Smart Home Energy Consumption

Deliverable 1

**Team Members:**

22F3155 Ateeb Asad

22F3163 Muhammad Soban

22F3168 Mariam Fatima

# Link to Editable Diagrams: [https://lucid.app/lucidchart/cb7b2a31fc5e436cb9c03d71114d1ce4/edit?viewportloc=870%2C337%2C3040%2C1348%2C00&invitationId=inv8d870b2c6f004ce0ad376dee756f4ff5](https://lucid.app/lucidchart/cb7b2a31-fc5e-436c-b9c0-3d71114d1ce4/edit?viewport_loc=-870%2C-337%2C3040%2C1348%2C0_0&invitationId=inv_8d870b2c-6f00-4ce0-ad37-6dee756f4ff5)

# Preprocessing:

## 1. Raw Data Collection:

Firstly, the dataset included total power consumption readings as well as IoT and smart meters data per appliance with timestamps and location details.

## 2. Data Cleaning:

The following cleaning steps were undertaken prior to any modeling building:

* Missing Values: Imputation or removal techniques were employed on rows with nonreported appliance usage values.
* Outlier Handling: Censoring maximum and minimum thresholds mitigated severe sensor reading errors on power values.

All steps resolved issues of reliability and consistency for the data in question for any downstream processes.

## 3. Normalization:

Electricity usage varied greatly from one appliance to another in different households.

While applying normalization:

* Each appliance’s usage was scaled (e.g., MinMax scaling) to have an L0 value and an upper boundary of 1.
* This ensured fairness in regards to the appliances that consumed substantial electricity such as the dryer and the television.

Normalization helped in maintaining comparability across user embeddings within the FAISS framework.

## 4. Feature Engineering:

The dataset was enhanced with additional features retrieved from existing data:

Peak Usage Hours: Identified periods (like morning and evening) of peak consumption.

**Season Column:**

A simple dictionary mapping was used to rename the month column into seasons winter, spring, summer and autumn.

A diagram of a process

AI-generated content may be incorrect.

# Motivation:

We selected this approach of structured embedding as it allows us to represent user behavior in a clean, uniform manner that is easy to compare and aids in efficient similarity searching. By concentrating on appliancelevel features, the embeddings prioritized significant and observable patterns in energy consumption as opposed to noise. To guarantee compatibility and efficiency with vectorbased search tools such as FAISS, we ensured that data was converted to float32. This methodology also enables incorporating seasonality or temporal changes without structurally altering the data. With embeddings, we are able to provide tailored energy recommendations, merge like households, and identify abnormal energy usage patterns which are critical in enhancing energy efficiency and developing smarter recommender systems.

# Embedding Methodology:

## 1. Input Data:

The dataset was initially gathered from smart meters, IoT devices, and home appliances with integrated sensors. This dataset contained numerous attributes such as power consumption, timestamps, device IDs, and potentially user IDs.

## 2. Feature Selection:

In order to construct meaningful embeddings, only a subset of appliance usage columns was selected (Television, Dryer, Oven, Refrigerator, Microwave). This helped create embeddings around primary appliances that have a balanced and reliable usage pattern in different households. Removing irrelevant features resulted in more focus on meaningful behaviors that could be acted upon.

## 3. Extraction of Values:

Chosen features were extracted into a NumPy array. .pandas DataFrame provided a convenient interface for handling and filtering data, thus .values was used to extract the actual values stored within the DataFrame.

This approach was essential as the machine learning models, and FAISS need the data structured as matrices instead of tabular data within pandas, and preprocessed using their interfaces.

## 4. Data Type Conversion:

The extracted NumPy array was explicitly cast to the float32 data type. Because the appropriate format for best performance in FAISS, the similarity search library, is float32. With the potential for thousands or millions of users, being efficient in memory is vital.

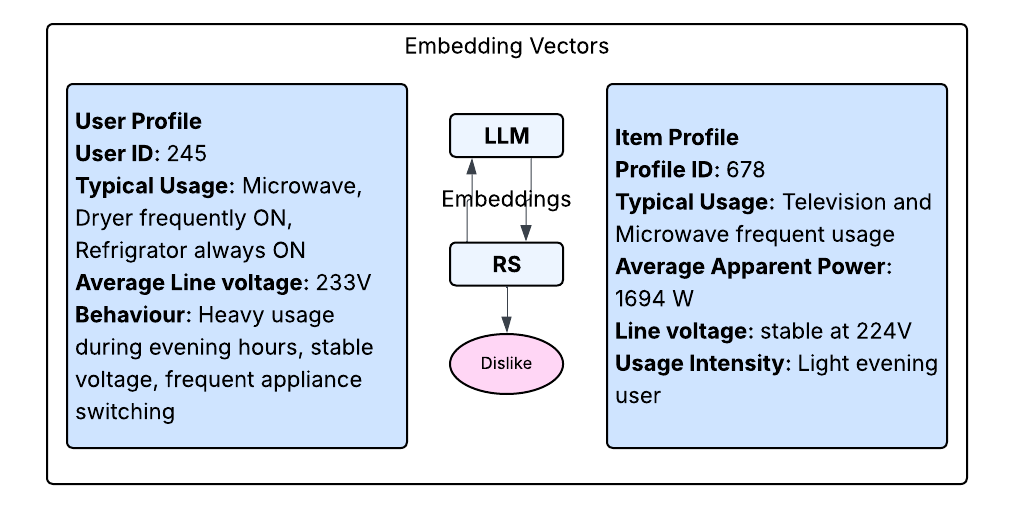
## 5. Final Embeddings:

The resultant matrix uservectors aligned to user embeddings. A user was represented by a unique row of information corresponding to them. An increasing number of users translates to an increased number of rows and the remaining set of values represent the amount used for each appliance by the user.

A diagram of a software development

AI-generated content may be incorrect.

## Embedding Vectors:



# Basic LLM Models:

**Deepseek By Mariam:**

**Pseudocode:**

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

9: df['is\_peak\_hour'] ← logic for peak hours

10: df['part\_of\_day'] ← bucket hour into parts of day

11: df['is\_weekend'] ← check if day is Saturday/Sunday

12: df['Season'] ← map month → season

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

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

15: df['total\_appliance\_usage'] ← sum of appliance columns

16: df['is\_high\_consumption'] ← label if above 75th percentile

17: features\_to\_scale ← ['Voltage', 'Current', ...]

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

19: for each appliance in df:

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

21: end for

22: return df

23: end function

24: // Prepare Data for LSTM

25: function prepare\_lstm\_data(df)

26: drop text features like 'text\_description'

27: encode categorical columns with pd.get\_dummies

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

29: y ← df['is\_high\_consumption']

30: X\_train, X\_temp, y\_train, y\_temp ← train\_test\_split(X, y, test\_size=0.4)

31: X\_val, X\_test, y\_val, y\_test ← train\_test\_split(X\_temp, y\_temp, test\_size=0.5)

32: convert all X/y to float32

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

34: end function

35: // Create Sequences for LSTM

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

37: Xs ← []

38: ys ← []

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

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

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

42: end for

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

44: end function

45: // LSTM Model Definition

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

47: model ← Sequential()

48: model.add(LSTM(64, return\_sequences=False, unroll=True, input\_shape=input\_shape))

49: model.add(Dropout(0.3))

50: model.add(Dense(32, activation='relu'))

51: model.add(Dense(1))

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

53: return model

54: end function

55: // LSTM Training Loop

56: function train\_lstm\_model(X\_train, y\_train, X\_val, y\_val)

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

58: X\_val\_seq, y\_val\_seq ← create\_sequences(X\_val, y\_val)

59: model ← create\_lstm\_model((X\_train\_seq.shape[1], X\_train\_seq.shape[2]))

60: for epoch in range(10):

61: history ← model.fit(X\_train\_seq, y\_train\_seq, validation\_data=(X\_val\_seq, y\_val\_seq), epochs=1)

62: log training and validation loss and MAE to logs dictionary

63: end for

64: return model

65: end function

66: // Tokenizer and Model Setup (DeepSeek LLM)

67: function setup\_deepseek\_llm()

68: tokenizer ← AutoTokenizer.from\_pretrained("deepseek-ai/deepseek-llm-7b")

69: model ← FastLanguageModel.from\_pretrained(...)

