Инструкции Copilot: Изучение и Подготовка RAG для AI Journey Contest — GigaMemory

Введение

AI Journey Contest 2025 предлагает три треки для участников, один из которых — **GigaMemory: global memory for LLM** — требует разработки долговременной персональной памяти для языковой модели GigaChat[1] [2]. Эта задача является идеальным применением технологии **Retrieval-Augmented Generation (RAG)**, которая позволяет языковым моделям эффективно работать с внешними базами знаний и долговременной памятью[3] [4] [5].

1. Основы RAG (Retrieval-Augmented Generation)

Что такое RAG?

RAG — это архитектурный подход, который объединяет мощь больших языковых моделей (LLM) с возможностью извлечения релевантной информации из внешних источников данных[3] [5]. Система RAG состоит из трех ключевых этапов:

1. **Indexing (Индексация)**: Подготовка и векторизация документов[4]
2. **Retrieval (Извлечение)**: Поиск релевантных фрагментов по запросу[4] [6]
3. **Generation (Генерация)**: Создание ответа на основе найденного контекста[4]

Зачем нужен RAG для GigaMemory?

Традиционные LLM не помнят пользователя между сессиями — теряются имя, контекст работы, предпочтения[1] [2]. RAG решает эту проблему, позволяя:

- Хранить персональную информацию о пользователе в векторной базе данных
- Извлекать релевантный контекст из долговременной памяти
- Генерировать персонализированные ответы с учетом истории взаимодействий
- Обновлять и консолидировать память без переобучения модели[7] [8]

2. Архитектура RAG для Конкурса

Компоненты системы

A. Подготовка данных (Data Preparation)

Chunking (Разбиение на фрагменты): Документы разбиваются на управляемые части — чанки[9] [10]

- **Fixed-size chunking**: Фрагменты фиксированной длины (512-1024 токенов)[11] [10]

- **Recursive chunking**: Иерархическое разбиение по естественным границам [11] [12]
- **Semantic chunking**: Разбиение по смысловым блокам с помощью NLP [11] [13]
- **Sentence-based chunking**: Разбиение по предложениям для сохранения семантики [13]

Рекомендация для GigaMemory: Используйте semantic chunking для сохранения контекста диалогов и фактов о пользователе [9] [13].

Embedding Generation (Генерация эмбеддингов): Преобразование текста в векторные представления [14] [15]

- Модели для русского языка: `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2` [16]
- Специализированные модели: `multi-qa-mpnet-base-dot-v1` (768 измерений) [16]
- GigaChat embeddings через API [17] [18]

B. Векторное хранилище (Vector Store)

Выбор векторной базы данных критичен для производительности [19] [20]:

- **ChromaDB**: Легкий, локальный, идеален для прототипирования [16] [21]
- **Pinecone**: Управляемое облачное решение с высокой производительностью [5]
- **Milvus/Qdrant**: Открытые решения для production [19] [22]
- **FAISS**: Библиотека от Facebook для эффективного поиска [5] [20]

Рекомендация: Для начала используйте ChromaDB локально, затем масштабируйте на Milvus/Qdrant [21].

C. Retrieval (Извлечение)

Базовый поиск:

- Векторный поиск по косинусному сходству или евклидовому расстоянию [6] [23]
- Top-K retrieval: извлечение N наиболее релевантных фрагментов [4] [24]

Продвинутые техники:

- **Hybrid search**: Комбинация векторного поиска и keyword search (BM25) [19] [20]
- **Multi-query**: Генерация нескольких вариантов запроса для улучшения recall [25]
- **HyDE (Hypothetical Document Embeddings)**: Генерация гипотетического ответа для поиска [25] [18]

D. Re-ranking (Переранжирование)

Re-ranking значительно улучшает точность, переупорядочивая найденные документы [26] [27] [28]:

