Large Event Model

Home-GenAI TF

SRI-B

February 17, 2025

Outline

Objective

- Develop a mathematical formulation for the Large Event Model (LEM) for smart homes.
- Integrate:
  - Multi-head attention transformer.
  - Knowledge Graph (KG) embeddings.
- Predict appropriate appliance settings based on interruptions, time, and context.

Motivation for Building a Large Event Model (LEM)

- Enhanced Predictive Capabilities
  - Anticipate user needs based on complex event patterns
  - Enable sophisticated, context-aware automation
- Improved User Experience (more ”Premium” services can be sold)
  - Seamless adaptation to interruptions and schedule changes
  - Personalized environment adjustments
- Safety and Security
  - Monitoring for child/elderly safety for unplanned scenarios
  - Anomaly detection for potential security issues
- Energy Efficiency
  - Optimize energy consumption based on usage patterns
  - Adapt to dynamic family schedules

Possible Use-Cases

What LEM will do	Under what scenario
Intelligent Interruption Handling	Without requiring any ”Routine” setup, automatically adjust home settings during interruptions (e.g., pausing TV when doorbell rings)
Child Safety and Monitoring	Child comes home ahead of scheduled, unplanned. LEM works by disabling dangerous appliances and monitoring activities
Coordinated Family Schedules	In a household with working parents and school-going children, schedules can often change unexpectedly. If a family member is delayed, the system adjusts home settings accordingly

Why we need to improve over Traditional LLMs

Aspect	Large Event Models (LEMs)	LLMs
Primary Focus	Real-time event interpretation & prediction in smart homes	Natural language processing & generation
Temporal Awareness	Understand event sequences & patterns in household context	Lack fine-grained temporal reasoning for real-world events
Device Integration	Direct interaction with smart home devices & sensors	Not designed for device control or sensor data interpretation
Data Processing	Local processing (edge computing) for enhanced privacy	Often require cloud processing, raising privacy concerns
Contextual Understanding	Grasp relationships between events, users, & home environment	Understand textual context, not real-world event context
Real-time Adaptation	Continuous adaptation to changing home dynamics	Static models, not for real-time learning & adaptation

Comparing to Large Language Models (LLMs)

Similarities:

- Both models process sequences of data.
- Use embeddings to represent input tokens.
- Employ attention mechanisms to model dependencies.

Differences:

- **Tokens in LLMs:** Words or subword units in natural language.
- **Tokens in LEM:** Embeddings of interruptions, time, and KG nodes.

Bag-of-Words Equivalent:

- In LEM, sequence elements act as context features.
- Unlike bag-of-words, LEM maintains sequence order and positional information.

Tokens and Embeddings in LEM

Definition of Tokens:

- Each element in the input sequence is analogous to a token.
- Examples include interruption embeddings, time embeddings, and KG node embeddings.

Embedding Process:

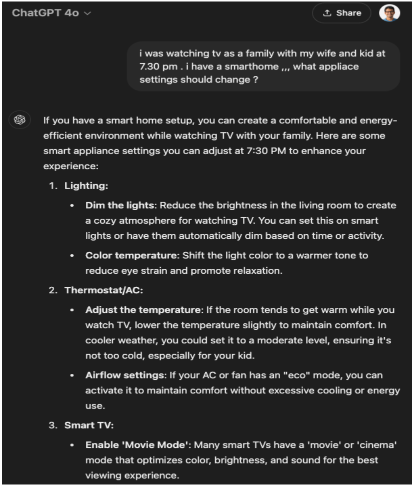
- Map tokens to high-dimensional vectors.
- Capture semantic and relational information.

Positional Encoding:

- Add positional embeddings to maintain order information.
- Crucial for the Transformer to model sequence dependencies.

LLM Output vs LEM Output

LLM Output



ChatGPT 4o

Share

i was watching tv as a family with my wife and kid at 7.30 pm . i have a smarthome ,,, what appliace settings should change ?

If you have a smart home setup, you can create a comfortable and energy-efficient environment while watching TV with your family. Here are some smart appliance settings you can adjust at 7:30 PM to enhance your experience:

1. **Lighting:**
   - **Dim the lights**: Reduce the brightness in the living room to create a cozy atmosphere for watching TV. You can set this on smart lights or have them automatically dim based on time or activity.
   - **Color temperature**: Shift the light color to a warmer tone to reduce eye strain and promote relaxation.
2. **Thermostat/AC:**
   - **Adjust the temperature**: If the room tends to get warm while you watch TV, lower the temperature slightly to maintain comfort. In cooler weather, you could set it to a moderate level, ensuring it's not too cold, especially for your kid.
   - **Airflow settings**: If your AC or fan has an "eco" mode, you can activate it to maintain comfort without excessive cooling or energy use.
3. **Smart TV:**
   - **Enable 'Movie Mode'**: Many smart TVs have a 'movie' or 'cinema' mode that optimizes color, brightness, and sound for the best viewing experience.

LEM Output

LEM Output Features:

- Automated prompting
- Will be signficantly smaller in size than LLMs, or SLMs, and can be run on the edge.
- Aware of what's available
- Generate JSON settings across appliances

Preview of LEM Theory

Model Predictions:

- The LEM outputs probabilities for appliance settings.
- Uses activation functions like sigmoid or softmax.

Calculating Probabilities:

- For each appliance, compute:

$$P(S_i = 1|\text{input}) = \sigma(\mathbf{h}_i^\top \mathbf{w}_i + b_i)$$

- Where σ is the sigmoid function, $\mathbf{h}_i$ is the hidden state, $\mathbf{w}_i$ and b_i are weights and biases.

Loss Function:

- Use binary cross-entropy loss for training:

$$\mathscr{L} = -\sum_i (S_i \log P(S_i) + (1 - S_i)\log(1 - P(S_i)))$$

- S_i is the actual appliance setting (0 or 1).

Detailed Theory of LEM

Problem Statement

Given:

- An interruption event I at time t.
- Knowledge Graph $G = (V, E)$ capturing relationships.
- Historical events and activities $\mathscr{H}$.

Goal:

- Predict appliance settings $\mathbf{S}$.
- Estimate $P(\mathbf{S} \mid I, t, G, \mathscr{H})$.

Mathematical Objective

Conditional Probability

$$\mathbf{S}^* = \arg\max_{\mathbf{S}} P(\mathbf{S} \mid I, t, G, \mathscr{H})$$

- $\mathbf{S}$: Vector of appliance settings.
- I: Interruption features.
- t: Time features.
- G: Knowledge Graph embeddings.
- $\mathscr{H}$: Historical context.

Interruption Representation

- Categorical features encoded as embeddings.
- Feature vector $\mathbf{e}_I \in \mathbb{R}^{d_I}$.

