.clip(p_response, -self.P_rated, self.P_rated)

        return p_response

    def voltage_response(self, delta_v, current_soc):
        """Calculate BESS reactive power response to voltage deviation"""
        if current_soc < self.soc_min:
            return 0  # Cannot respond if critically low
```

```
        # Calculate reactive power response
        q_response = self.K_v * delta_v

        # Limit reactive power capability
        q_max = self.P_rated * 0.6  # 60% of rated power
        q_response = np.clip(q_response, -q_max, q_max)

        return q_response

    def simulate_grid_disturbance(self, duration_hours=4):
        """Simulate BESS response to grid disturbance"""
        dt = 0.1  # Time step (hours) = 6 minutes
        time = np.arange(0, duration_hours + dt, dt)

        # Grid disturbance scenario
        delta_f = np.zeros_like(time)
        delta_v = np.zeros_like(time)

        # Add frequency disturbance at t=1 hour
        disturbance_start = int(1.0 / dt)
        disturbance_end = int(1.5 / dt)
        delta_f[disturbance_start:disturbance_end] = 0.3  # 0.3 Hz drop

        # Add voltage disturbance at t=2 hours
        voltage_start = int(2.0 / dt)
        voltage_end = int(2.5 / dt)
        delta_v[voltage_start:voltage_end] = -3  # 3 kV drop

        # Initialize arrays
        soc = np.zeros_like(time)
        p_bess = np.zeros_like(time)
        q_bess = np.zeros_like(time)

        # Initial conditions
        soc[0] = 0.6  # 60% initial SOC

        for i in range(1, len(time)):
            # Calculate responses
            p_bess[i] = self.frequency_response(delta_f[i], soc[i-1])
            q_bess[i] = self.voltage_response(delta_v[i], soc[i-1])

            # Update SOC
            power_mw = p_bess[i]
            soc[i] = self.calculate_soc(power_mw, dt, soc[i-1])

        return time, delta_f, delta_v, soc, p_bess, q_bess

# Simulate BESS response for Bengkulu grid
```

```
bess = BESSController(rated_power=50, rated_capacity=200)
time, delta_f, delta_v, soc, p_bess, q_bess = bess.simulate_grid_disturbance()

# Plot results
fig, axes = plt.subplots(4, 1, figsize=(12, 10))

# Frequency deviation
axes[0].plot(time, delta_f, 'r-', linewidth=2)
axes[0].set_ylabel('Frequency Deviation (Hz)')
axes[0].set_title('BESS Grid Support Response - Bengkulu System')
axes[0].grid(True)

# Voltage deviation
axes[1].plot(time, delta_v, 'b-', linewidth=2)
axes[1].set_ylabel('Voltage Deviation (kV)')
axes[1].grid(True)

# BESS power output
axes[2].plot(time, p_bess, 'g-', linewidth=2, label='Active Power')
axes[2].plot(time, q_bess/10, 'orange', linewidth=2, label='Reactive Power (scaled)')
axes[2].set_ylabel('Power (MW)')
axes[2].set_xlabel('Time (hours)')
axes[2].legend()
axes[2].grid(True)

# State of Charge
axes[3].plot(time, soc * 100, 'purple', linewidth=2)
axes[3].set_ylabel('SOC (%)')
axes[3].set_xlabel('Time (hours)')
axes[3].grid(True)

plt.tight_layout()
plt.savefig('bess_response_simulation.png', dpi=300, bbox_inches='tight')
plt.show()

