Performance Improvement of Utilities Power Grids with Virtual Power Plant Integration

1.1 Research Objectives

The study aims to develop a smart energy management system (EMS) for grid-connected Virtual Power Plants (VPPs) that efficiently coordinates PV generation, loads, battery energy storage system (BESS), and electric vehicles (EVs). It integrates advanced forecasting using AI-based models to improve predictions of renewable generation, load demand, EVs arrival/departure times, and electricity prices, and uses a Mixed-Integer Non-Linear Programming optimization framework to balance economic and technical objectives. The framework enhances PV integration, enables prosumers market participation, supports bidirectional EV-grid interaction (G2V/V2G), and evaluates trade-offs between cost minimization and grid-support.

1.2 VPP Component Modelling

The single-line diagram of the proposed system is produced with the power flows.

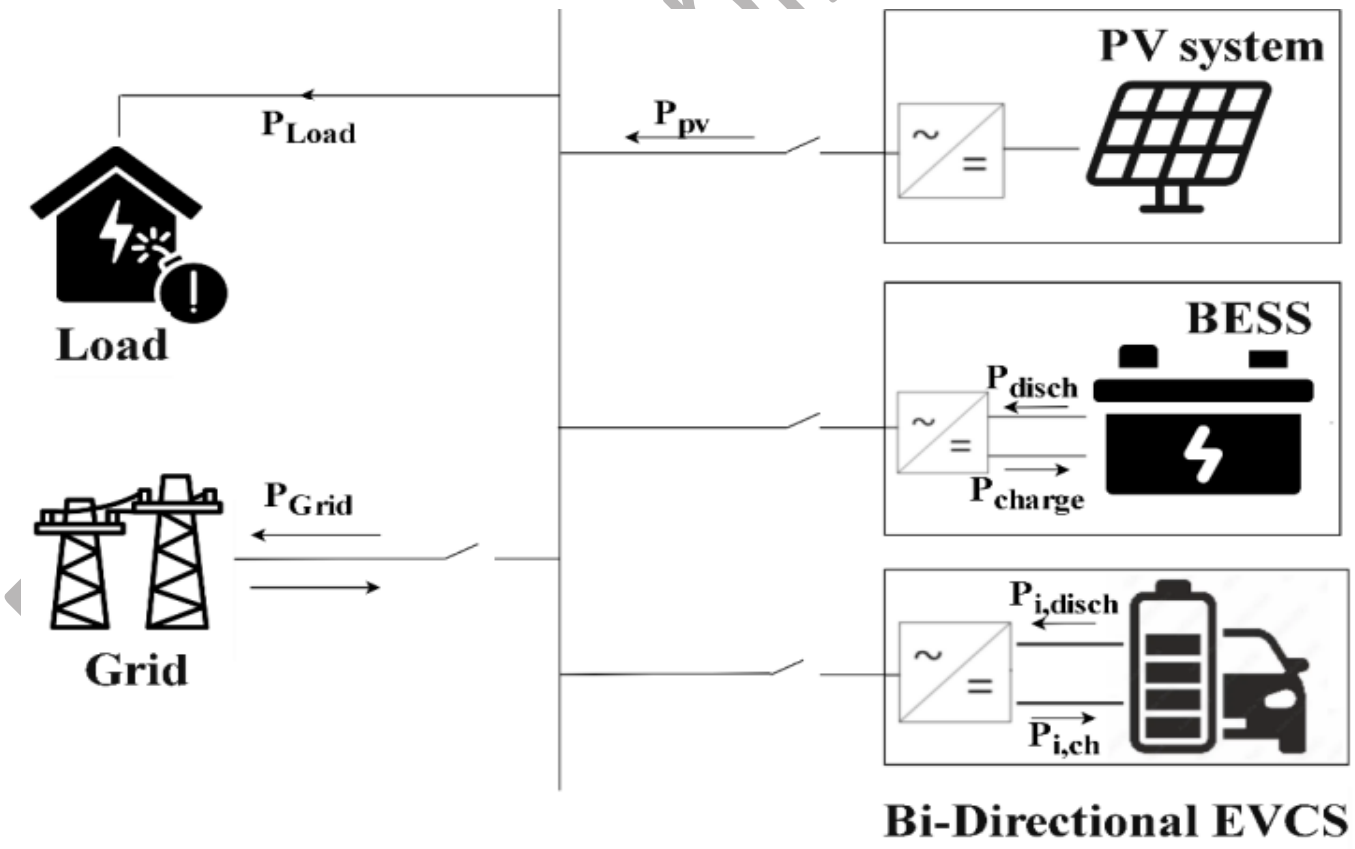


Fig. 3-1 Structure of the case study [69]

$$P_{grid}[t] + P_{pv}[t] + P_{BESS}^{dis}[t] * \eta_{inv}^{BESS} + \sum\left(P_{EV,i}^{dis}[t] * \eta_{inv}^{EV,i}\right) = P_{load}[t] + \frac{P_{BESS}^{ch}[t]}{\eta_{inv}^{BESS}} + \sum\left(\frac{P_{EV,i}^{ch}[t]}{\eta_{inv}^{EV,i}}\right) \quad (3\text{-}1)$$