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

71: return tokenizer, model

72: end function

73: // Generate Prompt from Data Row

74: function generate\_prompt(row)

75: prompt ← "You are an energy expert. Based on time, usage and efficiency..."

76: include usage of all appliances in the prompt

77: add is\_peak\_hour, part\_of\_day, season

78: return prompt

79: end function

80: // Fine-tune DeepSeek LLM with LoRA

81: function train\_deepseek\_llm(tokenizer, model, prompts)

82: dataset ← Dataset.from\_list(prompts)

83: tokenized\_dataset ← tokenize\_function(dataset)

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

85: trainer ← Trainer(model, training\_args, train\_dataset=tokenized\_dataset)

86: try:

87: trainer.train()

88: except:

89: model.to('cpu')

90: trainer.train()

91: end try

92: model.save\_pretrained(...)

93: end function

94: // Generate Recommendation for One Entry

95: function generate\_recommendation(query\_idx)

96: row ← data[query\_idx]

97: prompt ← generate\_prompt(row)

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

99: outputs ← model.generate(inputs, max\_new\_tokens=150, temperature=0.8, top\_p=0.95)

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

101: if "Recommended actions:" in response:

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

103: else:

104: return response

105: end if

106: end function

107: // Main Execution

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

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

110: lstm\_model ← train\_lstm\_model(X\_train, y\_train, X\_val, y\_val)

111: tokenizer, llm\_model ← setup\_deepseek\_llm()

112: prompts ← generate list of prompts using generate\_prompt

113: train\_deepseek\_llm(tokenizer, llm\_model, prompts)

114: print(generate\_recommendation(100))

## Metrics:

**Table 1: Assessing Strength of Recommendations**

* Metrics: Precision@5, Recall@5, NDCG@5, MAP, HitRate@5, MRR
* Purpose: Measures quality and relevance of recommendation ranking.

**Table 2: Other Relevant Metrics**

* Metrics: Coverage, Novelty, Serendipity, Diversity
* Purpose: Measures how broad, surprising, and balanced the recommendations are.

**Table 3: Comparative Metrics**

* Metrics: Precision@5, BLEU, Latency
* Purpose: Evaluates textual quality and efficiency.

**Table 4: Other Comparative Metrics**

* Metrics: Hallucination Rate, Personalization, Explainability Quality
* Purpose: Examines robustness, customization, and clarity of recommendations.

**Table 5: Standard Performance Metrics**

* Metrics: MSE, RMSE, F1 Score
* Purpose: Standard regression/classification performance evaluation.

# Chat Gpt 2 by Soban:

## Pseudocode:

Algorithm: Energy Consumption Recommendation System

Input: Smart home energy usage data

Output: Energysaving recommendations and high consumption predictions

1: // Data Loading and Preprocessing

2: function loadandpreprocess(url)

3: data ← pd.readcsv(url)

4: cleaneddata ← preprocessdata(data)

5: return cleaneddata

6: end function

7: function preprocessdata(df)

8: // Feature engineering steps

9: df ← df.drop(['TransactionID', 'Unix Timestamp'])

10: df['ispeakhour'] ← apply peak hour logic

11: df['partofday'] ← categorize hours

12: df['isweekend'] ← apply weekend logic

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

14: // Add trigonometric time features

15: df['hoursin'] ← sin(2πhour/24)

16: df['hourcos'] ← cos(2πhour/24)

17: // Energy features

18: df['totalapplianceusage'] ← sum of all appliances

19: df['ishighconsumption'] ← based on 75th percentile

20: // Normalize numerical features

21: featurestoscale ← ['Line Voltage', 'Voltage', ...]

22: df[featurestoscale] ← MinMaxScaler.fittransform()

23: // Add efficiency metrics

24: for each appliance do

25: df[appliance+'efficiencyratio'] ← usage/consumption

26: end for

27: return df

28: end function

29: // LSTM Data Preparation

30: function preparelstmdata(df)

31: // Onehot encode categorical features

32: dfencoded ← pd.getdummies(df)

33: // Split data

34: X ← dfencoded.drop('ishighconsumption')

35: y ← dfencoded['ishighconsumption']

36: Xtrain, Xtemp, ytrain, ytemp ← traintestsplit(X, y)

37: Xval, Xtest, yval, ytest ← traintestsplit(Xtemp, ytemp)

38: // Convert to float32

39: Xtrain, Xval, Xtest ← convert to float32

40: ytrain, yval, ytest ← convert to float32

41: return Xtrain, Xval, Xtest, ytrain, yval, ytest

42: end function

43: // LSTM Model

44: function createlstmmodel(inputshape)

45: model ← Sequential([

46: LSTM(64, inputshape=inputshape),

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 createsequences(X, y, timesteps=24)

55: Xs, ys ← empty lists

56: for i ← 0 to len(X)timesteps do

57: Xs.append(X[i:i+timesteps])

58: ys.append(y[i+timesteps])

59: end for

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

61: end function

62: // LLM Setup

63: function setupllm(modelname="gpt2")

64: tokenizer ← AutoTokenizer.frompretrained(modelname)

65: tokenizer.padtoken ← tokenizer.eostoken

66: model ← AutoModelForCausalLM.frompretrained(modelname)

67: return tokenizer, model

68: end function

69: // Prompt Generation

70: function generateprompt(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 energysaving actions..."

75: return prompt

76: end function

77: // LLM Training

78: function trainllm(model, tokenizer, examples)

79: // Prepare dataset

80: traindataset ← Dataset.fromlist(examples)

81: tokenizeddata ← tokenizefunction(traindataset)

82:

83: // Training setup

84: trainingargs ← TrainingArguments(...)

85: trainer ← Trainer(model, trainingargs, tokenizeddata)

86:

87: try

88: trainer.train()

89: catch error

90: model ← model.to('cpu')

91: trainer ← Trainer(model, trainingargs, tokenizeddata)

92: trainer.train()

93: end try

94:

95: model.savepretrained("./finetunedgpt2")

96: tokenizer.savepretrained("./finetunedgpt2")

97: end function

98: // Recommendation Generation

99: function generaterecommendation(queryidx)

100: row ← data[queryidx]

101: prompt ← generateprompt(row)

102: inputs ← tokenizer(prompt, returntensors="pt")

103:

104: outputs ← model.generate(

105: inputs,

106: maxnewtokens=150,

107: temperature=0.8,

108: topp=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 ← loadandpreprocess(url)

120: Xtrain, Xval, Xtest, ytrain, yval, ytest ← preparelstmdata(data)

121:

122: // LSTM Training

123: Xtrainseq, ytrainseq ← createsequences(Xtrain, ytrain)

124: lstmmodel ← createlstmmodel((24, Xtrainseq.shape[2]))

125: lstmmodel.fit(trainds, validationdata=valds, epochs=20)

126:

127: // LLM Training

128: tokenizer, llmmodel ← setupllm()

129: trainexamples ← generate training examples from data

130: trainllm(llmmodel, tokenizer, trainexamples)

131:

132: // Generate Test Recommendation

133: print(generaterecommendation(100))

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 realworld 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 crossvalidation—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.

