# Memory Decoder (MemDec) 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

Key Takeaway (conceptually):  
- 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. Loss Functions

2 complementary loss done:

1. Distribution Alignment Loss



* 1. KL divergence penalizes MemDec if its predicted probabilities differ from the retriever’s. **Ensures MemDec learns retrieval-style knowledge**

1. Language Modeling Loss



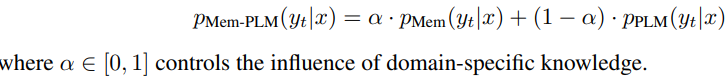
* 1. MemDec predicts the actual next token from the corpus
  2. Standard cross-entropy loss
  3. Ensures MemDec stays grounded in real domain text

1. Final Hybrid Loss



* 1. β ∈ [0,1]: hyper parameter controlling the mix (typically it is 0.5)
  2. Half of time imitate retriever’s distributions other half regular language model on real text

## Phase 2: Inference

Once pretrained, Memory Decoder exhibits a key plug-and-play capability that allows it to adapt any language model with a compatible tokenizer to the target domain via simple interpolation.  


α = 0.6 typically and high -> more domain-specific influence, low -> closer to base LLM

Takeaway

* Once pretrained, a single Memory Decoder can be attached to any base LLM that shares the same tokenizer, without further training.
* Unlike kNN-LM or RAG, no nearest-neighbor search or external datastore is used at inference.
* MemDec is just a small transformer forward pass — much cheaper than retrieval.

## Experimental Setup

1. Model
   1. Qwen2.5 family: 0.5B, 7B, 14B, 32B, 72B.
   2. LLaMA family: 7B, 13B.
   3. GPT-2 family: Small-scale baselines.
2. Domains
   1. Biomedical → PubMed abstracts + biomedical text corpora.
   2. Finance → Financial reports and news articles.
   3. Law → Legal case documents and statutes.
3. Baselines for Comparison
   1. DAPT (Domain-Adaptive Pretraining): Full model finetuning on domain text.
   2. RAG (Retrieval-Augmented Generation): Augments LLMs with retrieved passages.
   3. kNN-LM: Non-parametric retrieval using datastore at inference.
   4. Adapter Tuning: Lightweight finetuning with adapter layers.
4. Training Details
   1. 8× NVIDIA A800 GPUs (80GB each) a high-end cluster setup for large-scale LLM training.
   2. Standard Language Modeling & Downstream Tasks: Build the datastore and generate non-parametric distributions using a larger model (GPT2-xl). Then, they train the Memory Decoder using a smaller model (GPT2-small) with learning rate 1 ×10^-3
   3. Cross Model Adaptation: Datastore built using Qwen2.5-1.5B (bigger) and Memory Decoder trained on Qwen2.5-0.5B (smaller). Learning rate = 1 x10 ^ -4
   4. Cross-Vocabulary Adaptation (work across models with different tokenizers/vocabularies): Datastore built using LLaMA3.2-1B and continue training on the Memory Decoder from cross-model experiments. They reinitialize: Embedding layer (since tokens differ) and LM head (output layer).
   5. Same training budget
5. Evaluation Metrics
   1. Perplexity (PPL) on held-out test sets from each domain
   2. Downstream task metrics have their own benchmarks

## Results

1. Language Modeling on WikiText-103
   1. A 124M Memory Decoder improves all GPT-2 models.
   2. On GPT-2 small, outperforms DAPT by 15.1%, with far fewer parameters.
   3. Remains competitive with large GPT-2 models, beating all other parameter-efficient baselines.
2. Downstream Performance
   1. Evaluated on nine zero-shot tasks.
   2. Unlike DAPT (which suffers catastrophic forgetting, esp. on HYP/Yahoo), MemDec preserves or improves performance across all tasks.
   3. Achieves the highest average score, strongest on textual entailment (CB, RTE).
3. Cross Model Adaptation
   1. A single 0.5B MemDec boosts all Qwen2/2.5 models from 0.5B to 72B parameters.
   2. Dramatically reduces perplexity for small models; even large models benefit.
4. Cross-Vocabulary Adaptation
   1. Works across different tokenizers/vocabularies (e.g., LLaMA).
   2. By reinitializing embeddings + LM head, MemDec still transfers gains.
5. In-Context Learning (ICL)
   1. Zero-shot MemDec > base model’s best few-shot.
   2. Still benefits from demonstrations (0 → 8 shots).
   3. Crucially, does not harm ICL (unlike DAPT).