1.2.1 EMS Architecture

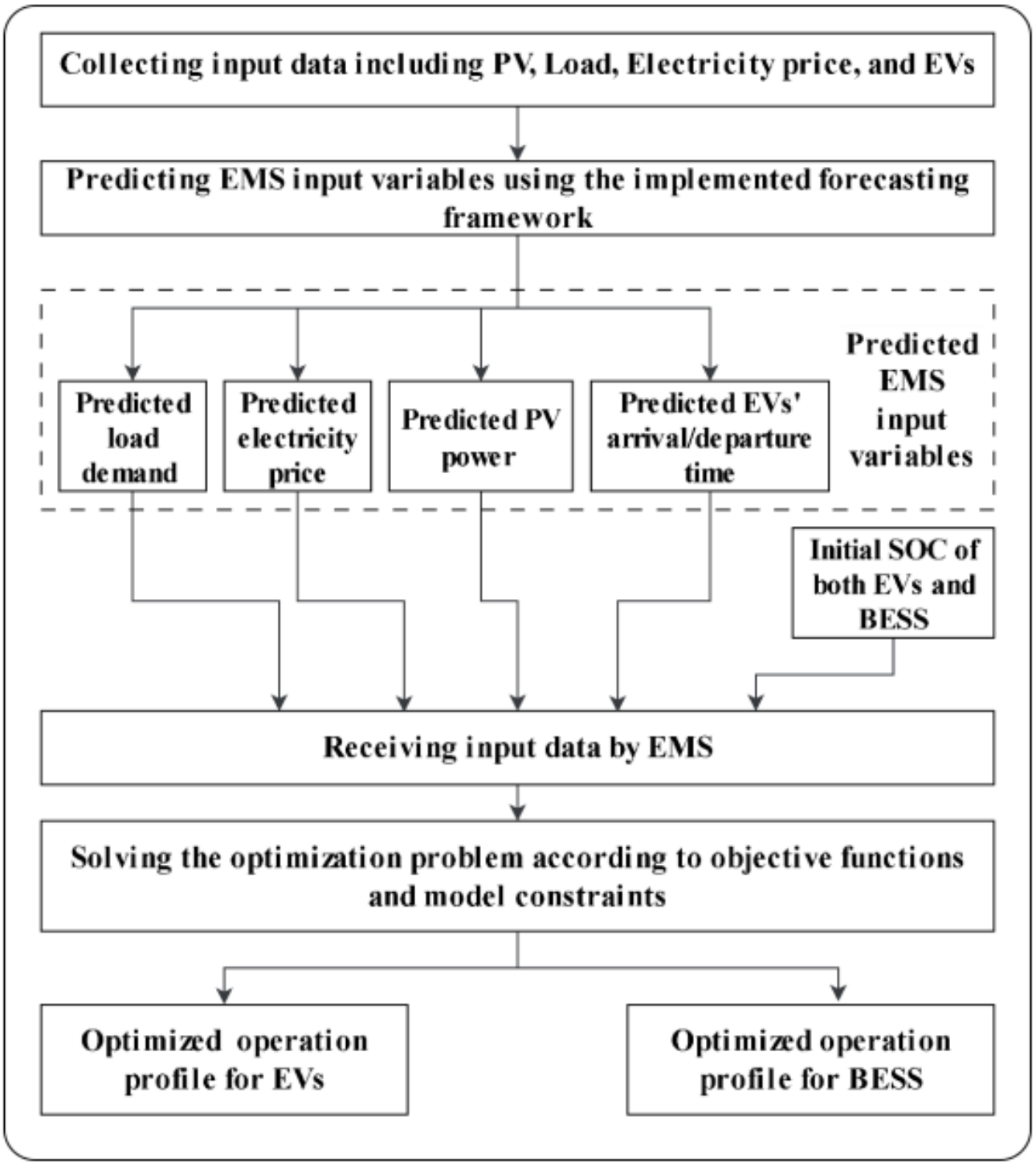


Fig. 3-4 Proposed EMS architecture within the VPP configuration

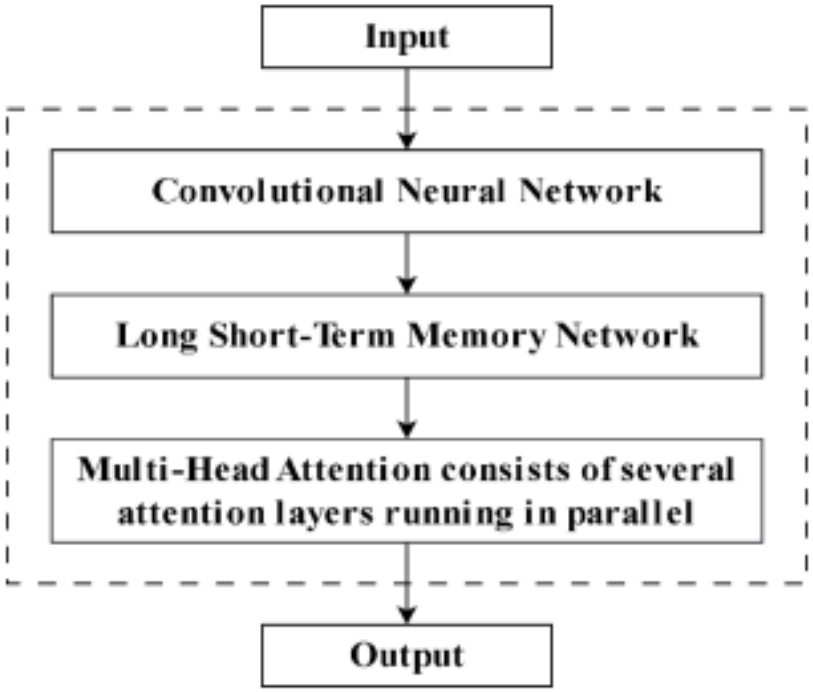


Fig. 4-18 CNN-LSTM with Multi-Head Attention architecture

1.3 Forecasting of EMS Input Variables

Table 4-15 Evaluation metrics of the predicted load demand

	LSTM	TRNN	CNN-LSTM	Prediction Model
MAPE (%)	0.85	0.83	0.58	0.45