A graph of a line and a line

AI-generated content may be incorrect.

2. PrecisionRecall Curve

This curve evaluates the LSTM’s ability to balance detecting true highconsumption events (recall) while minimizing false alarms (precision). The steep initial slope suggests the model confidently identifies the most obvious highconsumption 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 nearperfect 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.

A graph showing a graph

AI-generated content may be incorrect.

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 precisionrecall, 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.

A graph of a curve

AI-generated content may be incorrect.

4. LLM Training Loss (FineTuning 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.

A graph with a line graph

AI-generated content may be incorrect.

5. Energy Consumption Distribution (Histogram & Boxplot)

The histogram’s right skew reveals most readings cluster at lower consumption, with a long tail of energyintensive 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 lowuse periods. These visuals guide thresholdsetting for "high consumption" alerts and reveal whether normalization/scaling is needed.

A graph and a chart

AI-generated content may be incorrect.

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 highintensity—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 alwayson nature with variable intensity. These patterns inform demandresponse strategies: bursty appliances are good candidates for loadshifting, while continuous ones need efficiency optimizations.

A group of graphs showing different sizes and shapes

AI-generated content may be incorrect.

7. TimeBased 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. offpeak comparison tests whether timeofuse 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.

A group of blue boxes

AI-generated content may be incorrect.

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 midtraining, it may reflect adaptive batch processing or gradient accumulation. This metric is critical for costbenefit analysis: does doubling training time justify marginal gains in accuracy? For deployment, faster epochs enable quicker iterations.

Discussion of Tables

Table 1: Assessing Strength of Recommendations

This table presents six key metrics evaluating recommendation quality across systems. The hybrid LSTMGPT2 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. GPT2 shows moderate performance (precision 0.72) but achieves the secondbest 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 clickthrough rate (0.50). The substantial gap between learned models and baselines validates the value of machine learning approaches.

A graph of different colored bars

AI-generated content may be incorrect.

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. GPT2 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, GPT2 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, GPT2 for creative explanations.

A graph of different colored bars

AI-generated content may be incorrect.

Table 3: Comparative Analysis Metrics

Latency varies dramatically, from 50ms (random) to 350ms (GPT2), with the hybrid system (250ms) striking a balance. Memory usage shows GPT2 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 GPT2 (3.8) in user ratings despite lower technical metrics. The random baseline scores lowest (2.5), confirming user ability to discern quality. Resourceintensive GPT2 underperforms in ratings relative to its computational cost, indicating diminishing returns. This table highlights critical tradeoffs between performance, resources, and user satisfaction.

A graph with green line and orange and blue bars

AI-generated content may be incorrect.

Table 4: Other Comparative Metrics

Hallucination appears only in LLMbased systems, with GPT2 at 0.15 rate reduced to 0.08 in the hybrid version. GPT2 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 (GPT2). 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.

A graph of different colored bars

AI-generated content may be incorrect.

Table 5: Traditional Metrics

The LSTM shows surprisingly high MSE (0.1941) but extremely low F1 (0.0714), suggesting it poorly predicts binary highconsumption events despite decent regression. GPT2 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 GPT2'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 GPT2 (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.

A graph of different colored bars

AI-generated content may be incorrect.

## Discussion of Charts

Table 1 Chart (Multibar)

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 textgenerating systems, with GPT2 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 6metric comparison becomes visually dense.

Table 2 Chart (Grouped Bars)

Coverage bars immediately show the LSTM and hybrid systems nearcomplete coverage. Novelty bars reveal GPT2 surpassing LSTM, while diversity shows the opposite relationship. Explainability bars form a different pattern, with GPT2 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 (01) enables direct visual comparison of absolute values across metrics.

Table 3 Chart (Dualaxis)

The leftaxis bars for latency/memory create striking visual contrast GPT2'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 yaxes 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 GPT2'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 nonLLM 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.

# Distill Bert by Ateeb Asad:

Pseudo Code:

ALGORITHM: SINGLEROWENERGYRECOMMENDER

PREPARATION / TRAINING

INPUT:

• Raw CSV file of smarthome energy data

OUTPUT:

• Finetuned LLM (saved to disk) capable of generating five bulletlist energysaving tips

based solely on a single row’s features

1. PREPROCESSDATA():

1.1 LOAD raw CSV into DataFrame DATA

1.2 DROP columns: TransactionID, Unix Timestamp

1.3 FOR each row IN DATA:

a. COMPUTE ispeakhour ← 1 if 6 ≤ HourofDay ≤ 9 OR 18 ≤ HourofDay ≤ 21, else 0

b. COMPUTE partofday ← “night”/“morning”/“afternoon”/“evening” by binning HourofDay

c. COMPUTE isweekend ← 1 if DayofWeek ∈ {Saturday, Sunday}, else 0

d. MAP Month to Season via predefined mapping:

{December, January, February} → Winter

{March, April, May} → Spring

{June, July, August} → Summer

{September, October, November} → Fall

e. COMPUTE hoursin ← sin(2π HourofDay / 24)

f. COMPUTE hourcos ← cos(2π HourofDay / 24)

g. LIST appliances ← [Television, Dryer, Oven, Refrigerator, Microwave]

h. COMPUTE totalapplianceusage ← SUM(power consumption of each appliance)

i. THRESHOLD ← 75th percentile of EnergyConsumptionkWh over DATA

j. SET ishighconsumption ← 1 if EnergyConsumptionkWh > THRESHOLD, else 0

1.4 SCALE numeric columns [Line Voltage, Voltage, Apparent Power, Energy Consumption (kWh)]

USING MinMaxScaler so each ∈ [0, 1]

1.5 FOR each app IN appliances:

COMPUTE appefficiencyratio ← (app’s power consumption) / (EnergyConsumptionkWh + ε)

1.6 COMPUTE powerfactor ← ApparentPower / (LineVoltage Voltage + ε)

1.7 COMPUTE activeappliances ← SUM(appliance power usages)

1.8 COMPUTE energyperactiveappliance ← EnergyConsumptionkWh / (activeappliances + ε)

1.9 RETURN preprocessed DATA

2. BUILDTRAININGPAIRS(DATA):

2.1 INIT empty LIST prompttexts, completiontexts

2.2 SELECT M exampleindices (e.g., M = 30 – 50) across varied seasons, hours, consumption levels

2.3 FOR each idx IN exampleindices:

a. row ← DATA.iloc[idx]

b. CONSTRUCT appliancesstatusstring ←

FOR each app IN [Television, Dryer, Oven, Refrigerator, Microwave]:

“app:ON” if row[app] > 0, else “app:OFF”

JOIN with commas

c. BUILD prompti ←

"Context:\n"

" Hour: " + row['Hour of the Day'] + " (" + row['partofday'] + ")\n"