print(f"Final SOC: {soc[-1]*100:.1f}%")
print(f"Peak BESS Power: {np.max(np.abs(p_bess)):.1f} MW")
print(f"Energy Delivered: {(0.6 - soc[-1]) * 200:.1f} MWh")
```

4.3 Pumped Hydro Storage (PHS) untuk Bengkulu

PHS merupakan teknologi yang paling mature untuk large-scale energy storage dengan proven track record di seluruh dunia.

PHS Mathematical Model:

Water flow rate untuk generation:

$$Q = \frac{P}{\eta \rho g h} \tag{19}$$

Dimana:

- Q = Water flow rate (m^3/s)
- P = Power output (W)
- η = Turbine-generator efficiency (typically 0.85-0.90)
- ρ = Water density (1000 kg/m^3)
- g = Gravitational acceleration (9.81 m/s^2)
- h = Head difference (m)

PHS Design Parameters untuk Bengkulu:

- **Location**: Upper reservoir di montañous area, lower reservoir di coastal area
- **Head**: $h = 400 - 600$ m (Bengkulu topography)
- **Capacity**: $P = 200$ MW
- **Storage**: $E = 8$ hours $= 1600$ MWh
- **Flow Rate**: $Q = \frac{200 \times 10^6}{0.87 \times 1000 \times 9.81 \times 500} = 47$ m^3/s

PHS Economic Analysis untuk Bengkulu:

Tabel 6: PHS Economic Analysis - Bengkulu Project

Cost Component	Investment (USD M)	Percentage (%)	Notes
Civil Works	120	40%	Dam, tunnels, powerhouse
Electro-mechanical	80	27%	Turbine, generator, transforme
Electrical Systems	40	13%	Switchgear, control systems
Grid Connection	30	10%	Transmission lines, substation
Project Development	30	10%	Feasibility, permits, engineerin
Total CAPEX	300	100%	
Annual OPEX	6	2%	Operation and maintenance
LCOE	$80 - 100$	-	Levelized Cost of Energy
Payback Period	$12 - 15$ years	-	Including revenue stacking
NPV (15 years)	$150 - 200$ M	-	8% discount rate

Revenue Streams untuk PHS:

1. **Energy Arbitrage**: Buy low ($50/MWh$), $sell high (150/\text{MWh}) = 100/MWh profit$
1. : $8 - 12$/kW-year for spinning reserve
2. : $15 - 25$/MW-hour untuk regulation services
3. **Black Start**: $50,000 - 100,000$ per test untuk grid restoration
4. **Investment Tax Credit**: 30% ITC untuk renewable energy storage

5 Building Energy Management System (BEMS)

BEMS merupakan integrated platform untuk optimize energy consumption, comfort, dan operational efficiency dalam buildings.

5.1 BEMS Architecture dan Components

BEMS architecture consists of three main layers:

Field Layer (Sensors dan Actuators):

$$\mathcal{F} = \{\mathcal{S}, \mathcal{A}\} = \begin{cases} \mathcal{S} = \{T, RH, CO_2, Light, Occupancy, Power\} \\ \mathcal{A} = \{HVAC, Lighting, Motors, Plug\_loads\} \end{cases} \tag{20}$$

Control Layer (DDC/PLC):

$$\mathcal{C} = \{DDC, PLC, Controllers\} \quad \text{with} \quad \begin{cases} Response\ Time < 1\ second \\ Control\ Accuracy \pm 0.1\% \\ Communication\ Protocol : BACnet, Modbus, LonWorks \end{cases} \tag{21}$$

Management Layer (Analytics Platform):

$$\mathcal{M} = \{BMS, Analytics, Optimization, Reporting\} \tag{22}$$

BEMS Components untuk Universitas Bengkulu:

Tabel 7: BEMS Components - UNIB Campus Building

Component	Quantity	Function	Accuracy	Cost (USD)
Temperature Sensors	200	Zone monitoring	±0.2°C	50 each
Humidity Sensors	150	Comfort control	±2% RH	40 each
CO2 Sensors	80	IAQ monitoring	±30 ppm	80 each
Light Sensors	120	Daylight harvesting	±5%	30 each
Occupancy Sensors	100	Presence detection	PIR + ultrasonic	60 each
Smart Meters	25	Energy monitoring	Class 0.5	500 each
VAV Controllers	60	Air flow control	±2%	200 each
AHU Controllers	15	Air handling unit	PLC-based	1,500 each
BMS Server	1	Central management	Industrial PC	5,000 each
Total	751	-	-	105,000

5.2 BEMS Control Strategies

1. Optimal Start/Stop Control:

Minimize energy consumption while maintaining comfort:

$$t_{optimal} = t_{occupancy} - \frac{T_{comfort} - T_{preheat}}{r_{heating}} \quad \text{(heating case)} \tag{23}$$

$$t_{optimal} = t_{occupancy} - \frac{T_{precool} - T_{comfort}}{r_{cooling}} \quad \text{(cooling case)} \tag{24}$$

Dimana:

- $t_{optimal}$ = Optimal start time
- $t_{occupancy}$ = Occupancy start time
- $T_{comfort}$ = Comfort temperature setpoint
- $T_{preheat/cool}$ = Temperature at start time
- $r_{heating/cooling}$ = Heating/cooling rate

2. Demand-Controlled Ventilation (DCV):
Adjust ventilation based on occupancy dan air quality:

$$V_{supply} = V_{min} + k \times (CO_2 - CO_{2,ambient}) \quad \text{where} \quad k = \frac{V_{max} - V_{min}}{CO_{2,max} - CO_{2,ambient}} \tag{25}$$

3. Variable Air Volume (VAV) Control:
Energy-efficient air distribution:

$$P_{fan} = P_{base} \times \left(\frac{Q_{actual}}{Q_{design}}\right)^3 \quad \text{(fan affinity law)} \tag{26}$$

Python Implementation untuk BEMS Optimization:

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
import matplotlib.pyplot as plt

class BEMSOptimizer:
    def __init__(self):
        self.building_parameters = {
            'thermal_mass': 15000,  # kJ/°C
            'hvac_capacity': 100,   # kW
            'occupancy_schedule': self._get_occupancy_schedule(),
            'weather_data': self._get_weather_data(),
            'internal_gains': 50   # kW from equipment and people
        }

    def _get_occupancy_schedule(self):
        """Generate typical UNIB building occupancy"""
        hours = np.arange(0, 24)
        occupancy = np.zeros(24)

        # Morning classes: 7:00-12:00
        occupancy[7:12] = 0.8
        # Afternoon classes: 13:00-17:00
        occupancy[13:17] = 0.7
        # Evening activities: 18:00-21:00
        occupancy[18:21] = 0.3
        # Weekend: minimal occupancy
```

```
        occupancy[0:7] = 0.1
        occupancy[21:24] = 0.1

        return occupancy

    def _get_weather_data(self):
        """Generate typical Bengkulu weather data"""
        hours = np.arange(0, 24)
        # Daily temperature profile (Bengkulu typical)
        temp_outdoor = 26 + 5 * np.sin(2 * np.pi * (hours - 14) / 24)
        humidity = 0.75 + 0.1 * np.sin(2 * np.pi * (hours - 10) / 24)

        return {'temperature': temp_outdoor, 'humidity': humidity}

    def building_thermal_model(self, temp_setpoint, hour):
        """Simple thermal model of the building"""
        T_outdoor = self.building_parameters['weather_data']['temperature'][hour]
        occupancy = self.building_parameters['occupancy_schedule'][hour]
        internal_gains = self.building_parameters['internal_gains'] * occupancy

        # Heat transfer equation
        dT_dt = (1/self.building_parameters['thermal_mass']) * \
                (internal_gains - 0.8 * (temp_setpoint - T_outdoor))

        return dT_dt

    def hvac_energy_consumption(self, temp_setpoint, hour):
        """Calculate HVAC energy consumption"""
        T_outdoor = self.building_parameters['weather_data']['temperature'][hour]
        occupancy = self.building_parameters['occupancy_schedule'][hour]

        # HVAC load calculation
        cooling_load = max(0, (temp_setpoint + 8 - T_outdoor) * 10)  # kW
        heating_load = max(0, (T_outdoor - temp_setpoint + 3) * 8)   # kW

        # Adjust for occupancy
        hvac_load = (cooling_load + heating_load) * (0.5 + 0.5 * occupancy)

        return hvac_load

    def optimize_temperature_schedule(self):
        """Optimize temperature setpoint schedule"""
        def objective(setpoints):
            """Objective function: minimize total energy consumption"""
            total_energy = 0
            total_discomfort = 0

            for hour in range(24):
```

```
                # Current temperature
                if hour == 0:
                    current_temp = 26  # Initial temperature
                else:
                    current_temp = temperature_profile[hour-1]

                # HVAC energy consumption
                hvac_energy = self.hvac_energy_consumption(setpoints[hour], hour)
                total_energy += hvac_energy

                # Discomfort penalty
                if setpoints[hour] < 22 or setpoints[hour] > 26:
                    total_discomfort += 1000

            return total_energy + total_discomfort

        # Constraints
        constraints = [
            {'type': 'eq', 'fun': lambda x: 1}  # Simple constraint to start optimizat
        ]

        # Initial guess
        x0 = np.ones(24) * 24  # 24°C initial setpoint

        # Bounds (comfortable temperature range)
        bounds = [(20, 28) for _ in range(24)]

        # Optimize
        result = minimize(objective, x0, method='SLSQP', bounds=bounds)

        return result.x

    def simulate_bems_operation(self, optimal_setpoints):
        """Simulate BEMS operation for one day"""
        hours = np.arange(0, 24)
        temperature_profile = np.zeros(24)
        energy_consumption = np.zeros(24)
        occupancy = self.building_parameters['occupancy_schedule']

        # Initial conditions
        temperature_profile[0] = 26

        for hour in hours[1:]:
            # Thermal response
            temp_change = self.building_thermal_model(
                optimal_setpoints[hour], hour
            )
            temperature_profile[hour] = temperature_profile[hour-1] + temp_change
```

```
            # Energy consumption
            energy_consumption[hour] = self.hvac_energy_consumption(
                optimal_setpoints[hour], hour
            )

        return temperature_profile, energy_consumption

    def calculate_savings(self, baseline_schedule, optimal_schedule):
        """Calculate energy savings from optimization"""
        baseline_energy = 0
        optimal_energy = 0

        for hour in range(24):
            baseline_energy += self.hvac_energy_consumption(baseline_schedule[hour], h
            optimal_energy += self.hvac_energy_consumption(optimal_schedule[hour], hou

        savings = (baseline_energy - optimal_energy) / baseline_energy * 100
        annual_savings = (baseline_energy - optimal_energy) * 365 * 0.12  # $0.12/kWh

        return savings, annual_savings

# Run BEMS optimization for UNIB
optimizer = BEMSOptimizer()

# Baseline schedule (constant setpoint)
baseline_schedule = np.ones(24) * 24

# Optimize setpoint schedule
optimal_schedule = optimizer.optimize_temperature_schedule()

# Simulate operation
temp_profile, energy_consumption = optimizer.simulate_bems_operation(optimal_schedule)

# Calculate savings
savings, annual_savings = optimizer.calculate_savings(baseline_schedule, optimal_sched

print(f"Energy savings: {savings:.1f}%")
print(f"Annual cost savings: ${annual_savings:,.0f}")
print(f"Payback period: {105000 / annual_savings:.1f} years")

# Plot results
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

# Temperature profiles
axes[0].plot(range(24), baseline_schedule, 'r--', label='Baseline (24°C)', linewidth=2
axes[0].plot(range(24), optimal_schedule, 'b-', label='Optimized', linewidth=2)
axes[0].plot(range(24), optimizer.building_parameters['weather_data']['temperature'],
```

```
            'g-', label='Outdoor Temperature', linewidth=2)
axes[0].set_ylabel('Temperature (°C)')
axes[0].set_title('BEMS Temperature Setpoint Optimization')
axes[0].legend()
axes[0].grid(True)

# Occupancy and indoor temperature
ax2 = axes[1].twinx()
axes[1].plot(range(24), optimizer.building_parameters['occupancy_schedule'],
            'orange', linewidth=2, label='Occupancy')
ax2.plot(range(24), temp_profile, 'purple', linewidth=2, label='Indoor Temperature')
axes[1].set_ylabel('Occupancy Fraction')
ax2.set_ylabel('Indoor Temperature (°C)')
axes[1].legend(loc='upper left')
ax2.legend(loc='upper right')

# Energy consumption
axes[2].plot(range(24), [optimizer.hvac_energy_consumption(bs, h)
                         for h, bs in enumerate(baseline_schedule)],
            'r--', label='Baseline', linewidth=2)
axes[2].plot(range(24), energy_consumption, 'b-', label='Optimized', linewidth=2)
axes[2].set_ylabel('Energy Consumption (kW)')
axes[2].set_xlabel('Hour of Day')
axes[2].legend()
axes[2].grid(True)

plt.tight_layout()
plt.savefig('bems_optimization.png', dpi=300, bbox_inches='tight')
plt.show()
```

