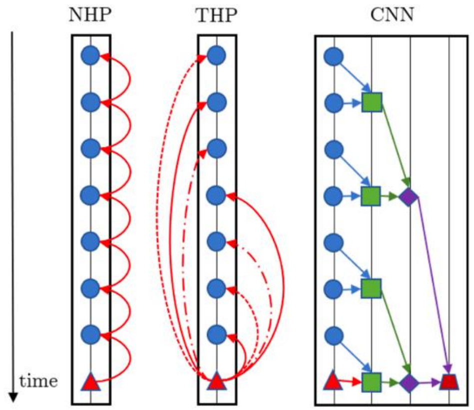
# Transformer Hawkes Process

Week 4: June 21st to June 27th

# Background

- Event sequence data:
  - time-stamped and labeled events with complex dependencies.
  - **Asynchronous**, with **variable time intervals**, unlike standard time series.




# Existing Methods & Limitations

- **Hawkes Process (HP):**
  - Assumes **all past events positively influence future ones**.
  - **Limitation**: Not all real-world events are mutually exciting (e.g., unrelated tweets); too strong assumption
- **Neural Hawkes Process (NHP) & RNN-based models:**
  - Improve on flexibility by learning representations via recurrent networks.
- **Two main limitations**:
  - **Weak long-term dependency modeling**: Information retention decays over time.
  - **Trainability**: Hard to train (vanishing/exploding gradients), and **not parallelizable** (sequential inputs required); Convolution-based models bring **inflexibility** in modeling events with **static/unnecessary dependencies**

# Proposed solution

- Transformer Hawkes Process (THP):
  - a **non-recurrent** model that integrates **Transformer-based self-attention** with **point process modeling (**borrows the **idea of history-dependent intensity,** but **not assume** your data must come from a Hawkes process**)**
  - models the **conditional intensity function** of temporal point processes using deep attention-based architecture
- Key components:
  - **Self-Attention Modules**: Captures both Short- and Long-Term Dependencies; Scales Efficiently
  - **Graph-Based Extension**: Structured-THP incorporates **relational graphs** among event generators (e.g., users), modeling similarity through learnable embeddings and matrix-based similarity.
  - Flexible for Continuous-Time Data: handles **asynchronous, irregular timestamps**

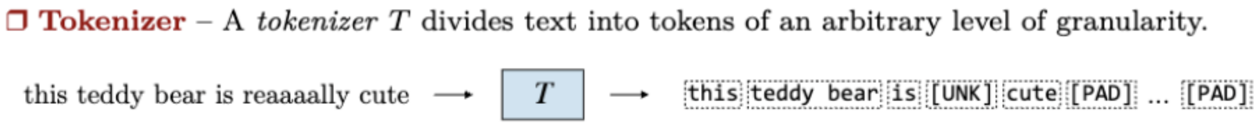
# Preliminaries

- **Hawkes Process:** $$\lambda(t) = \mu + \sum_{j:t_j<t} \psi(t - t_j).$$
  - **Intuition**: Past events **positively contribute** to the intensity of future events. The contribution typically decays over time via a kernel
  - **Limitation**: Assumes **only positive influence** from past events, too simplistic for many real-world scenarios where inhibition or complex patterns exist.
- **Neural Hawkes Process (NHP):**

$$\lambda(t) = \sum_{k=1}^{K} \lambda_k(t) = \sum_{k=1}^{K} f_k\left(\mathbf{w}_k^{\top}\mathbf{h}(t)\right), \quad t \in (0, T], \quad \text{where } f_k(x) = \beta_k \log\left(1 + \exp\left(\frac{x}{\beta_k}\right)\right).$$

- $K$: Number of event types.
- $\mathbf{h}(t)$: Hidden states from a continuous-time LSTM (CLSTM).
- $f_k(\cdot)$: A softplus variant ensuring positive intensities.