" Season: " + row['Season'] + "\n"

" Day of Week: " + row['Day of the Week'] + "\n"

" Active Appliances: " + appliancesstatusstring + "\n"

" Total Energy: " + format(row['Energy Consumption (kWh)'], ".2f") + " kWh\n"

" High Consumption?: " + ("Yes" if row['ishighconsumption'] == 1 else "No") + "\n\n"

"Now, recommend exactly five bulletlist energysaving tips:\n"

"Tips:"

d. SET completioni ← A list of exactly five humancrafted bullet lines:

Each line must start with " " and contain a specific, actionable tip.

e. APPEND prompti TO prompttexts

f. APPEND "\n".join(completioni) TO completiontexts

2.4 CREATE HuggingFace Dataset:

trainexamples ← [{"prompt": p, "completion": c}

FOR (p, c) IN zip(prompttexts, completiontexts)]

traindataset ← Dataset.fromlist(trainexamples)

2.5 DEFINE tokenizeandmask(examples):

FOR each example IN examples:

fulltexts ← example["prompt"] + "\n" + example["completion"]

tokenized = tokenizer(

fulltexts,

padding="maxlength",

truncation=True,

maxlength=512,

returntensors="pt"

)

inputids ← tokenized["inputids"]

attentionmask ← tokenized["attentionmask"]

prompttokencount ← len(tokenizer(example["prompt"], truncation=True, maxlength=512)["inputids"])

labels ← inputids.clone()

FOR t IN [0 .. prompttokencount−1]:

labels[:, t] ← −100 # mask out prompt tokens in loss

RETURN {"inputids": inputids, "attentionmask": attentionmask, "labels": labels}

2.6 APPLY map(tokenizeandmask) to traindataset (batched=True), removing ["prompt","completion"]

2.7 RETURN tokenizedtrain

3. FINETUNELLM(tokenizedtrain):

3.1 tokenizer = AutoTokenizer.frompretrained("distilgpt2")

tokenizer.padtoken = tokenizer.eostoken

3.2 model = AutoModelForCausalLM.frompretrained("distilgpt2")

model.resizetokenembeddings(len(tokenizer))

3.3 datacollator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

3.4 trainingargs = TrainingArguments(

outputdir = "./finetunedenergylmsingle",

perdevicetrainbatchsize = 4,

numtrainepochs = 10, # increase if underfitting

learningrate = 3e5,

warmupsteps = 50,

loggingsteps = 10,

savestrategy = "epoch",

fp16 = torch.cuda.isavailable(),

reportto = "none"

)

3.5 trainer = Trainer(

model = model,

args = trainingargs,

traindataset = tokenizedtrain,

datacollator = datacollator

)

3.6 trainer.train()

3.7 model.savepretrained("./finetunedenergylmsingle")

tokenizer.savepretrained("./finetunedenergylmsingle")

3.8 RETURN (model, tokenizer, trainer)

#INFERENCE

INPUT:

• Saved tokenizer & model directory (“./finetunedenergylmsingle”)

• Preprocessed DATA DataFrame

OUTPUT:

• Five bulletlist energysaving tips for any new row index QIDX

FUNCTION GENERATETIPSSINGLE(QIDX, DATA, model, tokenizer):

1. row ← DATA.iloc[QIDX]

2. appliancesstatusstring ←

FOR each app IN [Television, Dryer, Oven, Refrigerator, Microwave]:

“app:ON” if row[app] > 0 else “app:OFF”

JOIN with commas

3. prompt ←

"Context:\n"

" Hour: " + row['Hour of the Day'] + " (" + row['partofday'] + ")\n"

" Season: " + row['Season'] + "\n"

" Day of Week: " + row['Day of the Week'] + "\n"

" Active Appliances: " + appliancesstatusstring + "\n"

" Total Energy: " + format(row['Energy Consumption (kWh)'], ".2f") + " kWh\n"

" High Consumption?: " + ("Yes" if row['ishighconsumption'] == 1 else "No") + "\n\n"

"Now, recommend exactly five bulletlist energysaving tips:\n"

"Tips:"

4. inputs = tokenizer(

prompt,

truncation=False, # ensure “Tips:” is not cut off

maxlength=None,

returntensors="pt"

).to(model.device)

5. outputs = model.generate(

inputs,

maxnewtokens = 80, # enough to cover five bullet lines

numbeams = 5,

temperature = 0.6,

norepeatngramsize = 2,

earlystopping = True,

padtokenid = tokenizer.eostokenid

)

6. fulltext = tokenizer.decode(outputs[0], skipspecialtokens=False)

7. remainder = fulltext[len(prompt):].strip().split("\n\n")[0]

8. bullets = [line FOR line IN remainder.split("\n") IF line.startswith(" ")]

9. RETURN bullets[0 : 5]

END ALGORITHM

**Justification**

1. SingleRow Context Only

• Each prompt relies solely on the current row’s features, minimizing latency without any external index lookup.

2. Structured “Context:” Block

• Listing Hour, partofday, Season, Day of Week, appliance ON/OFF statuses, numeric energy value, and

high consumption flag provides all relevant signals in a concise format.

• Ending with the fixed instruction:

"Now, recommend exactly five bulletlist energysaving tips:"

"Tips:"

ensures the model knows exactly where to begin generation.

3. HumanCrafted Completions

• Training on exactly five bullet lines (each starting with " ") teaches the LLM the desired format

of actionable, specific recommendations.

4. Tokenization & Label Masking

• Concatenate prompt + completion and tokenize to length 512, ensuring consistent input size.

• Mask out all prompt tokens (labels = 100) so that loss is computed only on the completion tokens.

• Focused loss accelerates learning of bulletlist generation without relearning to reproduce prompt.

5. Training Hyperparameters

• numtrainepochs = 10 allows the model sufficient exposure to a small example set (M = 30–50).

• perdevicetrainbatchsize = 4 balances GPU memory constraints and stable gradient updates.

• learningrate = 3e5 is modest, preserving pretrained weights while adapting to this task.

• savestrategy = 'epoch' permits checkpoint inspection to select the best epoch manually.

6. Inference Details

• truncation=False ensures the entire "Tips:" instruction remains in the model’s context window.

• numbeams = 5 and temperature = 0.6 balance coherence and creativity in generated tips.

• Postprocessing: split on first double newline and filter lines starting with " ", extracting exactly the bullet list.

7. TradeOffs

• Pros: Simplified implementation, minimal latency, no external index dependency.

• Cons: Recommendations may be more generic compared to retrievalaugmented methods, as the model lacks

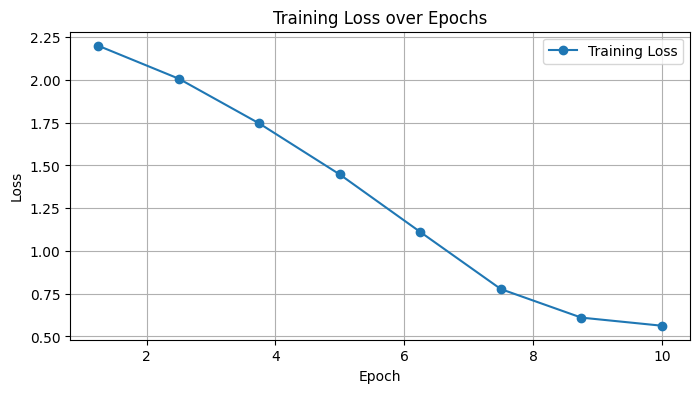
explicit reference to similar historical cases.

