# LLM GPT 2

# Psudocode

Algorithm: Enhanced Energy Recommendation System

Input: Smart home energy usage data

Output: Energy-saving recommendations and consumption predictions

1: // Main Data Pipeline

2: function MAIN()

3: // Data Loading and Preprocessing

4: data ← LOAD\_AND\_PREPROCESS("https://dataset.url")

5:

6: // LSTM Component

7: X\_train, X\_val, X\_test, y\_train, y\_val, y\_test ← PREPARE\_LSTM\_DATA(data)

8: lstm\_model ← CREATE\_LSTM\_MODEL((24, X\_train\_seq.shape[2]))

9: TRAIN\_LSTM(lstm\_model, X\_train\_seq, y\_train\_seq, X\_val\_seq, y\_val\_seq)

10:

11: // LLM Component

12: embedding\_model ← SentenceTransformer('all-MiniLM-L6-v2')

13: data['text\_description'] ← GENERATE\_TEXT\_DESCRIPTIONS(data)

14: embeddings ← embedding\_model.encode(data['text\_description'])

15: index ← BUILD\_FAISS\_INDEX(embeddings)

16:

17: // GPT-2 Setup

18: tokenizer, llm\_model ← INITIALIZE\_LLM("gpt2")

19: train\_examples ← GENERATE\_TRAINING\_EXAMPLES(data, embeddings, index)

20: TRAIN\_LLM(llm\_model, tokenizer, train\_examples)

21:

22: // Generate Recommendation

23: recommendation ← GENERATE\_RECOMMENDATION(100, data, embeddings, index, llm\_model, tokenizer)

24: PRINT(recommendation)

25: end function

26: // Data Preprocessing

27: function LOAD\_AND\_PREPROCESS(url)

28: df ← pd.read\_csv(url)

29: df ← DROP\_COLUMNS(df, ['Transaction\_ID', 'Unix Timestamp'])

30:

31: // Temporal Features

32: df['is\_peak\_hour'] ← APPLY\_PEAK\_HOUR\_LOGIC(df['Hour of the Day'])

33: df['part\_of\_day'] ← CATEGORIZE\_TIME\_OF\_DAY(df['Hour of the Day'])

34: df['is\_weekend'] ← IDENTIFY\_WEEKEND(df['Day of the Week'])

35:

36: // Seasonal Features

37: df['Season'] ← MAP\_TO\_SEASONS(df['Month'])

38:

39: // Cyclical Features

40: df['hour\_sin'] ← SIN\_CYCLE(df['Hour of the Day'], 24)

41: df['hour\_cos'] ← COS\_CYCLE(df['Hour of the Day'], 24)

42:

43: // Energy Features

44: appliances ← ['Television', 'Dryer', 'Oven', 'Refrigerator', 'Microwave']

45: df['total\_appliance\_usage'] ← SUM\_APPLIANCES(df, appliances)

46: threshold ← CALCULATE\_THRESHOLD(df['Energy Consumption (kWh)'], 0.75)

47: df['is\_high\_consumption'] ← FLAG\_HIGH\_CONSUMPTION(df['Energy Consumption (kWh)'], threshold)

48:

49: // Feature Scaling

50: features\_to\_scale ← ['Line Voltage', 'Voltage', 'Apparent Power', 'Energy Consumption (kWh)']

51: df[features\_to\_scale] ← SCALE\_FEATURES(df[features\_to\_scale])

52:

53: // Efficiency Metrics

54: for each appliance in appliances do

55: df[appliance+'\_efficiency\_ratio'] ← CALCULATE\_EFFICIENCY(df[appliance], df['Energy Consumption (kWh)'])

56: end for

57:

58: // Additional Metrics

59: df['power\_factor'] ← CALCULATE\_POWER\_FACTOR(df['Apparent Power'], df['Line Voltage'], df['Voltage'])

60: df['active\_appliances'] ← COUNT\_ACTIVE\_APPLIANCES(df, appliances)

61: df['energy\_per\_active\_appliance'] ← CALCULATE\_ENERGY\_PER\_APPLIANCE(df['Energy Consumption (kWh)'], df['active\_appliances'])

62:

63: return df

64: end function

65: // LSTM Component

66: function PREPARE\_LSTM\_DATA(df)

67: cat\_cols ← ['Season', 'part\_of\_day', 'Day of the Week', 'Offloading Decision']

68: df\_encoded ← ONE\_HOT\_ENCODE(df, cat\_cols)

69:

70: X ← DROP\_COLUMN(df\_encoded, 'is\_high\_consumption')

