Supplementary Material 1

Contents

1 Data Preprocessing and Quality Control **2**

2 Vector Database and Retrieval Pipeline **2**
- 2.1 Vector Encoding . . . 2
- 2.2 Database . . . 3
- 2.3 Retrieval Pipeline Implementation . . . 4

3 Query Preprocessing and Enrichment **6**
- 3.1 Router Verifier Architecture . . . 6
- 3.2 Context Verification and Refinement . . . 8

4 Implementation and Computational Resources **10**

5 Expert Model Configuration and Training **11**

6 Inter-Rater Agreement Analysis **11**
- 6.1 ICC$(2, k)$ Definition and Formula . . . 11
- 6.2 ICC(2,k) Results for All Evaluated Models . . . 11

1 Data Preprocessing and Quality Control

This section details the data preprocessing pipeline used to curate the AD-collection with 150,000+ PubMed abstracts and Gene-collection with 142 AD-related genes.

To ensure retrieval of high-quality evidence, we developed a 6-tier journal classification system based on domain-specific impact and relevance to Alzheimer's disease and neuroscience research.

For different journals, we assigned them with different tiers related to the time and import factors. The full journal ranking is provided in Table 1.

Table 1: Journal tier classification

Journal	**Tier**	**Score**
Nature	S	5
Science	S	5
Cell	S	5
Annual Review of Neuroscience	S	5
Annual Review of Medicine	S	5
Annual Review of Biochemistry	S	5
Annual Review of Genomics and Human Genetics	S	5
Annual Review of Biomedical Engineering	S	5
Annual Review of Public Health	S	5
NeuroImage	A+	4
Annals of Neurology	A+	4
Biological Psychiatry	A+	4
Neurobiology of Aging	A	3
Glia	A	3
Human Molecular Genetics	A	3
American Journal of Human Genetics	A	3
Cerebral Cortex	A	3
Journal of Neurochemistry	B	2
Journal of Neuroscience Methods	B	2
Journal of Neurology, Neurosurgery, and Psychiatry	B	2
European Journal of Neurology	B	2
Experimental Neurology	B	2
Free Radical Biology & Medicine	B	2
Alzheimer Disease and Associated Disorders	B	2
Dementia and Geriatric Cognitive Disorders	B	2
The European Journal of Neuroscience	B	2
Neuropsychologia	C	1
NeuroReport	C	1
Neurochemical Research	C	1
International Journal of Geriatric Psychiatry	C	1
Hippocampus	C	1
Gerontology	C	1

2 Vector Database and Retrieval Pipeline

2.1 Vector Encoding

We employ the BGE-M3 model [1], a state-of-the-art dual-encoder that produces both dense semantic vectors and sparse lexical vectors. The configuration of the BGE-M3 retriever is summarized in Table 2.

Table 2: BGE-M3 encoding parameters

Parameter	Value
Model Name	BAAI/bge-m3
Precision	FP16 (GPU inference)
Dense Vector Dimension	1024
Sparse Vector Type	Token ID → Weight mapping
Normalization	L2 norm (dense vectors only)
Output Format (Dense)	Float32 (for Milvus compatibility)
Output Format (Sparse)	Dict[int, float]

For the Gene-collection, each gene entry contains two text fields that are encoded separately: general gene information and aliases from NCBI and detailed functional annotation from OMIM. This dual encoding allows retrieval to match queries against either general information or specific functional descriptions.

2.2 Database

To support evidence-grounded retrieval under the RAG framework, we constructed two task-specific vector database schemas: a Gene collection for entity-level genomic queries and an AD-literature collection for large-scale literature retrieval.

The Gene collection is organized at the gene-entity level, where each record corresponds to a single human gene. As detailed in Table 3, the schema integrates structured identifiers with both dense and sparse embedding representations.

In contrast, the AD-literature collection is organized at the document-chunk level, where each record represents a MEDLINE-indexed PubMed abstract passage. The schema (Table 4) stores bibliographic metadata, journal-quality indicators, and hybrid embedding fields. This structure facilitates large-scale semantic search while enabling journal-aware re-ranking and citation verification during evidence synthesis.