RMSE	53.32	42.15	32.03	24
MAE	36.39	36.32	25.71	4.38
R^2 (%)	95.35	97.22	98.42	99.13

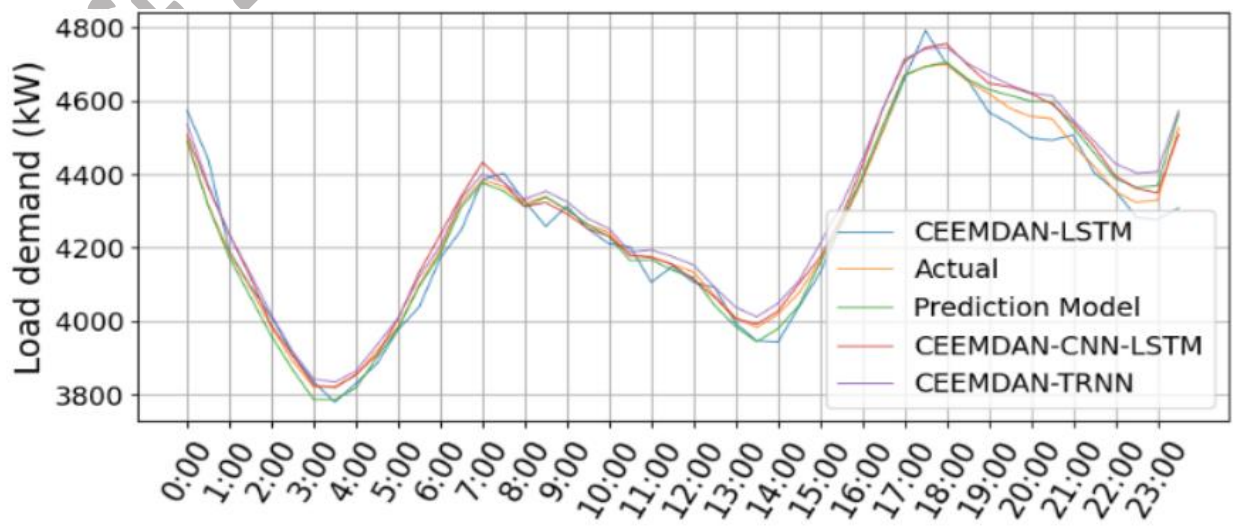


Fig. 4-30 Predicted load demand compared with validation models

Table 4-17 Evaluation metrics of the predicted PV generation

	LSTM	TRNN	CNN-LSTM	Prediction Model
RMSE	79.40	99.22	60.82	40.60
MAE	49.82	84.45	33.84	22.76
R^2 (%)	98.05	96.85	98.64	99.44