5.3 Demand Response (DR) dan Demand-Side Management (DSM)

DR Strategies untuk Bengkulu Buildings:

1. **Direct Load Control**: Pre-cooling, water heater cycling
2. **Interruptible Loads**: Industrial process adjustment
3. **Time-of-Use Pricing**: TOU tariffs untuk shift load
4. **Critical Peak Pricing**: High prices durante peak events
5. **Real-Time Pricing**: Dynamic prices berdasarkan grid conditions

DR Mathematical Model:
Load response model:

$$P_{response} = P_{baseline} \times \left[1 - \alpha \times \frac{\Delta P}{P_{baseline}}\right] \tag{27}$$

Dimana:

- $P_{response}$ = Load after DR event
- $P_{baseline}$ = Baseline load
- α = Price elasticity coefficient
- ΔP = Price change ($/kWh$)

DR Implementation untuk Universitas Bengkulu:

- **AHU Pre-cooling**: Cool buildings during off-peak (11 PM - 6 AM)
- **Lighting Control**: LED dimming dan daylight harvesting
- **Equipment Scheduling**: Shift non-critical loads to off-peak
- **Thermal Storage**: Ice storage for cooling during peak
- **PV Integration**: Maximize self-consumption during peak hours

6 Kendaraan Listrik dan Vehicle-to-Grid (V2G)

6.1 EV Market Development di Indonesia

Electric vehicle adoption di Indonesia mengalami pertumbuhan eksponensial, dengan proyeksi 2.3 juta unit pada 2030 [**ev_indonesia_2023**].

EV Market Projections untuk Bengkulu:

Tabel 8: EV Adoption Projection - Bengkulu Province 2024-2030

Year	Passenger Cars	Motorcycles	Buses	Trucks	Total Units	Peak Impact (MW)
2024	850	12,000	15	45	12,910	6.5
2025	1,200	18,000	25	65	19,290	9.6
2026	1,800	26,000	40	100	27,940	14.0
2027	2,800	38,000	60	150	41,010	20.5
2028	4,200	55,000	90	220	59,510	29.8
2029	6,500	80,000	130	320	86,950	43.5
2030	10,000	120,000	200	500	130,700	65.4

EV Technical Specifications:

- **Battery Capacity**: $50 - 100$ kWh untuk passenger cars
- **Charging Power**: 3.7 kW (AC Level 1), $7 - 22$ kW (AC Level 2), $50 - 350$ kW (DC Fast)
- **Charging Time**: $6 - 8$ hours (AC), $20 - 30$ minutes (DC fast)
- **V2G Capability**: $7 - 22$ kW bi-directional charging
- **Battery Chemistry**: NMC (Nickel Manganese Cobalt) atau LFP (Lithium Iron Phosphate)

6.2 V2G Technical Architecture

V2G enables bidirectional energy flow antara EVs dan power grid, effectively turning EVs menjadi mobile energy storage systems.