Table 3: Gene collection field specifications

Field Name	Data Type	Dimension	Purpose
id	INT64	—	Primary key (auto-generated)
gene_name	VARCHAR(32)	—	Gene symbol (uppercase)
description	VARCHAR(1024)	—	NCBI Gene description
gene_function_omim	VARCHAR(1024)	—	OMIM functional annotation
omim_number	INT64	—	OMIM ID
description_emb_dense	FLOAT_VECTOR	1024	Dense vector (description)
description_emb_sparse	SPARSE_FLOAT_VECTOR	—	Sparse vector (description)
gene_function_emb_dense	FLOAT_VECTOR	1024	Dense vector (function)
gene_function_emb_sparse	SPARSE_FLOAT_VECTOR	—	Sparse vector (function)

For the AD-literature collection, the parameter **nlist** is set to 4096, which partitions the database into 4096 clusters via k-means. The parameter **nprobe** is set to 20, so that at query time the 20 nearest clusters are searched. The sparse index drop ratio is configured as 0.2, pruning the lowest-weight 20% of sparse features during search, thereby reducing computational cost while

Table 4: AD-literature collection field specifications

Field Name	Data Type	Max Length	Purpose
id	INT64	—	Primary key (auto-generated)
pmid	VARCHAR	64	PubMed ID
title	VARCHAR	2048	Article title
abstract	VARCHAR	65535	Abstract text (chunk)
year	INT64	—	Publication year
month	INT64	—	Publication month
journal	VARCHAR	512	Journal full name
journal_tier	VARCHAR	8	S/A+/A/B/C/D
journal_score	INT64	—	0–5
is_supplement	BOOL	—	Supplement flag
keywords	ARRAY[VARCHAR]	128	MeSH terms
emb_dense	FLOAT_VECTOR	1024	Dense semantic vector
emb_sparse	SPARSE_FLOAT_VECTOR	—	Sparse lexical vector

maintaining recall for high-confidence lexical matches.

2.3 Retrieval Pipeline Implementation

We implement two dedicated retrieval pipelines corresponding to the Gene collection and the AD-literature collection. The Gene pipeline (Algorithm 1) is designed for entity-centric queries, prioritizing precise gene identification while supporting semantic similarity search when explicit symbols are absent. In contrast, the AD-literature pipeline (Algorithm 2) targets large-scale literature retrieval, combining hybrid semantic–lexical search with quality-aware re-ranking to enhance evidence reliability.

Both pipelines adopt dual dense–sparse representations and weighted fusion strategies, while maintaining consistency with the overall hallucination-reduction objective of AD-GPT. The detailed procedures are presented in the following algorithms, and the corresponding hyperparameter configurations are summarized in Table 5.

Algorithm 1 Gene information retrieval

Require: Query q, Top-k parameter

Step 1: Entity Extraction
symbols $\leftarrow$ `extract_gene_from_query`(q)
if symbols $\neq \emptyset$ **then**
    Direct Lookup: Query gene database by symbol
    return top-k exact matches
else
    Semantic Search: Encode query $\rightarrow (q_d, q_s)$
    Retrieve top-50 from each vector field:
        $R_1 \leftarrow$ `ANN`(description_emb_dense, q_d, 50)
        $R_2 \leftarrow$ `ANN`(description_emb_sparse, q_s, 50)
        $R_3 \leftarrow$ `ANN`(gene_function_emb_dense, q_d, 50)
        $R_4 \leftarrow$ `ANN`(gene_function_emb_sparse, q_s, 50)
    Fusion: $R \leftarrow$ `WeightedRanker`(R_1, R_2, R_3, R_4)
        with weights $(0.2, 0.4, 0.2, 0.4)$
    return top-k from R
end if

Table 5: Final hyperparameter configurations

Parameter	**Collection**	**Value**
nlist	AD	4096
nprobe	AD	20
α (hybrid weight)	Both	0.7
β (journal weight)	AD	0.3
drop_ratio_search	AD	0.2
k (top-k)	Both	5
τ_{rel}	Both	0.5
Gene ranker weights	Gene	(0.2, 0.4, 0.2, 0.4)
Article ranker weights	AD	(0.7, 0.3)