- **Cross-encoders**: Совместное кодирование запроса и документа (BERT-based) [26] [29]
- **ColBERT**: Быстрая модель для переранжирования [26]
- **Cohere Rerank**: API-решение с высокой точностью [27] [30]

Эффект: Увеличение accuracy на 20-35%[26] [30] [28]

E. Generation (Генерация)

Финальный этап — генерация ответа с использованием LLM:

```
# Пример промпта для GigaChat
context = retrieved_docs
query = user_query

prompt = f"""
Используя следующий контекст о пользователе:
{context}

Ответь на вопрос пользователя: {query}

Учитывай личные предпочтения и историю взаимодействий.
"""

response = gigachat.complete(prompt)
```

3. Фреймворки для Реализации

LangChain

Преимущества:

- Богатая экосистема компонентов[4] [31]
- Простая интеграция с GigaChat[32] [33]
- Поддержка цепочек (chains) и агентов[34]

Базовый пример:

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import Chroma
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains import RetrievalQA
from langchain_gigachat import GigaChat

# 1. Разбиение документов
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
)
chunks = text_splitter.split_documents(documents)

# 2. Создание векторного хранилища
vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=OpenAIEmbeddings()
)
```

```
# 3. Создание retriever
retriever = vectorstore.as_retriever(
    search_type="similarity",
    search_kwargs={"k": 5}
)

# 4. RAG chain
llm = GigaChat(credentials="YOUR_API_KEY")
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever
)

# 5. Использование
result = qa_chain.invoke({"query": "Какие мои предпочтения?"})
```

LlamaIndex

Преимущества:

- Специализация на RAG-задачах [35] [36] [37]
- Встроенная поддержка различных индексов [37]
- Простая интеграция с GigaChat [38] [17]

Базовый пример:

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.llms.gigachat import GigaChat
from llama_index.embeddings.gigachat import GigaChatEmbedding
from llama_index.core.settings import Settings

# Настройка
Settings.llm = GigaChat(credentials="YOUR_API_KEY")
Settings.embed_model = GigaChatEmbedding()

# Загрузка данных
documents = SimpleDirectoryReader('data').load_data()

# Создание индекса
index = VectorStoreIndex.from_documents(documents)

# Создание query engine
query_engine = index.as_query_engine()

# Запрос
response = query_engine.query("Что я предпочитаю на завтрак?")
```

4. GigaMemory: Специфика Задачи

Требования Конкурса

Согласно описанию задачи GigaMemory[1] [2] [39] [40]:

- Построить систему долговременной персональной памяти
- Хранить, обновлять и извлекать знания о конкретном пользователе
- Запоминать привычки, предпочтения, ограничения, факты
- Работать с многосессионными диалогами (десятки сессий, ~100K токенов)[40]
- Отвечать на вопросы по истории диалога

Архитектура Памяти

Multi-level Memory System (по аналогии с Mem0[41] [42] [43]):

1. **Short-term memory**: Контекст текущей сессии (последние N сообщений)
2. **Long-term memory**: Постоянные факты о пользователе (через RAG)
3. **Rules memory**: Предпочтения и ограничения пользователя
4. **Entities & Relationships**: Граф связей (люди, места, события)[41] [42]

Алгоритм работы:

```
1. User Query → система
2. Извлечь short-term context (последние сообщения)
3. RAG Retrieval: Найти релевантные long-term memories
4. Объединить contexts
5. GigaChat Generation с полным контекстом
6. Обновить память: извлечь новые факты → сохранить в векторную БД
```

Memory Extraction & Update

Автоматическое извлечение фактов:

```
# Промпт для извлечения фактов
extraction_prompt = f"""
Из следующего диалога извлеки ключевые факты о пользователе:
{conversation}

Формат: JSON
{{
  "facts": ["факт1", "факт2"],
  "preferences": ["предпочтение1"],
  "temporal_info": ["событие + дата"]
}}
"""

facts = gigachat.complete(extraction_prompt)
```

```
# Сохранить в векторную БД
vectorstore.add_texts(facts)
```

Консолидация памяти: Периодическое обновление для устранения дубликатов и противоречий[41] [42].

