Smart Home Energy Consumption

Deliverable 2

**Team Members:**

22F-3155 Ateeb Asad

22F-3163 Muhammad Soban

22F-3168 Mariam Fatima

Editable Format Diagrams:  
Deliverable 1: <https://lucid.app/lucidchart/c3583188-55b4-4033-8ef0-d468f4aeafa9/edit?viewport_loc=-1026%2C-383%2C2720%2C1259%2C0_0&invitationId=inv_21022dee-cea9-4d49-981d-dcb07f451d0f>

Deliverable 2: <https://lucid.app/lucidchart/450547ca-7599-418b-b1d5-975f2b130a80/edit?viewport_loc=1285%2C-610%2C3288%2C1522%2C0_0&invitationId=inv_6618aa24-b577-43db-ae3c-2983feea79a1>

# R2.1: FAISS Bucketing

## Conceptual Level:

We have partitioned high-dimensional embeddings into “buckets” or we can say clusters of similar vectors to facilitate fast similarity search and context retrieval. The recommender system technique that we used is FAISS in all three recommender systems, it leverages efficient indexing to group embeddings so the nearest neighbor queries run in sublinear time, which is crucial for large datasets.

## Programming Level:

We have performed embedding generation in which we encoded text descriptions into dense vectors using a transformer-based model. For index creation, we used FAISS index structure configures for L2-distance retrieval. Lastly, We added embeddings to the index and serialize it to disk for reuse.

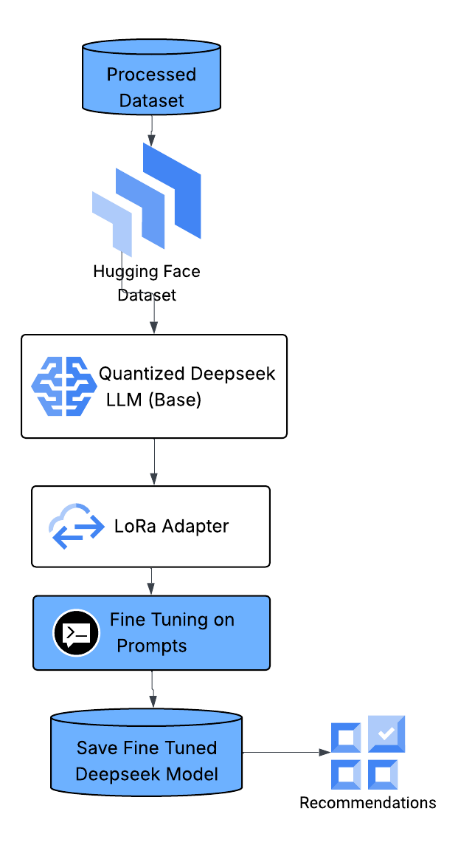
# R2.2: Deepseek R1 LLM Fine Tuning By Mariam:

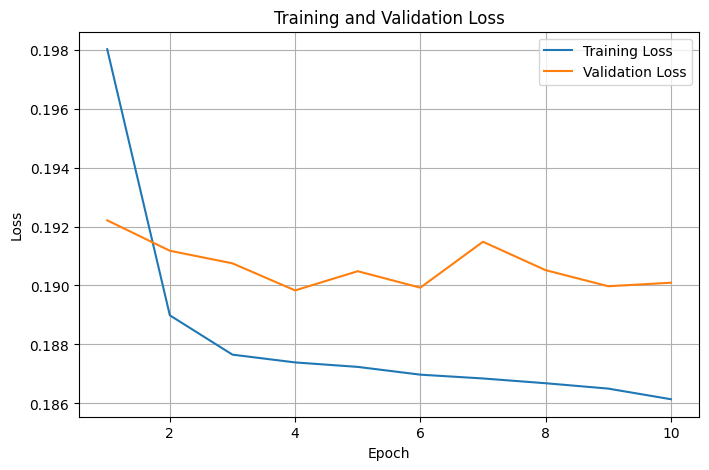
## Conceptual Level:

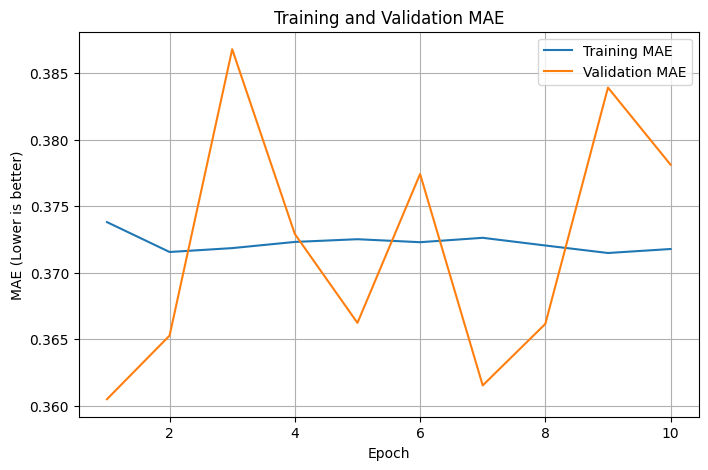
I have performed Fine-tuning on Deepseek R1 Distill Qwen large language model with 1.5 Billion parameters on smart home energy consumption data to enchance its recommendation accuracy. The purpose to fine tune is that, specialized models better understand dataset-specific vocabulary and patterns, yielding more relevant outputs.