Example Interruption:

- **Someone comes at the door while an user is watching TV**
  - Type: `Doorbell`
  - Initiator: `Visitor`
  - Time: `7:30 PM`
- **Feature Vector Representation:**

$$\begin{aligned}\mathbf{e}_I &= \text{Embedding}(\text{Type, Initiator}) \\ &= \text{Embedding}(\texttt{Doorbell}, \texttt{Visitor}) \\ &= \begin{bmatrix} 0.5 \\ -0.3 \\ 0.8 \\ \vdots \\ 0.1 \end{bmatrix} \in \mathbb{R}^{d_I}\end{aligned}$$

  - Where d_I is the embedding dimension (e.g., 16).
  - Embeddings are learned representations capturing interruption characteristics.
- **Explanation:**

Time Representation

- Cyclical encoding of time:

$$\text{Hour}_{\text{sin}} = \sin\left(\frac{2\pi\,\text{Hour}}{24}\right)$$
$$\text{Hour}_{\text{cos}} = \cos\left(\frac{2\pi\,\text{Hour}}{24}\right)$$

- Time embedding $\mathbf{e}_t \in \mathbb{R}^{d_t}$.
- **Simplified Circadian Rhythm Modeling:**
  - Captures daily patterns of human activity.
  - Placeholder for modeling human-time dependency.
  - Can be enhanced with wearable data in the future.

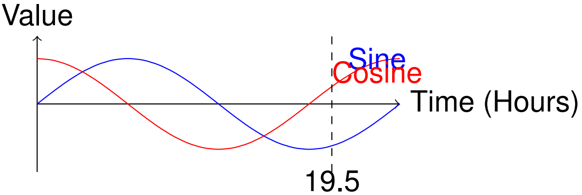
Time Representation

- **Cyclical Encoding of Time:**

$$\text{Hour} = 19.5 \quad (\text{7:30 PM})$$

$$\text{Hour}_{\sin} = \sin\left(\frac{2\pi \times 19.5}{24}\right) \approx -0.9659$$

$$\text{Hour}_{\cos} = \cos\left(\frac{2\pi \times 19.5}{24}\right) \approx -0.2588$$



- Sine and cosine functions provide complementary information due to their phase difference. Together, they capture both aspects of the positional information, enabling the model to understand the position more comprehensively.

Sinusoidal Positional Encoding in LEM

Challenge: Transformers process inputs in parallel, lacking inherent order awareness.

Need for Positional Encoding:

- Encode the position of each element in the sequence.
- Allow the model to capture temporal dependencies.

Sinusoidal Encoding Formulas:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

where:pos: Position in the sequence, i: Dimension index, d: Model dimensionality.

Key Advantages:

- **Unique Positions:** Provides distinct encoding for each position.
- **Relative Positioning:** Model can learn relative distances.
- **Generalization:** Handles unseen positions

Knowledge Graph G

- **Nodes:**
  - **People:** Alice (Parent), Bob (Parent), Charlie (Child), Daisy (Elderly).
  - **Appliances:** TV, Lights, Thermostat, Oven, Door Lock.
  - **Activities:** Watching Movie, Cooking, Sleeping, Studying.
  - **Rooms:** Living Room, Kitchen, Bedroom.
  - **Interruptions:** Doorbell, Phone Call.
- **Edges:**
  - **Participates In:** Alice $\rightarrow$ Watching Movie.
  - **Uses:** Watching Movie $\rightarrow$ TV.
  - **Located In:** TV $\rightarrow$ Living Room.
  - **Affects:** Doorbell $\rightarrow$ Watching Movie.
- **Feature Matrix ($\mathbf{X}$) Example:**

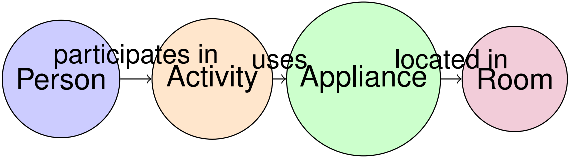
Node	Type	Feature Vector
Alice	Person	$[1, 0, 0, 0, 0]$
TV	Appliance	$[0, 1, 0, 0, 0]$
Watching Movie	Activity	$[0, 0, 1, 0, 0]$
Living Room	Room	$[0, 0, 0, 1, 0]$

Graph Neural Network (GNN)

- Use Relational Graph Convolutional Network (R-GCN).
- Handles multiple edge types (relations).

Graph Neural Network (GNN)

- **Purpose:** Generate embeddings for nodes in the KG.
- **Architecture:**
  - Input: Feature matrix $\mathbf{X}$ and adjacency matrices for each relation type.
  - Layers: Multiple GNN layers (e.g., R-GCN).
  - Output: Node embeddings $\mathbf{H}^{(L)}$.
- **Illustration:**



  - Shows how nodes are connected via edges (relations).

Relational Graph Convolutional Network (R-GCN)

- **What is R-GCN?**
  - Extension of standard GCN to handle multiple types of relations.
  - Learns relation-specific transformations.
- **How it Works:**
  - Aggregates messages from neighbors j connected via relation r.
- **Why it's Important:**
  - Captures the heterogeneity of the KG.
  - Allows the model to learn different importance levels for different relations.
- **Example:**
  - Node i: `Watching Movie` (Activity).
  - Relations:
    - `participated by` (from Persons).
    - `uses` (to Appliances).
  - Embedding $\mathbf{h}_i^{(l+1)}$ incorporates influences from both relations.

R-GCN Layer

Message Passing: For each node i and relation r:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{r\in\mathscr{R}}\sum_{j\in\mathscr{N}_i^r}\frac{1}{c_{i,r}}\mathbf{W}_r^{(l)}\mathbf{h}_j^{(l)} + \mathbf{W}_0^{(l)}\mathbf{h}_i^{(l)}\right)$$

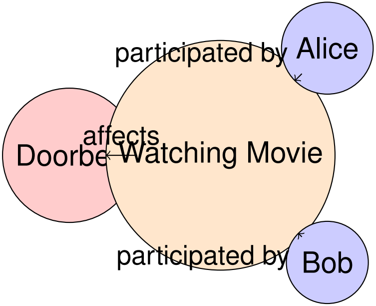
- $\mathscr{R}$: Set of relation types.
- $\mathscr{N}_i^r$: Neighbors of node i under relation r.
- $c_{i,r}$: Normalization constant.
- $\mathbf{W}_r^{(l)}$: Relation-specific weight matrix.
- σ: Activation function (e.g., ReLU).

Node Embeddings

- After L layers, obtain embeddings $\mathbf{H}^{(L)} \in \mathbb{R}^{|V| \times d_h}$.
- For nodes of interest (e.g., interruption node), extract embeddings.

Node Embeddings

- **After GNN Layers:**
  - Each node has an embedding $\mathbf{h}_i^{(L)} \in \mathbb{R}^{d_h}$.
- **Example: Interruption Node**
  - Node: `Doorbell` (Interruption).
  - Embedding $\mathbf{h}_{\text{Doorbell}}^{(L)}$ captures:
    - Its relationships with affected activities.
    - Impact on participants.
- **Graphical Representation:**



  - Visualizes how the interruption connects to activities and people.

Input Sequence

- Sequence $\mathbf{S} = [\mathbf{e}_I, \mathbf{e}_t, \mathbf{e}_G]$.
- $\mathbf{e}_G$: KG embeddings relevant to the interruption.