5. Оптимизация RAG

Chunking Strategies

Рекомендации для диалогов[11] [9] [10]:

- Разбивать по сессиям (session-based chunking)
- Включать overlap для сохранения контекста разговора
- Добавлять метаданные: timestamp, user_id, session_id

Пример:

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=100,
    separators=["\n\n", "\n", ". ", " ", ""]
)
```

Query Transformation

Multi-Query Approach[25]:

```
# Генерация нескольких вариантов запроса
multi_query_prompt = f"""
Сгенерируй 3 варианта этого вопроса:
{original_query}
"""

queries = gigachat.complete(multi_query_prompt)
# Поиск по каждому запросу и объединение результатов
```

HyDE (Hypothetical Document Embeddings)[25] [18]:

```
# Генерация гипотетического ответа
hyde_prompt = f"Сгенерируй подробный ответ на вопрос: {query}"
hypothetical_answer = gigachat.complete(hyde_prompt)

# Поиск похожих фрагментов на гипотетический ответ
results = vectorstore.similarity_search(hypothetical_answer)
```

Re-ranking Implementation

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import CohereRerank

# Базовый retriever
base_retriever = vectorstore.as_retriever(search_kwargs={"k": 20})

# Re-ranker
reranker = CohereRerank(top_n=5)

# Compression retriever
retriever = ContextualCompressionRetriever(
    base_compressor=reranker,
    base_retriever=base_retriever
)
```

6. Продвинутые Техники

Federated Learning для Приватности

Проект x0tta6bl4 демонстрирует подход к privacy-preserving RAG [44] [45]:

- Federated learning на распределенных узлах
- Privacy-preserving embeddings
- Локальное хранение персональных данных
- Mesh-native deployment

Применение для GigaMemory: Храните личные данные пользователя локально, синхронизируйте только агрегированные паттерны.

Self-Healing Memory

MAPE-K циклы для автоматического управления памятью [44] [45]:

- **Monitor**: Отслеживание качества retrieval
- **Analyze**: Выявление проблем (низкий recall, устаревшая информация)
- **Plan**: Планирование обновлений памяти
- **Execute**: Переиндексация, консолидация
- **Knowledge**: Накопление паттернов для улучшения

Temporal Memory Management

Для долговременной памяти критична работа с временем [40]:

```
# Добавление временных меток
memory_entry = {
    "content": "Пользователь любит кофе",
```

```
    "timestamp": "2025-10-15",
    "importance": 0.8,
    "category": "preference"
}

# Retrieval с учетом времени
recent_memories = vectorstore.similarity_search(
    query,
    filter={"timestamp": {"$gte": "2025-10-01"}},
    k=5
)
```

7. Метрики и Оценка

Retrieval Metrics [46] [4]

- **Precision**: Доля релевантных среди найденных
- **Recall**: Доля найденных среди всех релевантных
- **F1-Score**: Гармоническое среднее precision и recall
- **MRR (Mean Reciprocal Rank)**: Средний обратный ранг первого релевантного документа

Generation Quality [46]

- **BLEU**: Сравнение n-gram с эталоном
- **ROUGE**: Оценка overlap с reference
- **BERTScore**: Семантическое сходство

Memory-Specific Metrics [42] [47]

- **Correctness**: Точность извлечения фактов из памяти
- **Compression rate**: Эффективность хранения (output tokens / full context tokens)
- **Retention**: Способность помнить информацию через сессии
- **Temporal accuracy**: Правильность работы с временными запросами

8. Практический План Подготовки

Неделя 1: Основы

1. Изучить документацию LangChain/LlamaIndex [4] [35]
2. Установить окружение:
   ```
   pip install langchain langchain-gigachat chromadb sentence-transformers
   ```
3. Реализовать базовый RAG пайплайн на тестовых данных [31] [37]
4. Экспериментировать с chunking и embeddings [16] [9]