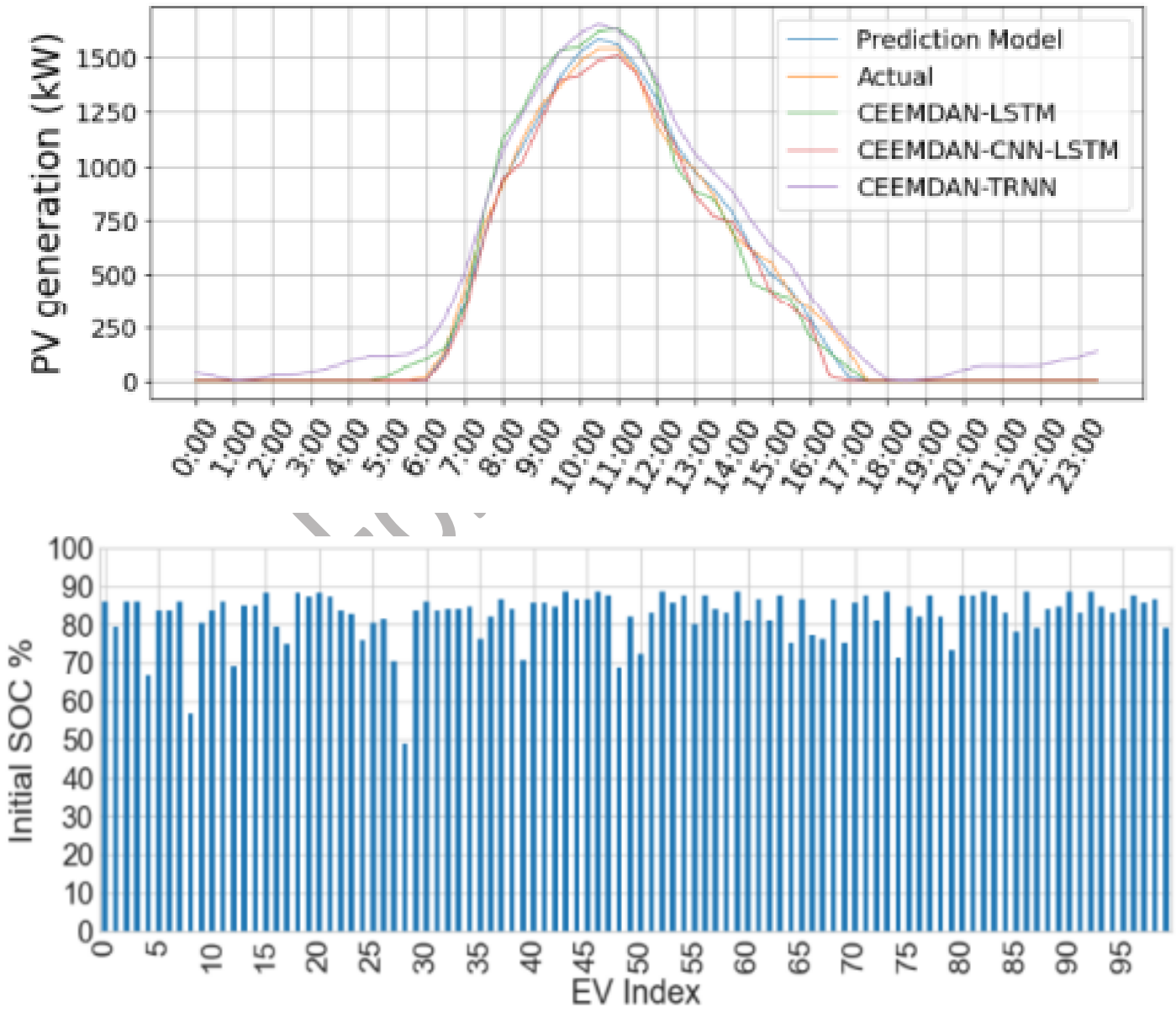


Fig. 5-4 Estimated SOC for each EV upon arrival as input to the EMS

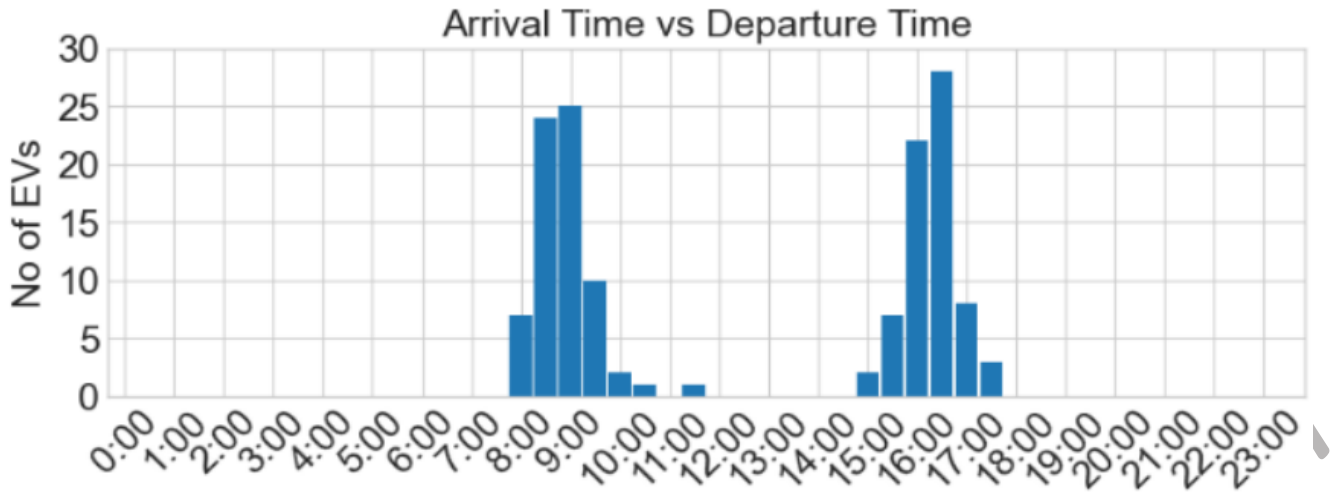


Fig. 5-5 Histogram of predicted EV arrival and departure times as input to the EMS

1.4 Case Studies

- **Base Case:** EVs only charge during their parking time, regardless of the electricity price, which slightly increased stress on the utility grid
- **Case 1**: Minimization of Power Imported from the Grid
- **Case 2:** Minimization of Electricity Cost
- **Case 3:** Cost Minimization with Peak Shaving Constraint
- **Case 4:** Multi-Objective Optimization (Case1 + Case2)

