**# Neural Memory for Long-Context LMs — Complete Research + Lecture Roadmap**

## North Star

Design & evaluate \*\*trainable + external memory\*\* that:

- (a) scales sub-linearly with context

- (b) resists representation drift

- (c) lifts Indic long-context tasks

\*\*Ship:\*\* LLCE-Indic benchmark + reference code + paper + 11 lecture videos

---

# PHASE A — CORE READING (skim → re-derive → reproduce)

## A1. Attention Is All You Need (Transformer)

\*\*Paper:\*\* https://arxiv.org/abs/1706.03762

### Tasks:

- [ ] \*\*Skim map:\*\* List core equations (Q,K,V projections; softmax; residual; layernorm)

- [ ] \*\*Assignment:\*\* Write 10-bullet "how info flows" map

- [ ] \*\*DoD:\*\* Explain attention as differentiable associative memory in ≤90 seconds

- [ ] \*\*Re-derive:\*\* Scaled dot-product attention from first principles (cosine similarity + temperature)

- [ ] \*\*Assignment:\*\* Derive gradients wrt Q and K; note where saturation happens

- [ ] \*\*Artifact:\*\* 1-page derivation PDF

- [ ] \*\*Mini-repro:\*\* Train tiny char-LM (1-2 layers), plot attention entropy vs token distance

- [ ] \*\*Acceptance:\*\* Entropy decays with distance in baseline

---

## 🎥 LECTURE 1: Attention as Differentiable Memory (30min + 10min demo)

\*\*Deliver after:\*\* A1 complete

### Learning Goals:

1. Attention = content-addressable memory

2. Gradients, saturation, entropy behavior

3. Where retention fails with distance; how residual stream helps

### Outline:

1. Q/K/V geometry → associative lookup (7 min)

2. Saturation & temperature; entropy as "focus meter" (8 min)

3. Residual connections as gradient highways (5 min)

4. Demo: entropy vs distance under different temps (10 min)

### Watch Before Recording:

- \*\*CS224N Transformers lectures\*\* (Stanford): https://www.youtube.com/@stanfordnlp

- \*\*Karpathy "Let's build GPT from scratch"\*\*: https://www.youtube.com/watch?v=kCc8FmEb1nY

- Search: "attention mechanism visualization" on YouTube

### Demo Requirements:

- Tiny 2-layer char-LM on Shakespeare or similar

- Plot: attention entropy vs position for layers 1 & 2

- Vary temperature: {0.5, 1.0, 2.0, 5.0}

- Show saturation at low temp, dispersion at high temp

### Deliverables:

- [ ] 8 slides: Q/K/V math, softmax derivation, entropy plots (2 you generated)

- [ ] Jupyter notebook: char-LM training + attention analysis

- [ ] 5-question quiz:

1. Derive ∂(softmax)/∂Q

2. Why does low temperature cause saturation?

3. What happens to gradient flow without residual connections?

4. How does attention entropy change with sequence length?

5. Why is scaled dot-product better than additive attention?

- [ ] 1-page cheat sheet: "Attention as Associative Memory"

---

## A2. Transformer-XL (recurrence)

\*\*Paper:\*\* https://arxiv.org/abs/1901.02860

### Tasks:

- [ ] \*\*Concept sketch:\*\* How memory segments roll; relative pos encodings

- [ ] \*\*Assignment:\*\* Draw time-unrolled diagram of segment caching

- [ ] \*\*Artifact:\*\* Diagram showing 3 timesteps with KV cache propagation

- [ ] \*\*Toy cache:\*\* Implement inference-only memory of past KVs; vary segment length

- [ ] \*\*Acceptance:\*\* Plot perplexity vs segment length {256, 512, 1024, 2048}; longer → better until saturation point

### Implementation Details:

```python

# Segment cache pseudo-code you must implement

class TransformerXLCache:

def \_\_init\_\_(self, n\_layers, segment\_len):

self.cache = [None] \* n\_layers

self.segment\_len = segment\_len

def update(self, layer\_idx, new\_kv):

if self.cache[layer\_idx] is None:

self.cache[layer\_idx] = new\_kv

else:

# Concatenate and keep only last segment\_len

concat\_cache = concat([self.cache[layer\_idx], new\_kv])

self.cache[layer\_idx] = concat\_cache[-self.segment\_len:]

```

---

## A3. Compressive Transformer

\*\*Paper:\*\* https://arxiv.org/abs/1911.05507

### Tasks:

- [ ] \*\*Mechanism capsule:\*\* How compression ratio & kernel work

- [ ] \*\*Assignment:\*\* Write pseudocode for "push old KVs through compressor and store"