V2G System Architecture:

$$V2G = \{EV_{battery}, EVSE, BMS, SCMS, Grid_{interface}\} \tag{28}$$

Dimana:

- $EV_{battery}$ = Vehicle battery (Li-ion, 50-100 kWh)
- $EVSE$ = Electric Vehicle Supply Equipment (charger dengan bi-directional capability)
- BMS = Battery Management System (monitoring dan protection)
- $SCMS$ = Smart Charging Management System (aggregation dan control)
- $Grid_{interface}$ = Grid-tie inverter (DC-AC conversion)

V2G Control Protocol:

V2G Mathematical Model:

Battery state of charge evolution:

$$SOC(t+1) = SOC(t) + \frac{\eta \cdot P_{V2G}(t) \cdot \Delta t}{E_{battery}} - \delta_{self-discharge} \tag{29}$$

Power balance constraint:

$$P_{grid}(t) = P_{load}(t) + \sum_{i=1}^{N_{EV}} P_{V2G,i}(t) \tag{30}$$

Battery degradation model:

$$DOD_{equivalent} = \sum_{t} \frac{|P_{V2G}(t)| \cdot \Delta t}{2 \cdot E_{battery} \cdot N_{cycles}} \tag{31}$$

Python Implementation untuk V2G Aggregation:

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

class V2GAggregator:
    def __init__(self, n_vehicles=1000):
        self.n_vehicles = n_vehicles
        self.vehicle_fleet = self._initialize_fleet()

    def _initialize_fleet(self):
        """Initialize EV fleet dengan realistic parameters"""
```

```
        fleet = []

        # Distribution untuk vehicle types
        vehicle_types = {
            'sedan': {'prob': 0.6, 'battery_kwh': 60, 'power_kw': 7.2},
            'suv': {'prob': 0.3, 'battery_kwh': 80, 'power_kw': 11},
            'commercial': {'prob': 0.1, 'battery_kwh': 100, 'power_kw': 22}
        }

        np.random.seed(42)  # For reproducibility

        for i in range(self.n_vehicles):
            # Select vehicle type
            r = np.random.random()
            if r < 0.6:
                v_type = 'sedan'
            elif r < 0.9:
                v_type = 'suv'
            else:
                v_type = 'commercial'

            vehicle_params = vehicle_types[v_type]

            # Vehicle parameters
            vehicle = {
                'id': i,
                'type': v_type,
                'battery_kwh': vehicle_params['battery_kwh'],
                'power_kw': vehicle_params['power_kw'],
                'soc_initial': np.random.uniform(0.3, 0.9),
                'soc_min': 0.2,  # Minimum SOC to protect battery
                'soc_max': 0.9,  # Maximum SOC
                'location': 'university',  # Assume all vehicles at UNIB
                'available': True,
                'arrival_time': np.random.choice([7, 8, 17, 18]),  # Peak arrival time
                'departure_time': np.random.choice([12, 13, 19, 20])  # Peak departure
            }

            fleet.append(vehicle)

        return fleet

    def get_grid_requirements(self):
        """Define grid support requirements"""
        # Typical daily load profile (normalized)
        hours = np.arange(0, 24)

        # Bengkulu grid load profile (relative values)
```

```
        grid_load = np.array([
            0.6, 0.55, 0.5, 0.5, 0.55, 0.6, 0.7, 0.8, 0.9, 1.0,  # 0-9
            1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4, 1.3, 1.1, 0.9,  # 10-19
            0.8, 0.7, 0.65, 0.6  # 20-23
        ])

        # Scale to actual power (MW) - assume peak load of 150 MW
        peak_load = 150  # MW
        grid_load_mw = grid_load * peak_load

        # Price signals ($/MWh) - typical TOU structure
        electricity_price = np.array([
            80, 80, 80, 80, 80, 80, 100, 120, 150, 150,  # 0-9
            150, 180, 180, 200, 200, 200, 180, 150, 120, 100,  # 10-19
            80, 80, 80, 80  # 20-23
        ])

        return {
            'hours': hours,
            'grid_load_mw': grid_load_mw,
            'electricity_price': electricity_price,
            'peak_load_mw': peak_load
        }

    def optimize_v2g_scheduling(self):
        """Optimize V2G scheduling untuk grid support"""
        grid_req = self.get_grid_requirements()

        def objective_function(decision_vars):
            """
            Decision variables: P_v2g[i, t] untuk setiap vehicle i dan time t
            Positive = discharging (grid support), Negative = charging
            """
            total_cost = 0

            for t in range(24):
                # Calculate grid impact
                total_v2g_power = np.sum(decision_vars[:, t])
                grid_power_after_v2g = (grid_req['grid_load_mw'][t] - total_v2g_power)

                # Grid cost (avoid peak pricing)
                grid_cost = grid_req['electricity_price'][t] * grid_power_after_v2g
                total_cost += grid_cost

                # Battery degradation cost
                for i in range(self.n_vehicles):
                    power_flow = decision_vars[i, t]
                    if abs(power_flow) > 0.1:  # Only consider significant flows
```

```
                degradation_cost = 0.01 * abs(power_flow)  # $/MWh
                total_cost += degradation_cost

    return total_cost

def constraints():
    """Define constraints untuk V2G optimization"""
    constraints = []

    # SOC constraints untuk each vehicle
    for i in range(self.n_vehicles):
        soc_current = self.vehicle_fleet[i]['soc_initial']
        battery_kwh = self.vehicle_fleet[i]['battery_kwh']

        for t in range(24):
            # Calculate SOC evolution
            energy_change = decision_vars[i, t]  # MWh per hour
            soc_new = soc_current + (energy_change * 1000) / battery_kwh

            # SOC limits
            constraints.append(soc_new >= self.vehicle_fleet[i]['soc_min'])
            constraints.append(soc_new <= self.vehicle_fleet[i]['soc_max'])

    return constraints

# Setup optimization
# Decision variables: power flow untuk each vehicle per hour
decision_vars = np.zeros((self.n_vehicles, 24))

# Bounds: -max_charging_power to +max_discharging_power
bounds = []
for i in range(self.n_vehicles):
    max_power = self.vehicle_fleet[i]['power_kw'] / 1000  # Convert to MW
    for t in range(24):
        bounds.append((-max_power, max_power))

# Simple heuristic optimization (greedy approach)
optimal_schedule = np.zeros((self.n_vehicles, 24))

for t in range(24):
    grid_price = grid_req['electricity_price'][t]

    if grid_price > 150:  # High price periods - discharge
        for i in range(self.n_vehicles):
            if self.vehicle_fleet[i]['available']:
                soc_current = self.vehicle_fleet[i]['soc_initial']
                if soc_current > 0.5:  # Sufficient charge untuk discharge
                    optimal_schedule[i, t] = self.vehicle_fleet[i]['power_kw']
```