Algorithm 2 AD-literature retrieval with journal quality re-ranking

Require: Query q, Top-k parameter, Relevance threshold τ_{rel}

Step 1: Auto-Keyword Extraction
keywords $\leftarrow$ `extract_keywords`$(q, \text{top-5})$
Build filter: `ARRAY_CONTAINS_ANY(keywords, keywords)`

Step 2: Hybrid Retrieval
Encode query: $(q_d, q_s) \leftarrow$ `encode_text_dual`(q)
Retrieve top-50 from each modality:
  $R_{\text{dense}} \leftarrow$ `IVF_Search`(emb_dense, q_d, 50)
  $R_{\text{sparse}} \leftarrow$ `Sparse_Search`(emb_sparse, q_s, 50)

Step 3: Hybrid Fusion
$R_{\text{fused}} \leftarrow$ `WeightedRanker`$(R_{\text{dense}}, R_{\text{sparse}})$
  with weights $(0.7, 0.3)$

Step 4: PMID Deduplication
for each document $d \in R_{\text{fused}}$ **do**
  if PMID already seen **then**
    Keep version with higher (year, month) or similarity score
  end if
end for

Step 5: Journal Quality Re-ranking
for each document d **do**
  $s_{\text{sim}} \leftarrow d$.similarity_score
  $s_{\text{journal}} \leftarrow d$.journal_score
  d.final_score $\leftarrow 0.7 \cdot s_{\text{sim}} + 0.3 \cdot \frac{s_{\text{journal}}}{5}$
end for
Sort by final_score (descending)
Filter: keep only documents with similarity_score $\geq \tau_{\text{rel}}$
return top-k documents

3 Query Preprocessing and Enrichment

3.1 Router Verifier Architecture

The RouterVerifier is an LLM-based module that performs three critical functions: First is Query Quality Assessment, see Algorithm 3, it is used to detect underspecified or ambiguous queries. Then, the Query Enrichment is aimed to expand short queries with task-appropriate keywords, we utilize the same LLM model with some system prompt to refine those queries. Detailed configuration is in Table 6. After BERT router predicts a task, the RouterVerifier, see Algorithm 4, validates the decision also using one LLM.

Table 6: RouterVerifier model specifications

Parameter	**Value**
Base Model	Llama 3.1-8B-Instruct
Quantization	4-bit (NF4 via BitsAndBytes)
Precision	FP16 compute
Max Context Length	2048 tokens
Generation Temperature	0.1 (for consistency)

Algorithm 3 Hybrid query quality assessment

Require: Query q

word_count $\leftarrow$ len(q.split())

if word_count ≤ 2 AND contains gene keywords **then**
    return `needs_refinement = True`
else if word_count ≥ 6 **then**
    return `needs_refinement = False`
else if word_count ≤ 2 **then**
    Gibberish Detection:
    Compute vowel ratio, detect character repetition
    if appears to be gibberish **then**
        return `needs_refinement = False, type = gibberish`
    end if
end if

Fallback: Use LLM for edge cases
return `llm_check_quality`(q)

Algorithm 4 Routing verification with confidence weighting

Require: Query q, BERT prediction t_{BERT}, BERT confidence c_{BERT}

Generate Verification Prompt:
response $\leftarrow$ `LLM`(verification_prompt(q, t_{BERT}))
Parse: $\{t_{\text{LLM}}, c_{\text{LLM}}, \text{is\_correct}\}$

if is_correct **then**
    return t_{BERT}
else
    Decision Logic Based on BERT Confidence:
    if $c_{\text{BERT}} < 0.60$ **then**
        BERT uncertain: trust verifier
        return t_{LLM}
    else if $c_{\text{BERT}} > 0.80$ AND $c_{\text{LLM}} < 0.85$ **then**
        BERT confident: keep unless verifier very certain
        return t_{BERT}
    else
        Moderate confidence: trust verifier
        return t_{LLM}
    end if
end if

3.2 Context Verification and Refinement

The LightweightVerifier assesses whether retrieved documents are relevant and sufficient to answer the user's query. If not, it generates a refined query for re-retrieval. And the verification workflow follows as described in Algorithm 5. After refinement and re-retrieval, a final check prevents "refinement drift" where refined queries retrieve documents that no longer answer the original question. The detail is in Algorithm 6.