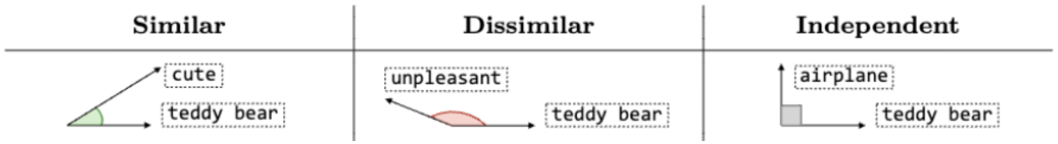
  - **Generalization**: Replaces the fixed kernel with a flexible, learned function using RNNs (specifically Continuous LSTM).
  - **Prediction**: The probability of event type k is computed as P[k_t=k]=λ_k(t)/λ(t).
  - **Limitation**: Inherits RNN weaknesses—difficulty in modeling **long-term dependencies** and **training instability** (e.g., gradient issues).
- **Transformer:**
  - attention-based architecture originally designed for NLP tasks
  - Key components
    - Token: invisible unit of text
      - Challenge:
      - In NLP:
        - Inputs (like words or subwords) are represented as **tokens** — discrete symbols indexed in a vocabulary.
        - Sequences are **regularly spaced** (e.g., each word occurs at step 1, 2, 3⋯).
        - Transformer inputs = **token embeddings** + **positional encodings**.
      - In event sequence modeling: Inputs are **events** that occur **irregularly** in continuous time.
        - A **type** (e.g. tweet, retweet, diagnosis) → like a token.
        - A **timestamp** → continuous, not uniformly spaced; instead of discrete positions



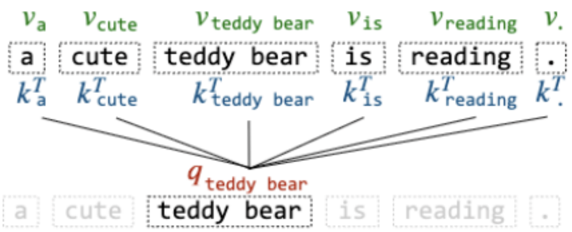
# Preliminaries

❒ **Similarity** – The *cosine similarity* between two tokens $t_1, t_2$ is quantified by:

$$\boxed{\text{similarity}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||} = \cos(\theta)} \in [-1, 1]$$

