# Basic LLM GPT 2

# Psudocode

Algorithm: Energy Consumption Recommendation System

Input: Smart home energy usage data

Output: Energy-saving recommendations and high consumption predictions

1: // Data Loading and Preprocessing

2: function load\_and\_preprocess(url)

3: data ← pd.read\_csv(url)

4: cleaned\_data ← preprocess\_data(data)

5: return cleaned\_data

6: end function

7: function preprocess\_data(df)

8: // Feature engineering steps

9: df ← df.drop(['Transaction\_ID', 'Unix Timestamp'])

10: df['is\_peak\_hour'] ← apply peak hour logic

11: df['part\_of\_day'] ← categorize hours

12: df['is\_weekend'] ← apply weekend logic

13: df['Season'] ← map months to seasons

14: // Add trigonometric time features

15: df['hour\_sin'] ← sin(2π\*hour/24)

16: df['hour\_cos'] ← cos(2π\*hour/24)

17: // Energy features

18: df['total\_appliance\_usage'] ← sum of all appliances

19: df['is\_high\_consumption'] ← based on 75th percentile

20: // Normalize numerical features

21: features\_to\_scale ← ['Line Voltage', 'Voltage', ...]

22: df[features\_to\_scale] ← MinMaxScaler.fit\_transform()

23: // Add efficiency metrics

24: for each appliance do

25: df[appliance+'\_efficiency\_ratio'] ← usage/consumption

26: end for

27: return df

28: end function

29: // LSTM Data Preparation

30: function prepare\_lstm\_data(df)

31: // One-hot encode categorical features

32: df\_encoded ← pd.get\_dummies(df)

33: // Split data

34: X ← df\_encoded.drop('is\_high\_consumption')

35: y ← df\_encoded['is\_high\_consumption']

36: X\_train, X\_temp, y\_train, y\_temp ← train\_test\_split(X, y)

37: X\_val, X\_test, y\_val, y\_test ← train\_test\_split(X\_temp, y\_temp)

38: // Convert to float32

39: X\_train, X\_val, X\_test ← convert to float32

40: y\_train, y\_val, y\_test ← convert to float32

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

42: end function

43: // LSTM Model

44: function create\_lstm\_model(input\_shape)

45: model ← Sequential([

46: LSTM(64, input\_shape=input\_shape),

47: Dropout(0.3),

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

49: Dense(1)

50: ])

51: model.compile(optimizer='adam', loss='mse')

52: return model

53: end function

54: function create\_sequences(X, y, time\_steps=24)

55: Xs, ys ← empty lists

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

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

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

59: end for

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

61: end function

62: // LLM Setup

63: function setup\_llm(model\_name="gpt2")

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

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

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

67: return tokenizer, model

68: end function

69: // Prompt Generation

70: function generate\_prompt(row)

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

72: prompt ← prompt + time and appliance information from row

73: prompt ← prompt + appliance status information

74: prompt ← prompt + "Suggest optimal energy-saving actions..."

75: return prompt

76: end function

77: // LLM Training

78: function train\_llm(model, tokenizer, examples)

79: // Prepare dataset

80: train\_dataset ← Dataset.from\_list(examples)

81: tokenized\_data ← tokenize\_function(train\_dataset)

82:

83: // Training setup

84: training\_args ← TrainingArguments(...)

85: trainer ← Trainer(model, training\_args, tokenized\_data)

86:

87: try

88: trainer.train()

89: catch error

90: model ← model.to('cpu')

91: trainer ← Trainer(model, training\_args, tokenized\_data)

92: trainer.train()

93: end try

94:

95: model.save\_pretrained("./fine\_tuned\_gpt2")

96: tokenizer.save\_pretrained("./fine\_tuned\_gpt2")

97: end function

98: // Recommendation Generation

99: function generate\_recommendation(query\_idx)

100: row ← data[query\_idx]

101: prompt ← generate\_prompt(row)

102: inputs ← tokenizer(prompt, return\_tensors="pt")

103:

104: outputs ← model.generate(

105: inputs,

106: max\_new\_tokens=150,

107: temperature=0.8,

108: top\_p=0.95

109: )

110:

111: response ← tokenizer.decode(outputs[0])

112: if "Recommended actions:" in response

113: return response.split("Recommended actions:")[-1]

114: else

115: return response

116: end if

117: end function

118: // Main Execution

119: data ← load\_and\_preprocess(url)

120: X\_train, X\_val, X\_test, y\_train, y\_val, y\_test ← prepare\_lstm\_data(data)