## Programming Level:

I have retrieved the base LLM with quantization for efficiency. Then Applied low-rank (LoRa) adapters to reduce trainable parameter count. The records are converted into (prompt, response) pairs for supervised fine tuning. To train, I executed an optimized training loop with configured hyperparameters and then saved the tuned model.

Charts:





A graph with lines and numbers

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A graph of error and classification metrics

AI-generated content may be incorrect.

# R2.2: GPT-2 LLM Fine Tuning By Soban:

## Conceptual Level:

This module introduces the fine-tuning of a GPT-2 language model for producing context-sensitive energy-conserving suggestions. The system combines smart home appliance usage habits and semantic embedding retrieval (FAISS) to enable personalized energy recommendations. The process includes numeric smart home data conversion to descriptive text, semantically similar example retrieval, prompt construction, and fine-tuning of GPT-2 for recommendation generation.

**Programming Level:**

1. **Generation of Text Description**

Numeric smart home information is translated into descriptive sentences. Every row of the data set is translated into a natural language style of presenting the context (time, appliances, season, usage, etc.).

Example:

"At 18:00 on Summer evening, appliances: TV: ON, Dryer: OFF,. Energy: 1.45kWh"

1. **Embedding & FAISS Indexing**

A SentenceTransformer (all-MiniLM-L6-v2) pretrained model translates the text to semantic embeddings. The embeddings are indexed with FAISS to allow for efficient similarity search. For the provided query (e.g., a new data point), top-k most similar usage patterns are fetched.

1. **Prompt Construction**

A prompt is dynamically generated with information from the top-k most similar examples. The entry contains hour, part of day, season, power factor, appliance use, and energy measurements.

1. **Prompt + Completion Fine-tuning**

With the prompts and fixed completions, GPT-2 is trained in a causal language modeling environment. The model trains to make suitable recommendations after contextual patterns in other similar entries.

1. **Model Training**

The GPT-2 model is fine-tuned with HuggingFace Trainer. Training is set with smaller batch sizes, FP16 disabled to ensure stability, and gradient accumulation to save memory. The model is saved for inference after training.

1. **Recommendation Generation**

With a new query, a similar context is fetched with FAISS. A prompt is created and sent to the fine-tuned GPT-2 model. The model produces individualized energy-saving suggestions.

**Outputs Example**

**Query Description:**

19:00 on Winter evening, appliances: TV: ON, Oven: ON, Microwave: OFF,. Energy: 2.10kWh

**Generated Recommendation:**

A diagram of a process

AI-generated content may be incorrect.1. Prevent peak hour appliance use. 2. Switch off unused kitchen appliances. 3. Make use of smart plugs with timed-off.

# Pseudocode:

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

**Charts**

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.

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?  
  
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?  
  
  
  
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.

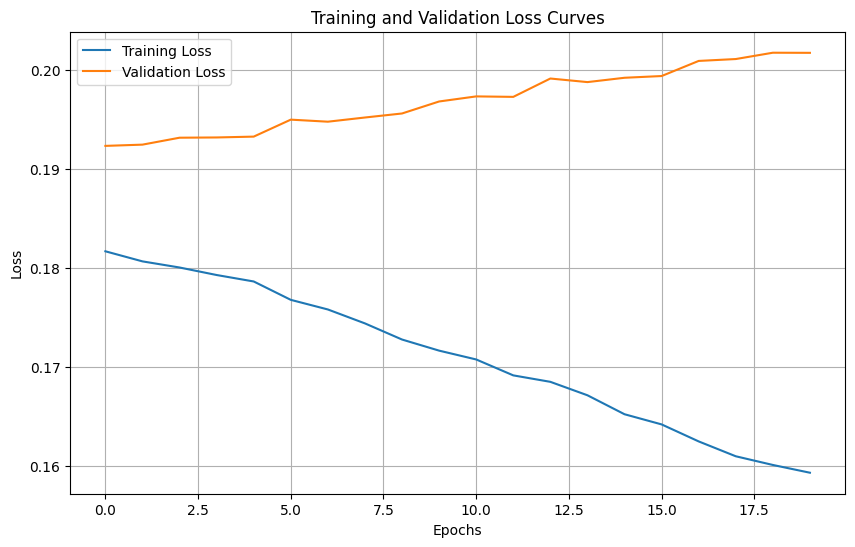
Training Time per Epoch – Highlights computational efficiency, crucial for real-time smart home applications.

