LUMOS: Large User MOdel Series

Building Foundation Models for User Behavior at Scale

Dhruv Nigam

March 4, 2026

Talk at Swiggy

Thank You, Swiggy

World class product. Great engineering culture.

What We'll Cover Today

1. **Systems & Scale** – brief overview (numbers, infra, pipeline)
2. **Scaling Laws** – power law relationships for user behavior models
3. **LLM Concepts Refresher** – tokenization, embeddings, attention
4. **LUMOS Architecture Deep Dive** – where we'll spend most time

Full details in our paper: **arXiv:2512.08957**

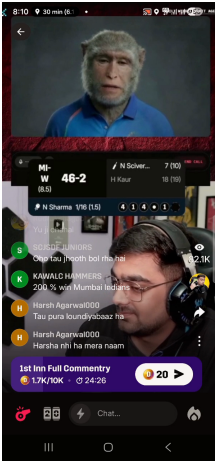
Is Dream11 Still Running?

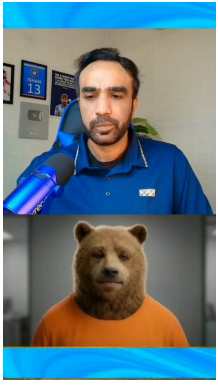
Yes! And we're now a **live streaming sports platform**!

Last India WC match: **15M users**, **360K concurrency**

If India reaches finals: Expecting **double concurrency**

AI Companion Demos





About Me

Currently @ Dream11:

- **AI Companion**: real-time sports AI, ~2s latency, 10M+ viewers
- **LUMOS**: 250M user foundation model (today's talk)
- **ForeCal**: probability calibration, published **KDD 2024**
- RL agent for discounts: **+2% WAU** uplift

Previously: Protium (12th employee, founding ML eng; DL models beat CIBIL by 12% AUC, $500M loans underwritten) and **CLSA** (quant trading, $1bn+/day sub-10ms; profitable stat-arb, Sharpe 1.3)

Education:

- **IIT Bombay** – B.Tech + M.Tech, Electrical Engineering (Signal Processing)

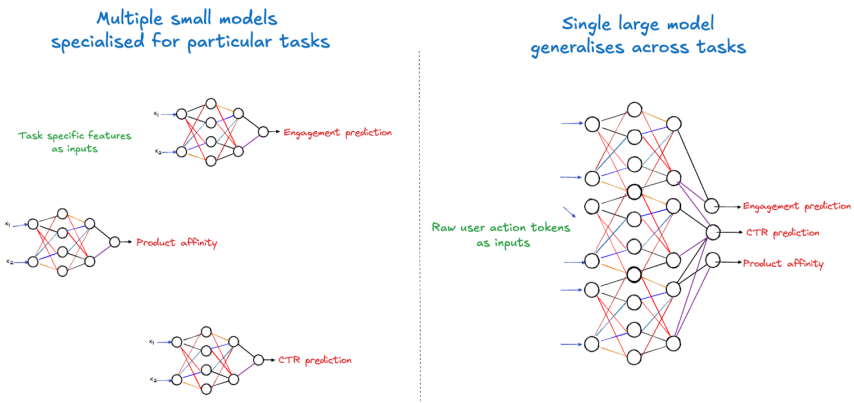
Open Source:

- **LiveKit Agents** (voice AI SDK)
- **veRL** (post-training RL, ByteDance)

Writing:

- **ML Trenches** – blog on deploying ML while staying sane

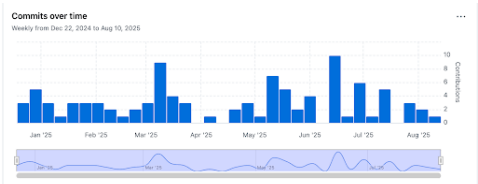
The Paradigm Shift



Many task-specific models → **One foundation model**

LUMOS: The Scale

Users:	**250M**
Data:	~**48 TB** processed
Tokens:	**1.7 trillion** activity tokens
Codebase:	**17,000+ LOC**
Models:	~**200 LOC (1%)**
New model:	<**10 LOC** to train
Inference:	~**50M users/day** scored