Algorithm 5 Context verification and refinement

Require: Query q, Retrieved contexts C, Task type t

if $C = \emptyset$ **then**
  return `action: refine, refined_query: emergency_refine(q)`
end if

Step 1: Relevance Assessment
result $\leftarrow$ `check_relevance`(q, C, t)
Parse: {action, confidence, reason, gaps}

if action = approve **then**
  return result
else if action = refine **then**
  Step 2: Generate Refined Query
  $q' \leftarrow$ `generate_refined_query`(q, gaps, t)
  return `action:refine, refined_query:` q'
else
  return `action:reject`
end if

Algorithm 6 Final relevance check

Require: Original query q_{orig}, Final contexts C_{final}

if $C_{\text{final}} = \emptyset$ **then**
  return `is_relevant: False`
end if

Simplified Relevance Prompt:
"Can these documents help answer: q_{orig}?"

response $\leftarrow$ `LLM`(simple_relevance_prompt)
Parse: {is_relevant, confidence, reason}

return response

4 Implementation and Computational Resources

We report the computational environment and runtime characteristics of AD-GPT to ensure reproducibility and transparency. The system is deployed on a single high-memory GPU server, with vector retrieval handled by Milvus and all models served within a unified inference pipeline.

We summarize (i) hardware specifications (Table 7), (ii) concurrent GPU memory footprint under practical deployment (Table 8), and (iii) stage-wise latency analysis for end-to-end query processing (Table 9). These results demonstrate that AD-GPT can operate within a single 40 GB GPU budget while maintaining interactive-level response time.

Table 7: System hardware specifications

Component	**Specification**
GPU	NVIDIA A100 (40 GB VRAM)
CPU	Intel Xeon (32 cores)
RAM	128 GB DDR4
Storage	2 TB NVMe SSD
Milvus Deployment	Standalone mode

Table 8: Actual concurrent GPU memory usage

Component	**Quantization**	**VRAM Usage**
BGE-M3 Encoder	FP16	4.0 GB
BERT Router (12-layer)	FP32	0.5 GB
Shared LLM for verifier(Llama 3.1-8B)	4-bit	8 GB
Task 1 Expert: Llama 3.2-3B	4-bit	4 GB
Task 2/3 Expert: Llama 3.1-8B	4-bit	8 GB
Total (peak concurrent)	—	**∼32.5 GB**

Memory Optimization: RouterVerifier and LightweightVerifier share the same Llama 3.1-8B instance. Then, experts are loaded dynamically. Thus, at the peak usage, the vRAM usage is ∼36.8 GB, still within 40 GB budget.

Table 9: Average query latency breakdown

Stage	**Latency (ms)**
BERT Routing	∼50
RouterVerifier (if triggered)	∼800
Query Encoding (BGE-M3)	∼100
Milvus Retrieval (hybrid)	∼1500
LightweightVerifier (if triggered)	∼900
Expert Generation (Llama 3.1-8B)	∼3000–5000
Total	∼5–8 seconds

5 Expert Model Configuration and Training

All expert models (Task 1, Task 2, Task 3) are fine-tuned using QLoRA (Quantized Low-Rank Adaptation), which enables efficient training of large language models on limited hardware by combining 4-bit quantization with low-rank adapters (Table 10).