121:

122: // LSTM Training

123: X\_train\_seq, y\_train\_seq ← create\_sequences(X\_train, y\_train)

124: lstm\_model ← create\_lstm\_model((24, X\_train\_seq.shape[2]))

125: lstm\_model.fit(train\_ds, validation\_data=val\_ds, epochs=20)

126:

127: // LLM Training

128: tokenizer, llm\_model ← setup\_llm()

129: train\_examples ← generate training examples from data

130: train\_llm(llm\_model, tokenizer, train\_examples)

131:

132: // Generate Test Recommendation

133: print(generate\_recommendation(100))

# Justification for each of 8 chosen chart

LSTM Training History

This chart was chosen to monitor model convergence and detect overfitting by comparing training vs validation metrics. The dual plots for loss and MAE provide complementary views of model learning progress.

Precision-Recall Curve

Selected because it effectively evaluates the LSTM's classification performance for our imbalanced energy consumption data. The area under the curve directly shows how well the model distinguishes high-consumption periods.

ROC Curve

Chosen as a standard diagnostic tool that complements the precision-recall curve by showing true vs false positive rates. The AUC metric helps quantify overall model discrimination ability.

LLM Training Loss

Essential for tracking the language model's learning progress during fine-tuning. The single loss curve clearly shows whether the training process is stable and converging.

Energy Distribution

Selected because the histogram and boxplot together reveal fundamental patterns in our target variable. The dual visualization shows both overall distribution and differences between high/low consumption groups.

Appliance Usage Patterns

Crucial for understanding how different devices contribute to energy consumption. The faceted histograms allow direct comparison of usage characteristics across all major appliances.

Time-Based Patterns

Chosen because energy usage is inherently temporal - these boxplots reveal consumption variations across hours, weekdays, seasons and peak periods. The multi-panel layout efficiently shows multiple temporal dimensions.

Training Time per Epoch

Included to monitor computational efficiency during model development. This helps identify potential bottlenecks and compare the resource requirements of different model configurations.

# 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. LSTM Training History (Loss & MAE)

The training history plot reveals the LSTM’s learning dynamics—loss decreasing smoothly indicates stable optimization, while erratic drops suggest noisy gradients. The gap between training and validation loss helps diagnose overfitting; a widening divergence implies memorization rather than generalization. MAE (Mean Absolute Error) complements loss by providing an interpretable scale (kWh), making it easier to assess real-world performance. Early convergence (plateauing curves) suggests the model may benefit from early stopping or learning rate adjustments. The simultaneous plotting of loss and MAE allows cross-validation—if loss improves but MAE stagnates, the model may be optimizing the wrong objective. This visualization is critical for hyperparameter tuning, as it reveals whether the model needs more capacity, regularization, or data diversity.

2. Precision-Recall Curve

This curve evaluates the LSTM’s ability to balance detecting true high-consumption events (recall) while minimizing false alarms (precision). The steep initial slope suggests the model confidently identifies the most obvious high-consumption cases. A plateau at higher recall values implies diminishing returns—capturing rare events requires sacrificing precision. The average precision (AP) score quantifies performance under all thresholds; an AP near 1 indicates near-perfect classification. Compared to accuracy, this metric is more informative for imbalanced data (where "high consumption" is rarer). The curve’s shape also hints at data quality—a jagged line may indicate inconsistent labeling or noisy measurements. For energy management, high recall is often prioritized to avoid missing critical consumption spikes.

3. ROC Curve (AUC Analysis)

The ROC curve evaluates how well the LSTM separates high/low consumption classes across all decision thresholds. A high AUC (e.g., >0.9) suggests the model distinguishes patterns effectively, while AUC ≈ 0.5 implies random guessing. The curve’s leftward bulge indicates strong true positive rates at low false positive costs—useful for systems where false alarms are expensive. The diagonal reference line (AUC = 0.5) provides a baseline; significant deviation confirms the model’s predictive power. Unlike precision-recall, ROC is less sensitive to class imbalance, making it a robust secondary metric. The "knee" of the curve (where slope flattens) suggests an optimal threshold for operational deployment.

4. LLM Training Loss (Fine-Tuning Progress)

The downward trend in loss indicates successful adaptation of the LLM to energy recommendation tasks. Rapid early improvements suggest the pretrained model already captures relevant linguistic patterns. Fluctuations may reflect the small training set (5 examples), causing instability in gradient updates. If loss plateaus too early, the model may need more diverse prompts or a higher learning rate. Unlike LSTM metrics, LLM loss lacks a validation curve—a limitation given the risk of overfitting on few examples. However, since the LLM generates text rather than predictions, qualitative evaluation (e.g., recommendation coherence) is equally important.