ML in production is 99% systems engineering

Training Infrastructure

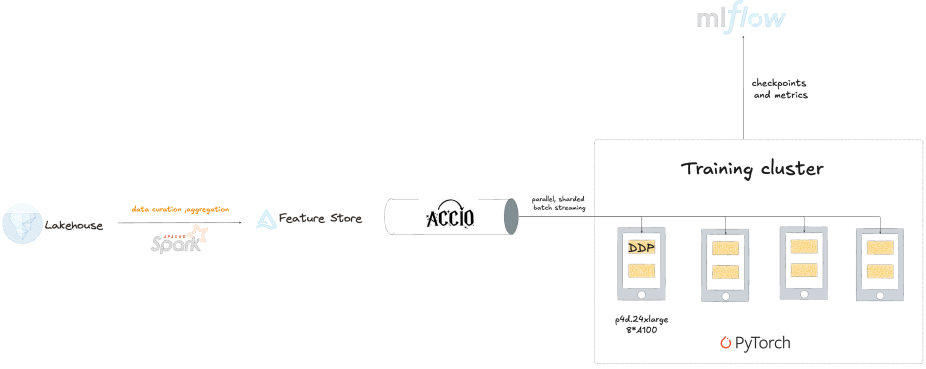
Hardware:

- **8× A100 40GB** (p4d.24xlarge)
- **576 A100 GPU-hours** (72h wall time)
- **BF16** mixed precision
- Batch size **4096** (512/GPU × 8)

Tech Stack:

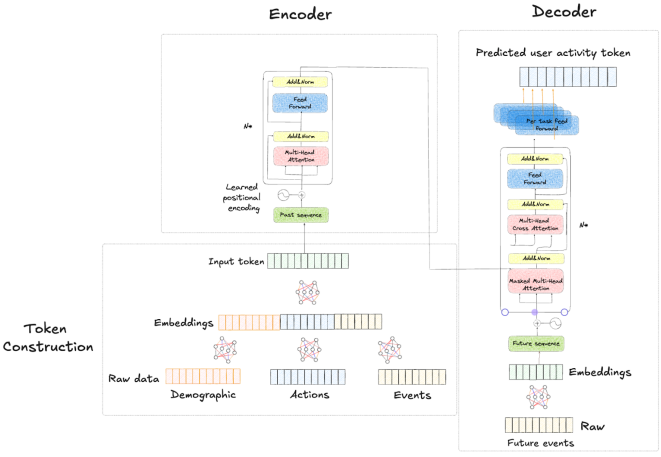
- **PySpark** – data processing
- **Delta Lake** – versioned storage
- **Accio** – custom streaming dataloader
- **PyTorch DDP** – distributed training
- **MLflow** + **Databricks**
- Migrating to **Ray Data** + **Ray Train**

Training Pipeline



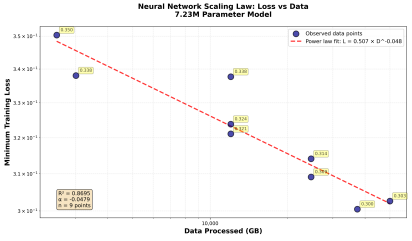
Raw Data → **PySpark** → **Delta Lake** → **Accio** → **DDP Training**

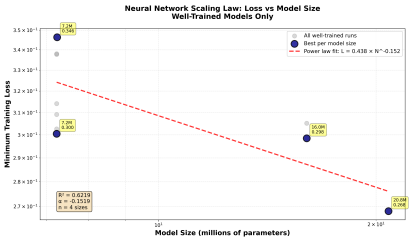
LUMOS Architecture



7.23M params · 6+6 layers · 8 heads · d_{model}=512 · SwiGLU · Pre-LayerNorm

Scaling Laws





Data scaling:

$L(D) = 0.507 \times D^{-0.048}$

Model scaling:

$L(N) = 0.438 \times N^{-0.152}$

(3× stronger than data)

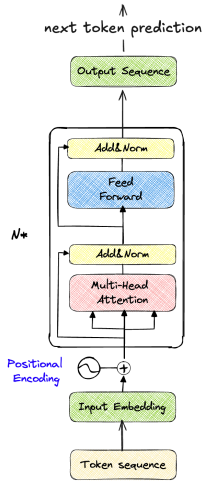
Config	Params	Loss
Small	7.23M	0.3347
Medium	12.18M	0.3298
Large	16.42M	0.3271
X-Large	**20.81M**	**0.3254**