- [ ] \*\*Hook proto:\*\* Implement downsample-old-KV hook (e.g., average pooling every n tokens)

- [ ] \*\*Acceptance:\*\* Measure Δquality at 8k→32k with/without compression

- [ ] \*\*Report:\*\* VRAM saved per token

### Compression Strategy:

```python

# Your compression hook

def compress\_old\_kv(kv\_cache, compression\_ratio=4):

"""

kv\_cache: [batch, seq\_len, d\_model]

Every compression\_ratio old tokens becomes 1 compressed token

"""

old\_tokens = kv\_cache[:, :-512, :] # Keep last 512 uncompressed

compressed = avg\_pool1d(old\_tokens, kernel=compression\_ratio)

recent = kv\_cache[:, -512:, :]

return concat([compressed, recent], dim=1)

```

### Experiments:

- [ ] Baseline (no compression): ppl at 8k, 16k, 32k

- [ ] Compression ratio=2: ppl + VRAM

- [ ] Compression ratio=4: ppl + VRAM

- [ ] Compression ratio=8: ppl + VRAM

- [ ] Plot: compression ratio vs (ppl\_degradation, VRAM\_saved)

---

## 🎥 LECTURE 2: Recurrent & Compressive Memory (30min + 10min demo)

\*\*Deliver after:\*\* A2-A3 complete

### Learning Goals:

1. Segment recurrence for unbounded context

2. Relative positional encodings math

3. Compressive buffers: kernels, ratios, quality vs VRAM trade-offs

### Outline:

1. Transformer limitations for long sequences (3 min)

2. Segment recurrence mechanism + relative positions (10 min)

3. Memory wall problem (5 min)

4. Compression strategies: average pooling, 1D conv, learned compression (7 min)

5. Demo: rolling cache + compression ablations (10 min)

### Watch Before Recording:

- Search: "Transformer-XL paper walkthrough" on YouTube

- Search: "Compressive Transformer explained"

- Look for conference talks (ICLR, NeurIPS presentations)

### Demo Requirements:

- Show segment rolling with 3-4 segments visually

- Live coding: implement compress\_old\_kv function

- Plot your perplexity vs segment\_length curve

- Plot your compression\_ratio vs (quality, VRAM) Pareto front

### Deliverables:

- [ ] Slide deck: rolling cache diagram (hand-drawn or animated), compression ablation figures (your data)

- [ ] Demo notebook: segment\_cache.py + compress\_old\_kv.py with profiling

- [ ] 5-question quiz on failure modes:

1. Why does relative position encoding matter for recurrence?

2. What happens if segment length = context length?

3. Derive VRAM savings: compression\_ratio=4 at seq\_len=32k

4. When does compression hurt more than help?

5. How to choose compression kernel (avg vs learned)?

---

## A4. RETRO (retrieval-enhanced)

\*\*Paper:\*\* https://arxiv.org/abs/2112.04426

### Tasks:

- [ ] \*\*Block diagram:\*\* encoder → nearest neighbor chunks → cross-attend

- [ ] \*\*Draw:\*\* Information flow from datastore to decoder

- [ ] \*\*RETRO-lite implementation:\*\*

- Build FAISS index on train splits

- Retrieve top-k chunks per input

- Prepend to input (simplification of cross-attention)

- [ ] \*\*Acceptance:\*\* Measure Δ(perplexity) vs added latency per token

- [ ] \*\*Report:\*\* Cache warm vs cold runs, retrieval overhead

### Implementation Specs:

```python

# RETRO-lite pipeline

class RETROLite:

def \_\_init\_\_(self, chunk\_size=64, top\_k=2):

self.chunk\_size = chunk\_size

self.top\_k = top\_k

self.datastore = None # FAISS index

def build\_index(self, corpus):

# Chunk corpus into chunk\_size tokens

chunks = chunkify(corpus, self.chunk\_size)

# Encode chunks with frozen LM

embeddings = encode(chunks)

# Build FAISS index (L2 or IP)

self.datastore = faiss.IndexFlatL2(d\_model)

self.datastore.add(embeddings)

def retrieve(self, query\_embedding):

distances, indices = self.datastore.search(query\_embedding, self.top\_k)

return chunks[indices]

def forward(self, input\_ids):

# Get query embedding

query\_emb = encode(input\_ids)

# Retrieve

retrieved\_chunks = self.retrieve(query\_emb)

# Prepend to input

augmented\_input = concat([retrieved\_chunks, input\_ids])

return augmented\_input

```

### Experiments:

- [ ] Baseline (no retrieval): perplexity

- [ ] RETRO k=1: ppl, latency (warm cache), latency (cold cache)