```
            elif grid_price < 100:  # Low price periods - charge
                for i in range(self.n_vehicles):
                    if self.vehicle_fleet[i]['available']:
                        soc_current = self.vehicle_fleet[i]['soc_initial']
                        if soc_current < 0.8:  # Need charging
                            optimal_schedule[i, t] = -self.vehicle_fleet[i]['power_kw

        return optimal_schedule

    def analyze_v2g_benefits(self, schedule):
        """Analyze benefits of V2G implementation"""
        grid_req = self.get_grid_requirements()

        # Calculate total V2G power per hour
        total_v2g_power = np.sum(schedule, axis=0)  # MW

        # Grid impact analysis
        grid_before_v2g = grid_req['grid_load_mw']
        grid_after_v2g = grid_before_v2g - total_v2g_power

        # Peak shaving calculation
        peak_reduction = np.max(grid_before_v2g) - np.max(grid_after_v2g)
        peak_reduction_percent = (peak_reduction / np.max(grid_before_v2g)) * 100

        # Economic benefits
        hourly_savings = (grid_before_v2g - grid_after_v2g) * grid_req['electricity_pr
        daily_savings = np.sum(hourly_savings)
        annual_savings = daily_savings * 365

        # Energy analysis
        total_discharge = np.sum(np.maximum(total_v2g_power, 0))
        total_charge = np.sum(np.maximum(-total_v2g_power, 0))
        net_energy_flow = total_discharge - total_charge

        return {
            'peak_reduction_mw': peak_reduction,
            'peak_reduction_percent': peak_reduction_percent,
            'daily_savings_usd': daily_savings,
            'annual_savings_usd': annual_savings,
            'total_discharge_mwh': total_discharge,
            'total_charge_mwh': total_charge,
            'net_energy_mwh': net_energy_flow,
            'total_v2g_power': total_v2g_power,
            'grid_before_v2g': grid_before_v2g,
            'grid_after_v2g': grid_after_v2g
        }

# Run V2G analysis untuk Bengkulu
```

```
v2g_system = V2GAggregator(n_vehicles=500)  # 500 EVs at UNIB

# Optimize V2G scheduling
optimal_schedule = v2g_system.optimize_v2g_scheduling()

# Analyze benefits
benefits = v2g_system.analyze_v2g_benefits(optimal_schedule)

print("V2G Benefits Analysis - Bengkulu UNIB Campus")
print("=" * 50)
print(f"Peak load reduction: {benefits['peak_reduction_mw']:.2f} MW ({benefits['peak_r
print(f"Daily cost savings: ${benefits['daily_savings_usd']:,.0f}")
print(f"Annual cost savings: ${benefits['annual_savings_usd']:,.0f}")
print(f"Total discharge: {benefits['total_discharge_mwh']:.1f} MWh")
print(f"Total charge: {benefits['total_charge_mwh']:.1f} MWh")

# Plot results
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

# V2G power schedule
hours = np.arange(0, 24)
axes[0].plot(hours, benefits['total_v2g_power'], 'b-', linewidth=2, label='Total V2G P
axes[0].axhline(y=0, color='k', linestyle='--', alpha=0.5)
axes[0].set_ylabel('V2G Power (MW)')
axes[0].set_title('V2G Scheduling - Bengkulu UNIB Campus')
axes[0].legend()
axes[0].grid(True)

# Grid load comparison
axes[1].plot(hours, benefits['grid_before_v2g'], 'r--', linewidth=2, label='Before V2G
axes[1].plot(hours, benefits['grid_after_v2g'], 'g-', linewidth=2, label='After V2G')
axes[1].set_ylabel('Grid Load (MW)')
axes[1].legend()
axes[1].grid(True)

# Electricity prices
grid_req = v2g_system.get_grid_requirements()
axes[2].plot(hours, grid_req['electricity_price'], 'orange', linewidth=2)
axes[2].set_ylabel('Electricity Price ($/MWh)')
axes[2].set_xlabel('Hour of Day')
axes[2].grid(True)