5. Energy Consumption Distribution (Histogram & Boxplot)

The histogram’s right skew reveals most readings cluster at lower consumption, with a long tail of energy-intensive outliers—typical for residential data. The boxplot’s interquartile range (IQR) shows where 50% of observations lie, while whiskers highlight extreme spikes (e.g., HVAC or dryer use). Bimodality in the histogram could suggest distinct operating modes (e.g., day/night). The boxplot’s median line, if close to the lower quartile, confirms most readings are below average—expected for energy data with frequent low-use periods. These visuals guide threshold-setting for "high consumption" alerts and reveal whether normalization/scaling is needed.

6. Appliance Usage Patterns (Faceted Histograms)

The binary peaks (e.g., TV’s on/off states) contrast with continuous distributions (e.g., refrigerator’s cyclical compressor spikes). Dryer/microwave plots show "bursty" usage—rare but high-intensity—while oven usage is less frequent but sustained. Gaps in intermediate values (e.g., near 0.5) suggest appliances operate near full capacity or not at all. The refrigerator’s smoother distribution reflects its always-on nature with variable intensity. These patterns inform demand-response strategies: bursty appliances are good candidates for load-shifting, while continuous ones need efficiency optimizations.

7. Time-Based Energy Patterns (Boxplots)

The hourly boxplot likely shows dual peaks (morning/evening) aligning with human activity—this intuition is confirmed if medians rise at 7–9 AM and 6–10 PM. Weekly patterns may show lower weekend consumption if households are away, or higher if more appliances are used. Seasonal variations (summer > winter or vice versa) reflect heating/cooling demands. The peak vs. off-peak comparison tests whether time-of-use pricing could effectively shift demand. Tight boxplot IQRs at night suggest predictable baseline usage, while wider daytime spreads imply behavioral variability. These insights guide when to deploy automated conservation measures.

8. Training Time per Epoch (Line Plot)

Linear time/epoch suggests stable computational load, while spikes may indicate memory bottlenecks (e.g., long sequences). Comparing LSTM vs. LLM times highlights hardware constraints—LLMs typically take longer per epoch due to larger parameters. Sudden slowdowns could trigger checks for GPU throttling or data pipeline issues. If time decreases mid-training, it may reflect adaptive batch processing or gradient accumulation. This metric is critical for cost-benefit analysis: does doubling training time justify marginal gains in accuracy? For deployment, faster epochs enable quicker iterations.

# Same goes for tables

## Discussion of Tables

### Table 1: Assessing Strength of Recommendations

This table presents six key metrics evaluating recommendation quality across systems. The hybrid LSTM-GPT-2 model demonstrates superior performance with the highest scores in precision (0.88), recall (0.85), NDCG (0.94), and hit rate (0.92), indicating it provides the most relevant recommendations. The LSTM alone performs exceptionally well in ranking metrics (NDCG 0.92) but scores zero in BLEU since it doesn't generate text. GPT-2 shows moderate performance (precision 0.72) but achieves the second-best BLEU score (0.65), reflecting its language generation capability. The simple rules baseline outperforms random and popular baselines across all metrics, suggesting even basic heuristics can be somewhat effective. Notably, CTR correlates strongly with other quality metrics, with the hybrid system achieving the highest simulated click-through rate (0.50). The substantial gap between learned models and baselines validates the value of machine learning approaches.

### Table 2: Other Important Metrics

Coverage metrics reveal the LSTM (0.95) and hybrid system (0.97) recommend from nearly the full solution space, while random baseline covers only half. GPT-2 shows higher novelty (0.80) than LSTM (0.75), suggesting it proposes less conventional recommendations. The hybrid system maintains strong diversity (0.90), crucial for avoiding repetitive suggestions. Interestingly, GPT-2 leads in explainability (0.85), benefiting from natural language outputs versus LSTM's numerical scores. The popular baseline scores lowest in novelty (0.20), expected since it recommends common solutions. Simple rules achieve reasonable novelty (0.65) by incorporating domain knowledge, outperforming random approaches. These metrics highlight complementary strengths - LSTM for broad coverage, GPT-2 for creative explanations.