Table 10: Expert model training hyperparameters

Hyperparameter	**Task 1 Expert**	**Task 2 Expert**	**Task 3 Expert**
Base Model	Llama 3.2-3B	Llama 3.1-8B	Llama 3.1-8B
LoRA Rank (r)	16	32	32
LoRA Alpha (α)	32	64	64
Learning Rate	3×10^{-4}	2×10^{-4}	2×10^{-4}
Batch Size (effective)	16	16	16
Training Epochs	5	3	3
Training Samples	∼5,000	∼8,000	∼6,000
Training Time	∼2 hours	∼6 hours	∼5 hours

6 Inter-Rater Agreement Analysis

6.1 ICC$(2, k)$ Definition and Formula

To quantify the agreement of the averaged ratings across the three domain experts, we adopted the two-way random-effects intraclass correlation coefficient for average ratings, denoted ICC$(2, k)$, following Shrout and Fleiss [4]. This form is appropriate for our design because (i) the same set of experts evaluated every query, which corresponds to a two-way ANOVA; and (ii) the final metric score reported in our evaluation is the mean across the k experts. Let n denote the number of queries (targets), k the number of experts (raters), BMS the between-target mean square, JMS the between-judge mean square, and EMS, the residual mean square obtained from a target by judge two-way ANOVA. The ICC$(2, k)$ is then defined as

$$\text{ICC}(2, k) = \frac{\text{BMS} - \text{EMS}}{\text{BMS} + (\text{JMS} - \text{EMS})/n}.$$

The numerator captures the target-related variance, and the denominator additionally accounts for residual variance and, through the term $(\text{JMS}-\text{EMS})/n$, any systematic differences among rater means. ICC$(2, k)$ therefore quantifies the absolute agreement of the averaged ratings across raters. Values close to 1 indicate strong agreement of the averaged ratings across raters, whereas values close to 0 indicate that the averaged ratings are dominated by rater-specific or residual variability. Following the benchmark by Koo and Li [5], ICC values below 0.5 indicate poor reliability, 0.5 to 0.75 moderate reliability, 0.75 to 0.9 good reliability, and above 0.9 excellent reliability.

6.2 ICC(2,k) Results for All Evaluated Models

Table 11 reports the ICC$(2, k)$ values for Faithfulness (F), CSR, and CVR across all models evaluated in our benchmark comparison. Across the evaluated models, ICC$(2, k)$ is generally in the good range, indicating that the expert annotation protocol yields averaged ratings with strong inter-expert agreement for the Faithfulness, CSR, and CVR metrics.

Table 11: ICC$(2, k)$ values for Faithfulness (F), CSR, and CVR across evaluated models. Higher values indicate stronger inter-expert agreement of the averaged expert ratings.

Model	**F**	**CSR**	**CVR**
AD-GPT	0.917	0.874	0.893
ChatGPT-o1	0.850	0.890	0.925
ChatGPT-5.2	0.928	0.891	0.922
Claude3.7-Sonnet	0.573	0.806	0.704
Claude4.5-Sonnet	0.604	0.726	0.668
DeepSeek-8B	0.707	1.000†	1.000†
Gemini1.5-Flash	0.847	0.884	0.753
Grok3	0.895	0.906	0.931
MedReason-8B	0.841	0.918	0.877

† For DeepSeek-8B, all three experts assigned a score of 0 to every query for both CSR and CVR, because the model consistently failed to produce any verifiable reference across the evaluated queries. In this case the observed variance is 0 and ICC$(2, k)$ is mathematically undefined. We report the value as 1.000 to indicate perfect consensus among experts.

References

[1] Chen, J., et al. (2024). *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation.* arXiv preprint arXiv:2402.03216.

[2] Robertson, S., & Zaragoza, H. (2009). *The Probabilistic Relevance Framework: BM25 and Beyond.* Foundations and Trends in Information Retrieval, 3(4), 333–389.

[3] Chen, Z., et al. (2025). *Symbolic Mixture-of-Experts: Adaptive Skill-Based Routing for Large Language Models.* arXiv preprint:2503.05641.

[4] Shrout, P. E., & Fleiss, J. L. (1979). *Intraclass Correlations: Uses in Assessing Rater Reliability.* Psychological Bulletin, 86(2), 420–428.

[5] Koo, T. K., & Li, M. Y. (2016). *A Guideline of Selecting and Reporting Intraclass Correlation Coefficients for Reliability Research.* Journal of Chiropractic Medicine, 15(2), 155–163.