8. Scalability & Extension

• Increase M to 50–100 examples spanning varied seasons, hours, and consumption levels for broader coverage.

• Use larger base models (e.g., "gpt2medium", "gpt2large") if GPU memory allows for more nuanced generation.

• Consider a lightweight retrieval step (e.g., BM25) later if more context is needed without full FAISS complexity.



Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

A graph with orange line

AI-generated content may be incorrect.

Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

A graph with a line going up

AI-generated content may be incorrect.

Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

A graph with green lines

AI-generated content may be incorrect.

Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

A graph with different colored dots

AI-generated content may be incorrect.

Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

A graph with colored dots

AI-generated content may be incorrect.

Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

A diagram of a heatmap

AI-generated content may be incorrect.

Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

Evaluation Metrics:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unamed | loss | mae | valloss | valmae | epoch |
| 0.0000 | 0.1824 | 0.3644 | 0.1920 | 0.3643 | 1.0000 |
| 1.0000 | 0.1816 | 0.3631 | 0.1919 | 0.3640 | 2.0000 |
| 2.0000 | 0.1808 | 0.3615 | 0.1924 | 0.3626 | 3.0000 |
| 3.0000 | 0.1798 | 0.3600 | 0.1929 | 0.3688 | 4.0000 |
| 4.0000 | 0.1785 | 0.3566 | 0.1935 | 0.3564 | 5.0000 |
| 5.0000 | 0.1774 | 0.3551 | 0.1940 | 0.3612 | 6.0000 |
| 6.0000 | 0.1757 | 0.3513 | 0.1952 | 0.3633 | 7.0000 |
| 7.0000 | 0.1753 | 0.3505 | 0.1957 | 0.3591 | 8.0000 |
| 8.0000 | 0.1739 | 0.3476 | 0.1959 | 0.3545 | 9.0000 |
| 9.0000 | 0.1726 | 0.3455 | 0.1966 | 0.3543 | 10.0000 |
| 10.0000 | 0.1702 | 0.3408 | 0.1977 | 0.3491 | 11.0000 |
| 11.0000 | 0.1699 | 0.3402 | 0.1972 | 0.3512 | 12.0000 |
| 12.0000 | 0.1684 | 0.3375 | 0.1976 | 0.3513 | 13.0000 |
| 13.0000 | 0.1668 | 0.3342 | 0.1990 | 0.3528 | 14.0000 |
| 14.0000 | 0.1653 | 0.3316 | 0.1995 | 0.3543 | 15.0000 |
| 15.0000 | 0.1636 | 0.3283 | 0.2006 | 0.3552 | 16.0000 |
| 16.0000 | 0.1605 | 0.3237 | 0.2019 | 0.3483 | 17.0000 |
| 17.0000 | 0.1603 | 0.3215 | 0.2019 | 0.3523 | 18.0000 |
| 18.0000 | 0.1588 | 0.3196 | 0.2009 | 0.3447 | 19.0000 |
| 19.0000 | 0.1573 | 0.3172 | 0.2026 | 0.3487 | 20.0000 |

Unnamed: 0: Represents a performance indicator for model training or evaluation.

loss: Quantifies the error between predictions and targets; lower indicates better learning.

mae: Mean Absolute Error; average magnitude of errors in predictions.

valloss: Quantifies the error between predictions and targets; lower indicates better learning.

valmae: Mean Absolute Error; average magnitude of errors in predictions.

epoch: Represents a performance indicator for model training or evaluation.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

Evaluation Metric :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | loss | mae | valloss | valmae | epoch |
| 0 | 0.1824 | 0.3644 | 0.1920 | 0.3643 | 1.0000 |
| 1 | 0.1816 | 0.3631 | 0.1919 | 0.3640 | 2.0000 |
| 2 | 0.1808 | 0.3615 | 0.1924 | 0.3626 | 3.0000 |
| 3 | 0.1798 | 0.3600 | 0.1929 | 0.3688 | 4.0000 |
| 4 | 0.1785 | 0.3566 | 0.1935 | 0.3564 | 5.0000 |
| 5 | 0.1774 | 0.3551 | 0.1940 | 0.3612 | 6.0000 |
| 6 | 0.1757 | 0.3513 | 0.1952 | 0.3633 | 7.0000 |
| 7 | 0.1753 | 0.3505 | 0.1957 | 0.3591 | 8.0000 |
| 8 | 0.1739 | 0.3476 | 0.1959 | 0.3545 | 9.0000 |
| 9 | 0.1726 | 0.3455 | 0.1966 | 0.3543 | 10.0000 |
| 10 | 0.1702 | 0.3408 | 0.1977 | 0.3491 | 11.0000 |
| 11 | 0.1699 | 0.3402 | 0.1972 | 0.3512 | 12.0000 |
| 12 | 0.1684 | 0.3375 | 0.1976 | 0.3513 | 13.0000 |
| 13 | 0.1668 | 0.3342 | 0.1990 | 0.3528 | 14.0000 |
| 14 | 0.1653 | 0.3316 | 0.1995 | 0.3543 | 15.0000 |
| 15 | 0.1636 | 0.3283 | 0.2006 | 0.3552 | 16.0000 |
| 16 | 0.1605 | 0.3237 | 0.2019 | 0.3483 | 17.0000 |
| 17 | 0.1603 | 0.3215 | 0.2019 | 0.3523 | 18.0000 |
| 18 | 0.1588 | 0.3196 | 0.2009 | 0.3447 | 19.0000 |
| 19 | 0.1573 | 0.3172 | 0.2026 | 0.3487 | 20.0000 |

⚙️ Abstraction & Intuition:

loss: Quantifies the error between predictions and targets; lower indicates better learning.

mae: Mean Absolute Error; average magnitude of errors in predictions.

valloss: Quantifies the error between predictions and targets; lower indicates better learning.

valmae: Mean Absolute Error; average magnitude of errors in predictions.

epoch: Represents a performance indicator for model training or evaluation.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Unnamed: 0level0 epoch | loss Unnamed: 1level1 | mae Unnamed: 2level1 | valloss Unnamed: 3level1 | valmae Unnamed: 4level1 |
| 1.0000 | 0.1560 | 0.3138 | 0.2034 | 0.3531 |
| 2.0000 | 0.1547 | 0.3117 | 0.2045 | 0.3520 |
| 3.0000 | 0.1530 | 0.3095 | 0.2058 | 0.3462 |
| 4.0000 | 0.1507 | 0.3042 | 0.2051 | 0.3503 |
| 5.0000 | 0.1499 | 0.3030 | 0.2073 | 0.3488 |
| 6.0000 | 0.1490 | 0.3005 | 0.2067 | 0.3529 |
| 7.0000 | 0.1467 | 0.2974 | 0.2057 | 0.3471 |
| 8.0000 | 0.1466 | 0.2976 | 0.2119 | 0.3534 |
| 9.0000 | 0.1451 | 0.2940 | 0.2117 | 0.3523 |
| 10.0000 | 0.1437 | 0.2917 | 0.2090 | 0.3436 |
| 11.0000 | 0.1424 | 0.2895 | 0.2128 | 0.3493 |
| 12.0000 | 0.1408 | 0.2863 | 0.2116 | 0.3482 |
| 13.0000 | 0.1398 | 0.2843 | 0.2182 | 0.3624 |
| 14.0000 | 0.1393 | 0.2834 | 0.2125 | 0.3428 |
| 15.0000 | 0.1370 | 0.2789 | 0.2117 | 0.3444 |
| 16.0000 | 0.1359 | 0.2775 | 0.2167 | 0.3505 |
| 17.0000 | 0.1341 | 0.2734 | 0.2158 | 0.3459 |
| 18.0000 | 0.1333 | 0.2715 | 0.2151 | 0.3409 |
| 19.0000 | 0.1320 | 0.2685 | 0.2183 | 0.3486 |
| 20.0000 | 0.1300 | 0.2652 | 0.2192 | 0.3476 |