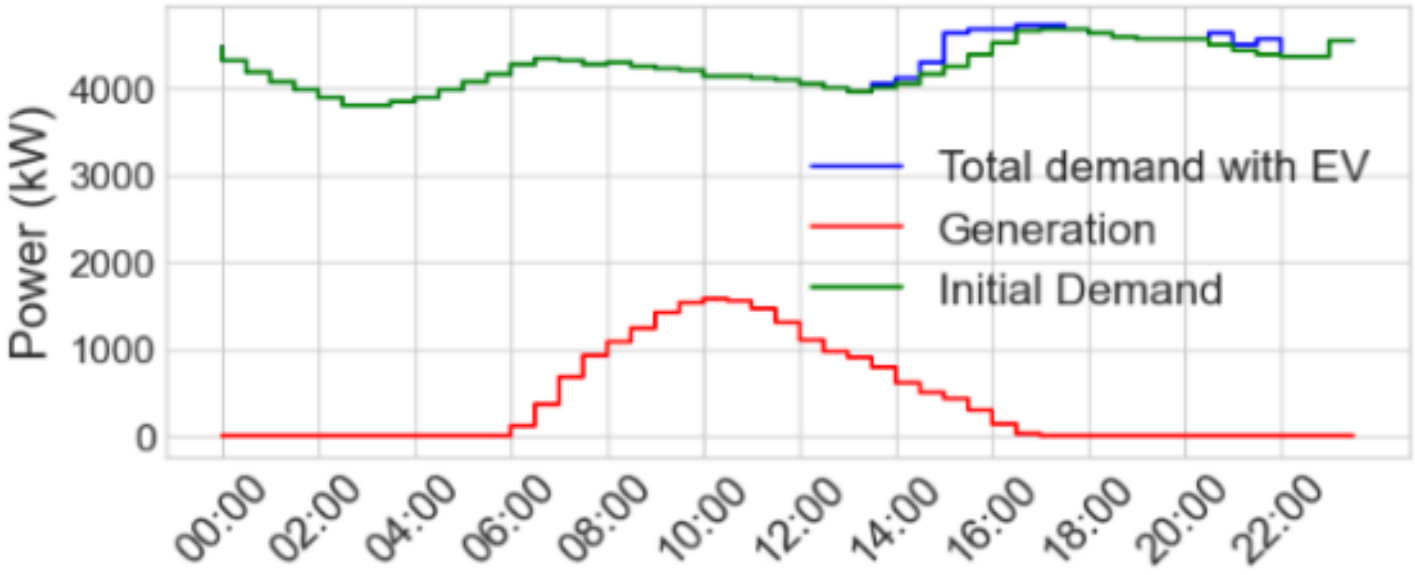


Fig. 5-8 Initial demand and generation in the base case

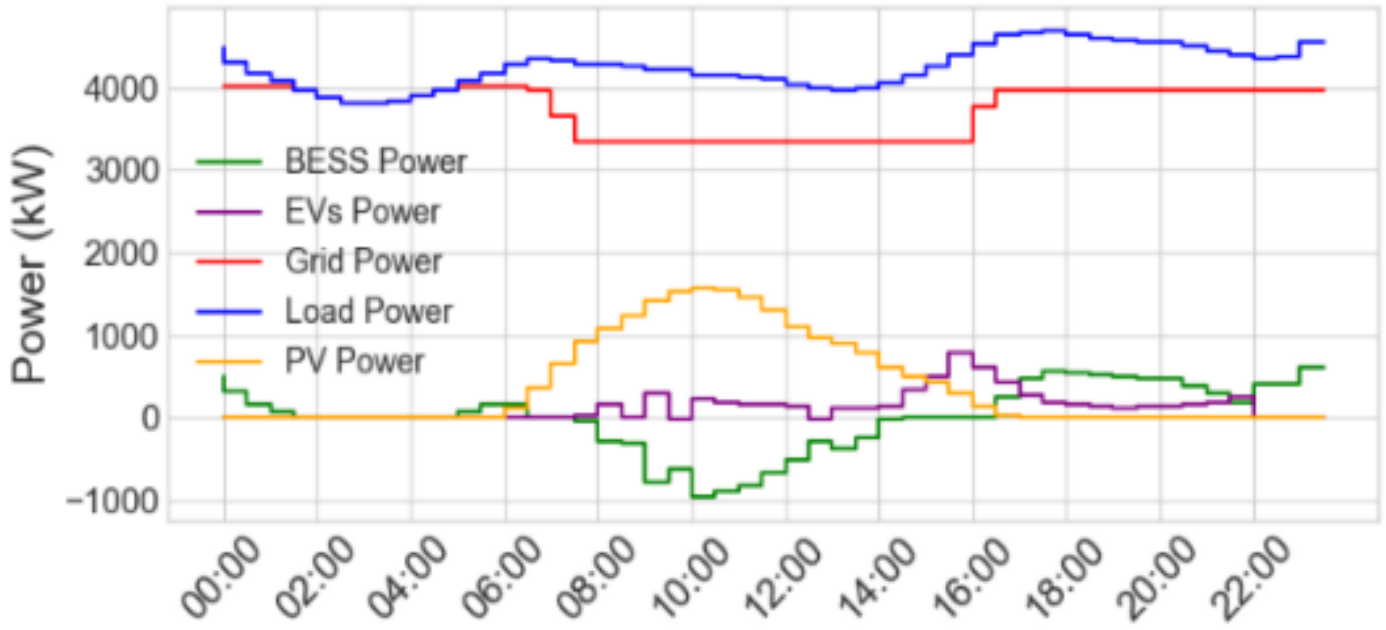