Неделя 2: Интеграция GigaChat

1. Получить API токены GigaChat [48] [49]
2. Интегрировать GigaChat для generation [38] [50] [32]
3. Использовать GigaChat embeddings для русского языка [17] [18]
4. Оптимизировать промпты для персонализации

Неделя 3: Memory System

1. Реализовать multi-level memory [8] [41] [43]
2. Создать систему извлечения фактов из диалогов
3. Реализовать обновление и консолидацию памяти [42] [51]
4. Добавить поддержку temporal queries [40]

Неделя 4: Оптимизация

1. Внедрить re-ranking для precision [26] [27] [28]
2. Экспериментировать с query transformation [25]
3. Тестировать на примерах из датасета конкурса [40]
4. Оптимизировать latency и compression rate [47]

9. Ресурсы и Ссылки

Официальные ресурсы конкурса

- **Регистрация**: https://dsworks.ru/group/aij2025 [48] [49]
- **GigaMemory задача**: https://dsworks.ru/champ/aij25-memory [39] [40]
- **Документация GigaChat**: https://developers.sber.ru/docs/ru/gigachat [52]

Туториалы и примеры

- **LangChain RAG tutorial**: Python AI Tutorial from LangChain Engineer [25]
- **LlamaIndex tutorial**: Build a RAG App with LlamaIndex [4]
- **RAG на GigaChat**: Реализация RAG на основе GigaChat (видео) [50]
- **LangChain + GigaChat пример**: Создаем свой RAG: LangChain + Python [31]

Статьи и гайды

- **RAG основы (русский)**: RAG (Retrieval-Augmented Generation): основы и подготовка [3]
- **Chunking strategies**: 5 Chunking Strategies For RAG Applications [11]
- **Reranking guide**: Re-ranking in RAG: Improve Retrieval [28]
- **Memory for chatbots**: How to Build a Chatbot with Long-Term Memory [8]

Репозитории и примеры кода

- **LangChain GitHub**: https://github.com/langchain-ai/langchain
- **LlamaIndex GitHub**: https://github.com/run-llama/llama_index
- **RAG from Scratch**: https://github.com/langchain-ai/rag-from-scratch [25]
- **Mem0 (memory framework)**: https://github.com/mem0ai/mem0 [41] [42]

10. Контрольный Список для Copilot

Теоретическая подготовка

- [ ] Понимание архитектуры RAG (Indexing → Retrieval → Generation)
- [ ] Знание векторных эмбеддингов и similarity search
- [ ] Изучение chunking strategies и их применимости
- [ ] Понимание re-ranking и его влияния на точность
- [ ] Знакомство с memory management для chatbots

Технические навыки

- [ ] Установка и настройка LangChain или LlamaIndex
- [ ] Работа с векторными базами данных (ChromaDB, Milvus)
- [ ] Интеграция GigaChat API для generation и embeddings
- [ ] Реализация базового RAG пайплайна
- [ ] Создание системы извлечения фактов из диалогов

Оптимизация

- [ ] Экспериментирование с различными chunking strategies
- [ ] Тестирование embedding моделей для русского языка
- [ ] Внедрение re-ranking для повышения precision
- [ ] Реализация query transformation (multi-query, HyDE)
- [ ] Оптимизация context window management

GigaMemory специфика

- [ ] Реализация multi-level memory (short-term + long-term)
- [ ] Система обновления памяти из диалогов
- [ ] Поддержка temporal queries
- [ ] Управление user profiles и preferences
- [ ] Механизм консолидации и устранения противоречий

Тестирование и метрики

- [ ] Настройка evaluation pipeline (Precision, Recall, F1)
- [ ] Тестирование на примерах из датасета конкурса
- [ ] Измерение latency и compression rate
- [ ] Оценка memory retention через сессии
- [ ] Проверка correctness извлечения фактов