Transformer Model

- **Input Sequence Example:**

$$\mathbf{S} = \left[\mathbf{e}_I, \mathbf{e}_t, \mathbf{e}_{\text{Doorbell}}, \mathbf{e}_{\text{Watching Movie}}, \mathbf{e}_{\text{Alice}}, \mathbf{e}_{\text{Bob}}\right]$$

  - $\mathbf{e}_I$: Interruption embedding.
  - $\mathbf{e}_t$: Time embedding.
  - $\mathbf{e}_{\text{Doorbell}}$: KG embedding of Doorbell node.
  - $\mathbf{e}_{\text{Watching Movie}}$, $\mathbf{e}_{\text{Alice}}$, $\mathbf{e}_{\text{Bob}}$: KG embeddings.
- **Transformer Mechanism:**
  - Attention weights determine the influence of each input element.
  - Learns which parts of the input are most relevant for prediction.

Input Sequence → Transformer Encoder → Appliance Settings

- **Periodicity:** Sinusoids naturally model cyclical behaviors (e.g., daily routines).
- **Temporal Dependencies:** Helps the model understand timing of events.
- **Example:** Recognizing that interruptions in the evening often require specific responses.
- **Generalization:** Model adapts to events occurring at times not seen during training.

Why sine functions are being used in LEM for Time Modeling:

- **No Additional Parameters:** Calculated using fixed formulas.
- **Smooth Transitions:** Continuous functions represent gradual changes over time.
- **Compatibility with Attention:** Facilitates computing attention scores based on position.

Multi-Head Attention in LEM's Transformer

Role in the Large Event Model (LEM):

- Processes combined input features to predict appliance settings.
- Input features include:
  - Interruption embedding ($\mathbf{e}_I$)
  - Time embedding ($\mathbf{e}_t$)
  - Knowledge Graph embeddings ($\mathbf{e}_G$)

Why Use Multi-Head Attention?

- Allows the model to focus on different aspects of the input simultaneously.
- Captures complex relationships and dependencies.
- Enhances the model's ability to make accurate predictions.

Multi-Head Attention in LEM's Transformer

Role in the Large Event Model (LEM):

- Processes combined input features to predict appliance settings.
- Input features include:
  - Interruption embedding ($\mathbf{e}_I$)
  - Time embedding ($\mathbf{e}_t$)
  - Knowledge Graph embeddings ($\mathbf{e}_G$)

Why Use Multi-Head Attention?

- Allows the model to focus on different aspects of the input simultaneously.
- Captures complex relationships and dependencies.
- Enhances the model's ability to make accurate predictions.

Input Embeddings in LEM

Combined Input Sequence:

$$\mathbf{S} = [\mathbf{e}_I, \mathbf{e}_t, \mathbf{e}_{G1}, \mathbf{e}_{G2}, \ldots, \mathbf{e}_{G_n}]$$

$\mathbf{e}_I$(Interruption)	$\mathbf{e}_t$(Time)	$\mathbf{e}_{G1}$(Doorbell)	$\mathbf{e}_{G2}$(Activity)	..

- Each embedding represents a different component of the context.
- These embeddings are combined to form the input sequence for the Transformer.

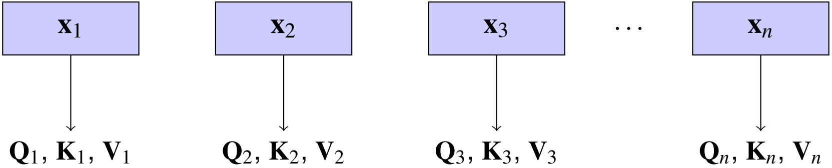
Multi-Head Attention: Processing the Inputs

Step 1: Linear Projections

- For each input embedding $\mathbf{x}_i \in \mathbf{S}$:

$$\mathbf{Q}_i = \mathbf{x}_i \mathbf{W}_Q \quad \text{(Query)}$$
$$\mathbf{K}_i = \mathbf{x}_i \mathbf{W}_K \quad \text{(Key)}$$
$$\mathbf{V}_i = \mathbf{x}_i \mathbf{W}_V \quad \text{(Value)}$$

- $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$: Projection matrices.



These projections enable the computation of attention scores.

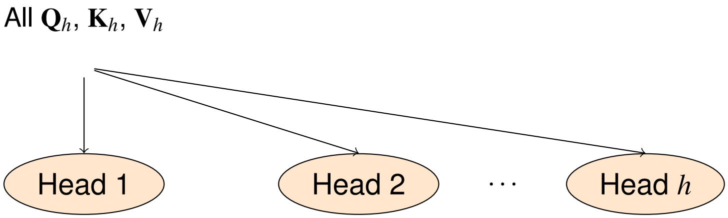
Multi-Head Attention: Computing Attention

Step 2: Scaled Dot-Product Attention

- For each head h, attention scores are computed:

$$\text{Attention}_h(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h) = \text{softmax}\left(\frac{\mathbf{Q}_h \mathbf{K}_h^\top}{\sqrt{d_k}}\right)\mathbf{V}_h$$

- d_k: Dimension of the Key vectors.



- Multiple attention heads operate in parallel.
- Each head computes attention using its own projections.

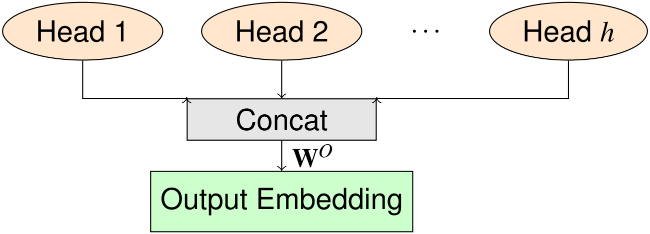
Multi-Head Attention: Combining Outputs

Step 3: Concatenation and Final Linear Layer

- Outputs from all heads are concatenated:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$

- $\mathbf{W}^O$: Output projection matrix.



- The concatenated vector is projected to the final output embedding.
- This embedding is used for predicting appliance settings.

Advantages of Multi-Head Attention in LEM

Contextual Understanding:

- Simultaneously captures various relationships and dependencies.
- Considers interruption type, time, and participant interactions.

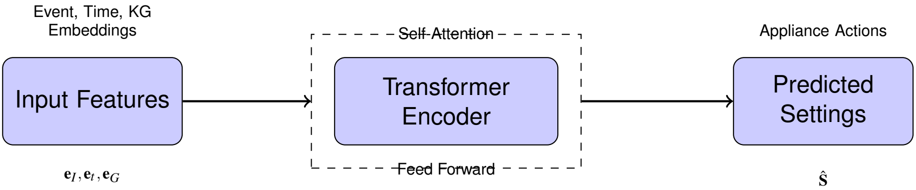
Focus on Relevant Features:

- Different heads can specialize in different features.
- Improves the model's ability to attend to important information.

Improved Predictions:

- Enhances the capacity to model complex patterns.
- Leads to more accurate and context-aware appliance settings.

Visualizing the Process



Key Components:

- **Input Features:** Combine interruption, time, and KG embeddings.
- **Transformer Encoder:** Processes input sequence with attention mechanisms.
- **Output:** Predicts the settings for each appliance.

Multi-Head Attention

- Multi-head attention is integral to the Transformer's ability to process complex input sequences.
- In LEM, it allows the model to consider multiple aspects of events and contexts simultaneously.
- This results in more accurate predictions for smart home appliance settings.

Scaled Dot-Product Attention

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Multi-Head Attention

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$
$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Model Dimensionality (d) in LEM

What is Model Dimensionality?