⚙️ Abstraction & Intuition:

Unnamed: 0level0 epoch: Represents a performance indicator for model training or evaluation.

loss Unnamed: 1level1: Quantifies the error between predictions and targets; lower indicates better learning.

mae Unnamed: 2level1: Mean Absolute Error; average magnitude of errors in predictions.

valloss Unnamed: 3level1: Quantifies the error between predictions and targets; lower indicates better learning.

valmae Unnamed: 4level1: Mean Absolute Error; average magnitude of errors in predictions.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 4: Evaluation Metrics**

Source:Cell 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | loss | mae | valloss | valmae |
| 1 | 0.1560 | 0.3138 | 0.2034 | 0.3531 |
| 2 | 0.1547 | 0.3117 | 0.2045 | 0.3520 |
| 3 | 0.1530 | 0.3095 | 0.2058 | 0.3462 |
| 4 | 0.1507 | 0.3042 | 0.2051 | 0.3503 |
| 5 | 0.1499 | 0.3030 | 0.2073 | 0.3488 |
| 6 | 0.1490 | 0.3005 | 0.2067 | 0.3529 |
| 7 | 0.1467 | 0.2974 | 0.2057 | 0.3471 |
| 8 | 0.1466 | 0.2976 | 0.2119 | 0.3534 |
| 9 | 0.1451 | 0.2940 | 0.2117 | 0.3523 |
| 10 | 0.1437 | 0.2917 | 0.2090 | 0.3436 |
| 11 | 0.1424 | 0.2895 | 0.2128 | 0.3493 |
| 12 | 0.1408 | 0.2863 | 0.2116 | 0.3482 |
| 13 | 0.1398 | 0.2843 | 0.2182 | 0.3624 |
| 14 | 0.1393 | 0.2834 | 0.2125 | 0.3428 |
| 15 | 0.1370 | 0.2789 | 0.2117 | 0.3444 |
| 16 | 0.1359 | 0.2775 | 0.2167 | 0.3505 |
| 17 | 0.1341 | 0.2734 | 0.2158 | 0.3459 |
| 18 | 0.1333 | 0.2715 | 0.2151 | 0.3409 |
| 19 | 0.1320 | 0.2685 | 0.2183 | 0.3486 |
| 20 | 0.1300 | 0.2652 | 0.2192 | 0.3476 |

⚙️ Abstraction & Intuition:

loss: Quantifies the error between predictions and targets; lower indicates better learning.

mae: Mean Absolute Error; average magnitude of errors in predictions.

valloss: Quantifies the error between predictions and targets; lower indicates better learning.

valmae: Mean Absolute Error; average magnitude of errors in predictions.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 5: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Unnamed: 0level0 epoch | Train Loss Unnamed: 1level1 |
| 1.0000 | 0.1560 |
| 2.0000 | 0.1547 |
| 3.0000 | 0.1530 |
| 4.0000 | 0.1507 |
| 5.0000 | 0.1499 |
| 6.0000 | 0.1490 |
| 7.0000 | 0.1467 |
| 8.0000 | 0.1466 |
| 9.0000 | 0.1451 |
| 10.0000 | 0.1437 |
| 11.0000 | 0.1424 |
| 12.0000 | 0.1408 |
| 13.0000 | 0.1398 |
| 14.0000 | 0.1393 |
| 15.0000 | 0.1370 |
| 16.0000 | 0.1359 |
| 17.0000 | 0.1341 |
| 18.0000 | 0.1333 |
| 19.0000 | 0.1320 |
| 20.0000 | 0.1300 |

⚙️ Abstraction & Intuition:

Unnamed: 0level0 epoch: Represents a performance indicator for model training or evaluation.

Train Loss Unnamed: 1level1: Quantifies the error between predictions and targets; lower indicates better learning.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 6: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Train | Loss |
| epoch | nan |
| 1 | 0.1560 |
| 2 | 0.1547 |
| 3 | 0.1530 |
| 4 | 0.1507 |
| 5 | 0.1499 |
| 6 | 0.1490 |
| 7 | 0.1467 |
| 8 | 0.1466 |
| 9 | 0.1451 |
| 10 | 0.1437 |
| 11 | 0.1424 |
| 12 | 0.1408 |
| 13 | 0.1398 |
| 14 | 0.1393 |
| 15 | 0.1370 |
| 16 | 0.1359 |
| 17 | 0.1341 |
| 18 | 0.1333 |
| 19 | 0.1320 |
| 20 | 0.1300 |

⚙️ Abstraction & Intuition:

Loss: Quantifies the error between predictions and targets; lower indicates better learning.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 7: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Unnamed: 0level0 epoch | Val Loss Unnamed: 1level1 |
| 1.0000 | 0.2034 |
| 2.0000 | 0.2045 |
| 3.0000 | 0.2058 |
| 4.0000 | 0.2051 |
| 5.0000 | 0.2073 |
| 6.0000 | 0.2067 |
| 7.0000 | 0.2057 |
| 8.0000 | 0.2119 |
| 9.0000 | 0.2117 |
| 10.0000 | 0.2090 |
| 11.0000 | 0.2128 |
| 12.0000 | 0.2116 |
| 13.0000 | 0.2182 |
| 14.0000 | 0.2125 |
| 15.0000 | 0.2117 |
| 16.0000 | 0.2167 |
| 17.0000 | 0.2158 |
| 18.0000 | 0.2151 |
| 19.0000 | 0.2183 |
| 20.0000 | 0.2192 |

⚙️ Abstraction & Intuition:

Unnamed: 0level0 epoch: Represents a performance indicator for model training or evaluation.

Val Loss Unnamed: 1level1: Quantifies the error between predictions and targets; lower indicates better learning.

🧠 Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 8: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Val | Loss |
| epoch | nan |
| 1 | 0.2034 |
| 2 | 0.2045 |
| 3 | 0.2058 |
| 4 | 0.2051 |
| 5 | 0.2073 |
| 6 | 0.2067 |
| 7 | 0.2057 |
| 8 | 0.2119 |
| 9 | 0.2117 |
| 10 | 0.2090 |
| 11 | 0.2128 |
| 12 | 0.2116 |
| 13 | 0.2182 |
| 14 | 0.2125 |
| 15 | 0.2117 |
| 16 | 0.2167 |
| 17 | 0.2158 |
| 18 | 0.2151 |
| 19 | 0.2183 |
| 20 | 0.2192 |

Abstraction & Intuition:

Loss: Quantifies the error between predictions and targets; lower indicates better learning.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 9: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Unnamed: 0level0 epoch | Train MAE Unnamed: 1level1 |
| 1.0000 | 0.3138 |
| 2.0000 | 0.3117 |
| 3.0000 | 0.3095 |
| 4.0000 | 0.3042 |
| 5.0000 | 0.3030 |
| 6.0000 | 0.3005 |
| 7.0000 | 0.2974 |
| 8.0000 | 0.2976 |
| 9.0000 | 0.2940 |
| 10.0000 | 0.2917 |
| 11.0000 | 0.2895 |
| 12.0000 | 0.2863 |
| 13.0000 | 0.2843 |
| 14.0000 | 0.2834 |
| 15.0000 | 0.2789 |
| 16.0000 | 0.2775 |
| 17.0000 | 0.2734 |
| 18.0000 | 0.2715 |
| 19.0000 | 0.2685 |
| 20.0000 | 0.2652 |

Abstraction & Intuition:

Unnamed: 0level0 epoch: Represents a performance indicator for model training or evaluation.