Fig. 5-10 Overview of EMS performance in Case 1

Fig. 5-15 Scheduled power for the BESS and EVs in Case 2

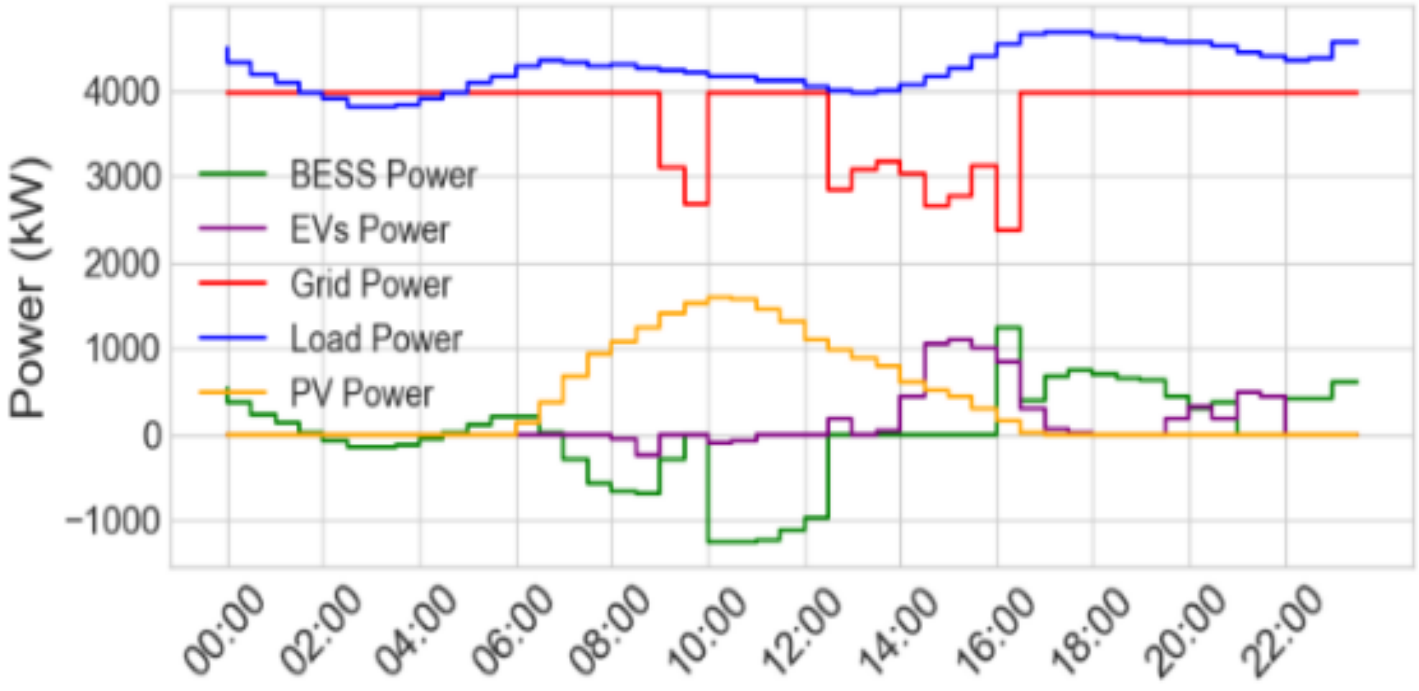
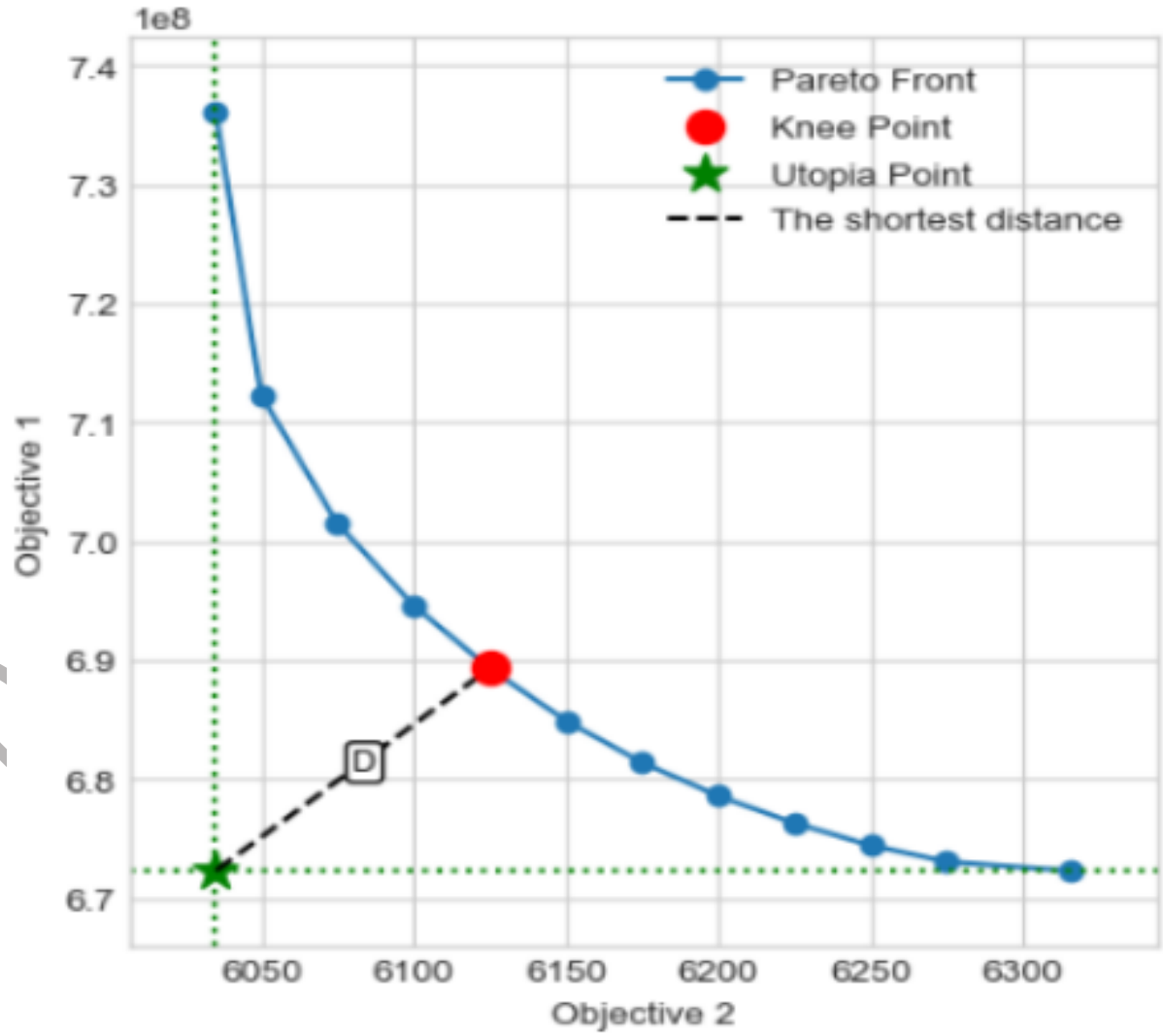