- d represents the number of dimensions in embedding vectors.
- Determines the size of vectors used to represent inputs and positional encodings.

Role in the Model:

- **Input Embeddings:** Events, time, participants, etc., are encoded as vectors of size d.
- **Positional Encodings:** Sinusoidal functions generate position-specific vectors of size d.
- **Combination:** Ensures embeddings and positional encodings can be combined (e.g., added together).

Impact on LEM:

- **Expressiveness:** Higher d allows capturing more complex patterns and relationships.
- **Model Capacity:** Influences the ability to learn and generalize from data.
- **Computational Cost:** Larger d increases memory usage and computation time.

Example Values:

- Common choices for d include 128, 256, 512, or 1024.

Transformer Encoder Layer

1. Multi-head attention sublayer with residual connection:

$$\mathbf{Z}_1 = \text{LayerNorm}(\mathbf{X} + \text{MultiHead}(\mathbf{X}, \mathbf{X}, \mathbf{X}))$$

2. Position-wise feed-forward network with residual connection:

$$\mathbf{Z}_2 = \text{LayerNorm}(\mathbf{Z}_1 + \text{FFN}(\mathbf{Z}_1))$$

Feed-Forward Network (FFN) in LEM's Transformer

Role in LEM:

- After processing input embeddings (interruption, time, KG embeddings) through attention mechanisms, the Transformer applies an FFN to each position.
- The FFN introduces non-linearity, allowing the model to learn complex patterns.

Structure of FFN:

$$\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

Key Points:

- Applied independently to each position in the sequence.
- Comprises two linear transformations with a ReLU activation in between.
- Enhances the model's ability to capture non-linear relationships.

Example Scenario for FFN Processing

Context:

- **Interruption:** Doorbell rings during family movie night.
- **Activity:** Watching Movie in the Living Room.
- **Participants:** Alice, Bob, Charlie.
- **Time:** 7:30 PM.

Input Embeddings:

- $\mathbf{e}_I$: Interruption embedding (Doorbell).
- $\mathbf{e}_t$: Time embedding.
- $\mathbf{e}_{G1}$: KG embedding for Doorbell node.
- $\mathbf{e}_{G2}$: KG embedding for Watching Movie activity.
- $\mathbf{e}_{G3}, \mathbf{e}_{G4}, \mathbf{e}_{G5}$: KG embeddings for participants (Alice, Bob, Charlie).

Processing Flow:

1. Input embeddings are processed through multi-head attention.
2. The outputs from attention are then passed through the FFN.

Input to the FFN

Output from Attention Mechanism:

- Let $\mathbf{h}_i$ be the output embedding at position i after the attention layer.
- For example, at the position corresponding to $\mathbf{e}_I$ (Interruption embedding), we have $\mathbf{h}_I$.

Example Embedding Vector:

$$\mathbf{h}_I = \begin{bmatrix} 0.6 \\ -0.2 \\ 0.3 \end{bmatrix}$$

Goal:

- Apply the FFN to $\mathbf{h}_I$ to obtain a transformed representation.
- This transformed embedding captures non-linear interactions and is used for prediction.

Calculations in the FFN

Given:

- **Input vector:** $\mathbf{h}_I = \begin{bmatrix} 0.6 \\ -0.2 \\ 0.3 \end{bmatrix}$
- **Weights and biases:**

$$\mathbf{W}_1 = \begin{bmatrix} 0.5 & -0.3 & 0.2 \\ -0.4 & 0.6 & -0.1 \\ 0.3 & -0.2 & 0.5 \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.05 \end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix} 0.2 & -0.5 & 0.3 \\ -0.3 & 0.4 & 0.6 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} -0.05 \\ 0.02 \end{bmatrix}$$

Step-by-Step Calculations:

1. Compute $\mathbf{z}_1 = \mathbf{h}_I \mathbf{W}_1 + \mathbf{b}_1$
2. Apply ReLU: $\mathbf{a}_1 = \text{ReLU}(\mathbf{z}_1)$
3. Compute output: $\mathbf{y} = \mathbf{a}_1 \mathbf{W}_2 + \mathbf{b}_2$

Detailed Calculations

1. Compute $\mathbf{z}_1$:

$$\mathbf{z}_1 = \mathbf{h}_I \mathbf{W}_1 + \mathbf{b}_1$$

$$\mathbf{z}_1 = \begin{bmatrix} 0.6 & -0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 0.5 & -0.3 & 0.2 \\ -0.4 & 0.6 & -0.1 \\ 0.3 & -0.2 & 0.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.1 \\ 0.05 \end{bmatrix}$$

$$\mathbf{z}_1 = \begin{bmatrix} 0.3 + 0.08 + 0.09 + 0.1 = 0.57 \\ -0.18 - 0.12 - 0.06 - 0.1 = -0.46 \\ 0.12 + 0.02 + 0.15 + 0.05 = 0.34 \end{bmatrix}$$

2. Apply ReLU:

$$\mathbf{a}_1 = \text{ReLU}(\mathbf{z}_1) = \begin{bmatrix} \text{ReLU}(0.57) \\ \text{ReLU}(-0.46) \\ \text{ReLU}(0.34) \end{bmatrix} = \begin{bmatrix} 0.57 \\ 0 \\ 0.34 \end{bmatrix}$$

Final Computation in the FFN

3. Compute Output $\mathbf{y}$:

$$\mathbf{y} = \mathbf{a}_1 \mathbf{W}_2 + \mathbf{b}_2$$

$$\mathbf{y} = \begin{bmatrix} 0.57 & 0 & 0.34 \end{bmatrix} \begin{bmatrix} 0.2 & -0.5 \\ -0.3 & 0.4 \\ 0.3 & 0.6 \end{bmatrix} + \begin{bmatrix} -0.05 \\ 0.02 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 0.114 + 0 + 0.102 - 0.05 = 0.166 \\ -0.285 + 0 + 0.204 + 0.02 = -0.061 \end{bmatrix}$$

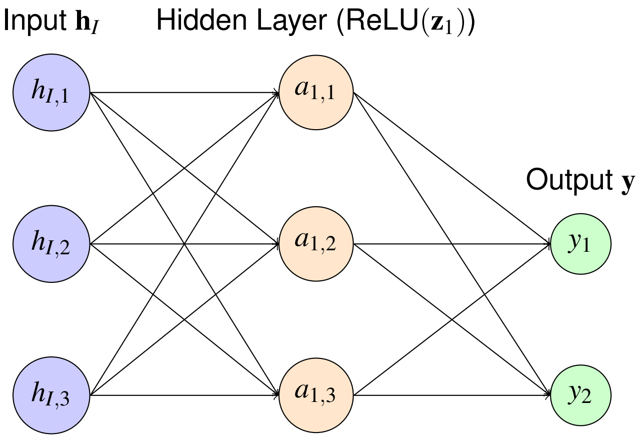
Result:

$$\mathbf{y} = \begin{bmatrix} 0.166 \\ -0.061 \end{bmatrix}$$

Interpretation:

- $\mathbf{y}$ is the transformed embedding at the position corresponding to the interruption.
- This embedding incorporates non-linear interactions and is used in subsequent layers or for prediction.