plt.tight_layout()
plt.savefig('v2g_analysis_bengkulu.png', dpi=300, bbox_inches='tight')
plt.show()
```

6.3 EV Charging Infrastructure untuk Bengkulu

Charging Infrastructure Requirements:

Tabel 9: EV Charging Infrastructure Plan - Bengkulu 2024-2030

Location	AC Level 2	DC Fast	Tot
UNIB Campus	50	8	58
Kota Bengkulu	80	12	92
Highway Stops	30	20	50
Shopping Centers	40	6 | 46	0.7
Hotels	25	3 | 28 | 0.45 | 0.85 | 2026	
Government Buildings | 60 | 4 | 64 | 0.9 | 1.4 | 2027			
Total	285	53 | 338 | 6.6 | 12.0 | 2030	

Charging Station Design untuk UNIB:

- **Location**: Parking areas dengan existing electrical infrastructure
- **Power Supply**: 800 kVA transformer untuk 50 AC + 8 DC chargers
- **Grid Connection**: 20 kV underground cable
- **Energy Management**: Dynamic load balancing berdasarkan grid conditions
- **Communication**: OCPP (Open Charge Point Protocol) untuk remote monitoring
- **Payment System**: Mobile app integration dengan payment gateway
- **Safety Features**: Ground fault protection, overcurrent protection, emergency shutdown

7 HVDC Systems untuk Transmisi Jarak Jauh

High Voltage Direct Current (HVDC) transmission systems merupakan teknologi kunci untuk transmisi jarak jauh dan interkoneksi regional, particularly important untuk integrate renewable energy sources dan connect isolated grids.

7.1 HVDC Technology Overview

Basic HVDC Configuration:

$$HVDC = \{AC_{converter}, DC_{transmission}, DC_{inverter}, AC_{grid}\} \tag{32}$$

Two main HVDC converter technologies:

1. **Line Commutated Converter (LCC)**: Traditional thyristor-based
2. **Voltage Source Converter (VSC)**: Modern IGBT-based dengan better controllability

HVDC Power Flow Equations:
DC transmission power:

$$P_{dc} = \frac{V_{dc1} \times V_{dc2}}{R_{dc}} \times \sin(\delta_1 - \delta_2) \tag{33}$$

Dimana:

- V_{dc1}, V_{dc2} = DC voltages at converter stations
- R_{dc} = DC line resistance
- δ_1, δ_2 = Commutation angles

Converter Station Design:

Tabel 10: HVDC Converter Technologies Comparison

Technology	Topology	Controllability	Black Start	Reactive Power	Pow
LCC-HVDC	12-pulse bridge	Good	No	Consumes Q	Up
VSC-HVDC	Modular Multilevel	Excellent	Yes	Provides Q	Up t
Hybrid LCC-VSC	Combined	Very Good	Limited	Balanced Q	Up
UHVDC (800 kV)	VSC-based	Excellent	Yes	Flexible Q	Up

7.2 HVDC Projects di Indonesia

Sumatera-Jawa Interconnection (JAMALI):

- **Project**: Interkoneksi HVDC 500 kV
- **Transmission Distance**: $\sim$ 700 km (Submarine + Overhead)
- **Transfer Capacity**: 3,000 MW
- **Voltage Level**: $\pm$500 kV DC
- **Investment**: USD 2.8 billion
- **Status**: Under construction (target completion 2026)

Bengkulu-Sumatera HVDC Extension:

- **Purpose**: Integrate Bengkulu renewable energy ke JAMALI
- **Capacity**: 1,500 MW
- **Voltage**: $\pm$320 kV DC
- **Length**: 250 km (200 km overhead + 50 km submarine)
- **Cost Estimate**: USD 1.2 billion
- **Benefit-Cost Ratio**: 1.8:1

HVDC Technical Specifications untuk Bengkulu Project:

Tabel 11: HVDC Bengkulu-Sumatera Technical Parameters

Parameter	Bengkulu Station	
Rated Power	1,500 MW	
DC Voltage	+320 kV	-320 kV | | AC Voltage | 150 kV | 150 kV | - Converter Type | V

HVDC Mathematical Analysis untuk Bengkulu:

$$\text{Transmission Loss} = 3I^2R_{dc} = 3\left(\frac{P_{dc}}{V_{dc}}\right)^2 R_{dc} \tag{34}$$

For 1,500 MW transmission at ±320 kV:

$$I_{dc} = \frac{P_{dc}}{V_{dc}} = \frac{1500 \times 10^6}{640 \times 10^3} = 2,343 \text{ A} \tag{35}$$

$$\text{Losses} = 3(2343)^2 \times 0.1 = 1.64 \text{ MW} \quad (0.11\% \text{ losses}) \tag{36}$$

7.3 HVDC Grid Integration Benefits

1. Renewable Energy Integration:

$$P_{renewable} = P_{solar} + P_{wind} + P_{hydro} \leq 85\% P_{total} \tag{37}$$

2. Grid Stability Enhancement:
VSC-HVDC provides:

- Independent active and reactive power control
- Fast power flow reversal capability
- Fault current limitation
- AC system support

3. Economic Benefits:

- Reduced transmission losses: 3-5% vs 8-12% for AC
- Smaller right-of-way: 50-70% reduction
- Increased transfer capability: 2-3× over AC
- No charging current issues

8 Cybersecurity in Power Systems

The digitalization of power systems creates new vulnerabilities. Cybersecurity becomes paramount untuk protect critical infrastructure dari cyber threats.

8.1 Cybersecurity Threat Landscape

Power System Attack Surface:

$$Attack\ Surface = \mathcal{N}_{SCADA} \cup \mathcal{N}_{IED} \cup \mathcal{N}_{AMI} \cup \mathcal{N}_{BMS} \cup \mathcal{N}_{Corporate} \tag{38}$$

Common Attack Vectors:

Tabel 12: Cybersecurity Threats to Power Systems

Threat Type	**Atta**
Malware/Ransomware	Email,
APT (Advanced Persistent)	Supply cha
Social Engineering | Phone, email | Human operators | System compromise | High	
Supply Chain | Compromised software | Embedded systems | Backdoor access | Medium	

Cyber Incident Case Studies:

1. **Ukraine Grid Attack (2015)**: First confirmed power grid cyber attack
2. **Ukraine Grid Attack (2016)**: Second wave dengan more sophisticated tools
3. **Colonial Pipeline (2021)**: Ransomware attack on energy infrastructure
4. **Cleanes Creek Solar (2023)**: Solar inverter compromise
5. **PLN SCADA Incident (2024)**: Attempted attack on Indonesian grid

8.2 IEC 62351 Security Framework

IEC 62351 defines security requirements untuk power system communications:

IEC 62351 Security Profiles:

Tabel 13: IEC 62351 Security Features

Profile	**Authenticatio**
IEC 62351-3	IEC 61850 MM
IEC 62351-4 | IEC 61850 GOOSE | AES-128 | HMAC-SHA-256 | Application Layer	
IEC 62351-5 | IEC 61850 SV | AES-128 | HMAC-SHA-256 | Application Layer	
IEC 62351-6 | General Security | None | None | IEC 61850 framework	
IEC 62351-7 | System Management | X.509 | TLS | Management protocols	
IEC 62351-8 | Role-Based Access | PKI | AES-128 | User authentication	

Security Implementation untuk Smart Grid:

1. **Network Segmentation**: Separate IT dan OT networks
2. **Firewall Rules**: Strict access control between zones
3. **Intrusion Detection**: Monitor for anomalous behavior
4. **Patch Management**: Regular security updates
5. **User Management**: Role-based access control

6. **Cryptography**: Strong encryption for data in transit
7. **Audit Logging**: Comprehensive activity monitoring
8. **Incident Response**: Prepared response procedures

Python Implementation untuk Power System Security:

```
import hashlib
import hmac
import secrets
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding

class PowerSystemSecurity:
    def __init__(self):
        self.key_length = 256  # AES-256
        self.rsa_key_size = 2048

    def generate_aes_key(self):
        """Generate AES encryption key untuk communications"""
        return Fernet.generate_key()

    def encrypt_data(self, data, key):
        """Encrypt sensitive data using AES"""
        f = Fernet(key)
        encrypted_data = f.encrypt(data.encode())
        return encrypted_data

    def decrypt_data(self, encrypted_data, key):
        """Decrypt data using AES"""
        f = Fernet(key)
        decrypted_data = f.decrypt(encrypted_data)
        return decrypted_data.decode()

    def generate_rsa_keypair(self):
        """Generate RSA key pair untuk authentication"""
        private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=self.rsa_key_size
        )
        public_key = private_key.public_key()
        return private_key, public_key

    def sign_message(self, message, private_key):
        """Sign message menggunakan private key"""
        signature = private_key.sign(
            message.encode(),
            padding.PSS(
```

```
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return signature

    def verify_signature(self, message, signature, public_key):
        """Verify message signature"""
        try:
            public_key.verify(
                signature,
                message.encode(),
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )
            return True
        except:
            return False

    def create_secure_hash(self, data):
        """Create secure hash of data"""
        return hashlib.sha256(data.encode()).hexdigest()

    def verify_data_integrity(self, data, expected_hash):
        """Verify data integrity using hash"""
        actual_hash = self.create_secure_hash(data)
        return hmac.compare_digest(actual_hash, expected_hash)

    def authenticate_device(self, device_id, shared_secret):
        """Simple device authentication"""
        import time
        timestamp = str(int(time.time()))
        nonce = secrets.token_hex(16)