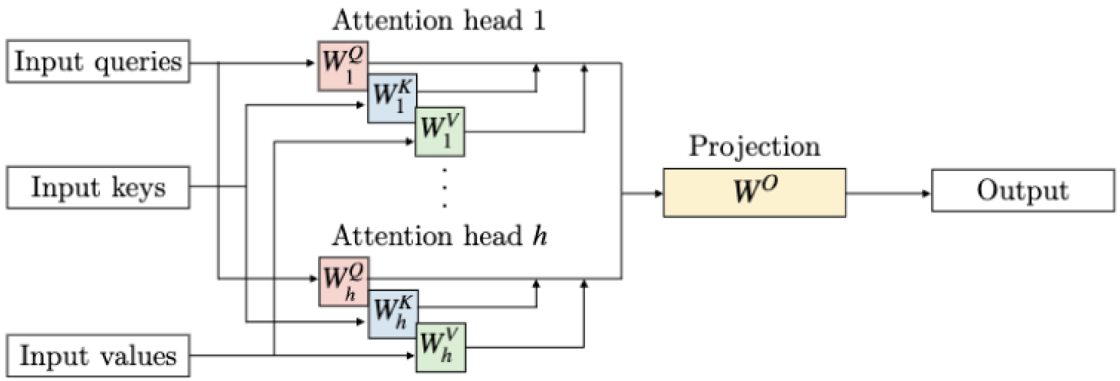



- **Transformer:**
  - Key components
    - **Token**: invisible unit of text
      - How to overcome: **replace/supplement positional encodings** with:
        - **Time encodings** (e.g., sinusoidal encodings of timestamps)
        - Or learnable functions like z(ti−tj) to model time difference directly in attention
    - **Input Embedding + Positional Encoding: "This type of event at this point in the sequence**."
      - **Transform input into embeddings:** $\mathbf{x}_i = \text{Embed}(w_i) + \text{PosEnc}(i)$
        - **Embedding:**
          - **Numerical vector representations** of the input elements (e.g. words, events, tokens, or features) in a continuous space.
          - Encode **semantic similarity**: similar inputs → similar embeddings.
        - **Positional encoding:** give the model a **sense of order** or **distance** between elements
    - **Multi-head self-attention**
      - **Core idea**: Each element (word, event, etc.) can attend to every other element in the sequence.
      - **Multi-head attention** runs several attention operations in parallel and concatenates the results



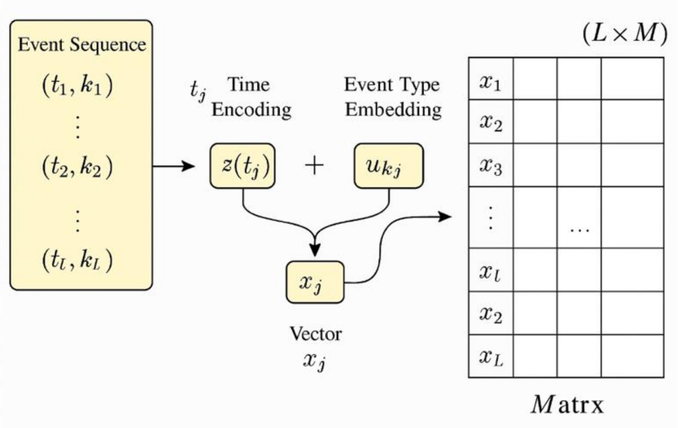

Attention can be efficiently computed using matrices $Q, K, V$ that contain queries $q$, keys $k$ and values $v$ respectively, along with the dimension $d_k$ of keys:

$$\boxed{\text{attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$$



# Model-THP

Objective: Design a neural model that effectively captures both short- and long-term dependencies in **continuous-time event sequences** using the **self-attention mechanism**, avoiding the limitations of RNN-based models.




- Data format:
  - Each event sequence is a list of event tuples (t_j,k_j)
- $t_j$: timestamp of event
- $k_j \in \{1, ..., K\}$: event type
- Input Encoding:
  - **Event Type Embedding** Uk_j:
    - representing the **semantic meaning of the event type**.
  - Temporal Encoding z(t_j): Uniform input processing
    - dot product of their sinusoidal encodings captures their **relative temporal distance**
    - decide **how far back in time** to look, and **how much** weight to assign to past events
    - answers "how long since the last event?" or "at what time did this occur?" injects when something happened so that sequence order and gaps matter.

$$[\mathbf{z}(t_j)]_i = \begin{cases} \cos\left(t_j/10000^{\frac{i-1}{M}}\right), & \text{if } i \text{ is odd,} \\ \sin\left(t_j/10000^{\frac{i}{M}}\right), & \text{if } i \text{ is even.} \end{cases}$$

  - Embedding of the event sequence S：

$$\boldsymbol{x}_j = \boldsymbol{U}\boldsymbol{k}_j + \boldsymbol{z}(t_j) \qquad \boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_L]^\top \in \mathbb{R}^{L \times M}$$

- Self-Attention Module:
  - Input X is transformed into Query Q, Key K, and Value V matrices
  - Designed to match and retrieve relevant parts

$$\mathbf{S} = \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{M_K}}\right)\mathbf{V}, \quad \text{where } \mathbf{Q} = \mathbf{X}\mathbf{W}^Q,\ \mathbf{K} = \mathbf{X}\mathbf{W}^K,\ \mathbf{V} = \mathbf{X}\mathbf{W}^V.$$

| | | |
|---|---|---|
| **Query (Q)** | What this current position/event is **trying to retrieve** | "What I need to know" |
| **Key (K):** discriminative | Encodes what each other position/event has to **offer** | "What I can provide" |
| **Value (V):** representational | The actual **information content** available at that event | "Here's what I'll contribute if chosen" |