Visualization of the FFN Processing



- The FFN processes the embedding $\mathbf{h}_I$ through two layers.
- Non-linear activation (ReLU) is applied after the first layer.
- The final output $\mathbf{y}$ is a transformed representation used by the model.

Importance of FFN in LEM

Capturing Complex Patterns:

- The FFN allows the model to learn non-linear relationships between features.
- Enhances the ability to model interactions between interruptions, activities, and other contextual information.

Position-wise Processing:

- The FFN operates on each position independently.
- Preserves the sequence information while enabling parallel computation.

Overall Impact on LEM:

- Contributes to more accurate predictions of appliance settings.
- Helps the model understand how different factors combine to influence outcomes.

Summary of FFN Processing

Processing Steps:

1. **Input:** Embedding from the attention mechanism.
2. **First Linear Transformation:** $\mathbf{z}_1 = \mathbf{h}_I \mathbf{W}_1 + \mathbf{b}_1$
3. **Activation Function:** $\mathbf{a}_1 = \text{ReLU}(\mathbf{z}_1)$
4. **Second Linear Transformation:** $\mathbf{y} = \mathbf{a}_1 \mathbf{W}_2 + \mathbf{b}_2$

Key Takeaways:

- The FFN adds depth and non-linearity to the model.
- It operates efficiently by processing each position independently.
- Critical for capturing intricate patterns in the data.

Combining GNN and Transformer in LEM

Objective:

- Integrate Knowledge Graph (KG) embeddings obtained from a Graph Neural Network (GNN) with a Transformer model.
- Use combined embeddings to predict appropriate appliance settings in response to interruptions.

Process Overview:

1. Generate KG embeddings using a GNN (e.g., R-GCN).
2. Construct input sequence by combining interruption, time, and KG embeddings.
3. Use a Transformer to model dependencies and make predictions.

Benefits:

- Leverages relational information from the KG.
- Captures temporal and contextual dependencies.
- Enhances prediction accuracy by combining structural and sequential modeling.

Generating Knowledge Graph Embeddings

Knowledge Graph (KG):

- Nodes: Interruption, Activities, Participants, Appliances, Rooms
- Edges: Relationships (e.g., `participates in`, `uses`, `located in`, `affects`)

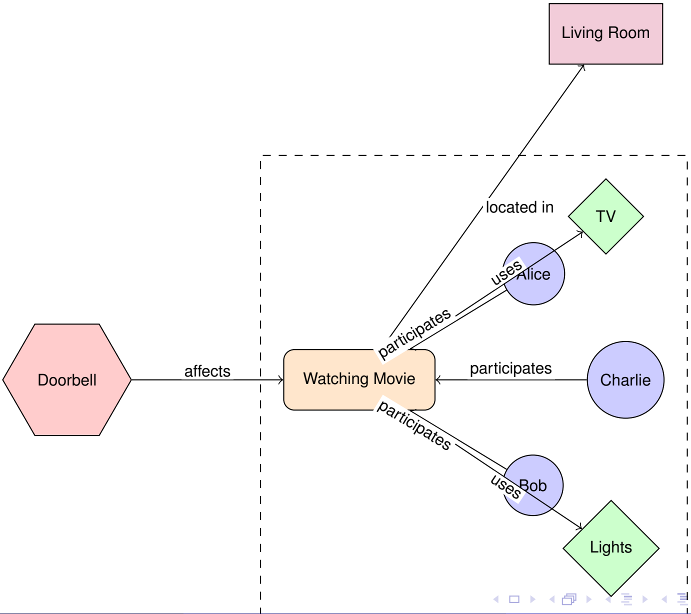
Using GNN to Obtain Embeddings:

- Apply Relational Graph Convolutional Network (R-GCN) to KG.
- For each node v, compute embedding $\mathbf{h}_v$ that captures its features and relationships.

Example Node Embeddings:

- $\mathbf{h}_{\text{Doorbell}}$: Embedding for the Doorbell node.
- $\mathbf{h}_{\text{Watching Movie}}$: Embedding for the activity.
- $\mathbf{h}_{\text{Alice}}, \mathbf{h}_{\text{Bob}}, \mathbf{h}_{\text{Charlie}}$: Embeddings for participants.

Visualization of Knowledge Graph Embeddings



Constructing the Input Sequence

Components of the Input Sequence:

- **Interruption Embedding ($\mathbf{e}_I$)**: Represents the doorbell interruption.
- **Time Embedding ($\mathbf{e}_t$)**: Encodes the time of the event (7:30 PM).
- **KG Embeddings ($\mathbf{e}_G$)**: Embeddings of relevant nodes from the KG.

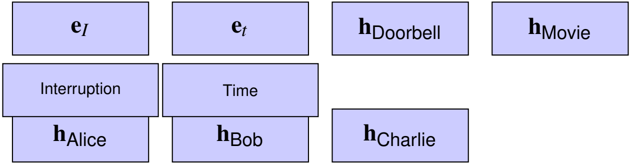
Combined Input Sequence:

$$\mathbf{S} = \left[\mathbf{e}_I, \mathbf{e}_t, \mathbf{h}_{\text{Doorbell}}, \mathbf{h}_{\text{Watching Movie}}, \mathbf{h}_{\text{Alice}}, \mathbf{h}_{\text{Bob}}, \mathbf{h}_{\text{Charlie}}\right]$$

Purpose:

- The Transformer will process this sequence to model dependencies.
- Each element provides different contextual information.

Visualization of the Input Sequence



- Sequence elements are processed by the Transformer.
- Positional encodings are added to maintain order information.

How GNN and Transformer Work Together

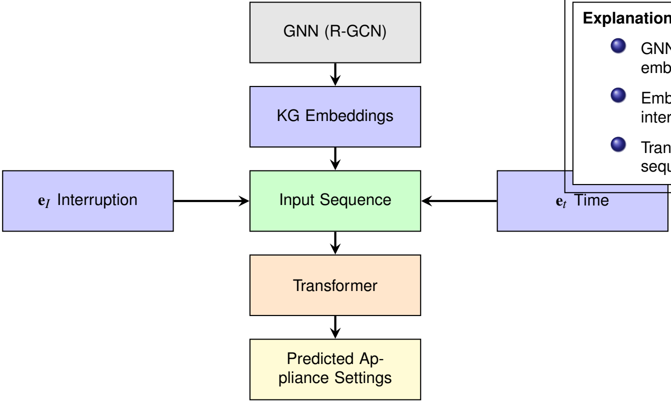
Step-by-Step Integration:

1. **GNN Processing:**
   - KG is processed by the GNN to obtain node embeddings.
   - Embeddings capture relational and structural information.
2. **Sequence Construction:**
   - Combine interruption and time embeddings with KG embeddings.
   - Form a sequence suitable for the Transformer.
3. **Transformer Processing:**
   - The Transformer models dependencies within the sequence.
   - Attention mechanisms focus on relevant elements.
4. **Prediction:**
   - Transformer outputs are used to predict appliance settings.

Key Point:

- The GNN enriches the input to the Transformer with relational context.

Visualization of the Integrated Model



Explanation:

- GNN processes KG for embeddings
- Embeddings combined with interruption and time info
- Transformer uses enriched sequence for predictions

Advantages of Combining GNN and Transformer

Rich Contextual Representation:

- GNN captures structural and relational information from the KG.
- Transformer models sequential dependencies and attention mechanisms.