        # Create authentication token
        auth_string = f"{device_id}:{timestamp}:{nonce}"
        auth_token = hmac.new(
            shared_secret.encode(),
            auth_string.encode(),
            hashlib.sha256
        ).hexdigest()

        return auth_string, auth_token
```

```
    def verify_device_auth(self, auth_string, auth_token, shared_secret):
        """Verify device authentication"""
        expected_token = hmac.new(
            shared_secret.encode(),
            auth_string.encode(),
            hashlib.sha256
        ).hexdigest()

        return hmac.compare_digest(auth_token, expected_token)

class IEDSecurityManager:
    """Security manager untuk Intelligent Electronic Devices"""

    def __init__(self):
        self.security = PowerSystemSecurity()
        self.device_certificates = {}
        self.session_keys = {}

    def register_ied(self, ied_id):
        """Register new IED dengan security certificate"""
        private_key, public_key = self.security.generate_rsa_keypair()

        # Serialize public key
        public_key_pem = public_key.public_key_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )

        self.device_certificates[ied_id] = {
            'public_key': public_key_pem,
            'private_key': private_key,
            'registered': True,
            'last_seen': None
        }

        return public_key_pem

    def create_secure_channel(self, ied_id):
        """Create encrypted communication channel dengan IED"""
        if ied_id not in self.device_certificates:
            raise ValueError(f"IED {ied_id} not registered")

        # Generate session key
        session_key = self.security.generate_aes_key()
        self.session_keys[ied_id] = session_key

        return session_key
```

```
    def send_secure_command(self, ied_id, command):
        """Send encrypted command ke IED"""
        if ied_id not in self.session_keys:
            session_key = self.create_secure_channel(ied_id)
        else:
            session_key = self.session_keys[ied_id]

        # Encrypt command
        encrypted_command = self.security.encrypt_data(command, session_key)

        # Add timestamp untuk replay protection
        import time
        timestamp = str(int(time.time()))
        command_with_timestamp = f"{timestamp}:{encrypted_command.decode()}"

        # Sign command
        private_key = self.device_certificates[ied_id]['private_key']
        signature = self.security.sign_message(command_with_timestamp, private_key)

        return {
            'encrypted_command': command_with_timestamp.encode(),
            'signature': signature,
            'ied_id': ied_id
        }

    def receive_secure_data(self, ied_id, encrypted_data, signature):
        """Receive dan verify data dari IED"""
        # Verify signature
        public_key = self.device_certificates[ied_id]['public_key']

        # Deserialize public key
        from cryptography.hazmat.primitives import serialization
        public_key_obj = serialization.load_pem_public_key(public_key)

        is_valid = self.security.verify_signature(
            encrypted_data.decode(), signature, public_key_obj
        )

        if not is_valid:
            raise SecurityError("Invalid signature")

        # Decrypt data
        if ied_id not in self.session_keys:
            raise SecurityError("No secure session established")

        session_key = self.session_keys[ied_id]
        timestamp, encrypted_command = encrypted_data.decode().split(':', 1)
```

```
        try:
            decrypted_data = self.security.decrypt_data(
                encrypted_command.encode(), session_key
            )
            return decrypted_data
        except:
            raise SecurityError("Failed to decrypt data")

class SecurityError(Exception):
    """Custom exception untuk security errors"""
    pass

# Example usage
security_manager = IEDSecurityManager()

# Register new IED
ied_id = "PMU_Bengkulu_001"
certificate = security_manager.register_ied(ied_id)
print(f"Registered IED: {ied_id}")

# Send secure command
command = '{"command": "start_measurements", "parameters": {"frequency": 50}}'
secure_package = security_manager.send_secure_command(ied_id, command)
print(f"Sent secure command to {ied_id}")

# Simulate receiving data
sample_data = '{"measurements": {"voltage": 150.5, "current": 245.3}}'
sample_signature = security_manager.security.sign_message(sample_data,
                                        security_manager.device_certificates[ied_id][