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?

A graph with a line graph and numbers

AI-generated content may be incorrect.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?

A graph of a graph

AI-generated content may be incorrect.

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?

A graph of a positive rate

AI-generated content may be incorrect.

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?

A graph with a line

AI-generated content may be incorrect.

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?

A screen shot of a graph

AI-generated content may be incorrect.

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?  
**Mertics Table:**

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.

A screenshot of a graph

AI-generated content may be incorrect.

# R2.2: Distill GPT-2 LLM Fine Tuning By Ateeb:

## Conceptual Level:

The goal of using DisttillGPT2 LLM is to make the model capable of generating meaningful text summaries or recommendations that are relevant to our data. Only the original features of the dataset are used. By doing this, the language model becomes more knowledgeable and specific to our data’s domain.

## Programming Level:

The process starts by taking each row from the dataset and converting its feature values into a readable prompt, using natural language. This prompt is paired with a text description, which is what we want the language model to learn to generate. The entire dataset is prepared in this way: prompt as input, text description as target output. Then, both are tokenized (converted to the format the language model understands), and the language model is fine-tuned on these pairs. After training, the model is able to produce relevant energy usage reports for new, similar prompts.

## Pseudocode:

ALGORITHM: SINGLE\_ROW\_ENERGY\_RECOMMENDER

# ─── PREPARATION / TRAINING ────────────────────────────────────────────────────

INPUT:

• Raw CSV file of smart‑home energy data

OUTPUT:

• Fine‑tuned LLM (saved to disk) capable of generating five bullet‑list energy‑saving tips

based solely on a single row’s features

1. PREPROCESS\_DATA():

1.1 LOAD raw CSV into DataFrame DATA

1.2 DROP columns: Transaction\_ID, Unix Timestamp

1.3 FOR each row IN DATA:

a. COMPUTE is\_peak\_hour ← 1 if 6 ≤ Hour\_of\_Day ≤ 9 OR 18 ≤ Hour\_of\_Day ≤ 21, else 0

b. COMPUTE part\_of\_day ← “night”/“morning”/“afternoon”/“evening” by binning Hour\_of\_Day

c. COMPUTE is\_weekend ← 1 if Day\_of\_Week ∈ {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 hour\_sin ← sin(2π \* Hour\_of\_Day / 24)

f. COMPUTE hour\_cos ← cos(2π \* Hour\_of\_Day / 24)

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

h. COMPUTE total\_appliance\_usage ← SUM(power consumption of each appliance)

i. THRESHOLD ← 75th percentile of Energy\_Consumption\_kWh over DATA

j. SET is\_high\_consumption ← 1 if Energy\_Consumption\_kWh > 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 app\_efficiency\_ratio ← (app’s power consumption) / (Energy\_Consumption\_kWh + ε)

1.6 COMPUTE power\_factor ← Apparent\_Power / (Line\_Voltage \* Voltage + ε)

1.7 COMPUTE active\_appliances ← SUM(appliance power usages)

1.8 COMPUTE energy\_per\_active\_appliance ← Energy\_Consumption\_kWh / (active\_appliances + ε)

1.9 RETURN preprocessed DATA

2. BUILD\_TRAINING\_PAIRS(DATA):

2.1 INIT empty LIST prompt\_texts, completion\_texts

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

2.3 FOR each idx IN example\_indices:

a. row ← DATA.iloc[idx]

b. CONSTRUCT appliances\_status\_string ←

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

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

JOIN with commas

c. BUILD prompt\_i ←

"Context:\n"

"- Hour: " + row['Hour of the Day'] + " (" + row['part\_of\_day'] + ")\n"

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

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

"- Active Appliances: " + appliances\_status\_string + "\n"

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

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

"Now, recommend exactly five bullet‑list energy‑saving tips:\n"

"Tips:"

d. SET completion\_i ← A list of exactly five human‑crafted bullet lines:

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

e. APPEND prompt\_i TO prompt\_texts

f. APPEND "\n".join(completion\_i) TO completion\_texts

2.4 CREATE HuggingFace Dataset:

train\_examples ← [{"prompt": p, "completion": c}

FOR (p, c) IN zip(prompt\_texts, completion\_texts)]

train\_dataset ← Dataset.from\_list(train\_examples)