Improved Prediction Accuracy:

- Combining both models leverages strengths of each.
- Provides a holistic understanding of the context.

Scalability and Flexibility:

- Can incorporate additional nodes and relations in the KG.
- Transformer handles varying sequence lengths and complex dependencies.

Example Prediction Using the Integrated Model

Input:

- Interruption embedding ($\mathbf{e}_I$) for Doorbell.
- Time embedding ($\mathbf{e}_t$) for 7:30 PM.
- KG embeddings for relevant nodes.

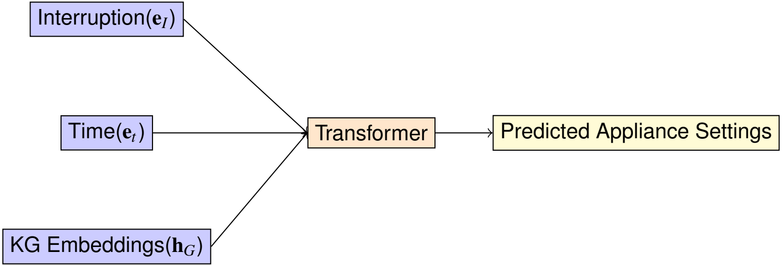
Transformer Output:

- Predicts appliance settings based on combined input.
- Example predictions:
  - $\hat{S}_{\text{TV}} = 0.1$ (Pause TV)
  - $\hat{S}_{\text{Lights}} = 0.7$ (Adjust Lights)
  - $\hat{S}_{\text{DoorLock}} = 0.9$ (Unlock Door)

Interpretation:

- The model understands that when the doorbell rings during a movie, it's appropriate to pause the TV, adjust the lights, and unlock the door.
- This understanding comes from both the relational context (GNN) and sequence modeling (Transformer).

Visualization of Prediction Flow



Flow Explanation:

- Inputs are fed into the Transformer.
- The Transformer processes the inputs to output predictions.
- The model effectively combines temporal, relational, and interruption data.

Mathematical Formulation

Overall Objective:

$$\hat{\mathrm{S}} = \sigma\left(\text{Transformer}\left([\mathbf{e}_I, \mathbf{e}_t, \mathbf{h}_G]\right)\mathbf{W}_S + \mathbf{b}_S\right)$$

Where:

- $\mathbf{e}_I$: Interruption embedding.
- $\mathbf{e}_t$: Time embedding.
- $\mathbf{h}_G$: KG embeddings from GNN.
- σ: Activation function (e.g., sigmoid).
- $\mathbf{W}_S, \mathbf{b}_S$: Weights and biases for the output layer.

Explanation:

- The Transformer processes the combined input sequence to produce an output vector.
- The output vector is transformed into predicted appliance settings using a linear layer and activation function.

Final Output Layer

- Transformer outputs $\mathbf{O} \in \mathbb{R}^{d_{\text{model}}}$.
- Apply a linear layer and sigmoid activation for binary appliance settings:

$$\hat{\mathbf{S}} = \sigma(\mathbf{O}\mathbf{W}_S + \mathbf{b}_S)$$

- $\hat{\mathbf{S}} \in [0,1]^n$, where n is the number of appliances.

Loss Function

- **Goal: Minimize loss to improve prediction accuracy.**
- Use Binary Cross-Entropy Loss:

$$\mathscr{L} = -\sum_{i=1}^{n} \left(S_i \log \hat{S}_i + (1 - S_i) \log(1 - \hat{S}_i) \right)$$

- S_i: True state of appliance i.

Loss Function and Training: Example

- **Actual Appliance Settings:**
  - TV: Paused ($S_{\text{TV}} = 0$),Lights: Dimmed ($S_{\text{Lights}} = 0.5$),Door Lock: Unlocked ($S_{\text{DoorLock}} = 1$)
- **Predicted Settings $\hat{\mathbf{S}}$:**

$$\hat{\mathbf{S}} = \begin{bmatrix} \hat{S}_{\text{TV}} \\ \hat{S}_{\text{Lights}} \\ \hat{S}_{\text{DoorLock}} \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.6 \\ 0.9 \end{bmatrix}$$

- **Compute Loss:**

$$\mathscr{L} = -\left(S_{\text{TV}} \log \hat{S}_{\text{TV}} + (1 - S_{\text{TV}}) \log(1 - \hat{S}_{\text{TV}}) + \dots\right)$$

- **Training Steps:**
  1. **Forward Pass:** Compute predictions using current model parameters.
  2. **Loss Computation:** Calculate the discrepancy between actual and predicted settings.
  3. **Backward Pass:** Compute gradients of loss w.r.t. model parameters.
  4. **Parameter Update:** Adjust parameters using Adam optimizer

Optimization

- Use gradient descent algorithms (e.g., Adam).
- Update model parameters to minimize $\mathscr{L}$.

Example Scenario: Doorbell Interrupts Movie Night

Scenario:

- **Time:** 7:30 PM
- **Activity:** Family is watching a movie in the **Living Room.**
- **Participants:** Alice (Parent), Bob (Parent), Charlie (Child)
- **Appliances in use:** TV (On), Lights (Dimmed)

Interruption:

- **Type:** Doorbell rings
- **Initiator:** Visitor
- **Affected Activity:** Watching Movie

Objective:

- Predict the appropriate appliance settings in response to the interruption.

Constructing Input Features (Part 1)

1. **Interruption Embedding ($\mathbf{e}_I$):**
   - **Interruption Type:** `Doorbell`
   - **Embedding Vector (Example):**

$$\mathbf{e}_I = \text{Embedding}(\texttt{Doorbell}) = \begin{bmatrix} 0.5 \\ -0.3 \\ 0.8 \\ \vdots \\ 0.1 \end{bmatrix} \in \mathbb{R}^{d_I}$$

   - Captures the characteristics of the interruption.

2. **Time Embedding ($\mathbf{e}_t$):**
   - **Time:** 7:30 PM ($t = 19.5$ hours)
   - **Cyclical Encoding:**

$$\text{Hour}_{\sin} = \sin\left(\frac{2\pi \times 19.5}{24}\right) \approx -0.9659$$

$$\text{Hour}_{\cos} = \cos\left(\frac{2\pi \times 19.5}{24}\right) \approx -0.2588$$

$$\mathbf{e}_t = \begin{bmatrix} -0.9659 \\ -0.2588 \end{bmatrix}$$

   - Models the cyclical nature of time (e.g., daily patterns).

Constructing Input Features (Part 2)

3. Knowledge Graph Embeddings ($\mathbf{e}_G$):

- **Nodes Involved:**
  - **Interruption Node:** `Doorbell`
  - **Activity Node:** `Watching Movie`
  - **Participant Nodes:** `Alice`, `Bob`, `Charlie`
- **Compute Embeddings Using R-GCN:**

$$
\begin{aligned}
\mathbf{h}_{\text{Doorbell}} &= \text{GNN}(\text{Doorbell}) \\
\mathbf{h}_{\text{Watching Movie}} &= \text{GNN}(\text{Watching Movie}) \\
\mathbf{h}_{\text{Alice}} &= \text{GNN}(\text{Alice}) \\
\mathbf{h}_{\text{Bob}} &= \text{GNN}(\text{Bob}) \\
\mathbf{h}_{\text{Charlie}} &= \text{GNN}(\text{Charlie})
\end{aligned}
$$

- Embeddings capture relationships and context.