Fig. 5-12 Overview of EMS performance in Case 3



1.5 Results

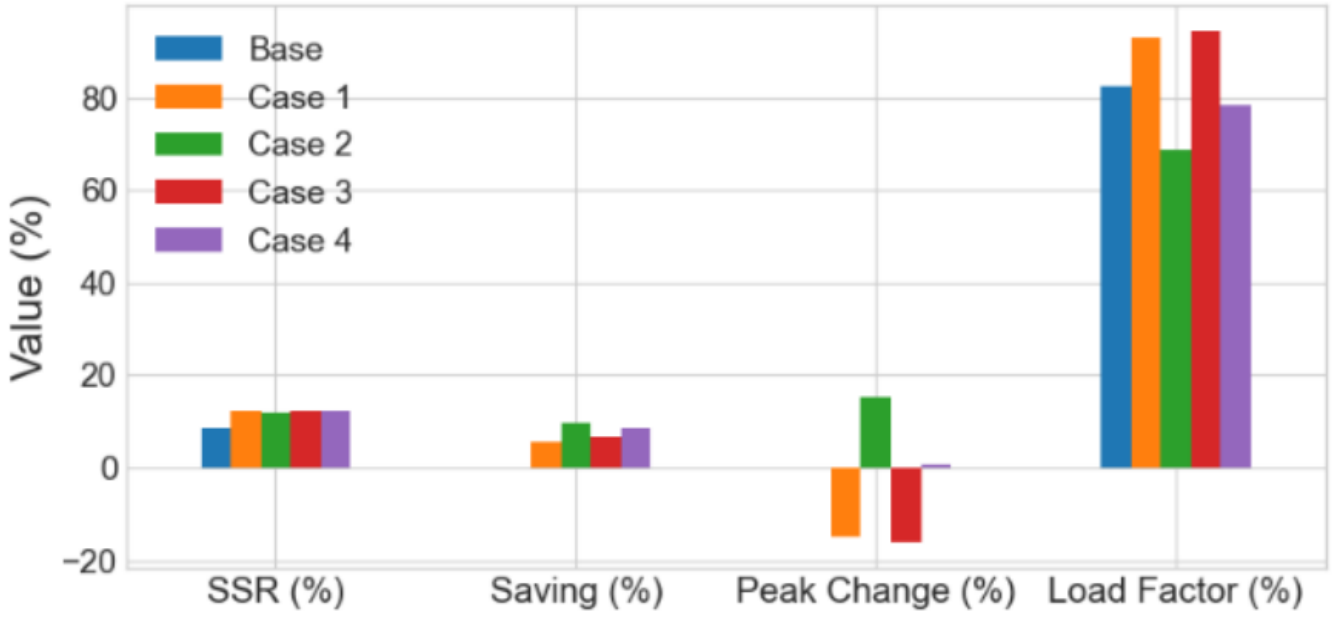


Fig. 5-30 Comparison of key performance metrics

1.6 Code Snippets

```
demand_df=pd.read_csv(r"C:\Users\Basel\Desktop\Optimization Model V13\Input data\Day-Ahead-Load.csv",index_col=[0])
```

```
price_df=pd.read_csv(r"C:\Users\Basel\Desktop\Optimization Model V13\Input data\Day-Ahead-Price.csv",index_col=[0])
```

```
vpp = ConcreteModel()
opt = SolverFactory('scip', executable=r'C:\Program Files\SCIPOptSuite 9.2.2\bin\scip.exe')
```

```
def Balance_Constraint(vpp, t):
    EVs_Sum_Discharging = sum((getattr(vpp, f'EV_Discharge_Power_{i}')[t] * EV_INV_EFFICIENCY )    for i in vpp.SetIndex)
    EVs_Sum_Charging = sum((getattr(vpp, f'EV_Charge_Power_{i}')[t] / EV_INV_EFFICIENCY )   for i in vpp.SetIndex)
    return vpp.Grid[t] + vpp.PV[t] +(vpp.BESS_Discharge_Power[t] * BESS_INV_EFFICIENCY )
+ EVs_Sum_Discharging == vpp.Load[t]+(vpp.BESS_Charge_Power[t] / BESS_INV_EFFICIENCY )+ EVs_Sum_Charging


vpp.Balance_Constraint = Constraint(vpp.Period, rule=Balance_Constraint)
```

```
vpp.BESS_Energy = Var(vpp.Period, bounds=(BESS_MIN_ENERGY, BESS_MAX_ENERGY))
vpp.BESS_SOC = Var(vpp.Period, bounds=(BESS_MIN_SOC, BESS_MAX_SOC))
vpp.BESS_Charge_Power = Var(vpp.Period, bounds=(0,BESS_MAX_CH_POWER))
vpp.BESS_Discharge_Power = Var(vpp.Period, bounds=(0,BESS_MAX_DIS_POWER))
vpp.BESS_Discharge_State=Var(vpp.Period,domain=Binary)
vpp.BESS_Charge_State=Var(vpp.Period,domain=Binary)
```

```
def BESS_Energy_Constraint(vpp, t):
    if t == vpp.Period.first():
        return vpp.BESS_Energy[t] == BESS_INITIAL_ENERGY
    return vpp.BESS_Energy[t] == (vpp.BESS_Energy[t-1]
                                  + (vpp.BESS_Charge_Power[t-1] * dt * BESS_CH_EFFICIENCY )
                                  - (vpp.BESS_Discharge_Power[t-1]  * dt / BESS_DIS_EFFICIENCY))
```

```
for i in vpp.SetIndex:

    setattr(vpp, f'EV_Energy_{i}', Var(vpp.Period, bounds=(EV_MIN_ENERGY, EV_MAX_ENERGY)))
    setattr(vpp, f'EV_SOC_{i}', Var(vpp.Period, bounds=(EV_MIN_SOC, EV_MAX_SOC)))