- [ ] RETRO k=2: same metrics

- [ ] RETRO k=5: same metrics

- [ ] Plot: k vs (Δppl, retrieval\_latency\_ms)

---

## A5. kNN-LM (non-parametric memory)

\*\*Paper:\*\* https://openreview.net/pdf?id=HklBjCEKvH

### Tasks:

- [ ] \*\*Datastore plan:\*\* Choose which hidden layer (typically last or second-to-last)

- [ ] \*\*Design:\*\* L2 vs inner product (IP), normalize embeddings or not

- [ ] \*\*Interpolation sweep:\*\*

- λ ∈ {0, 0.2, 0.5, 0.8, 1.0} (interpolation between LM and kNN)

- k ∈ {8, 32, 64} (number of neighbors)

- [ ] \*\*Acceptance:\*\* Curve showing diminishing returns

- [ ] \*\*Report:\*\* Latency budget for each (k, λ) pair

### kNN-LM Math:

```

p\_final(w) = λ \* p\_kNN(w) + (1-λ) \* p\_LM(w)

where:

p\_kNN(w) is proportional to sum over i in kNN of:

exp(-distance(h\_query, h\_i) / T) \* indicator(w\_i = w)

```

### Implementation:

```python

class kNNLM:

def \_\_init\_\_(self, base\_lm, datastore, k=64, lambda\_val=0.5, T=1.0):

self.lm = base\_lm

self.datastore = datastore # FAISS index

self.k = k

self.lambda\_val = lambda\_val

self.T = T

def forward(self, context):

# Get LM distribution

p\_lm = self.lm(context) # [vocab\_size]

# Get hidden state for kNN lookup

h\_query = self.lm.get\_hidden(context) # [d\_model]

# Retrieve k nearest neighbors

distances, indices = self.datastore.search(h\_query, self.k)

# Compute kNN distribution

p\_knn = self.compute\_knn\_dist(distances, indices)

# Interpolate

p\_final = self.lambda\_val \* p\_knn + (1 - self.lambda\_val) \* p\_lm

return p\_final

```

### Experiments Grid:

```

k=8 k=32 k=64

λ=0.0 ppl,lat ppl,lat ppl,lat

λ=0.2 ppl,lat ppl,lat ppl,lat

λ=0.5 ppl,lat ppl,lat ppl,lat

λ=0.8 ppl,lat ppl,lat ppl,lat

λ=1.0 ppl,lat ppl,lat ppl,lat

```

- [ ] Build datastore on 100k samples

- [ ] Build datastore on 1M samples (compare build time, RAM, disk size)

- [ ] Test retrieval with warm cache (index in RAM)

- [ ] Test retrieval with cold cache (index on disk)

---

## 🎥 LECTURE 3: Retrieval-Augmented Memory (kNN-LM & RETRO) (30min + 10min demo)

\*\*Deliver after:\*\* A4-A5 complete

### Learning Goals:

1. Non-parametric memory: scaling without retraining

2. Latency budgets and when retrieval pays off

3. Retrieval policies: when to retrieve, what to retrieve

### Outline:

1. Parametric vs non-parametric memory (5 min)

2. kNN-LM: interpolation math, datastore design (10 min)

3. RETRO: chunk retrieval, cross-attention vs prepending (8 min)

4. Retrieval policy: learned gating (7 min)

5. Demo: λ sweep and latency analysis (10 min)

### Watch Before Recording:

- Search: "RETRO deepmind paper talk"

- Search: "kNN-LM Urvashi Khandelwal"

- Look for ICLR/EMNLP presentation videos

### Demo Requirements:

- Live build FAISS index on small corpus (show timing)

- Run kNN-LM with different λ, show perplexity improvement

- Show retrieval timing breakdown: encode query, search index, augment context

- Compare warm vs cold cache (have index in RAM vs disk)

### Deliverables:

- [ ] Slides with trade-off triangle (quality, latency, infrastructure cost)

- [ ] Demo notebook: kNN-LM implementation + RETRO-lite

- [ ] Your experimental data:

- Table: (k, λ) → (ppl, latency\_ms)

- Chart: Pareto frontier

- [ ] 5-question quiz:

1. Why does kNN-LM improve over LM alone?

2. Derive retrieval cost: FAISS search for k=64 in index of size 1M

3. When does domain shift hurt retrieval?

4. Compare: prepending vs cross-attention for retrieved chunks

5. How to decide λ: validation set or learned gating?

---

## A6. Product-Key Memory (PKM)

\*\*Paper:\*\* https://arxiv.org/abs/1907.05242

### Tasks:

- [ ] \*\*Design choice:\*\* Additive residual branch alongside FFN (don't replace FFN initially)