Заключение

Подготовка к треку GigaMemory требует глубокого понимания RAG-систем и специфики долговременной памяти для LLM. Ключевые факторы успеха:

1. **Качественный retrieval**: Правильный chunking + эффективная векторизация + re-ranking = высокий recall и precision [3] [5] [26]
2. **Интеллектуальная память**: Автоматическое извлечение, обновление и консолидация фактов о пользователе [41] [42]
3. **Персонализация**: Учет предпочтений, истории взаимодействий и временного контекста [1] [2] [40]
4. **Оптимизация**: Эффективная работа с длинными диалогами (~100K токенов) через compression и memory management [47] [40]

Следуя этим инструкциям и активно экспериментируя, вы сможете создать конкурентоспособное решение для трека GigaMemory на AI Journey Contest 2025.

Удачи в соревновании!

Дедлайн регистрации и подачи решений: 30 октября 2025 [48] [49] [53] [54]

Призовой фонд: 6 500 000 рублей [48] [49]
[55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82]

⁂

1. https://habr.com/ru/companies/sberbank/articles/957292/
2. https://habr.com/ru/amp/publications/957292/
3. https://habr.com/ru/articles/871226/
4. https://python.langchain.com/docs/tutorials/rag/
5. https://bigdataschool.ru/wiki/retrieval-augmented-generation/
6. https://habr.com/ru/articles/779526/
7. https://yandex.cloud/ru/blog/posts/2025/05/retrieval-augmented-generation-basics
8. https://www.designveloper.com/blog/ai-chatbot-with-memory/
9. https://docs.compressa.ai/cloud/guides/langchain_advanced_chunking/
10. https://weaviate.io/blog/chunking-strategies-for-rag
11. https://airbyte.com/data-engineering-resources/chunk-text-for-rag

12. https://www.f22labs.com/blogs/7-chunking-strategies-in-rag-you-need-to-know/
13. https://habr.com/ru/articles/881268/
14. https://learn.microsoft.com/ru-ru/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings
15. https://18f.ru/rag/obzor-embedding-modeley-dlya-rag-sistem-keysy-i-rekomendatsii/
16. https://habr.com/ru/articles/955798/
17. https://developers.llamaindex.ai/python/examples/embeddings/gigachat/
18. https://courses.sberuniversity.ru/llm-gigachat/2/6/4
19. https://habr.com/ru/companies/ruvds/articles/924100/
20. https://blog.deepschool.ru/llm/rag-ot-pervoj-versii-k-rabochemu-resheniyu/
21. https://www.pvsm.ru/python/433707
22. https://milvus.io/ru/blog/optimize-vector-databases-enhance-rag-driven-generative-ai.md
23. https://huggingface.co/blog/ngxson/make-your-own-rag
24. https://galileo.ai/blog/mastering-rag-how-to-select-a-reranking-model
25. https://www.youtube.com/watch?v=sVcwVQRHIc8
26. https://www.unite.ai/ru/power-of-rerankers-and-two-stage-retrieval-for-retrieval-augmented-generation/
27. https://www.chatbase.co/blog/reranking
28. https://www.chitika.com/re-ranking-in-retrieval-augmented-generation-how-to-use-re-rankers-in-rag/
29. https://adasci.org/a-hands-on-guide-to-enhance-rag-with-re-ranking/
30. https://jina.ai/ru/news/maximizing-search-relevancy-and-rag-accuracy-with-jina-reranker/
31. https://www.youtube.com/watch?v=LdIOy-XhPnw
32. https://habr.com/ru/articles/907844/
33. https://www.youtube.com/watch?v=0QUFKTfzrVg
34. https://habr.com/ru/articles/862870/
35. https://learn.microsoft.com/ru-ru/azure/developer/javascript/ai/get-started-app-chat-template-llamaindex
36. https://www.studywithgpt.com/ru/tutorial/sf6wq8
37. https://nuancesprog.ru/p/21099/
38. https://developers.llamaindex.ai/python/framework-api-reference/llms/gigachat/
39. https://dsworks.ru/champ/aij25-memory
40. https://dsworks.ru/champ/aij25-memory/data
41. https://github.com/GibsonAI/memori
42. https://arxiv.org/abs/2504.19413
43. https://github.com/unclecode/mem4ai
44. ector-index-rag-intelligence-AulyF.mQRoiNVAUPiOyEXg.md
45. x0tta6bl4.md
46. https://www.scalefree.com/blog/architecture/chatbot-implementation-using-retrieval-augmented-generation/
47. https://aws.amazon.com/blogs/machine-learning/building-smarter-ai-agents-agentcore-long-term-memory-deep-dive/
48. https://aij.ru/en/contest
49. https://aij.ru/contest