LLM Architecture Foundation

Standard LLM Architecture: Decoder-Only

4 Core Components:

1. Tokenization
2. Token Embeddings
3. Positional Embeddings
4. Self-Attention



Tokenization & Embeddings

Tokenization: Text $\rightarrow$ discrete tokens via BPE/WordPiece

$$\text{Text} \rightarrow T = \{t_1, t_2, \ldots, t_n\}$$

Token Embeddings: One-hot $\rightarrow$ dense vectors via learnable matrix

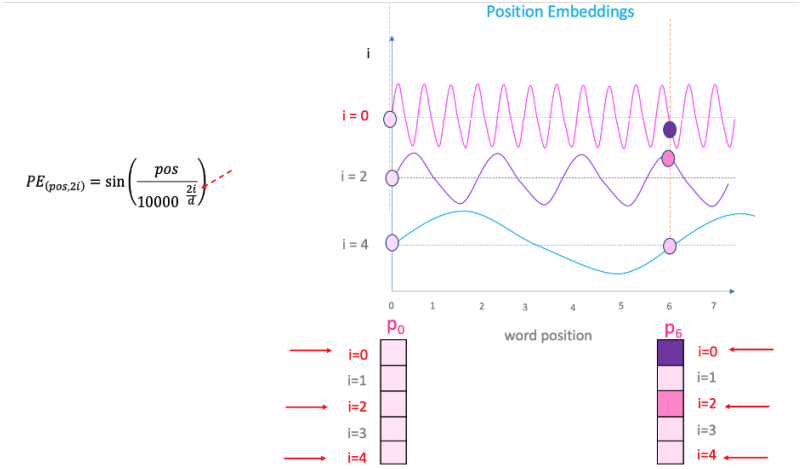
$$E = W_e \cdot t_i$$

Positional Embeddings: Inject sequence order (self-attention is permutation-invariant)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

Positional Embeddings: Illustration

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



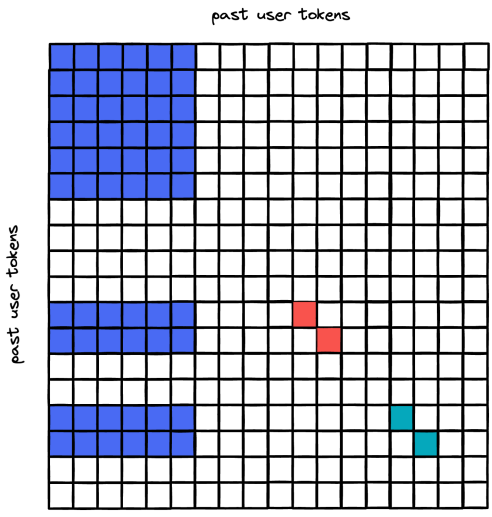
Self-Attention

Each token attends to all others.

Weights = similarity between query and keys.

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

$$z_i = \sum_{j=1}^{N} \alpha_{ij} v_j$$



LUMOS Architecture Deep Dive

LUMOS Tokenization: Multi-Modal Daily Tokens

User Behavior

13 Features

- Joins, deposits
- Withdrawals, wins
- Activity counts

Supply Data

32 Features

- Sports calendar
- Event counts
- Event timing

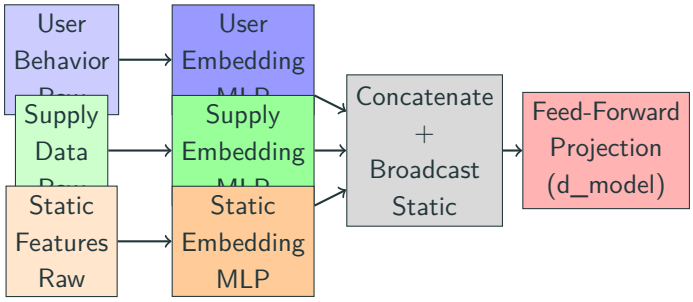
Static Features

9 Attributes

- Demographics
- Device info
- Registration data

Embed $\rightarrow$ Concatenate $\rightarrow$ Feed-Forward $\rightarrow$ **Rich Daily Token**

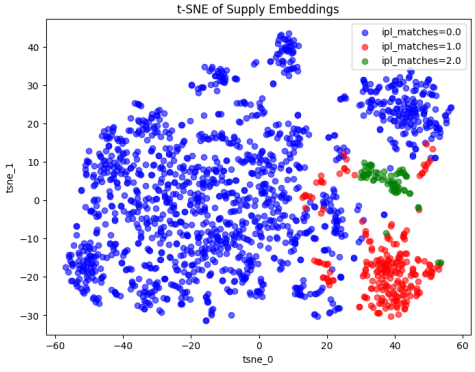
Token Construction Process



vs. LLMs

LLMs: Discrete text tokens from fixed vocabulary **LUMOS:** Continuous multi-modal daily vectors