2.5 DEFINE tokenize\_and\_mask(examples):

FOR each example IN examples:

full\_texts ← example["prompt"] + "\n" + example["completion"]

tokenized = tokenizer(

full\_texts,

padding="max\_length",

truncation=True,

max\_length=512,

return\_tensors="pt"

)

input\_ids ← tokenized["input\_ids"]

attention\_mask ← tokenized["attention\_mask"]

prompt\_token\_count ← len(tokenizer(example["prompt"], truncation=True, max\_length=512)["input\_ids"])

labels ← input\_ids.clone()

FOR t IN [0 .. prompt\_token\_count−1]:

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

RETURN {"input\_ids": input\_ids, "attention\_mask": attention\_mask, "labels": labels}

2.6 APPLY map(tokenize\_and\_mask) to train\_dataset (batched=True), removing ["prompt","completion"]

2.7 RETURN tokenized\_train

3. FINE\_TUNE\_LLM(tokenized\_train):

3.1 tokenizer = AutoTokenizer.from\_pretrained("distilgpt2")

tokenizer.pad\_token = tokenizer.eos\_token

3.2 model = AutoModelForCausalLM.from\_pretrained("distilgpt2")

model.resize\_token\_embeddings(len(tokenizer))

3.3 data\_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

3.4 training\_args = TrainingArguments(

output\_dir = "./fine\_tuned\_energy\_lm\_single",

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

num\_train\_epochs = 10, # increase if underfitting

learning\_rate = 3e-5,

warmup\_steps = 50,

logging\_steps = 10,

save\_strategy = "epoch",

fp16 = torch.cuda.is\_available(),

report\_to = "none"

)

3.5 trainer = Trainer(

model = model,

args = training\_args,

train\_dataset = tokenized\_train,

data\_collator = data\_collator

)

3.6 trainer.train()

3.7 model.save\_pretrained("./fine\_tuned\_energy\_lm\_single")

tokenizer.save\_pretrained("./fine\_tuned\_energy\_lm\_single")

3.8 RETURN (model, tokenizer, trainer)

# ─── INFERENCE ──────────────────────────────────────────────────────────────────

INPUT:

• Saved tokenizer & model directory (“./fine\_tuned\_energy\_lm\_single”)

• Preprocessed DATA DataFrame

OUTPUT:

• Five bullet‑list energy‑saving tips for any new row index Q\_IDX

FUNCTION GENERATE\_TIPS\_SINGLE(Q\_IDX, DATA, model, tokenizer):

1. row ← DATA.iloc[Q\_IDX]

2. appliances\_status\_string ←

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['part\_of\_day'] + ")\n"

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

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

"- Active Appliances: " + appliances\_status\_string + "\n"

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

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

"Now, recommend exactly five bullet‑list energy‑saving tips:\n"

"Tips:"

4. inputs = tokenizer(

prompt,

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

max\_length=None,

return\_tensors="pt"

).to(model.device)

5. outputs = model.generate(

inputs,

max\_new\_tokens = 80, # enough to cover five bullet lines

num\_beams = 5,

temperature = 0.6,

no\_repeat\_ngram\_size = 2,

early\_stopping = True,

pad\_token\_id = tokenizer.eos\_token\_id

)

6. full\_text = tokenizer.decode(outputs[0], skip\_special\_tokens=False)

7. remainder = full\_text[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. Single-Row 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, part\_of\_day, 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 bullet‑list energy‑saving tips:"

"Tips:"

ensures the model knows exactly where to begin generation.

3. Human‑Crafted 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 bullet‑list generation without re‑learning to reproduce prompt.

5. Training Hyperparameters

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

• per\_device\_train\_batch\_size = 4 balances GPU memory constraints and stable gradient updates.

• learning\_rate = 3e-5 is modest, preserving pre‑trained weights while adapting to this task.

• save\_strategy = '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.

• num\_beams = 5 and temperature = 0.6 balance coherence and creativity in generated tips.

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

7. Trade‑Offs

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

• Cons: Recommendations may be more generic compared to retrieval‑augmented 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.

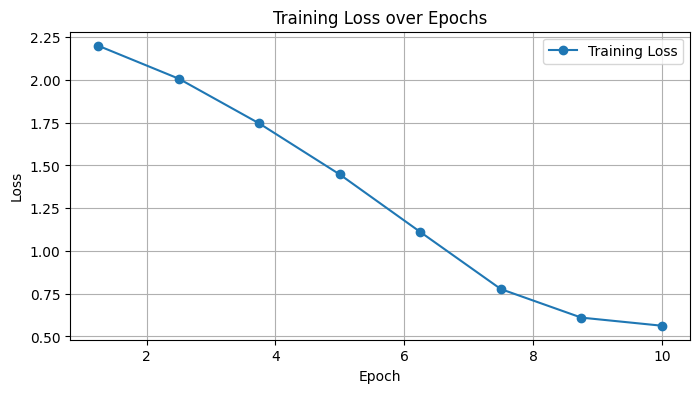
• Use larger base models (e.g., "gpt2-medium", "gpt2-large") if GPU memory allows for more nuanced generation.