### Table 3: Comparative Analysis Metrics

Latency varies dramatically, from 50ms (random) to 350ms (GPT-2), with the hybrid system (250ms) striking a balance. Memory usage shows GPT-2 requiring 4× more RAM (2048MB) than LSTM (512MB), reflecting LLM complexity. User ratings favor the hybrid system (4.5/5), suggesting users value combined strengths over individual models. Surprisingly, simple rules (3.7) outperform GPT-2 (3.8) in user ratings despite lower technical metrics. The random baseline scores lowest (2.5), confirming user ability to discern quality. Resource-intensive GPT-2 underperforms in ratings relative to its computational cost, indicating diminishing returns. This table highlights critical tradeoffs between performance, resources, and user satisfaction.

### Table 4: Other Comparative Metrics

Hallucination appears only in LLM-based systems, with GPT-2 at 0.15 rate reduced to 0.08 in the hybrid version. GPT-2 leads personalization (0.85), benefiting from language understanding capabilities. All systems show high robustness (>0.75), with simple rules (0.92) nearly matching random (0.95). The LSTM's zero hallucination reflects its deterministic nature. Personalization scores correlate with model complexity, from 0.3 (random) to 0.85 (GPT-2). Robustness inversely relates to model sophistication - simpler systems handle edge cases better. These metrics reveal fundamental tensions between accuracy, safety, and adaptability in recommendation systems.

### Table 5: Traditional Metrics

The LSTM shows surprisingly high MSE (0.1941) but extremely low F1 (0.0714), suggesting it poorly predicts binary high-consumption events despite decent regression. GPT-2 achieves strong F1 (0.72) without direct optimization, likely from meaningful text patterns. The hybrid model delivers best F1 (0.78), combining LSTM's detection with GPT-2's interpretation. Simple rules (F1 0.65) outperform random (0.45) and popular (0.60) baselines through domain knowledge. RMSE values follow similar patterns to MSE, with GPT-2 (0.5) underperforming hybrid (0.4472). These results challenge assumptions that lower MSE necessarily translates to better practical performance, highlighting the importance of metric selection.

## Discussion of Charts

### Table 1 Chart (Multi-bar)

The clustered bars visually emphasize the hybrid system's dominance across all metrics. Precision, recall, NDCG and hit rate bars form a consistent pattern, showing these metrics correlate strongly. BLEU scores stand out as only applying to text-generating systems, with GPT-2 and hybrid showing activity. CTR bars mirror quality metrics, validating it as a meaningful proxy. The chart effectively shows how baselines cluster at the bottom, with simple rules consistently above random/popular. Color differentiation helps track each metric across systems, though the 6-metric comparison becomes visually dense.

### Table 2 Chart (Grouped Bars)

Coverage bars immediately show the LSTM and hybrid systems near-complete coverage. Novelty bars reveal GPT-2 surpassing LSTM, while diversity shows the opposite relationship. Explainability bars form a different pattern, with GPT-2 leading, demonstrating these metrics capture orthogonal qualities. The popular baseline's extremely low novelty bar visually confirms its conventional nature. Grouping allows easy comparison of systems across different evaluation dimensions. The consistent scale (0-1) enables direct visual comparison of absolute values across metrics.

### Table 3 Chart (Dual-axis)

The left-axis bars for latency/memory create striking visual contrast - GPT-2's tall red memory bar dwarfs others. User rating line on the right axis shows an inverse relationship to resource usage - hybrid achieves highest ratings without maximum resource cost. Random baseline's tiny latency/memory bars juxtaposed with its low rating line effectively show cheap ≠ good. The dual y-axes successfully communicate two distinct dimensions without distortion. Marker points on the rating line clearly show exact values. This format excellently captures the cost/quality tradeoff.

### Table 4 Chart (Grouped Bars)

Hallucination bars create immediate visual warning for GPT-2's red segment. Personalization bars grow with model complexity, forming a clear progression. Robustness bars remain high throughout but dip slightly for advanced systems. The zero hallucination bars for non-LLM systems create strong visual baseline. Color consistency allows tracking each metric across systems. The chart effectively shows that no system excels at all three metrics simultaneously, highlighting design tradeoffs.

### Table 5 Chart (Grouped Bars)

MSE and RMSE bars follow nearly identical patterns, confirming their mathematical relationship. F1 scores show completely different pattern, revealing the limitation of regression metrics. The LSTM's tall MSE bars versus tiny F1 bar creates striking visual paradox. Hybrid system's balanced performance shows in medium MSE/RMSE with tall F1 bar. Simple rules' relatively strong F1 performance stands out versus its baselines peers. This chart powerfully demonstrates how metric choice dramatically affects system assessment.