- [ ] \*\*Architecture:\*\*

```

x → LayerNorm → Attention → residual

↓

→ LayerNorm → FFN → residual

↓

→ LayerNorm → PKM → residual (NEW BRANCH)

```

- [ ] \*\*Slot design:\*\* Pick codebook sizes (A×B), key dims, product quantization

- Typical: A=256, B=256 → 256×256 = 65k slots

- Key dim: 64 or 128

- Query: project hidden state to key\_dim

- Product quantization: query\_a @ codebook\_A, query\_b @ codebook\_B

- Top-k product scores → retrieve values

- [ ] \*\*Probe:\*\* Histogram of accessed keys over training

- [ ] \*\*Measure:\*\* Collision rate (multiple queries → same slot)

- [ ] \*\*Acceptance:\*\* Memory-size ↔ accuracy/latency Pareto; stable addressing distribution

### PKM Math:

```

Query: q = W\_q @ h in R^(key\_dim)

Split: q\_a = q[:key\_dim//2], q\_b = q[key\_dim//2:]

Scores\_A = q\_a @ Codebook\_A^T in R^A

Scores\_B = q\_b @ Codebook\_B^T in R^B

Product scores: S[i,j] = Scores\_A[i] + Scores\_B[j] in R^(A×B)

Top-k indices: idx = topk(S.flatten(), k)

Retrieved values: V = sum over i in idx of softmax(S[i]) \* Memory[i]

Output: h\_out = h + W\_o @ V

```

### Implementation:

```python

class ProductKeyMemory(nn.Module):

def \_\_init\_\_(self, d\_model=768, key\_dim=64, n\_keys\_A=256, n\_keys\_B=256, top\_k=32):

super().\_\_init\_\_()

self.query\_proj = nn.Linear(d\_model, key\_dim)

self.codebook\_A = nn.Parameter(torch.randn(n\_keys\_A, key\_dim // 2))

self.codebook\_B = nn.Parameter(torch.randn(n\_keys\_B, key\_dim // 2))

self.memory = nn.Parameter(torch.randn(n\_keys\_A \* n\_keys\_B, d\_model))

self.output\_proj = nn.Linear(d\_model, d\_model)

self.top\_k = top\_k

def forward(self, h):

# Query

q = self.query\_proj(h) # [batch, seq, key\_dim]

q\_a, q\_b = q.chunk(2, dim=-1)

# Product quantization

scores\_a = q\_a @ self.codebook\_A.T # [batch, seq, n\_keys\_A]

scores\_b = q\_b @ self.codebook\_B.T # [batch, seq, n\_keys\_B]

# Outer product scores

scores = scores\_a.unsqueeze(-1) + scores\_b.unsqueeze(-2)

scores = scores.flatten(-2) # [batch, seq, A\*B]

# Top-k selection

top\_scores, top\_idx = scores.topk(self.top\_k, dim=-1)

# Retrieve and aggregate

retrieved = self.memory[top\_idx] # [batch, seq, top\_k, d\_model]

weights = F.softmax(top\_scores, dim=-1).unsqueeze(-1)

aggregated = (retrieved \* weights).sum(dim=-2)

# Output

return self.output\_proj(aggregated)

```

### Experiments:

- [ ] \*\*Placement ablation:\*\* Add PKM at layers {2, 6, 10} (for 12-layer model)

- [ ] \*\*Size ablation:\*\* A×B ∈ {128×128, 256×256, 512×512}

- [ ] \*\*Key dim ablation:\*\* {32, 64, 128}

- [ ] \*\*Top-k ablation:\*\* {8, 16, 32, 64}

- [ ] \*\*Collision study:\*\* Log which slots are accessed, plot access distribution

- [ ] \*\*Edit robustness:\*\* After PKM training, run ROME/MEMIT; measure side-effect reduction compared to baseline

---

## A7-A9. Positional Encodings (RoPE, ALiBi, YaRN)

### A7. RoPE (Rotary Position Embedding)

\*\*Paper:\*\* https://arxiv.org/abs/2104.09864

\*\*Tasks:\*\*

- [ ] Plot rotation angle vs position index

- [ ] Verify relative invariance property

- [ ] Train at 2048, test at 4096, 8192, 16384

### A8. ALiBi (Attention with Linear Biases)

\*\*Paper:\*\* https://arxiv.org/abs/2108.12409

\*\*Tasks:\*\*

- [ ] Train at 4k, test at 32k

- [ ] Compare to vanilla attention

- [ ] Plot perplexity drift

### A9. YaRN (Yet another RoPE extensioN)

\*\*Paper:\*\* https://arxiv.org/abs/2309.00071

\*\*Tasks:\*\*

- [ ] Apply YaRN scaling to RoPE model

- [ ] Test 16k→128k with different scale factors

- [ ] Note instability thresholds

\*\*Deliverable:\*\* One figure comparing all three at 32k/64k/128k

---

## 🎥 LECTURE 4: Positional Geometry & Context Extension (30min + 10min demo)

\*\*Deliver after:\*\* A7-A9 complete

### Learning Goals:

1. Positional encoding design space

2. RoPE rotary invariances

3. ALiBi train-short-test-long capability

4. YaRN scaling for extreme lengths

### Watch Before Recording:

- Search: "RoPE rotary position embedding explained"