Specialized Embeddings



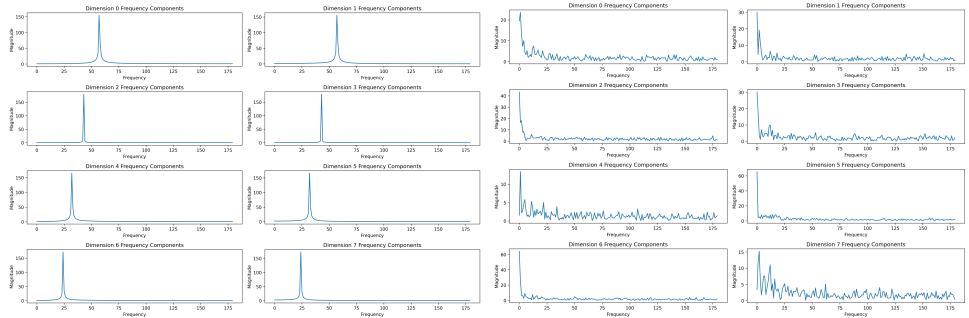
1. **Static User Embeddings**
   - New user personalization
   - Cold-start scenarios
2. **Supply Embeddings**
   - Supply similarity analysis
   - “What-if” scenarios

Positional Embeddings: Ablation Study

Method	Validation Loss	ROC-AUC
Learned	**0.3347**	**0.9296**
Sinusoidal	0.3372	0.9284
TAPE	0.3394	0.9275
Absolute	0.3556	0.9207

Learned embeddings win – adaptable to dataset-specific temporal patterns.

Sinusoidal vs. Learned Embeddings



Learned embeddings capture **lower frequencies** – precise positional encoding less critical for user behavior sequences.

Cross-Attention: The Key Innovation

Conditioning predictions on future known events

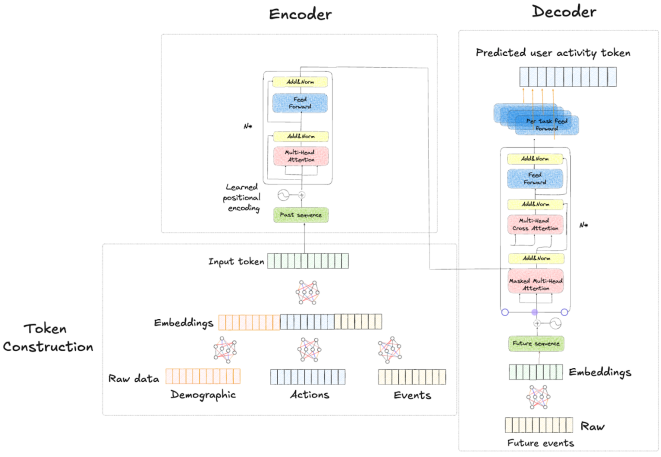
Self-Attention (Encoder):

- Historical context processing
- Information diffusion within sequence

Cross-Attention (Decoder):

- **Queries**: Future supply data
- **Keys/Values**: Historical user context
- **Innovation**: Condition on known future events

LUMOS Architecture (Recap)

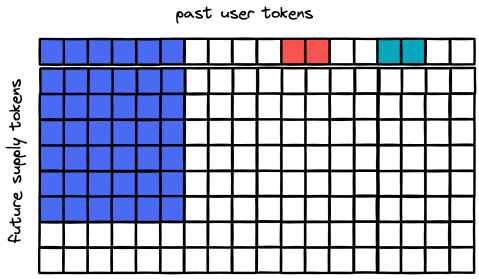


Encoder processes history via self-attention → Decoder uses cross-attention to condition on future supply → Masked prediction