4. Combined Input Sequence ($\mathbf{S}$):

$$
\mathbf{S} = \left[\mathbf{e}_I, \mathbf{e}_t, \mathbf{h}_{\text{Doorbell}}, \mathbf{h}_{\text{Watching Movie}}, \mathbf{h}_{\text{Alice}}, \mathbf{h}_{\text{Bob}}, \mathbf{h}_{\text{Charlie}}\right]
$$

Training and Loss Calculation

Actual Appliance Settings (S):

- TV: Paused ($S_{\text{TV}} = 0$)
- Lights: Brightened ($S_{\text{Lights}} = 1$)
- Door Lock: Unlocked ($S_{\text{DoorLock}} = 1$)

Loss Function: Binary Cross-Entropy

$$\mathscr{L} = -\sum_i \left(S_i \log \hat{S}_i + (1 - S_i)\log(1 - \hat{S}_i)\right)$$

Compute Loss (Example):

$$\begin{aligned}\mathscr{L} = -\Big(& S_{\text{TV}} \log \hat{S}_{\text{TV}} + (1 - S_{\text{TV}})\log(1 - \hat{S}_{\text{TV}}) \\ &+ S_{\text{Lights}} \log \hat{S}_{\text{Lights}} + (1 - S_{\text{Lights}})\log(1 - \hat{S}_{\text{Lights}}) \\ &+ S_{\text{DoorLock}} \log \hat{S}_{\text{DoorLock}} + (1 - S_{\text{DoorLock}})\log(1 - \hat{S}_{\text{DoorLock}})\Big)\end{aligned}$$

Objective: Minimize $\mathscr{L}$ to improve model predictions.

Where will get the training data?

We need to generate synthetic data

Synthetic Data Generation for LEM Training

Why Synthetic Data?

- **Limited Real Data:** Collecting real-world smart home data can be challenging due to privacy concerns and logistical constraints.
- **Controlled Environment:** Synthetic data allows us to create diverse scenarios systematically.
- **Scalability:** Enables generation of large datasets necessary for training deep learning models.

Objectives:

- Simulate realistic smart home environments and events.
- Include various interruptions, activities, and participant interactions.
- Generate data suitable for training and evaluating the LEM.

Data Generation Overview

Purpose of Data Generation:

- Create a comprehensive dataset to train the Large Event Model (LEM).
- Simulate diverse scenarios in a smart home environment.

Key Considerations:

- Number of data points required.
- Variety of personas and activities.
- Alignment with model training requirements.

Components of the Synthetic Data

Key Elements to Simulate:

- **Participants:** Individuals in the household (e.g., Alice, Bob, Charlie).
- **Activities:** Common household activities (e.g., watching TV, cooking).
- **Interruptions:** Events that disrupt activities (e.g., doorbell, phone call).
- **Appliances:** Devices used during activities (e.g., TV, lights, coffee maker).
- **Time Information:** Timestamps for activities and events.
- **Knowledge Graph Structure:** Relationships between participants, activities, appliances, and interruptions.

Data Formats:

- **Structured Data:** Tabular data for events and states.
- **Graph Data:** Representing the Knowledge Graph for relational context.

Number of Data Points to Generate

Determining Dataset Size:

- **Model Complexity:** Deep learning models like LEM require large datasets to generalize well.
- **Goal:** Generate sufficient data to cover diverse scenarios and reduce overfitting.

Estimated Data Points:

- **Baseline:** Start with 100,000 data points.
- **Scaling Up:** Potentially increase to 1 million data points for extensive coverage.

Data Point Definition:

- Each data point represents a unique event or scenario:
  - Specific interruption.
  - Time stamp.
  - Participant involvement.
  - Appliance settings before and after the event.

Types of Personas to Consider

Diversity in Personas:

- Simulate various household members with different characteristics.
- Include demographic diversity: age, occupation, preferences.

Example Personas:

1. **Parents:**
   - Working professionals.
   - Varying work schedules (e.g., 9-5 jobs, shift work).
2. **Children:**
   - Different age groups (toddlers, school-aged, teenagers).
   - School and extracurricular activities.
3. **Elderly Members:**
   - Retirees with specific routines.
   - Potential mobility considerations.
4. **Housemates:**
   - Young adults sharing a home.
   - Diverse schedules and habits.

Activities and Interruptions

Variety of Activities:

- Daily routines (e.g., cooking, cleaning).
- Leisure activities (e.g., watching TV, gaming).
- Work-related tasks (e.g., conference calls).

Types of Interruptions:

- **External:** Doorbell, phone calls, deliveries.
- **Internal:** Appliance malfunctions, power outages.
- **Emergency:** Fire alarms, medical alerts.

Combining Personas and Activities:

- Map personas to activities based on profiles.
- Simulate interruptions relevant to each activity and persona.

Methodology for Data Generation

Step 1: Define Scenarios and Activities

- List common activities and routines in a household.
- Define possible interruptions and their characteristics.

Step 2: Create Synthetic Participants

- Generate participant profiles with attributes (e.g., age, occupation).
- Assign roles and preferences to influence activity participation.

Step 3: Simulate Daily Schedules

- Use probabilistic models to assign activities over time.
- Incorporate variability to mimic real-life unpredictability.

Methodology for Data Generation (Continued)

Step 4: Introduce Interruptions

- Define interruption types and their likelihood at different times.
- Randomly insert interruptions into activity timelines.

Step 5: Generate Appliance Usage Data

- Map activities to required appliances and their settings.
- Simulate changes in appliance states due to interruptions.

Step 6: Construct the Knowledge Graph

- Represent entities (participants, activities, appliances) as nodes.
- Define edges based on relationships and interactions.

Example Scenario for Data Generation

Participants:

- **Alice:** Parent, works from home.
- **Bob:** Parent, leaves for office.
- **Charlie:** Child, attends school.

Daily Activities:

- Morning routines, school drop-off, work, leisure time.
- Evening activities include family dinner, watching TV.

Possible Interruptions:

- Doorbell rings (e.g., delivery, visitor).
- Phone calls.
- Power outages.

Generating Synthetic Data (Example)

Simulating an Evening Scenario:

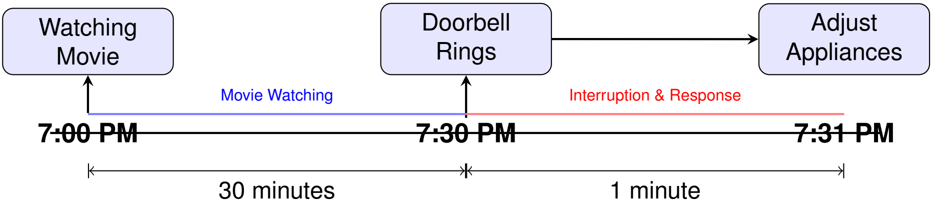
- **Time:** 7:00 PM - Family starts watching a movie.
- **Participants:** Alice, Bob, Charlie.
- **Appliances:** TV (On), Lights (Dimmed), Sound System (On).