71: y ← EXTRACT\_COLUMN(df\_encoded, 'is\_high\_consumption')

72:

73: // Month Mapping

74: month\_mapping ← CREATE\_MONTH\_MAPPING()

75: X['Month'] ← MAP\_VALUES(X['Month'], month\_mapping)

76:

77: // Data Splitting

78: X\_train, X\_temp, y\_train, y\_temp ← SPLIT\_DATA(X, y, test\_size=0.4)

79: X\_val, X\_test, y\_val, y\_test ← SPLIT\_DATA(X\_temp, y\_temp, test\_size=0.5)

80:

81: // Type Conversion

82: X\_train, X\_val, X\_test ← CONVERT\_TO\_FLOAT32(X\_train, X\_val, X\_test)

83: y\_train, y\_val, y\_test ← CONVERT\_TO\_FLOAT32(y\_train, y\_val, y\_test)

84:

85: return X\_train, X\_val, X\_test, y\_train, y\_val, y\_test

86: end function

87: function CREATE\_LSTM\_MODEL(input\_shape)

88: model ← Sequential([

89: LSTM(64, input\_shape=input\_shape, return\_sequences=False),

90: Dropout(0.3),

91: Dense(32, activation='relu'),

92: Dense(1)

93: ])

94: model.compile(optimizer='adam', loss='mse', metrics=['mae'])

95: return model

96: end function

97: function CREATE\_SEQUENCES(X, y, time\_steps=24)

98: Xs, ys ← [], []

99: for i ← 0 to len(X)-time\_steps do

100: Xs.append(X[i:i+time\_steps])

101: ys.append(y[i+time\_steps])

102: end for

103: return np.array(Xs), np.array(ys)

104: end function

105: function TRAIN\_LSTM(model, X\_train, y\_train, X\_val, y\_val)

106: train\_ds ← CREATE\_TF\_DATASET(X\_train, y\_train, batch\_size=32, shuffle=True)

107: val\_ds ← CREATE\_TF\_DATASET(X\_val, y\_val, batch\_size=32)

108: model.fit(train\_ds, validation\_data=val\_ds, epochs=20)

109: end function

110: // LLM Component

111: function GENERATE\_TEXT\_DESCRIPTIONS(df)

112: descriptions ← []

113: for each row in df do

114: appliances\_status ← JOIN(

115: FORMAT\_APPLIANCE\_STATUS(appliance, row[appliance])

116: for appliance in ['Television', 'Dryer', 'Oven', 'Refrigerator', 'Microwave']

117: )

118: desc ← FORMAT(

119: "At {hour}:00 during {season} {part\_of\_day}, appliances: {status}. Energy: {energy:.2f}kWh",

120: hour=row['Hour of the Day'],

121: season=row['Season'],

122: part\_of\_day=row['part\_of\_day'],

123: status=appliances\_status,

124: energy=row['Energy Consumption (kWh)']

125: )

126: descriptions.append(desc)

127: end for

128: return descriptions

129: end function

130: function BUILD\_FAISS\_INDEX(embeddings)

131: index ← faiss.IndexFlatL2(embeddings.shape[1])

132: index.add(embeddings)

133: return index

134: end function

135: function INITIALIZE\_LLM(model\_name)

136: tokenizer ← AutoTokenizer.from\_pretrained(model\_name)

137: tokenizer.pad\_token ← tokenizer.eos\_token

138: model ← AutoModelForCausalLM.from\_pretrained(model\_name)

139: return tokenizer, model

140: end function

141: function GENERATE\_TRAINING\_EXAMPLES(data, embeddings, index, k=5)

142: train\_examples ← []

143: sample\_indices ← RANDOM\_SAMPLE(data, size=5)

144:

145: for each idx in sample\_indices do

146: query\_embedding ← embeddings[idx:idx+1]

147: distances, indices ← index.search(query\_embedding, k)

148: similar\_data ← data.iloc[indices[0]]

149:

150: prompt ← GENERATE\_PROMPT\_WITH\_EMBEDDINGS(similar\_data, distances)

151: recommendations ← GENERATE\_DYNAMIC\_RECOMMENDATIONS(similar\_data.iloc[0])

152: full\_text ← prompt + " " + recommendations

153: train\_examples.append({"text": full\_text})

154: end for

155: return train\_examples

156: end function

157: function GENERATE\_PROMPT\_WITH\_EMBEDDINGS(similar\_data, distances)

158: prompt ← "You are an AI assistant specialized in smart home energy recommendations...\n"

159: prompt ← prompt + "Based on the following similar appliance usage patterns:\n\n"

160:

161: for idx, (i, row) in similar\_data.iterrows() do

162: prompt ← prompt + FORMAT\_SIMILAR\_PATTERN(row, idx+1)

163: end for

164:

165: prompt ← prompt + "Recommended energy-saving actions:"

166: return prompt

167: end function

168: function TRAIN\_LLM(model, tokenizer, examples)

169: train\_dataset ← CONVERT\_TO\_DATASET(examples)

170: tokenized\_data ← TOKENIZE\_DATA(tokenizer, train\_dataset)

171:

172: training\_args ← TrainingArguments(

173: output\_dir="./gpt2-energy-finetuned",

174: per\_device\_train\_batch\_size=4,

175: num\_train\_epochs=40,

176: learning\_rate=3e-5,

177: fp16=False

178: )

179:

180: trainer ← Trainer(

181: model=model,

182: args=training\_args,

183: train\_dataset=tokenized\_data,

184: data\_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False)

185: )

186:

187: try

188: trainer.train()

189: catch error

190: model ← model.to('cpu')

191: trainer ← REINITIALIZE\_TRAINER(model, training\_args, tokenized\_data, tokenizer)

192: trainer.train()

193: end try

194:

195: SAVE\_MODEL(model, tokenizer, "./fine\_tuned\_gpt2")

196: end function

197: function GENERATE\_RECOMMENDATION(query\_idx, data, embeddings, index, model, tokenizer, k=3)

198: query\_embedding ← embeddings[query\_idx:query\_idx+1]

199: distances, indices ← index.search(query\_embedding, k)

200: similar\_data ← data.iloc[indices[0]]

201:

202: prompt ← GENERATE\_PROMPT\_WITH\_EMBEDDINGS(similar\_data, distances)

203:

204: inputs ← TOKENIZE\_PROMPT(tokenizer, prompt)

205: outputs ← GENERATE\_TEXT(model, inputs)

206: response ← DECODE\_OUTPUTS(tokenizer, outputs)

207:

208: if "Recommended energy-saving actions:" in response then

209: return EXTRACT\_RECOMMENDATIONS(response)

210: else

211: return response

212: end if

213: end function

# Justification for each of 8 chosen chart

Training & Validation Loss Curves – Shows model convergence and detects overfitting by comparing training vs. validation loss over epochs.

Training & Validation Accuracy (MAE) Curves – Tracks prediction error reduction during training, ensuring the model learns effectively.

Precision-Recall Curve – Evaluates recommendation quality under different thresholds, important for imbalanced energy consumption data.

ROC Curve – Measures the trade-off between true positive rate (correct high-consumption predictions) and false alarms.

Cumulative Gain Chart – Shows how quickly the model identifies high-energy usage instances, useful for prioritizing recommendations.

t-SNE Embeddings – Visualizes semantic patterns in appliance usage descriptions, helping validate the LLM's understanding.

Training Time per Epoch – Highlights computational efficiency, crucial for real-time smart home applications.

Recommendation Examples – Provides qualitative insights into the LLM’s reasoning beyond quantitative metrics.

# 6 lines on each chart is k parallel pictures waly folders me se utha kr report me la ga lai each explanation heading k

1. Training & Validation Loss Curves

This chart reveals the model’s learning dynamics by plotting loss over epochs. The training loss (blue) represents how well the model fits the data, while validation loss (orange) tests generalization. A converging gap between curves suggests stable learning, whereas divergence hints at overfitting—critical for ensuring the model doesn’t just memorize energy usage patterns. The smooth descent implies effective gradient updates, and plateaus may signal the need for architectural tweaks. Ultimately, it answers: Does the model learn principles or noise?

2. Training & Validation MAE Curves

Mean Absolute Error (MAE) measures practical utility—how far predictions deviate from true energy values in kWh. Unlike loss, MAE is interpretable: a value of 0.1 means ~100 Wh error per prediction. Parallel curves indicate consistent generalization, while erratic validation MAE suggests unstable temporal patterns. The slope reflects learning speed, and final values contextualize real-world reliability (e.g., 0.05 MAE ≈ 5% error). It asks: Can users trust these predictions to manage consumption?

3. Precision-Recall Curve

This curve dissects the model’s trade-offs in identifying high-energy events. High precision (few false alarms) is vital to avoid unnecessary user interventions, while recall ensures no critical events are missed. The area under the curve (AUC) quantifies robustness—closer to 1 means the model balances both well. A steep drop at high recall implies the model struggles with edge cases (e.g., rare appliance combinations). It probes: How wisely does the system prioritize alerts?