- Search: "ALiBi attention linear biases"

- \*\*Stanford CS25 Transformers United\*\* playlist

### Deliverables:

- [ ] Side-by-side comparison at 32k/64k/128k

- [ ] 16k→128k stability plot

- [ ] 5-question quiz on when each method fails

---

## A10-A11. Systems Optimization (FlashAttention-2, vLLM)

### A10. FlashAttention-2

\*\*Paper:\*\* https://arxiv.org/abs/2307.08691

\*\*Tasks:\*\*

- [ ] Install and integrate FA-2

- [ ] Profile HBM reads/writes at 8k, 16k, 32k

- [ ] Measure: bytes/token reduction

### A11. vLLM & PagedAttention

\*\*Paper:\*\* https://arxiv.org/abs/2309.06180

\*\*Code:\*\* https://github.com/vllm-project/vllm

\*\*Tasks:\*\*

- [ ] Setup vLLM server

- [ ] Compare HF vs vLLM: throughput, p50/p90/p99 latency

- [ ] Test with 16 concurrent requests

\*\*Deliverable:\*\* Table with tokens/s, latency percentiles, VRAM footprint

---

## 🎥 LECTURE 5: IO-Aware Attention & Serving Systems (30min + 10min demo)

\*\*Deliver after:\*\* A10-A11 complete

### Learning Goals:

1. IO bottleneck: HBM bandwidth matters more than FLOPs

2. FlashAttention algorithm

3. PagedAttention and KV cache as virtual memory

4. Continuous batching

### Watch Before Recording:

- \*\*Tri Dao FlashAttention talks\*\* (Stanford MLSys)

- \*\*vLLM talks\*\* (Anyscale presentations)

### Deliverables:

- [ ] HBM bandwidth analysis slides

- [ ] Nsight profiler screenshots

- [ ] HF vs vLLM benchmark comparison

---

## A12-A14. KV Cache Management

### A12. Infini-attention

\*\*Paper:\*\* https://arxiv.org/abs/2404.07143

\*\*Tasks:\*\*

- [ ] Implement local window + global summary

- [ ] Test quality vs memory at 64k, 128k

### A13. SnapKV

\*\*Paper:\*\* https://arxiv.org/abs/2404.14469

\*\*Tasks:\*\*

- [ ] Head-wise importance scoring

- [ ] Sweep top-p: {0.3, 0.5, 0.7, 0.9, 1.0}

- [ ] Measure: EM/F1 vs VRAM saved

### A14. KVQuant

\*\*Paper:\*\* https://proceedings.neurips.cc/paper\_files/paper/2024/file/028fcbcf85435d39a40c4d61b42c99a4-Paper-Conference.pdf

\*\*Tasks:\*\*

- [ ] Quantize KV cache: 8→4→2 bit

- [ ] Measure quality vs VRAM vs tokens/s

- [ ] Pareto fronts

\*\*Deliverable:\*\* Policy comparison at 8k/32k/128k

---

## 🎥 LECTURE 6: KV-Cache Policies (30min + 10min demo)

\*\*Deliver after:\*\* A12-A14, Phase E complete

### Learning Goals:

1. KV cache as bottleneck

2. Policy interface: keep/compress/evict/quantize

3. Head-wise importance

4. Graceful degradation

### Watch Before Recording:

- Search: "SnapKV paper talk"

- Search: "Infini-attention Google"

### Deliverables:

- [ ] Pareto fronts for all policies

- [ ] Adversarial stress test results

---

## A15-A17. Mechanistic Memory & Editing

### A15. ROME (Rank-One Model Editing)

\*\*Paper:\*\* https://proceedings.neurips.cc/paper\_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf

\*\*Tasks:\*\*

- [ ] Edit 100 India-centric facts

- [ ] Measure: success rate, locality, side-effects

### A16. MEMIT (Mass Editing)

\*\*Paper:\*\* https://openreview.net/forum?id=MkbcAHIYgyS

\*\*Tasks:\*\*

- [ ] Batch edit 1000 facts

- [ ] Compare to ROME

- [ ] Failure taxonomy

### A17. Toy Models of Superposition

\*\*Paper:\*\* https://transformer-circuits.pub/2022/toy\_model/index.html

\*\*Tasks:\*\*

- [ ] Reproduce polysemantic packing

- [ ] Show phase transition with sparsity

- [ ] 1-page note: "Where memory lives & how it's edited"

---

## 🎥 LECTURE 7: Mechanistic Memory & Editing (30min + 10min demo)

\*\*Deliver after:\*\* A15-A17 complete

### Learning Goals:

1. Locating factual knowledge

2. Editing without fine-tuning

3. Polysemantic neurons and superposition

### Watch Before Recording:

- Search: "ROME rank one model editing"

- \*\*Anthropic mechanistic interpretability\*\* content

### Deliverables:

- [ ] Before/after activation visualizations

- [ ] 100-fact India edit results

- [ ] Superposition experiment notebook

---

## A18-A20. Benchmark Runners

### A18. LRA (Long Range Arena)

\*\*Paper:\*\* https://arxiv.org/abs/2011.04006

\*\*Code:\*\* https://github.com/google-research/long-range-arena

### A19. SCROLLS

\*\*Paper:\*\* https://arxiv.org/abs/2201.03533

\*\*Data:\*\* https://huggingface.co/datasets/tau/scrolls

### A20. LongBench

\*\*Paper:\*\* https://arxiv.org/abs/2308.14508

\*\*Deliverable:\*\* Unified runner: one CLI for all benchmarks

---

# PHASE B — LEAN BASELINE

## Tasks:

- [ ] Lock environment: CUDA, PyTorch, flash-attn, vLLM

- [ ] Fix data corpus & tokenizer

- [ ] Model grid: {hidden 512/768, layers 6/12, heads 8/12}

- [ ] Baseline CSV: context\_len, tokens/s, VRAM\_MB, bytes/token, perplexity

- [ ] Profiler pass: Nsight trace at 32k

- [ ] Reproducibility: 3 seeds, mean±std

---

# PHASE C — PARAMETRIC MEMORY (PKM)

## Tasks:

- [ ] Integration: PKM as residual branch

- [ ] Placement ablation: early vs mid vs late layers

- [ ] Keying ablation: codebook sizes, temperature, top-r

- [ ] Collision study: key reuse vs factual QA

- [ ] Edit robustness: ROME/MEMIT post-PKM

\*\*Deliverable:\*\* 3-panel figure (accuracy-latency, collisions, edit robustness)

---

## 🎥 LECTURE 8: Product-Key Memory Implementation (30min + 10min demo)

\*\*Deliver after:\*\* Phase C complete

### Learning Goals:

1. PKM architecture

2. Slot design and addressing

3. Collision analysis

4. Memory-accuracy-latency trade-offs

### Deliverables:

- [ ] 3-panel figure from Phase C

- [ ] PKM implementation notebook

- [ ] Training logs and analysis

---

# PHASE D — NON-PARAMETRIC MEMORY

## Tasks:

- [ ] FAISS index: build for 100k, 1M entries

- [ ] Hot/cold cache: measure retrieval timing

- [ ] Retrieval gate: MLP predicts benefit

- [ ] Costed eval: accuracy vs latency budgets

\*\*Deliverable:\*\* Retrieval policy ROC + budgeted accuracy table

---

# PHASE E — KV-CACHE POLICIES

## Tasks:

- [ ] Score functions: attention weight, gradient, novelty

- [ ] Window vs sink: compare strategies

- [ ] Quantization schedule: age-based bit-width

- [ ] Adversarial tests: distractors, code-mix, repeating patterns

\*\*Deliverable:\*\* Robustness chart under stress at 8k/32k/128k

---

# PHASE F — LLCE-INDIC BENCHMARK

## F1. Corpus & Licensing

### Data Sources:

\*\*Government & Legal:\*\*

- Government circulars: https://egazette.gov.in

- RBI: https://www.rbi.org.in

- SEBI: https://www.sebi.gov.in

- Supreme Court: https://main.sci.gov.in

\*\*Educational:\*\*

- NCERT: https://ncert.nic.in/textbook.php (CC-BY)