Introducing an Interruption:

- **Time:** 7:30 PM - Doorbell rings.
- **Interruption Type:** Visitor arrives.
- **Appliance Adjustments:**
  - Pause TV.
  - Adjust lights (e.g., brighten).
  - Unlock door.

Visualization of Synthetic Event

Event Timeline:



Appliance State Changes:

- **TV:** From ”Playing” to ”Paused”.
- **Lights:** From ”Dimmed” to ”Bright”.
- **Door Lock:** From ”Locked” to ”Unlocked”.

Feeding the Dataset for Training

Data Preparation:

- Organize data into training, validation, and test sets.
- Ensure sets are representative of the overall data distribution.

Data Format:

- **Structured Input:** Encoded sequences of embeddings.
- **Labels/Targets:** Desired appliance settings or actions.

Batch Processing:

- Use mini-batches for efficient training.
- Shuffle data to ensure randomness.

Data Feeding Pipeline:

- Utilize data loaders to feed data into the model.
- Implement pre-processing steps (e.g., normalization, padding).

Training Process

Feedback Data into the Model:

- Input sequences are passed through the Transformer.
- Model learns to map inputs to appliance setting probabilities.

Optimization:

- Use optimization algorithms like Adam or SGD.
- Adjust model parameters to minimize the loss function.

Evaluation Metrics:

- Monitor accuracy, precision, recall, F1-score.
- Use validation set to prevent overfitting.

Building the Knowledge Graph

Nodes:

- Participants: Alice, Bob, Charlie.
- Activity: Watching Movie.
- Interruption: Doorbell.
- Appliances: TV, Lights, Door Lock.

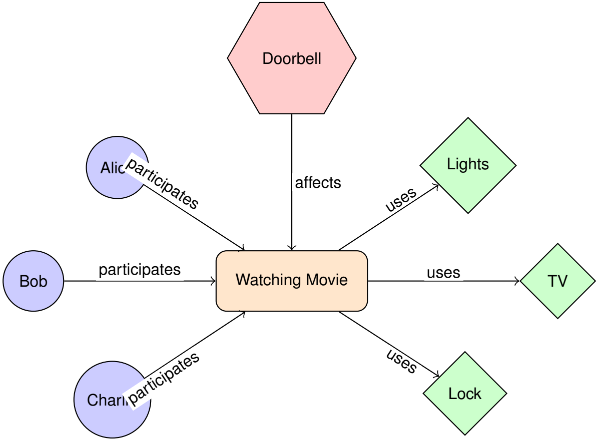
Edges:

- **participates in:** Links participants to the activity.
- **uses:** Connects activity to appliances.
- **affects:** Interruption affects the activity.
- **controls:** Participants control appliances.

Graph Construction:

- Represent these entities and relationships in the KG.
- Use the KG for generating embeddings via the GNN.

KG Created: GNN will use this to generate context-aware embeddings



Comparison to Bag-of-Words Models

Bag-of-Words (BoW) Characteristics:

- Represents text as unordered collection of words.
- Ignores word order and context.

LEM's Approach:

- Maintains sequence order and positional context.
- Uses embeddings that capture relationships and interactions.

Advantages Over BoW:

- Better captures complex dependencies.
- More suitable for modeling sequences in smart home events.

Example Input for Training the LEM

Structure of the Input:

- **Sequence of Embeddings:**
  - Interruption Embedding ($\mathbf{e}_I$)
  - Time Embedding ($\mathbf{e}_t$)
  - Knowledge Graph Embeddings ($\mathbf{h}_G$)
- **Positional Encodings:** Added to each embedding to maintain order.

Example Input Sequence:

$$\mathbf{S} = \left[\mathbf{e}_I, \mathbf{e}_t, \mathbf{h}_{\text{Doorbell}}, \mathbf{h}_{\text{Watching Movie}}, \mathbf{h}_{\text{Alice}}, \mathbf{h}_{\text{Bob}}, \mathbf{h}_{\text{Charlie}}\right]$$

- $\mathbf{e}_I$: Embedding for the interruption (Doorbell rings)
- $\mathbf{e}_t$: Embedding for the time (7:30 PM)
- $\mathbf{h}_{\text{Doorbell}}$: KG embedding for the Doorbell node
- $\mathbf{h}_{\text{Watching Movie}}$: KG embedding for the activity
- $\mathbf{h}_{\text{Alice}}, \mathbf{h}_{\text{Bob}}, \mathbf{h}_{\text{Charlie}}$: KG embeddings for participants

Feeding into the Model

- The input sequence $\mathbf{S}$ is fed into the Transformer model.
- Positional encodings are added to each embedding to preserve order.
- The model processes this sequence to predict appropriate appliance settings.

Example of Synthetically Generated Training Data

Structure of the Synthetic Data:

- Each data point represents an event or scenario in the smart home.
- Data includes features and labels: (1) **Features:** Input data to the model (2) **Labels:** Target appliance settings or actions.

Sample Data Point:

Feature	**Value**
Interruption Type	Doorbell Ringing
Time	7:30 PM
Participants	Alice, Bob, Charlie
Activity	Watching Movie
Room	Living Room
Appliances in Use	TV (On), Lights (Dimmed)
KG Relationships	As per KG structure
Label (Target)	
TV	Pause
Lights	Brighten
Door Lock	Unlock

Advantages of Synthetic Data Generation

Diversity of Scenarios:

- Ability to simulate a wide range of events and interactions.
- Enhance model robustness by including rare or edge cases.

Ethical Considerations:

- Avoids privacy issues associated with collecting real user data.
- Ensures compliance with data protection regulations.

Cost-Effectiveness:

- Reduces the need for expensive data collection efforts.
- Accelerates the development and testing process.

Question to explore: How can we merge this with other Samsung ST events data?

Why we need it:

- Ensure that synthetic data reflects realistic behaviors.
- Validate against known patterns and statistics.

Next Steps

Implementation:

- Develop scripts to automate data generation.
- Test the synthetic data for consistency and realism.

Model Training:

- Use the synthetic data to train the Large Event Model.
- Evaluate model performance and iterate as necessary.

Validation:

- Compare model predictions with expected outcomes.
- Adjust data generation parameters based on results.

Summary

- **Model Workflow:**
  1. **Input:** Interruption event (`Doorbell`), time (`7:30 PM`), KG embeddings.
  2. **GNN:** Generates embeddings capturing relationships in the KG.
  3. **Transformer:** Processes the combined input sequence to model dependencies.
  4. **Output:** Predicts appliance settings (e.g., pause TV, dim lights).

Input → GNN → Transformer → Output

- **Benefits:**
  - **Context-Aware Predictions:** Incorporates rich contextual information.
  - **Personalization:** Adapts to the specific household and occupants.
  - **Scalability:** Model can be extended with more data and relations.

What has been done so far?

- Identified public forums (Reddit, SmartThings community, App Store Reviews, etc.) to understand consumer painpoints, and what's missing in SmartThings.
- All TF-members have ensured they have access to the ML Platform for training and next-steps
- Identified research papers for potential activities, events.
- Developed a Monte-Carlo based script to generate synthetic datasets - 10,000 data points generated.