# Model-THP

- Self-Attention Module:
  - Use **multi-head attention** for richer representations and apply **causal masking** to avoid looking into the future.

$$\mathbf{S} = \left[\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_H\right]\mathbf{W}^O,$$

- Feedforward Layers:
  - attention output is passed through position-wise feedforward layers to yield hidden representations H of the sequence

$$\mathbf{H} = \mathrm{ReLU}\left(\mathbf{S}\mathbf{W}_1^{\mathrm{FC}} + \mathbf{b}_1\right)\mathbf{W}_2^{\mathrm{FC}} + \mathbf{b}_2, \quad \mathbf{h}(t_j) = \mathbf{H}(j,:).$$

**discrete** hidden representations for the observed event times t_j

- Stacked Layers :
  - mimicking deep neural networks

# Model-Continuous Time Conditional Intensity

- Motivation:
  - Real-world TPP modeling requires **continuous intensity estimation** over any time t, not just event times.
  - interpolate between discrete timestamps to define a **continuous conditional intensity function;** does not rely on RNNs or fixed time steps.
- Conditional Intensity:

$$\lambda_k(t|\mathcal{H}_t) = f_k\Bigg(\underbrace{\alpha_k \frac{t - t_j}{t_j}}_{current} + \underbrace{\mathbf{w}_k^\top \mathbf{h}(t_j)}_{history} + \underbrace{b_k}_{base}\Bigg).$$

- $f_k(\cdot)$ is a **softplus** function $\beta_k \log(1 + \exp(\cdot/\beta_k))$, ensuring:
  - Positivity of intensity.
  - Smooth growth to prevent instability (gradient explosion).

  - **Current term**: captures influence as an **interpolation** between the two nearest event times t_j and t_{j+1}.
  - **History term:** scalar projection of hidden state, summarizing past influences.
  - **Base term**: background bias term; reflects base intensity without history.
- Prediction formula:
  - next time stamp prediction and next event type prediction:

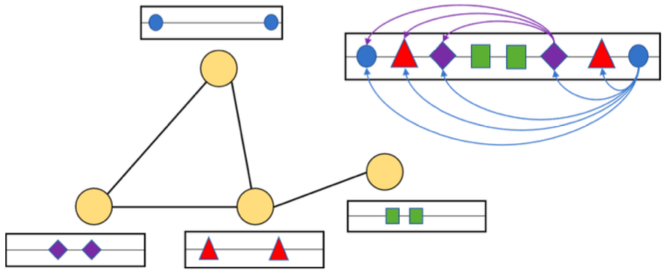
$$p(t|\mathcal{H}_t) = \lambda(t|\mathcal{H}_t)\exp\left(-\int_{t_j}^{t} \lambda(\tau|\mathcal{H}_\tau)d\tau\right),$$

$$\widehat{t}_{j+1} = \int_{t_j}^{\infty} t \cdot p(t|\mathcal{H}_t)dt, \quad \text{and } \widehat{k}_{j+1} = \operatorname*{argmax}_k \frac{\lambda_k(t_{j+1}|\mathcal{H}_{j+1})}{\lambda(t_{j+1}|\mathcal{H}_{j+1})}.$$

# Model-Training

- Objective:
  - maximizing the **log-likelihood** of observed event sequences over a time interval [t_1,t_L], based on the conditional intensity function λ(t | Ht)

$$\ell(\mathcal{S}) = \underbrace{\sum_{j=1}^{L} \log \lambda(t_j|\mathcal{H}_j)}_{\text{event log-likelihood}} - \underbrace{\int_{t_1}^{t_L} \lambda(t|\mathcal{H}_t)dt}_{\text{non-event log-likelihood}} .$$



- Components:
  - Event log-likelihood:
    - rewarding the model for correctly predicting event occurrences at observed times.
  - Non-event log-likelihood:
    - penalizing the model for assigning high intensity at times where no events occurred.
- Optimization: $\max \sum_{i=1}^{N} \ell(\mathcal{S}_i),$
- Challenge:
  - Intractable Integral of non event log likelihood: Monte Carlo (MC) Integration; Numerical Integration (e.g., Trapezoidal Rule)