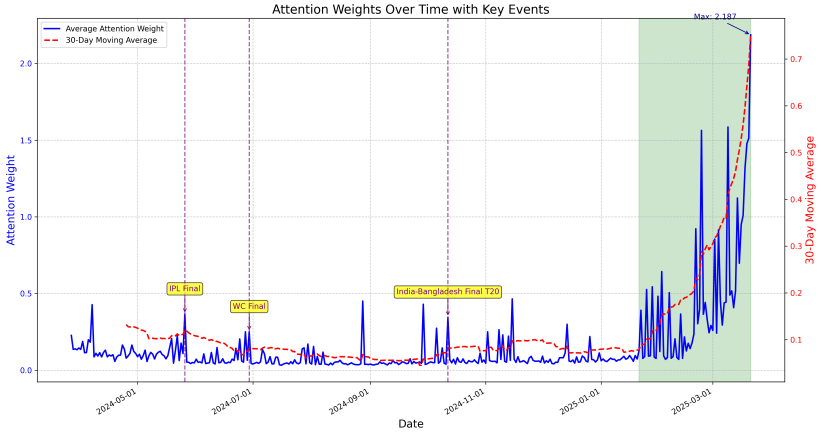
Visualizing Cross-Attention

Non-square matrix: Future Supply $\times$ History

- Dimensions: 28×360
- Rows = future supply queries
- Columns = historical user behavior
- Cell (i, j) = attention from future day i to past day j



Attention Pattern Analysis



Ablation Study: Role of Supply Data

Configuration	Validation Loss	ROC-AUC
Full Model	**0.3331**	**0.9300**
No Past Supply	0.3349	0.9297
No Future Supply	0.3350	0.9294
No Supply Data	0.3345	0.9298

Both past and future supply data improve predictions.

Beyond Next Token Prediction

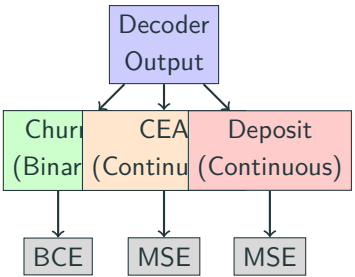
Masked Prediction:

- Predict **full cardinality** of user raw tokens daily
- Mirrors input transaction data

Multi-Task Learning:

- **Churn** (binary)
- **CEA** (continuous)
- **Deposit amount** (continuous)

Loss: MultiTaskLoss with uncertainty weighting



Uncertainty-Based Loss Function

$$\mathcal{L} = \sum_{i=1}^{d_u} \underbrace{\frac{1}{2\sigma_i^2}}_{\text{dynamic weight}} \mathcal{L}_i(\mathbf{U}_i^{fut}, \hat{\mathbf{U}}_i^{fut}) + \underbrace{\log \sigma_i^2}_{\text{regularizer}}$$

$\mathcal{L}_i$	Task-specific loss (BCE for binary, MSE for continuous)
σ_i^2	**Learnable** uncertainty per behavior dimension
$\frac{1}{2\sigma_i^2}$	High uncertainty $\rightarrow$ lower weight (auto-balancing)
$\log \sigma_i^2$	Prevents $\sigma_i^2 \rightarrow \infty$ (can't ignore any task)

Model jointly learns **predictions + task uncertainties**.
Hard tasks get lower weight automatically.

User Embeddings

Extracting User Embeddings: Key Considerations

1. **Extraction Point:**
   - Pre-Encoder vs. **Post-Encoder**
2. **Aggregation:**
   - Mean, Max, or Exp. Weighted Avg
3. **Recency (λ):**
   - Low λ = long-term patterns
   - High λ = recent behavior
4. **Training Objective:**
   - **Multi-Task** vs. Single-Task

User Embedding Performance

Model	Method (Extraction, λ)	Churn AUC	CEA Spear.	Pub. CEA Spear.	Net Win AUC	Withdraw AUC
Multi-Task	('post_encoder', 0)	0.9086	**0.3198**	0.4414	0.7121	0.9054
Multi-Task	('post_encoder', 0.1)	0.9087	0.3172	**0.4429**	0.7128	0.9059
Multi-Task	('post_encoder', 1)	0.9090	0.3047	0.4422	0.7141	0.9768
Multi-Task	**('post_encoder', 10)**	**0.9802**	0.1518	0.4403	**0.7219**	**0.9077**
Single-Task	('post_encoder', 0)	0.9063	0.3820	0.4107	0.6544	0.9735
Single-Task	('post_encoder', 0.1)	0.9063	0.3737	0.4125	0.6543	0.9736
Single-Task	('post_encoder', 1)	0.9067	0.3764	0.4087	0.6574	0.9740
Single-Task	('post_encoder', 10)	0.9077	0.3333	0.3931	0.6708	0.9048

Do You Need a Transformer?

Architecture	Parameters	Validation Loss
Transformer (LUMOS)	**2.49M**	**0.3347**
Large Deep NN	11.57M	0.3566
Small Deep NN	2.30M	0.3575

2.49M params beats 11.57M params. Transformers win.

