# Memory Decoder Notes

## Better than DAPT + RAG

- DAPT: Retains all model parameters on domain text  
 • Expensive = ↑ parameters  
 • Catastrophic forgetting (losing generalizability)  
- RAG: Keeps model frozen but adds an external retrieval step  
 • Slows inference (expensive nearest neighbour search + longer context)  
 • Maintaining huge datastore (500 GB for GPT-2 small)

Challenge: Achieve efficient domain adaptation without retaining on retrieval overhead

## Memory Decoder

Small transformer decoder trained to mimic probability distribution of KNN retrievers.  
- Learns retrieval-like behaviour during pre-training  
- At inference, interpolates its predictions with base LLM’s prediction  
- No datastore search → avoid latency  
- Plug-and-Play: works with any model using same tokenizer, no parameter change

## Phase 1: Pre-training

• Build datastore of hidden states & tokens from a domain corpus  
• For each context, compute KNN distribution (neighbours’ continuation probabilities)  
• Train MemDec to align its output with these KNN distributions using:  
 - KL divergence loss (distribution alignment)  
 - Language modelling loss (stay grounded in corpus stats)  
 - Final loss = weighted combination  
In short: absorbs retriever’s knowledge into its own parameters, so it won’t need retrieval later.

### 1. Build Datastore

Key (k) = embedding of the context  
Value (v) = next token  
xi = context (sequence of tokens before position i)  
yi = actual next token in the corpus after xi  
ϕ(xi) = hidden representation (vector) of xi extracted from a pretrained LLM layer

### 2. Generate KNN Distribution

Equation:  
P\_KNN(yt | x) ∝ Σ (exp(-d(ϕ(x), ki) / T)) for neighbours (ki, vi) where vi = yt  
  
Definitions:  
• ϕ(x) = query embedding for context x  
• N(ϕ(x), k) = set of k nearest neighbours  
• d(ϕ(x), ki) = distance between query & neighbour (e.g., Euclidean)  
• T = temperature (controls sharpness)  
• 1\_{yt = vi} = indicator adds weight if neighbour’s next token matches candidate yt

Intuition:  
- Look up nearest neighbours of the context  
- Gather their next tokens  
- Weight them by similarity  
- Build probability distribution over tokens

### 3. Construct Training Pairs

For each context xi:  
- Input = the context xi  
- Supervision target = distribution P\_KNN(· | xi)  
  
Note: To avoid cheating, if datastore returns the exact same entry as the query (top-1), it is excluded.  
Otherwise, the model could just memorize exact matches instead of generalizing.

### 4. Pretraining Objective

Combine KL divergence loss and language modelling loss (weighted).