# Model-Structured THP

- Motivation:
  - Instead of training a separate THP for each vertex (e.g., user, sensor, etc.), the authors propose using a *single shared THP model* to handle all point processes across vertices
  - heterogeneity across vertices is captured using **vertex embeddings**.
- Extended Input Representation: (t_j,k_j,v_j)
  - The input embedding X is extended to incorporate:

$$X = (\mathbf{U}Y + \mathbf{E}V + Z)^\top \Rightarrow \mathbf{x}_j = \underbrace{\mathbf{U} \cdot \text{type}}_{\text{event}} + \underbrace{\mathbf{E} \cdot \text{vertex}}_{\text{who}} + \underbrace{\mathbf{z}(t_j)}_{\text{when}}$$

- $t_j$: timestamp
- $k_j$: event type
- $v_j$: vertex index (indicating where the event occurs)

  - U: event type embeddings, E: vertex embeddings, Z: temporal encodings.

# Model-Structured THP

- Graph Attention Mechanism:
  - learn interactions between events occurring at different vertices.
  - Attention similarity matrix A: $A = (\mathbf{E}V)^\top \Omega (\mathbf{E}V)$
    - the entities have **related event behaviors** — they may co-evolve or influence each other
    - Ω∈R^{M×M} is a **learnable matrix:**
      - Learn **which dimensions are more informative**.
      - Capture **directional or asymmetric** similarities if Ω≠Ω^T
      - Incorporate **task-specific relational structure**

$$\mathbf{S} = \text{Softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{M_K}} + \mathbf{A}\right)\mathbf{V}_{\text{value}},$$

- Multi-head Graph Attention
- Updated Conditional Intensity Function

$$\lambda(t|\mathcal{H}_t) = \sum_{k=1}^{K}\sum_{v=1}^{|\mathcal{V}|} \lambda_{k,v}(t|\mathcal{H}_t)$$

- $k$: event type
- $v$: vertex (i.e., the "owner" of the point process)
- $\lambda_{k,v}(t|\mathcal{H}_t)$: intensity of observing **event type** $k$ on **vertex** $v$ at time $t$

$$\lambda_{k,v}(t|\mathcal{H}_t) = f_{k,v}\left(\alpha_{k,v} \cdot \frac{t - t_j}{t_j} + \mathbf{w}_{k,v}^\top \mathbf{h}(t) + b_{k,v}\right)$$

  - lets the model distinguish:
    - **same event type on different nodes**
    - **same node with different event types**

- Updated Optimization

$$\max \sum_{i=1}^{N} \ell(\mathcal{S}_i) + \mu L_{\text{graph}}(\mathbf{V}, \mathbf{\Omega}),$$

$$\boxed{L_{\text{graph}}(\mathbf{V}, \mathbf{\Omega})} = \boxed{\sum_{k=1}^{|\mathcal{V}|}\sum_{j=1}^{k} -\log\left(1 + \exp(\mathbf{V}_j \mathbf{\Omega} \mathbf{V}_k)\right)} + \boxed{\mathbf{1}\{(v_j, v_k) \in \mathcal{E}\}\left(\mathbf{V}_j \mathbf{\Omega} \mathbf{V}_k\right)}.$$

"soft-margin" **penalty: discourages** similarity
- If there's **no edge**, this term **stays** in the loss.
- If there's **an edge**, this term is **partially cancelled out**
- If s is **positive**, gradient is **negative** → gradient descent **lowers s**.
- If s is **negative**, gradient is **small** → s is pushed further negative, but slowly.

**adds back the similarity**, but **only if an edge exists**

$$\frac{d}{ds}\left[-\log(1 + \exp(s))\right] = -\frac{\exp(s)}{1 + \exp(s)} = -\sigma(s)$$

increases **monotonically** and **approaches 0** from below as s→+∞.