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

**Enhanced Abstraction: Basic LLM Notebook Evaluation Metrics & Diagrams**

**Figure 1: Diagram**

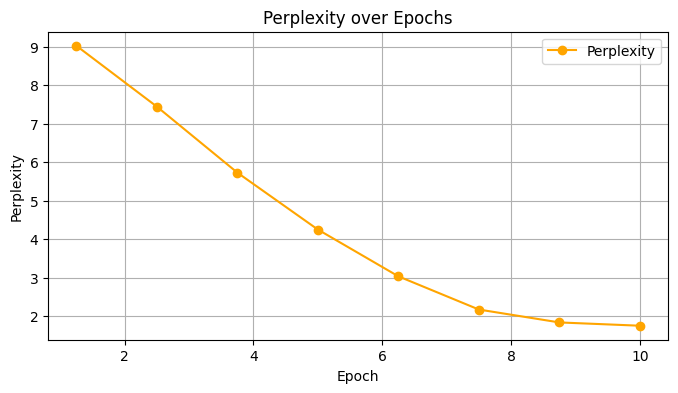
***Source: Cell 8***



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

**Figure 2: Diagram**

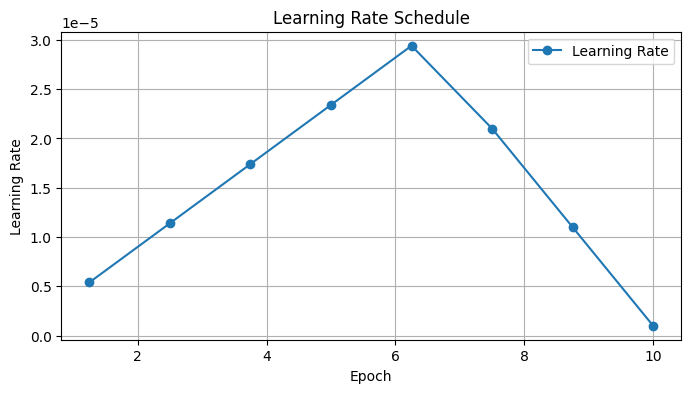
***Source: Cell 8***



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

**Figure 3: Diagram**

***Source: Cell 8***



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

**Figure 4: Diagram**

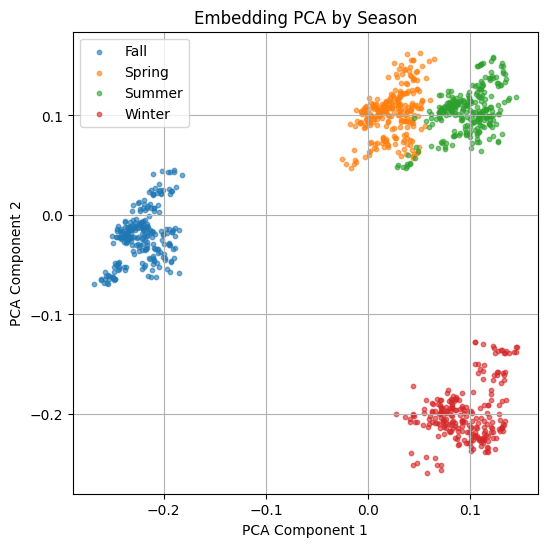
***Source: Cell 8***



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

**Figure 5: Diagram**

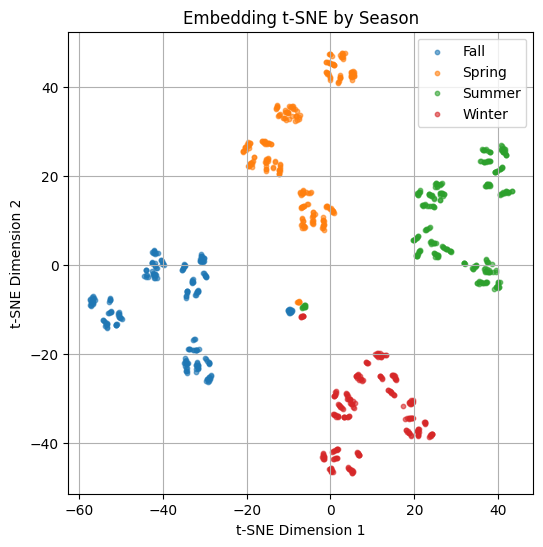
***Source: Cell 8***



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

**Figure 6: Diagram**

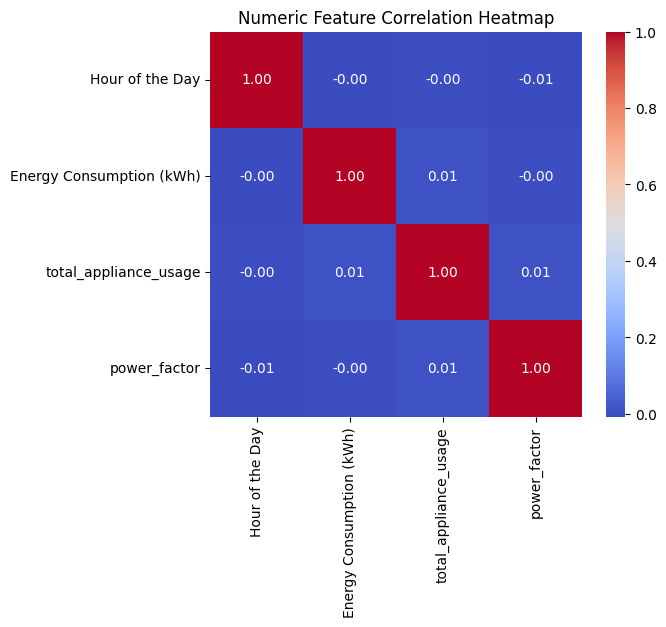
***Source: Cell 8***



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

**Figure 7: Diagram**

***Source: Cell 8***



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

**Table 1: Evaluation Metrics**

Source: Cell 11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Unnamed: 0** | **loss** | **mae** | **val\_loss** | **val\_mae** | **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** |

Abstraction & Intuition:

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.

val\_loss: Quantifies the error between predictions and targets; lower indicates better learning.

val\_mae: 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.

**Table 2: Evaluation Metrics**

Source: Cell 11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Index** | **loss** | **mae** | **val\_loss** | **val\_mae** | **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.

val\_loss: Quantifies the error between predictions and targets; lower indicates better learning.

val\_mae: 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.

**Table 3: Evaluation Metrics**

Source: Cell 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Unnamed: 0\_level\_0 epoch** | **loss Unnamed: 1\_level\_1** | **mae Unnamed: 2\_level\_1** | **val\_loss Unnamed: 3\_level\_1** | **val\_mae Unnamed: 4\_level\_1** |
| **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: 0\_level\_0 epoch: Represents a performance indicator for model training or evaluation.

loss Unnamed: 1\_level\_1: Quantifies the error between predictions and targets; lower indicates better learning.