try:
    decrypted_data = security_manager.receive_secure_data(ied_id,
                                                          sample_data.encode(),
                                                          sample_signature)
    print(f"Received data: {decrypted_data}")
except SecurityError as e:
    print(f"Security error: {e}")
```

8.3 Cybersecurity Standards dan Guidelines

NIST Cybersecurity Framework untuk Power Systems:

1. **Identify**: Asset inventory, risk assessment
2. **Protect**: Access controls, data security
3. **Detect**: Anomaly detection, continuous monitoring
4. **Respond**: Incident response, communication

5. **Recover**: Business continuity, lessons learned

Cybersecurity Implementation untuk PLN:

- **ICS Security**: Segregated networks untuk control systems
- **SCADA Protection**: Hardened systems, regular patching
- **User Management**: Multi-factor authentication, role-based access
- **Network Monitoring**: SIEM systems, intrusion detection
- **Incident Response**: 24/7 SOC, incident response team
- **Training**: Regular cybersecurity awareness programs
- **Vendor Security**: Supply chain security requirements
- **Physical Security**: Access controls, surveillance systems

9 Peraturan Pemerintah Indonesia untuk Sistem Tenaga

Modern power systems memerlukan regulatory framework yang comprehensive untuk ensure reliability, safety, dan environmental protection.

9.1 UU No. 30 Tahun 2009 tentang Ketenagalistrikan

UU No. 30/2009 Key Provisions:

Tabel 14: UU No. 30/2009 Key Articles for Smart Grid Development

Article	Topic	Relevance
Article 7	Government Authority	License approval ur
Article 23	PLN Authority	Distribution pla
Article 35	Energy Conservation	Smart Grid untuk
Article 38	Renewable Energy	Integration requ
Article 46	Grid Access	Open access untuk
Article 68	Safety Standards	Cyber security unt
Article 72	Environmental | Environmental compliance untuk new technologies	

UU 30/2009 Implications untuk Bengkulu Smart Grid:

1. **Licensing**: PLN memerlukan izin untuk AMI deployment
2. **Tariff Structure**: New pricing mechanisms untuk TOU dan demand response
3. **Net Metering**: Revised regulations untuk rooftop PV dan distributed generation
4. **Safety Standards**: Compliance dengan cybersecurity requirements
5. **Environmental**: Life cycle assessment untuk new technologies

9.2 Permen ESDM Regulations

Key Permen ESDM untuk Modern Grid:

Tabel 15: Permen ESDM Regulations for Smart Grid

Permen	
Permen ESDM 12/2021 | Purchase of Renewable Energy	Articles 1
Permen ESDM 26/2021 | Distributed Generation | Articles 8-12 | Net metering rules	
Permen ESDM 15/2023 | Energy Conservation | Articles 5-10 | BEMS requirements	
Permen ESDM 8/2024 | Electric Vehicle Charging | Articles 12-18 | V2G regulations	
Permen ESDM 12/2024 | Smart Grid Standards | All articles | AMI, PMU, cybersecurity	

Permen ESDM 12/2024: Smart Grid Implementation:

- **AMI Requirements**: Rollout plan, technical standards, data privacy
- **PMU Deployment**: Phase 1 (100 units), Phase 2 (500 units)
- **Cybersecurity Standards**: Compliance dengan IEC 62351
- **Data Management**: Customer data protection, analytics requirements
- **Interoperability**: Standards untuk different vendors
- **Performance Monitoring**: KPIs untuk grid reliability improvement

9.3 RUPTL PLN 2021-2030

RUPTL PLN defines investment strategy untuk power system development including Smart Grid technologies:

RUPTL Investment Priorities untuk Smart Grid:

Tabel 16: RUPTL PLN 2021-2030 Smart Grid Investments

Program
AMI Deployment | 2.5 | 2024-2030 | 35 million meters | 99.5% automation
PMU Network | 0.8 | 2024-2028 | 600 units | Real-time monitoring
Cybersecurity | 1.2 | 2024-2030 | All systems | 99.9% availability
BEMS Implementation | 0.6 | 2025-2030 | 10,000 buildings | 15% energy savings
EV Charging Infrastructure | 3.2 | 2024-2030 | 50,000 stations | 2.3 million EVs | Energy Storage | 8.5
HVDC Transmission | 12.8 | 2024-2030 | 4,000 MW | Regional integration
Smart Substations | 2.1 | 2024-2030 | 500 substations | 25% efficiency | height**Total Smart Grid** | 3

Bengkulu-Specific RUPTL Implementation:

- **AMI Rollout**: 150,000 smart meters (2024-2027)
- **PMU Deployment**: 24 units untuk grid observability
- **BEMS Implementation**: 50 government buildings
- **EV Charging**: 100 charging stations
- **Energy Storage**: 100 MW / 400 MWh BESS
- **Smart Grid Lab**: UNIB collaboration untuk R&D

9.4 Regulations for Distributed Energy Resources (DER)

Net Metering Regulations:

$$Billing\ Credit = Energy\ Exported\ (kWh) \times Tariff\ Rate\ (\$/kWh) \tag{39}$$

Indonesia Net Metering Rules (Permen ESDM 26/2021):

- **Capacity Limit**: Up to 65% of connected load
- **Technology**: Solar, wind, hydro, biomass
- **Billing Period**: 3-month cycles
- **Credit Expiration**: Unused credits expire after 3 months
- **Tariff Structure**: Average electricity price

Prosumer Business Model untuk Bengkulu:

- **Residential Prosumer**: 5 kW rooftop solar + 10 kWh battery
- **Commercial Prosumer**: 50 kW solar + 100 kWh battery
- **Agricultural Prosumer**: 20 kW solar + wind hybrid
- **University Prosumer**: 500 kW solar + 1 MWh storage
- **Industrial Prosumer**: 2 MW solar + 5 MWh storage

Carbon Pricing Regulations:

$$Carbon\ Credit = \frac{Emissions\ Avoided\ (ton\ CO_2) \times Carbon\ Price\ (\$/ton\ CO_2)}{Revenue} \tag{40}$$

Indonesia Carbon Pricing (2024): $20 - 50/tonCOExpectedincreaseto100$/ton CO by 2030

9.5 Environmental Regulations

AMDAL Requirements untuk Smart Grid:

- **Electronic Waste**: Proper disposal of smart meters, batteries
- **Electromagnetic Fields**: EMF compliance untuk AMI deployment
- **Resource Consumption**: Material usage assessment
- **Energy Balance**: Lifecycle energy analysis
- **Smart Grid Carbon Footprint**: Operational emissions

Environmental Impact untuk Bengkulu Projects:

Tabel 17: Environmental Impact Assessment - Bengkulu Smart Grid

Project
AMI Rollout | 15,000 | 80% paper reduction | 95% data accuracy | A+
PMU Network | 8,500 | Minimal waste | 99% observability | A | BESS (100 MW) | 45,000 | Battery r
HVDC Project | 25,000 | Minimal land use | 70% lower losses | A | height**Total** | 190,500 | Significant

10 Kesimpulan

Transformasi sistem tenaga listrik menuju era digital dan berkelanjutan memerlukan pendekatan holistik yang mengintegrasikan teknologi canggih dengan regulatory framework yang comprehensive.

10.1 Lima Pilar Utama Transformasi

1. Smart Grid Foundation:

$$Efficiency\ Gain = \eta_{traditional} \times (1 + \alpha_{smart\ grid}) = 0.75 \times 1.25 = 93.75\% \tag{41}$$

2. Renewable Energy Integration:

$$VRE\ Penetration\ Target = \frac{Capacity_{renewable}}{Capacity_{total}} = \frac{3.5\ GW}{5.0\ GW} = 70\%\ by\ 2030 \tag{42}$$

3. Energy Storage Deployment:

$$Storage\ Requirement = \frac{VRE\ Capacity \times Duration}{Efficiency} = \frac{3.5\ GW \times 4\ h}{0.9} = 15.6\ GWh \tag{43}$$

4. Digitalization Implementation:

- AMI penetration: 99.5% by 2027
- PMU deployment: 600 units by 2028
- Cybersecurity compliance: 100% by 2026
- BEMS adoption: 10,000 buildings by 2030

5. Policy dan Regulatory Support:

- UU No. 30/2009 amendments untuk Smart Grid
- Permen ESDM updates untuk DER integration
- RUPTL PLN alignment dengan NZE 2060
- Carbon pricing mechanism implementation

10.2 Bengkulu's Roadmap ke Smart Grid

Phase 1 (2024-2025): Foundation

- AMI pilot deployment (25,000 meters)
- PMU installation (6 units)
- BEMS implementation (10 government buildings)

Phase 2 (2026-2027): Expansion

- AMI full rollout (150,000 meters)

- PMU network completion (24 units)
- V2G pilot (100 EVs)
- 50 MW BESS deployment

Phase 3 (2028-2030): Integration

- Full Smart Grid operations
- 70% renewable energy penetration
- 200 MWh energy storage
- Vehicle-to-grid (V2G) network (500 EVs)

Economic Impact untuk Bengkulu:

- **Job Creation**: 5,000 direct jobs, 15,000 indirect jobs
- **Cost Savings**: $150M annually dalam energy efficiency
- **GDP Contribution**: $800M additional economic output
- **Investment Attraction**: $2.5B dalam Smart Grid technologies
- **Energy Independence**: 70% renewable energy by 2030