4. ROC Curve

The ROC curve evaluates discriminative power—how well the model separates high/low consumption days. The diagonal represents random guessing; curves above it show predictive value. AUC near 1 suggests near-perfect separation, while a shallow curve indicates confusion (e.g., overlapping voltage patterns). The "knee" of the curve reveals optimal thresholds for actionability. It questions: Can the model distinguish critical vs. normal usage scenarios?

5. Cumulative Gain Chart

This plot measures efficiency in ranking high-consumption instances. The ideal line (top-left to top-right) would flag all true events immediately. Our model’s curve shows how much faster it identifies issues versus random inspection—e.g., capturing 80% of problems in 20% of time. The gap from the diagonal reflects value-add. It challenges: Does this save users time in diagnosing energy waste?

6. t-SNE Embeddings

This visualization maps semantic relationships between usage descriptions in 2D. Clusters imply similar appliance patterns (e.g., kitchen devices grouped), while outliers may represent anomalies. Overlapping regions suggest the LLM conflates distinct behaviors, and voids reveal underrepresented scenarios. Colors (energy intensity) should gradient smoothly if the model captures physics. It asks: Does the AI "understand" energy contexts spatially?

7. Training Time per Epoch

This bar chart exposes computational scalability. Linear growth suggests stable resource use, while spikes may indicate bottlenecks (e.g., memory swaps). Times >1s/epoch could hinder real-time updates. Compared to model accuracy, it answers: Is the performance gain worth the wait? For edge devices, this dictates deployability.

8. Recommendation Examples

These textual outputs bridge metrics and usability. Coherent explanations (e.g., "Shift dryer use to off-peak") validate the LLM’s reasoning, while hallucinations (e.g., "Turn off unused solar panels") reveal training gaps. Consistency across examples reflects stability, and specificity (e.g., kWh savings estimates) builds trust. It questions: Would a homeowner find this advice actionable?  
  
  
  
Same goes for tables

### ****Table 1: Strength of Recommendations****

The zero-values in Precision@5, Recall@5, and Hit Rate@5 suggest the model fails to identify high-consumption events in top recommendations—either due to imbalanced data or insufficient feature learning. The modest MAP (0.208) indicates sporadic correctness in ranking, while the near-zero MRR (0.028) implies rare early hits in ordered lists. NDCG@5’s null score confirms poor ranking quality, likely because the model prioritizes low-impact events. This table exposes a critical gap: the system detects energy anomalies but cannot prioritize them effectively, necessitating threshold tuning or representation learning improvements.

### ****Table 2: Other Important Metrics****

Coverage (0.4) reveals the model recommends only 40% of possible appliance combinations, risking repetitive suggestions. Novelty (1.0) paradoxically indicates recommendations are all "unexpected"—likely an artifact of poor appliance-frequency calibration. Diversity (0.0) shows no variety in recommended actions (e.g., always suggesting "turn off TV"), while the high Explainability Score (80.33) suggests verbose but potentially generic justifications. Inference time (95ms) is viable for real-time use but could mask inefficiencies in batch processing. This table highlights a trade-off between novelty and practicality, where unusual recommendations lack actionable diversity.

### ****Table 3: Comparative Analysis Metrics****

Reiterating Precision@5’s zero-value underscores the model’s universal struggle with top-k relevance. The inference latency (95ms) is competitive for edge deployment but may not scale for multi-user scenarios. Memory usage (3.2GB) is high for embedded devices but reasonable for cloud-based systems. Together, these metrics frame a hardware-accuracy trade-off: the model is deployable but not yet precise, suggesting a need for lightweight architectures or hybrid approaches.

### ****Table 4: Additional Comparative Metrics****

The Personalization Score (0.85) is optimistic but likely assumes uniform user behavior—real-world variability could degrade performance. Explainability Quality (1.0) suggests perfect clarity, but this may reflect superficial coherence rather than true utility (e.g., "save energy" without specifics). Alignment with Energy Goals (0.9) implies the model’s logic matches theoretical efficiency principles, though its practical impact (Table 1) remains weak. This table reveals a disconnect between subjective and objective metrics: the system "sounds" convincing but lacks empirical effectiveness.

### ****Table 5: Standard Performance Metrics****

MSE (0.189) and RMSE (0.435) indicate moderate error magnitudes, but the F1 score (0.0) exposes catastrophic failure in binary classification—likely due to imbalanced thresholds or inadequate feature representation. The MSE/RMSE values, while seemingly acceptable, are misleading without context: a 0.43 kWh error could be trivial for a dryer but critical for a lightbulb. This table’s apparent neutrality masks critical flaws, emphasizing the need for problem-specific error interpretation.