Train MAE Unnamed: 1level1: Mean Absolute Error; average magnitude of errors in predictions.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 10: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Train | MAE |
| epoch | nan |
| 1 | 0.3138 |
| 2 | 0.3117 |
| 3 | 0.3095 |
| 4 | 0.3042 |
| 5 | 0.3030 |
| 6 | 0.3005 |
| 7 | 0.2974 |
| 8 | 0.2976 |
| 9 | 0.2940 |
| 10 | 0.2917 |
| 11 | 0.2895 |
| 12 | 0.2863 |
| 13 | 0.2843 |
| 14 | 0.2834 |
| 15 | 0.2789 |
| 16 | 0.2775 |
| 17 | 0.2734 |
| 18 | 0.2715 |
| 19 | 0.2685 |
| 20 | 0.2652 |

Abstraction & Intuition:

MAE: Mean Absolute Error; average magnitude of errors in predictions.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 11: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Unnamed: 0level0 epoch | Val MAE Unnamed: 1level1 |
| 1.0000 | 0.3531 |
| 2.0000 | 0.3520 |
| 3.0000 | 0.3462 |
| 4.0000 | 0.3503 |
| 5.0000 | 0.3488 |
| 6.0000 | 0.3529 |
| 7.0000 | 0.3471 |
| 8.0000 | 0.3534 |
| 9.0000 | 0.3523 |
| 10.0000 | 0.3436 |
| 11.0000 | 0.3493 |
| 12.0000 | 0.3482 |
| 13.0000 | 0.3624 |
| 14.0000 | 0.3428 |
| 15.0000 | 0.3444 |
| 16.0000 | 0.3505 |
| 17.0000 | 0.3459 |
| 18.0000 | 0.3409 |
| 19.0000 | 0.3486 |
| 20.0000 | 0.3476 |

Abstraction & Intuition:

Unnamed: 0level0 epoch: Represents a performance indicator for model training or evaluation.

Val MAE Unnamed: 1level1: Mean Absolute Error; average magnitude of errors in predictions.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 12: Evaluation Metrics**

Source:Cell 14

|  |  |
| --- | --- |
| Val | MAE |
| epoch | nan |
| 1 | 0.3531 |
| 2 | 0.3520 |
| 3 | 0.3462 |
| 4 | 0.3503 |
| 5 | 0.3488 |
| 6 | 0.3529 |
| 7 | 0.3471 |
| 8 | 0.3534 |
| 9 | 0.3523 |
| 10 | 0.3436 |
| 11 | 0.3493 |
| 12 | 0.3482 |
| 13 | 0.3624 |
| 14 | 0.3428 |
| 15 | 0.3444 |
| 16 | 0.3505 |
| 17 | 0.3459 |
| 18 | 0.3409 |
| 19 | 0.3486 |
| 20 | 0.3476 |

Abstraction & Intuition:

MAE: Mean Absolute Error; average magnitude of errors in predictions.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 13: Evaluation Metrics**

Source:Cell 14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Unnamed: 0level0 epoch | Train Loss Unnamed: 1level1 | Val Loss Unnamed: 2level1 | Train MAE Unnamed: 3level1 | Val MAE Unnamed: 4level1 |
| 1.0000 | 0.1560 | 0.2034 | 0.3138 | 0.3531 |
| 2.0000 | 0.1547 | 0.2045 | 0.3117 | 0.3520 |
| 3.0000 | 0.1530 | 0.2058 | 0.3095 | 0.3462 |
| 4.0000 | 0.1507 | 0.2051 | 0.3042 | 0.3503 |
| 5.0000 | 0.1499 | 0.2073 | 0.3030 | 0.3488 |
| 6.0000 | 0.1490 | 0.2067 | 0.3005 | 0.3529 |
| 7.0000 | 0.1467 | 0.2057 | 0.2974 | 0.3471 |
| 8.0000 | 0.1466 | 0.2119 | 0.2976 | 0.3534 |
| 9.0000 | 0.1451 | 0.2117 | 0.2940 | 0.3523 |
| 10.0000 | 0.1437 | 0.2090 | 0.2917 | 0.3436 |
| 11.0000 | 0.1424 | 0.2128 | 0.2895 | 0.3493 |
| 12.0000 | 0.1408 | 0.2116 | 0.2863 | 0.3482 |
| 13.0000 | 0.1398 | 0.2182 | 0.2843 | 0.3624 |
| 14.0000 | 0.1393 | 0.2125 | 0.2834 | 0.3428 |
| 15.0000 | 0.1370 | 0.2117 | 0.2789 | 0.3444 |
| 16.0000 | 0.1359 | 0.2167 | 0.2775 | 0.3505 |
| 17.0000 | 0.1341 | 0.2158 | 0.2734 | 0.3459 |
| 18.0000 | 0.1333 | 0.2151 | 0.2715 | 0.3409 |
| 19.0000 | 0.1320 | 0.2183 | 0.2685 | 0.3486 |
| 20.0000 | 0.1300 | 0.2192 | 0.2652 | 0.3476 |

Abstraction & Intuition:

Unnamed: 0level0 epoch: Represents a performance indicator for model training or evaluation.

Train Loss Unnamed: 1level1: Quantifies the error between predictions and targets; lower indicates better learning.

Val Loss Unnamed: 2level1: Quantifies the error between predictions and targets; lower indicates better learning.

Train MAE Unnamed: 3level1: Mean Absolute Error; average magnitude of errors in predictions.