50. https://www.youtube.com/watch?v=kxQ3qfryEHE
51. https://llmmultiagents.com/en/blogs/memos-revolutionizing-llm-memory-management-as-a-first-class-operating-system
52. https://tadviser.com/index.php/Product:Sberbank:_GigaChat
53. https://vc.ru/sber/2262432-ai-journey-kontest-2025-kak-kod-stanovitsya-iskusstvom
54. https://student.csi.tsu.ru/node/2422
55. https://github.com/trustbit/enterprise-rag-challenge/
56. https://dsworks.ru/en/group/aij2025
57. https://www.singlestore.com/blog/a-guide-to-retrieval-augmented-generation-rag/
58. https://www.youtube.com/watch?v=EUxkKELGChM
59. https://www.superteams.ai/blog/newsletter-august-2025-issue-not-just-reasoning-but-retrieval-is-the-biggest-challenge-of-building-modern-ai
60. https://www.youtube.com/watch?v=sr2iWz133eg
61. https://habr.com/ru/companies/pgk/articles/913912/
62. https://www.librechat.ai/docs/configuration/rag_api
63. https://learn.microsoft.com/ru-ru/shows/generative-ai-for-beginners/retrieval-augmented-generation-rag-and-vector-databases-generative-ai-for-beginners
64. https://milvus.io/ru/blog/how-to-choose-the-right-embedding-model-for-rag.md
65. https://flycode.ru/как-оптимизировать-сегментацию-текс-ai/
66. https://learn.microsoft.com/ru-ru/azure/architecture/ai-ml/guide/rag/rag-preparation-phase
67. https://habr.com/ru/companies/raft/articles/954158/
68. https://customgpt.ai/rag-reranking-techniques/
69. https://www.reddit.com/r/LLMDevs/comments/1i7olf1/how_are_you_handling_memory_and_personalization/
70. https://www.philschmid.de/gemini-with-memory
71. https://www.linkedin.com/pulse/comprehensive-guide-chatbot-memory-techniques-ai-cloudkitect-prmqe
72. https://russiancourses.net/premium_courses/rag-prilozheniya-na-langchain-i-llamaindex-dva-freymvorka-v-odnom-kurse
73. https://habr.com/ru/articles/755186/
74. https://www.ksolves.com/blog/artificial-intelligence/adding-memory-to-your-chatbot
75. https://developers.llamaindex.ai/typescript/framework/tutorials/rag/
76. https://arxiv.org/pdf/2503.08026.pdf
77. https://www.reddit.com/r/ChatGPTCoding/comments/1cigqb9/using_system_role_to_provide_long_term_memory/
78. privet-ia-dania-nik-x0tta6bl4-rwlAvMPoR2mdrmKHPGSKtA.md
79. eto-proekt-stelly-na-vezde-v-g-qD.Uav2USGyipcr836rZPg.md
80. accessibility-digital-inclusio-xBpw.ZOoQhmOeV0an8XZ9Q.md
81. digital-rights-anti-censorship-X8WRl89PRD2qblLaLv2p2Q.md
82. stil-x0tta6bl4-sozdai-svoi-sob-Y05KfNWrRiSH1GdvzLR2Kw.md