mae Unnamed: 2\_level\_1: Mean Absolute Error; average magnitude of errors in predictions.

val\_loss Unnamed: 3\_level\_1: Quantifies the error between predictions and targets; lower indicates better learning.

val\_mae Unnamed: 4\_level\_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.

**Table 4: Evaluation Metrics**

Source: Cell 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **epoch** | **loss** | **mae** | **val\_loss** | **val\_mae** |
| **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.

val\_loss: Quantifies the error between predictions and targets; lower indicates better learning.

val\_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 5: Evaluation Metrics**

Source: Cell 14

|  |  |
| --- | --- |
| **Unnamed: 0\_level\_0 epoch** | **Train Loss Unnamed: 1\_level\_1** |
| **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: 0\_level\_0 epoch: Represents a performance indicator for model training or evaluation.

Train Loss Unnamed: 1\_level\_1: 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: 0\_level\_0 epoch** | **Val Loss Unnamed: 1\_level\_1** |
| **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: 0\_level\_0 epoch: Represents a performance indicator for model training or evaluation.

Val Loss Unnamed: 1\_level\_1: 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: 0\_level\_0 epoch** | **Train MAE Unnamed: 1\_level\_1** |
| **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: 0\_level\_0 epoch: Represents a performance indicator for model training or evaluation.

Train MAE Unnamed: 1\_level\_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.

**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: 0\_level\_0 epoch** | **Val MAE Unnamed: 1\_level\_1** |
| **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: 0\_level\_0 epoch: Represents a performance indicator for model training or evaluation.

Val MAE Unnamed: 1\_level\_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.

**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: 0\_level\_0 epoch** | **Train Loss Unnamed: 1\_level\_1** | **Val Loss Unnamed: 2\_level\_1** | **Train MAE Unnamed: 3\_level\_1** | **Val MAE Unnamed: 4\_level\_1** |
| **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: 0\_level\_0 epoch: Represents a performance indicator for model training or evaluation.

Train Loss Unnamed: 1\_level\_1: Quantifies the error between predictions and targets; lower indicates better learning.