Closing

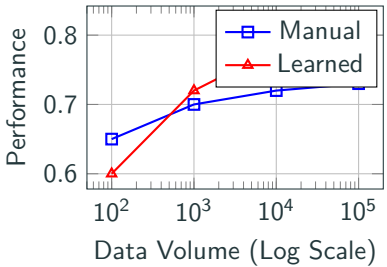
End Feature Engineering

Traditional:

- Manual feature engineering
- Domain expert required
- Brittle, hard to maintain

LUMOS:

- **Learned representations**
- End-to-end optimization
- Scales with data volume



Learned representations scale.
Manual features plateau.

Key Takeaways

1. **ML in production = 99% systems engineering**
   17K LOC platform, models are 1%
2. **Foundation models enable rapid iteration**
   $<$10 LOC to train a new model variant
3. **Scaling laws hold for user behavior**
   Power law relationships, model capacity $>$ data quantity
4. **One model to rule them all**
   250M users, 48TB, 576 GPU-hours $\rightarrow$ churn + CEA + deposits + embeddings

Thank You

Questions?

Online A/B Test Results

Experiment Design:

- Deployed via **Pacman** (intelligent offers engine)
- **700K users/cohort**, 18-day controlled test
- LUMOS predictions vs. legacy specialized models

Metric	Result
DAU Increase	**+3.15%**
Offer Cost Reduction	**-2.47%**
Avg AUC Improvement (offline)	**+0.025**
MAPE Reduction (offline)	**-4.6%**

More users, less cost. Offline gains translate to online impact.

Accio: Custom Streaming DataLoader

Problem: Dataset too large for memory + DDP needs balanced sharding.

Solution: Stream partition-by-partition from Delta Lake tables.

- PyTorch-compatible, designed for large sequence models on Delta tables
- Worker-aware sharding: each DDP process gets disjoint partitions
- No redundant data loading across workers
- Deterministic and balanced across all GPUs

Delta Tables $\rightarrow$ **Accio** $\rightarrow$ DDP Workers (8$\times$ A100)

Forward Pass: CrossAttentionForecaster

```
def forward(self, user_history,
            supply_history,
            static_features,
            future_supply):
  B, T_hist, _ = user_history.shape
  _, T_future, _ = future_supply.shape

  # 1) Encoder Input
  uh = self.user_embed(user_history)
  sh = self.supply_embed(supply_history)
  st = self.static_embed(
      static_features.unsqueeze(1))
  st = st.repeat(1, T_hist, 1)

  hist_concat = torch.cat(
      [uh, st, sh], dim=-1)
  hist_enc_in = self.hist_proj(hist_concat)
```

```
# Add positional embeddings
hist_pos = self.pos_embed.get_embedding(
    0, T_hist)
hist_enc_in = hist_enc_in + hist_pos

# Run encoder
enc_out = self.encoder(hist_enc_in)

# 2) Decoder Input (future supply)
fs = self.supply_embed(future_supply)
dec_in = self.future_proj(fs)
future_pos = self.pos_embed.get_embedding(
    T_hist, T_future)
dec_in = dec_in + future_pos

# 3) Cross-attention decoder
dec_out = self.decoder(
    x=dec_in, enc_out=enc_out)

# 4) Final projection
return self.out(dec_out)
```

User Embedding Extraction

```
def historical_user_embedding(self, user_hist_raw, supply_hist_raw, static_raw,
                              reduction="mean", exp_alpha=1.0,
                              embedding_stage="post_encoder"):
    B, T, D = user_hist_raw.shape
    uh = self.user_embed(user_hist_raw)
    sh = self.supply_embed(supply_hist_raw)
    st = self.static_embed(static_raw.unsqueeze(1)).repeat(1, T, 1)
    hist_concat = torch.cat([uh, st, sh], dim=-1)
    hist_enc_in = self.hist_proj(hist_concat)

    to_reduce = hist_enc_in
    if embedding_stage == "post_encoder":
        hist_pos = self.pos_embed.get_embedding(0, T)
        to_reduce = self.encoder(hist_enc_in + hist_pos)

    if reduction == "mean":     return to_reduce.mean(dim=1)
    elif reduction == "max":    return to_reduce.max(dim=1).values
    elif reduction == "expavg":
        t = torch.linspace(0, 1, T, device=to_reduce.device)
        weights = torch.softmax(exp_alpha * t, dim=0).unsqueeze(0).unsqueeze(-1)
        return (to_reduce * weights).sum(dim=1)
```