Experiment plan

Complicating the model

1) Adding attention into gnn
    a) ~~Replace SAGEConv with GATConv or HeteroConv with multihead attention and edge_dim so that the model itself weighs the importance of neighbors for different types of feedback.~~
    b) Use HGTConv (or HeteroConv with specialized TransformerConv on each edge) – an architecture specifically designed for heterogeneous (multi-typed) graphs, as proposed in the “Heterogeneous Graph Transformer” paper (Ziniu Hu).
    c) We can add temporality in another way by introducing a learnable temporal decay. To do this, we need to:
        1. ~~During data preparation, record the “age” of each edge, i.e. `Δt=current_time − event_time`.~~ - не дало результатов
        2. In `HeteroGNN.__init__`, define for each relation (or even for each attention head) a learnable parameter `λ`.
        3. ~~Replace SAGEConv with TemporalGATConv — a custom convolutional layer in which, after computing the attention coefficients, we multiply each `αij` by `exp(−λ·Δtij)`.~~ - tried, failed: difficult debugging + problems with launching heteroconv
2) Adding events to our THP
   Ideas for development (some can be combined, some separately):
    a) In THPEncoder, after the final normalization (self.final_norm), add a small MLP head that, from the output representation `x[:, -1, :]`, predicts the event (feedback category). In the loss function, in addition to the main ranking loss (BPR or BCE), add a cross-entropy term on feedback_logits. There are two ways to implement this:
        i) ~~Next-event prediction:~~
            ~~(1) Feed in the first L events (`event_type[:, :L]`).~~
            ~~(2) Train the feedback_head on the label `event_type[:, L]` (the next position).~~
        ii) Reconstruction-style: At each time step t, attempt to reconstruct `event_type[:, t]` from the context up to t (using teacher forcing).
    b) Inject the event-type embedding into the THP: We already have `event_type: LongTensor [B, L]`. Simply create `nn.Embedding(num_event_types, d_model)` and add it to `x = emb + pe + te`.
    c) For personalization, inject a static user embedding (from the graph component) into the THP.
    d) Add three kinds of temporal embeddings (HTP):

    i) Absolute time: from times, extract not just seconds since the epoch but also features like `hour_sin`, `hour_cos`, `day_of_week`, and even seasonality (month).
    ii) Relative interval to the previous event: compute `Δt_prev = times[i] - times[i-1]` and project it via `nn.Linear(1, d_model)`.
    iii) Relative interval to the next event: precompute `Δt_next = times[i+1] - times[i]` during data preparation (for the last event, fill with the average interval).

e) Learnable Hawkes kernel per head: currently the decay is fixed; make it a learnable parameter for each attention head.

f) Relative positional embeddings: Instead of absolute `pos_emb`, use the Shaw et al. scheme of `rel_pos_emb[i-j]` to better localize attention within the window.

g) Multi-head with different kernel functions. Assign different decay types to different heads:
    i) Exponential decay
    ii) Linear kernel `1/(1+λ·Δt)`
    iii) A small MLP over `Δt`

h) Temporal DropNet: For “old” positions (very distant events), randomly zero out their attention weights, forcing the model to focus on more recent events—analogous to DropToken but in time.

i) Contrastive loss inside the THP: take pairs of neighboring positions `(i,j=i+1)` and train the THP so their representations are closer to each other than to more distant events, while repelling negatives.

j) Layer-wise attention fusion: instead of only using the last THP layer’s output, concatenate or sum representations from all heads or from intermediate layers, with learnable weights.

k) Sub-interest modeling: inspired by interest clustering, introduce k prototype vectors as “attractor centers” of interests. For each session, compute its similarity to each center, letting the model focus on different “aspects” of preferences.

l) Future-masking as in NFARec / SasRec: to prevent peeking at future events when predicting the next one, add a subsequent mask alongside the existing padding mask.