- ICMR: https://main.icmr.nic.in

\*\*Multilingual:\*\*

- IndicGenBench: https://arxiv.org/abs/2404.16816

- Indic-QA: https://arxiv.org/abs/2407.13522

- IndicNLP/IndicCorp: https://indicnlp.ai4bharat.org

- Samanantar: https://huggingface.co/datasets/ai4bharat/samanantar

- FLORES-200: https://ai.meta.com/tools/flores/

\*\*Artifact:\*\* SOURCES.md with licensing table

---

## F2. Schema

```json

{

"doc\_id": "RBI-2024-03-MC-001",

"lang\_primary": "hi",

"lang\_secondary": "en",

"is\_code\_mix": true,

"transliteration\_flag": false,

"update\_chain\_id": "RBI-2024-MC-series-01",

"task\_type": "temporal\_update",

"question": "Question text",

"answer": "Answer text",

"provenance\_offsets": [[45234, 45289]],

"distance\_bucket": "32k+",

"doc\_date": "2024-03-15",

"effective\_date": "2024-04-01"

}

```

\*\*Artifact:\*\* schema.json + validate.py

---

## F3. Seed Set & Annotation

### Task Coverage:

1. \*\*Needle-in-haystack:\*\* Bury fact at 70-90% depth

2. \*\*Temporal consistency:\*\* Multiple updates, "which is current?"

3. \*\*Cross-lingual recall:\*\* Question/answer in different languages

4. \*\*Aggregation:\*\* Combine values from distant sections

5. \*\*Contradiction resolution:\*\* Older clause vs newer superseding

6. \*\*Retrieval toggle:\*\* Mark if answer absent without retrieval

7. \*\*KV-stress:\*\* Documents ≥64k tokens

\*\*Goal:\*\* 500-1000 Q/A pairs, IAA ≥0.8

---

## F4. Metrics & Harness

### Quality Metrics:

- EM/F1

- Recall@Distance (bucketed)

- Temporal Accuracy

- Cross-Lingual Transfer

- Contradiction Rate

### Systems Metrics:

- Tokens/s

- p50/p90/p99 latency

- VRAM/token

- Δquality under KV policy

\*\*Artifact:\*\* eval.py produces results JSON + plots

---

## F5. Baselines

### Models:

- Llama-3.1-8B (GQA, 128k)

- Mistral-7B-v0.3

- Mixtral-8x7B

- Your PKM/kNN/RETRO/SnapKV variants

### Configs:

- RoPE-scaled vs ALiBi

- FlashAttention-2

- vLLM

\*\*Artifact:\*\* Leaderboard with quality + systems columns

---

## 🎥 LECTURE 9: LLCE-Indic Benchmark Launch (30min + 10min demo)

\*\*Deliver after:\*\* Phase F complete

### Learning Goals:

1. Why Indic long-context matters

2. LLCE-Indic task taxonomy

3. Baseline results

4. Using the benchmark

### Deliverables:

- [ ] Dataset schema + examples

- [ ] Live evaluator demo

- [ ] Recall@distance, temporal accuracy plots

- [ ] 5-question quiz on pitfalls

---

# PHASE G — SYNTHESIS & PAPER

## G1. Master Results Table

Rows: Models/Policies

Columns: LLCE-Indic metrics + systems (EM, F1, Recall@distance, Temporal, XLing, Tokens/s, Latency, VRAM)

## G2. Ablation Grids

- PKM: slots × key\_dim × temp

- kNN/RETRO: k × λ × index size

- KV policy: SnapKV p × Infini × KV bits

- Positional: RoPE vs YaRN vs ALiBi at long lengths

## G3. Visualizations

- Spider plot: 5-6 dimensions of trade-offs

- Pareto frontiers: PKM, kNN, KV policies

## G4. Paper Structure (6-8 pages)

1. Introduction (1 page)

2. Related Work (1 page)

3. LLCE-Indic Benchmark (1.5 pages)

4. Methods (1.5 pages)

5. Experiments (1.5 pages)

6. Results & Analysis (1 page)

7. Ethics & Limitations (0.5 page)

8. Conclusion (0.5 page)

9. References + Appendix

## G5. Reproducibility Package

- GitHub repo with code

- Docker container

- repro.sh script

- All config files (YAML)

- Seeds and RNG state management

- Results archive

---

# STRETCH — MoE AS ORTHOGONAL CAPACITY

\*\*Do after LLCE-Indic is complete\*\*

## S1. MoE FFN

- [ ] 8 experts, top-2 routing

- [ ] Load-balancing auxiliary loss

- [ ] No expert collapse

## S2. Domain Experts

- [ ] Legal/medical/code specialization

- [ ] Measure routing skew

- [ ] Per-domain gains

## S3. MoE × Memory

- [ ] PKM inside experts vs shared PKM

- [ ] Per-expert memory hit-rates

- [ ] Specialization analysis

---

# ENGINEERING PRINCIPLES

## 1. HBM is the Bottleneck

- Track bytes/token before FLOPs

- Use FlashAttention-2

- Optimize for HBM access patterns

## 2. Batching > Everything

- Implement continuous batching

- Use vLLM for higher batch sizes

- Profile throughput vs batch size

## 3. KV Cache as Virtual Memory

- Policy interface: keep/compress/evict/quantize

- Log cache hits, misses, evictions

- Graceful degradation under pressure

## 4. Observability is Mandatory

- Export: attention entropy, gradient norms, KV bytes/token, retrieval counts, tokens/s, latency