Val Loss Unnamed: 2\_level\_1: Quantifies the error between predictions and targets; lower indicates better learning.

Train MAE Unnamed: 3\_level\_1: Mean Absolute Error; average magnitude of errors in predictions.

Val MAE Unnamed: 4\_level\_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.

**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.

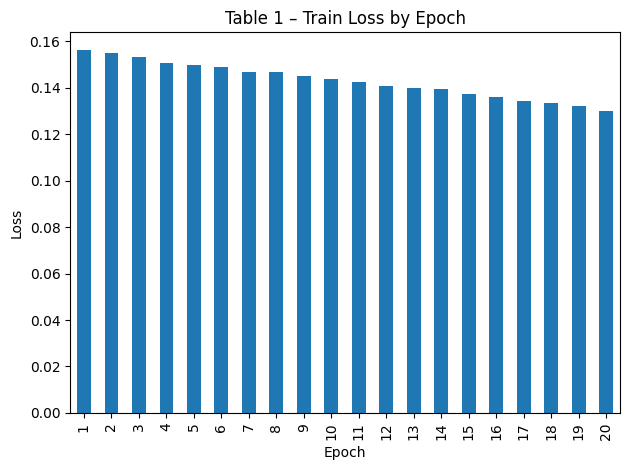
Val.1: Represents a performance indicator for model training or evaluation.