m) Adaptive window (learnable window_mask): instead of a hard limit of “last L events,” learn to forget old events by defining an attention mask whose values are learned during training. The model itself learns how “important” each position in the history is (old vs. new). For example:

```
self.window_weights = nn.Parameter(torch.ones(max_len))
w_i = self.window_weights.unsqueeze(0).unsqueeze(0)  # for keys
w_j = self.window_weights.unsqueeze(0).unsqueeze(2)  # for
queries
win_mask = (w_i + w_j) / 2.0
scores = scores * win_mask
```

3) Adding temporal features/info to graph
   Calculate absolute features `(hour_min, hour_cost, day_of_week)` and relative deltas (the interval before and after the event) from the date field.
   a) Chen et al. (HTP) distinguish three types of information: absolute time (for example, seasonality or time of day), the relative interval between viewed items, and the relative interval until the next recommendation.
   b) The sub-interest concept (Pan et al. (the SINE model)): Users are divided into several "sub-interests" (prototypes), and omissions are used to adjust the boundaries of which aspect they dislike.
   c) KHGT (AAAI'22) – combines a graph encoder with time coding of interactions, reflecting the dynamics of multi-behavior.
4) (Optional) Port the model to the PyTorch Geometric Temporal extension.
5) Adaptive neighbor mask:
   Instead of building a full user graph (`build_full_graph`), introduce a learnable mask:
   `self.user_window = nn.Parameter(torch.ones(max_neighbors))`
   During user aggregation, multiply each attention weight by the corresponding entry in this window mask so that the model can "forget" less informative neighbors.
6) Adding different types of interactions and edges
   For example, you can create separate "like" and "dislike" relationships between users and products. Graph models will then aggregate information in different ways for different relationships.
7) Edge embedding vs edge type vs node type
   If the feedback has a numerical score or category, you can add vector features to edge_attr (edge feature). For example, weight or one-hot for "implicit_negative", "explicit_positive", etc. In GAT or HeteroConv, such signs affect the mechanism of attention.
   Examples:
   a) SIGformer: builds a signed graph and trains Transformer over it, capturing the global structure of both positive and negative connections.
   b) Pone-GNN: it uses two types of node-level embeddings (interest/disinterest) and contrastive pooling to extract discriminating information from negative feedback.
   c) NFARec: introduces "feedback-aware correlation", effectively setting a different message distribution path for opposing reviews in the hypergraph.
   d) KHGT: it simulates multi-type interactions (for example, viewing/liking/buying) in a single Graph Transformer, including time encoding (see below).
   Edge dropout by feedback type: introduce separate dropout probabilities for positive and negative edges so that the GNN does not overfit on overly repetitive "like" or "dislike" signals.
8) Learn and add Laplacian positional encodings: based on Dwivedi & Bresson's "Equivariant and Stable Positional Encoding for More Powerful Graph Neural Networks," as well as random-walk positional encodings for our graph.
9) Contrastive pre-training (GraphCL): generate two "views" of the graph (drop-edge, mask-node) and train the model to align the embeddings of the same node across these augmentations.

Training

1) Classification task (multilabel)
2) ~~Multitasking (thp and gnn with two different losses)~~
3) Pre-training:
    a) GraphCL: contrastive learning on augmented versions of your heterogeneous graph (drop-edge, mask-node) using an InfoNCE loss between the representations of the same nodes.
    b) DGI (Deep Graph Infomax): maximize mutual information between the global graph embedding and the local node embeddings.

Losses

1) Semi-hard bpr
2) LambdaRankLoss
3) LambdaLoss
4) TOP1-max
5) Softmax/InfoNCE
6) «Not-to-recommend» loss: Wang et al. (RecSys'23)
7) WARPLoss
8) Focal-BPR
9) ListMLE/ListNet

Datasets

1) ~~all datasets from NFARec~~

Baselines (for compare)

1) ~~ALS~~
2) ~~LightFM~~
3) DSSM
4) LigthGCN
5) ~~Our final model without emotion info~~