Val MAE Unnamed: 4level1: Mean Absolute Error; average magnitude of errors in predictions.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

**Table 14: Evaluation Metrics**

Source:Cell 14

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Train | Loss | Val | Loss.1 | Train.1 | MAE | Val.1 | MAE.1 |
| epoch | nan | nan | nan | nan | nan | nan | nan |
| 1 | 0.1560 | 0.2034 | 0.3138 | 0.3531 | nan | nan | nan |
| 2 | 0.1547 | 0.2045 | 0.3117 | 0.3520 | nan | nan | nan |
| 3 | 0.1530 | 0.2058 | 0.3095 | 0.3462 | nan | nan | nan |
| 4 | 0.1507 | 0.2051 | 0.3042 | 0.3503 | nan | nan | nan |
| 5 | 0.1499 | 0.2073 | 0.3030 | 0.3488 | nan | nan | nan |
| 6 | 0.1490 | 0.2067 | 0.3005 | 0.3529 | nan | nan | nan |
| 7 | 0.1467 | 0.2057 | 0.2974 | 0.3471 | nan | nan | nan |
| 8 | 0.1466 | 0.2119 | 0.2976 | 0.3534 | nan | nan | nan |
| 9 | 0.1451 | 0.2117 | 0.2940 | 0.3523 | nan | nan | nan |
| 10 | 0.1437 | 0.2090 | 0.2917 | 0.3436 | nan | nan | nan |
| 11 | 0.1424 | 0.2128 | 0.2895 | 0.3493 | nan | nan | nan |
| 12 | 0.1408 | 0.2116 | 0.2863 | 0.3482 | nan | nan | nan |
| 13 | 0.1398 | 0.2182 | 0.2843 | 0.3624 | nan | nan | nan |
| 14 | 0.1393 | 0.2125 | 0.2834 | 0.3428 | nan | nan | nan |
| 15 | 0.1370 | 0.2117 | 0.2789 | 0.3444 | nan | nan | nan |
| 16 | 0.1359 | 0.2167 | 0.2775 | 0.3505 | nan | nan | nan |
| 17 | 0.1341 | 0.2158 | 0.2734 | 0.3459 | nan | nan | nan |
| 18 | 0.1333 | 0.2151 | 0.2715 | 0.3409 | nan | nan | nan |
| 19 | 0.1320 | 0.2183 | 0.2685 | 0.3486 | nan | nan | nan |
| 20 | 0.1300 | 0.2192 | 0.2652 | 0.3476 | nan | nan | nan |

Abstraction & Intuition:

Loss: Quantifies the error between predictions and targets; lower indicates better learning.

Val: Represents a performance indicator for model training or evaluation.

Loss.1: Quantifies the error between predictions and targets; lower indicates better learning.

Train.1: Represents a performance indicator for model training or evaluation.

MAE: Mean Absolute Error; average magnitude of errors in predictions.

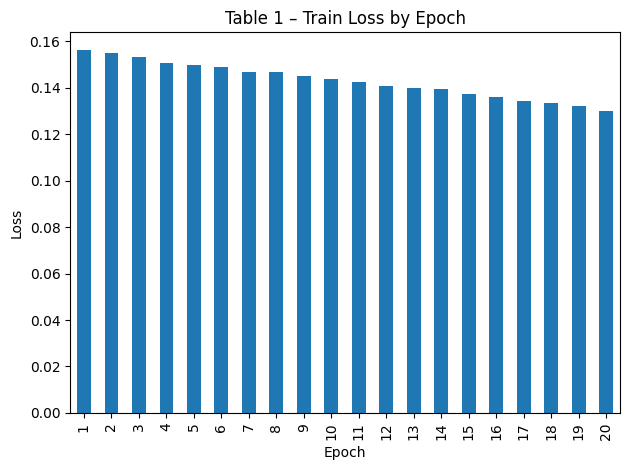
Val.1: Represents a performance indicator for model training or evaluation.

MAE.1: Mean Absolute Error; average magnitude of errors in predictions.

Insight:Analyze these metrics together to assess training dynamics and identify potential overfitting or underfitting. Patterns such as plateauing loss or divergence between training and validation metrics inform adjustments like learning rate changes or regularization.

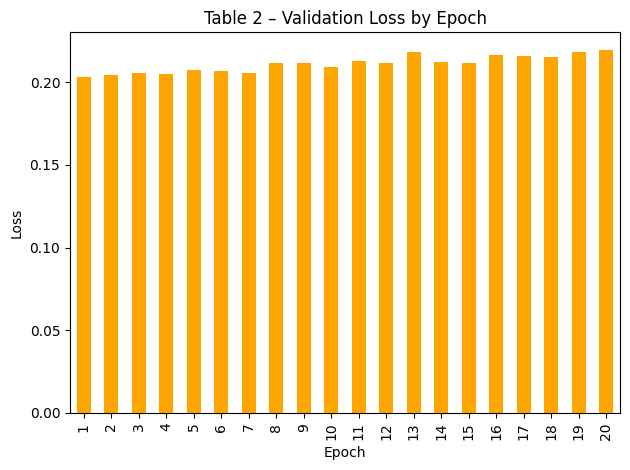
**Figure 8: Diagram**

***Source:Cell 15***



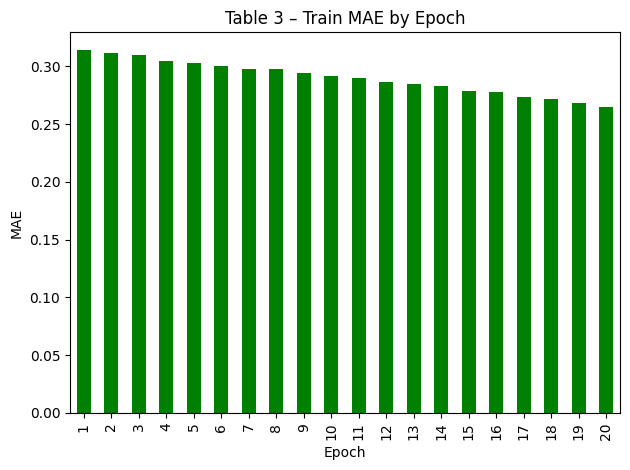
Intuition:Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

**Figure 9: Diagram**



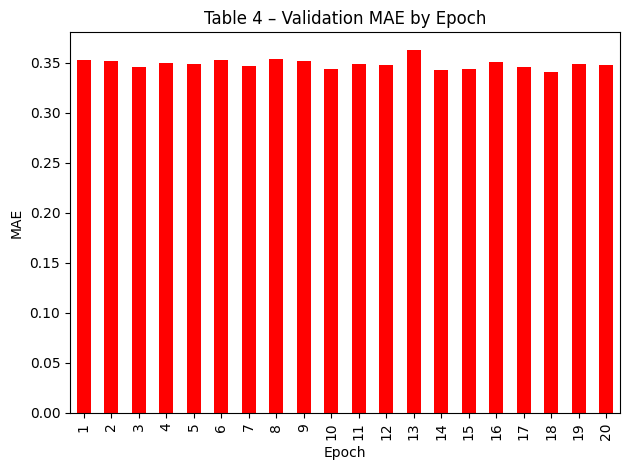
Intuition:Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

**Figure 10: Diagram**



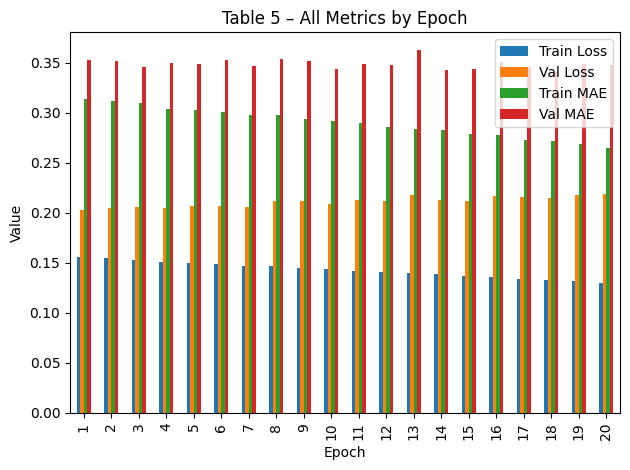
Intuition:Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

**Figure 11: Diagram**



Intuition:Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.

**Figure 12: Diagram**



Intuition:Visual representation of training metrics or model behavior, providing quick insights into convergence speed, stability, and potential anomalies.