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**

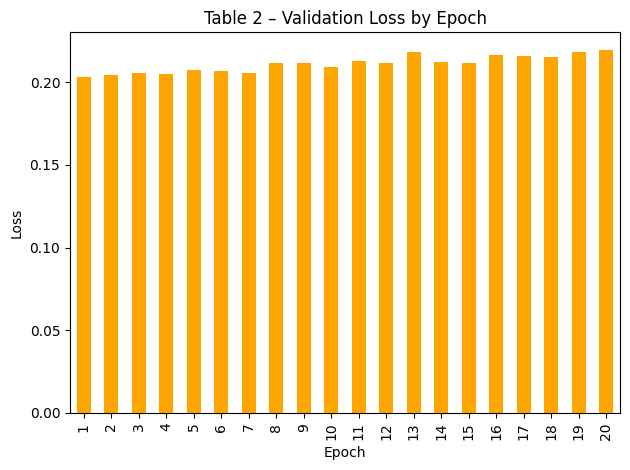
***Source: Cell 15***



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

**Figure 9: Diagram**

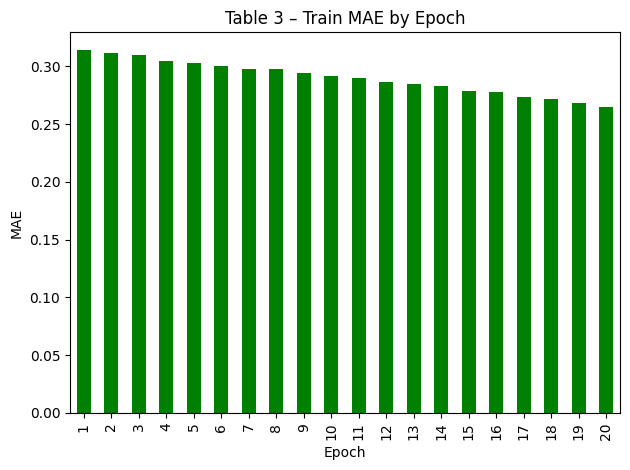
***Source: Cell 15***



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

**Figure 10: Diagram**

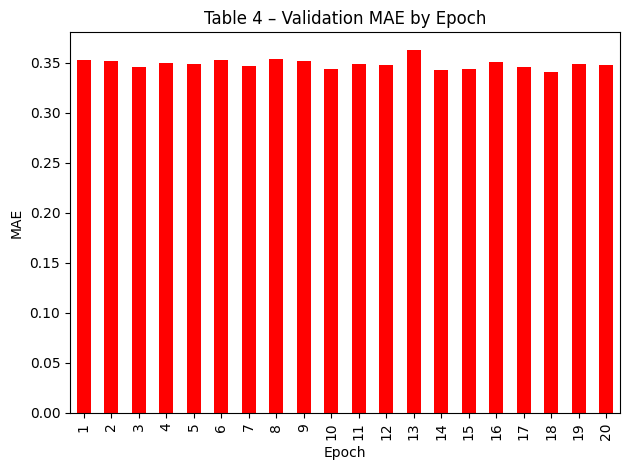
***Source: Cell 15***



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

**Figure 11: Diagram**

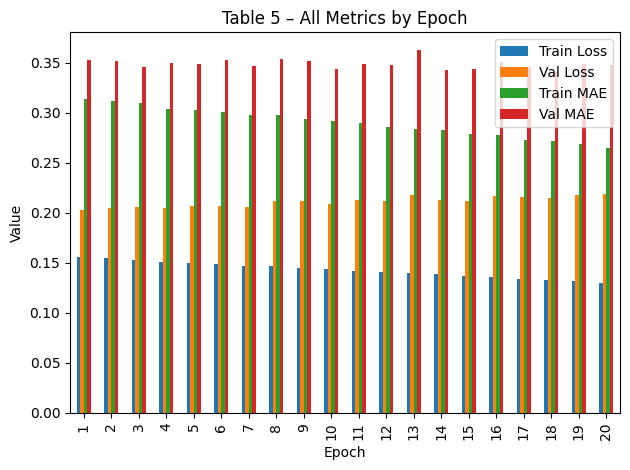
***Source: Cell 15***



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

**Figure 12: Diagram**

***Source: Cell 15***



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

# R2.3: Two-Step Tuning Process:

## Conceptual Level:

The initial Fine-tuning of LLM on prompts derived from dataset to capture foundational associations. Then the fine tuned model is enhanced by augmentation each input with similar case context which is retrieved via FAISS.

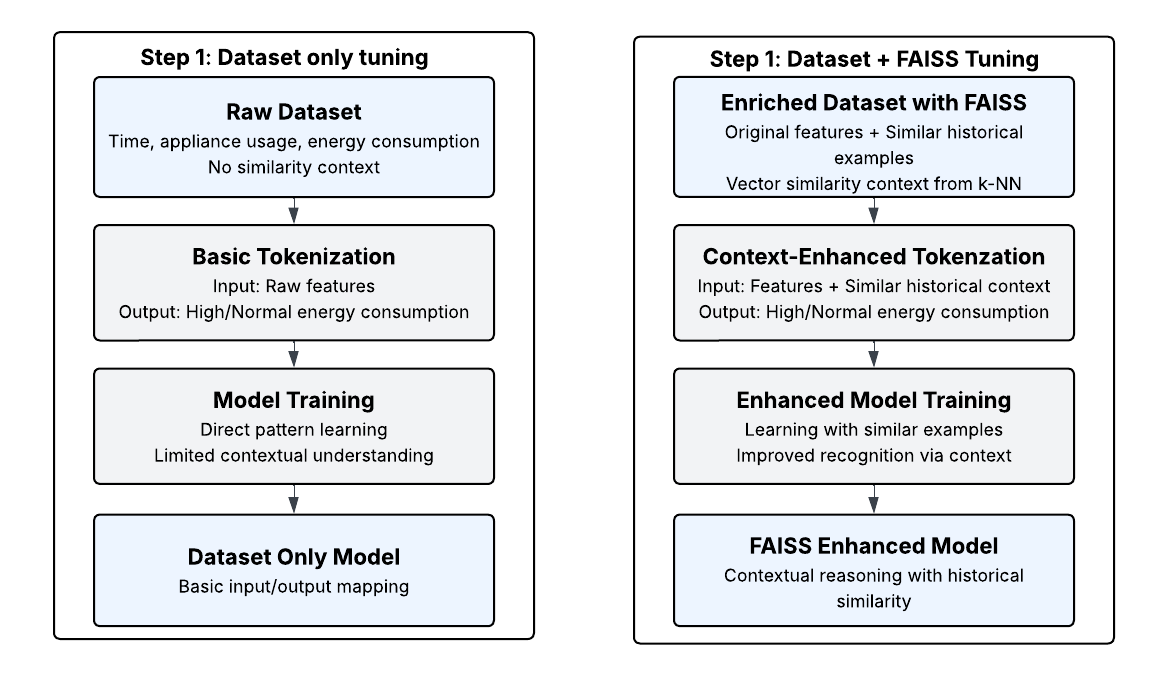
A diagram of a data processing process

AI-generated content may be incorrect.

## Programming Level:

Step 1: Apply LoRa (Low Rank) fine tuning on basic prompts.

Step 2: Retrieve similar prompts using FAISS, enrich the current input and fine tune further on these enhanced samples.



# R 2.4:

## Conceptual Level:

The purpose is to generate dynamic contextual energy consumption recommendations by leveraging both FAISS-based retrieval and generative Large Language Model capabilities. It integrates the strength of memory based reasoning with generative intelligence like deepseek, gpt for highly personalized outputs according to the problem niche.

## Programming Level:

Firstly the raw data is converted into descriptive text. This text is embedded and used to query the FAISS index to check for similar examples. Then the input and retrieved contexts via FAISS are then merged into a prompt, Then the merged prompt is fed into the fine tuned LLM to generate the final recommendations.

A diagram of a process

AI-generated content may be incorrect.