- Dashboard: CSV + plots

## 5. Graceful Degradation

- Never fail hard

- Quantize → compress → evict

- Recent context = highest priority

## 6. Seed Discipline

- Set all seeds (Python, NumPy, PyTorch, CuDNN)

- Run 3× with different seeds

- Report mean ± std

---

# PAPER LINKS REFERENCE

## Core Papers:

- Transformer: https://arxiv.org/abs/1706.03762

- Transformer-XL: https://arxiv.org/abs/1901.02860

- Compressive: https://arxiv.org/abs/1911.05507

- RETRO: https://arxiv.org/abs/2112.04426

- kNN-LM: https://openreview.net/pdf?id=HklBjCEKvH

- PKM: https://arxiv.org/abs/1907.05242

- RoPE: https://arxiv.org/abs/2104.09864

- ALiBi: https://arxiv.org/abs/2108.12409

- YaRN: https://arxiv.org/abs/2309.00071

- FlashAttention-2: https://arxiv.org/abs/2307.08691

- vLLM: https://arxiv.org/abs/2309.06180

- Infini-attention: https://arxiv.org/abs/2404.07143

- SnapKV: https://arxiv.org/abs/2404.14469

- KVQuant: https://proceedings.neurips.cc/paper\_files/paper/2024/file/028fcbcf85435d39a40c4d61b42c99a4-Paper-Conference.pdf

- ROME: https://proceedings.neurips.cc/paper\_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf

- MEMIT: https://openreview.net/forum?id=MkbcAHIYgyS

- Superposition: https://transformer-circuits.pub/2022/toy\_model/index.html

## Benchmarks:

- LRA: https://arxiv.org/abs/2011.04006

- SCROLLS: https://arxiv.org/abs/2201.03533

- LongBench: https://arxiv.org/abs/2308.14508

## Indic Resources:

- IndicGenBench: https://arxiv.org/abs/2404.16816

- Indic-QA: https://arxiv.org/abs/2407.13522

- IndicNLP: https://indicnlp.ai4bharat.org

- Samanantar: https://huggingface.co/datasets/ai4bharat/samanantar

- FLORES-200: https://ai.meta.com/tools/flores/

---

# VIDEO LECTURE RESOURCES

## Foundational:

- CS224N (Stanford): https://www.youtube.com/@stanfordnlp

- CS25 Transformers: https://www.youtube.com/@stanfordcs25

- Karpathy GPT: https://www.youtube.com/watch?v=kCc8FmEb1nY

## Systems:

- Search: "Tri Dao FlashAttention"

- Search: "vLLM PagedAttention"

- MLSys conference: https://www.youtube.com/@MLSys

## Advanced:

- Search: "RetNet attention free"

- Search: "Albert Gu Mamba"

- Search: "Memorizing Transformers Yuhuai Wu"

## Mechanistic:

- Anthropic: https://www.anthropic.com/research

- Search: "ROME MEMIT model editing"

---

# TODAY'S QUICK START

## Get Your First Baseline (One Sitting)

```python

from transformers import AutoModelForCausalLM, AutoTokenizer

import torch

import time

model\_name = "gpt2"

model = AutoModelForCausalLM.from\_pretrained(model\_name).cuda()

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

results = []

for ctx\_len in [8192, 32768]:

input\_ids = torch.randint(0, 50000, (1, ctx\_len)).cuda()

torch.cuda.reset\_peak\_memory\_stats()

start = time.time()

with torch.no\_grad():

outputs = model(input\_ids)

end = time.time()

vram\_mb = torch.cuda.max\_memory\_allocated() / (1024\*\*2)

tokens\_per\_sec = ctx\_len / (end - start)

bytes\_per\_token = vram\_mb \* 1024 / ctx\_len

results.append({

'context\_len': ctx\_len,

'tokens\_per\_sec': tokens\_per\_sec,

'peak\_vram\_mb': vram\_mb,

'bytes\_per\_token': bytes\_per\_token,

})

import pandas as pd

df = pd.DataFrame(results)

df.to\_csv('baseline.csv', index=False)

print(df)

```

This gives you your "before" column for all improvements!

---

\*\*Complete roadmap. Copy everything and start checking boxes!\*\*