    setattr(vpp, f'EV_Charge_Power_{i}', Var(vpp.Period, bounds=(0,EV_MAX_CH_POWER)))
    setattr(vpp, f'EV_Discharge_Power_{i}', Var(vpp.Period, bounds=(0,EV_MAX_DIS_POWER)))
    setattr(vpp, f'EV_Discharge_State_{i}', Var(vpp.Period,domain=Binary))
    setattr(vpp, f'EV_Charge_State_{i}', Var(vpp.Period,domain=Binary))
```

```
def EV_Energy_Constraint_Rule(vpp, t):
    if t == vpp.Period.first():
        return getattr(vpp, f'EV_Energy_{i}')[t] == evs_initial_energy_df.loc[i,"Initial_Energy"]
    return getattr(vpp, f'EV_Energy_{i}')[t] == (
        getattr(vpp, f'EV_Energy_{i}')[t - 1]
        + (getattr(vpp, f'EV_Charge_Power_{i}')[t - 1] * dt * EV_CH_EFFICIENCY)
        - (getattr(vpp, f'EV_Discharge_Power_{i}')[t - 1]  * dt / EV_DIS_EFFICIENCY))
```

```
def Grid_Power(vpp,t):
    gridPower = sum(vpp.Grid[t] ** 2  for t in vpp.Period)
    return (1e-1) * gridPower
```

```
def Grid_Cost(vpp,t):
    gridCost = sum(vpp.Grid[t] * df.price[t] * dt  for t in vpp.Period)
    return gridCost
```

```
UTH=3965
```

```
def UTH_Constraint(vpp, t):
    return abs(vpp.Grid[t]) <= UTH
```

```
#vpp.UTH_Con = Constraint(vpp.Period, rule=UTH_Constraint)
#vpp.Epilson_Constraint = Constraint(vpp.Period, rule=epilson_constraint)
vpp.objective = Objective(rule=Grid_Power , sense=minimize )
#vpp.objective = Objective(rule=Grid_Cost , sense=minimize)
```

```
SSR = (1 - (df['Grid'] * dt).sum() / (df['Load'] * dt).sum()) * 100
Bill = df['Cost'].sum()
Peak = df['Grid'].max()
Saving = abs((Bill - Base_Bill) / Base_Bill) * 100
Load_Factor = df['Grid'].mean() / df['Grid'].max() * 100
SCR = ((df['PV'] * dt).sum() - abs((df.loc[df['Grid'] < 0, 'Grid'] * dt)).sum()) / (df['PV'] * dt).sum() * 100
Peak_Reduction = (Peak - Base_Peak) / Base_Peak * 100
```

- **Used Tools**
  - **Python** – for coding, data analysis, and model implementation
  - **Pyomo** – for modelling optimization problems
  - **SCIP** – for solving optimization problems
  - **Pandas & NumPy** – for data manipulation and numerical calculations
  - **Matplotlib & Seaborn** – for visualization of results and analysis
  - **Machine learning libraries:** scikit-learn and TensorFlow