Absolutely! Here's a complete, step-by-step example of building a Retrieval-Augmented Generation (RAG) pipeline in C# using .NET 9, PostgreSQL with pgvector, Semantic Kernel, and GPT-4.5 or GPT-5 via Azure OpenAI.

**🧠 RAG Pipeline in C# (.NET 9 + PostgreSQL + GPT-4.5/5)**

**🗃️ Step 1: PostgreSQL Setup with pgvector**

**1.1 Install pgvector Extension**

Run this in your PostgreSQL shell:

CREATE EXTENSION IF NOT EXISTS vector;

**1.2 Create Table for Embeddings**

CREATE TABLE documents (

id SERIAL PRIMARY KEY,

content TEXT NOT NULL,

embedding vector(1536) -- assuming OpenAI embedding size

);

**⚙️ Step 2: .NET 9 Project Setup**

**2.1 Create Console App**

dotnet new console -n RAGDemo

cd RAGDemo

**2.2 Add NuGet Packages**

dotnet add package Microsoft.SemanticKernel

dotnet add package Microsoft.SemanticKernel.Connectors.AzureOpenAI

dotnet add package Microsoft.SemanticKernel.Connectors.PgVector

dotnet add package Npgsql

dotnet add package Pgvector

**🧬 Step 3: Generate Embeddings with Semantic Kernel**

**3.1 Configure Kernel**

using Microsoft.SemanticKernel;

var kernelBuilder = Kernel.CreateBuilder();

#pragma warning disable SKEXP0010

kernelBuilder.AddAzureOpenAITextEmbeddingGeneration(

deploymentName: "text-embedding-ada-002",

endpoint: "https://your-openai-endpoint.openai.azure.com/",

apiKey: "your-api-key",

modelId: "text-embedding-ada-002",

serviceId: "embedding"

);

#pragma warning restore SKEXP0010

var kernel = kernelBuilder.Build();

var embeddingService = kernel.GetRequiredService<ITextEmbeddingGenerationService>();

**3.2 Generate Embedding**

var text = "The mitochondria is the powerhouse of the cell.";

var embedding = await embeddingService.GenerateEmbeddingAsync(text);

**🗄️ Step 4: Store Embedding in PostgreSQL**

**4.1 Connect and Insert**

using Npgsql;

using Pgvector;

var connString = "Host=localhost;Port=5432;Username=postgres;Password=yourpassword;Database=yourdb";

var dataSourceBuilder = new NpgsqlDataSourceBuilder(connString);

dataSourceBuilder.UseVector();

await using var dataSource = dataSourceBuilder.Build();

await using var conn = dataSource.OpenConnection();

await using var cmd = new NpgsqlCommand("INSERT INTO documents (content, embedding) VALUES ($1, $2)", conn);

cmd.Parameters.AddWithValue(text);

cmd.Parameters.AddWithValue(new Vector(embedding.ToArray()));

await cmd.ExecuteNonQueryAsync();

**🔍 Step 5: Retrieve Similar Records**

**5.1 Query Nearest Neighbors**

await using var queryCmd = new NpgsqlCommand("SELECT content FROM documents ORDER BY embedding <-> $1 LIMIT 3", conn);

queryCmd.Parameters.AddWithValue(new Vector(embedding.ToArray()));

await using var reader = await queryCmd.ExecuteReaderAsync();

List<string> results = new();

while (await reader.ReadAsync()) {

results.Add(reader.GetString(0));

}

**💬 Step 6: Feed Context into GPT-4.5/5**

**6.1 Add Chat Completion Service**

#pragma warning disable SKEXP0010

kernelBuilder.AddAzureOpenAIChatCompletion(

deploymentName: "gpt-4-5-deployment",

endpoint: "https://your-openai-endpoint.openai.azure.com/",

apiKey: "your-api-key",

modelId: "gpt-4-5",

serviceId: "chat"

);

#pragma warning restore SKEXP0010

var chatService = kernel.GetRequiredService<IChatCompletionService>();

**6.2 Build Prompt and Get Response**

var context = string.Join("\n", results);

var prompt = $"Use the following context to answer the question:\n{context}\n\nQuestion: What is the role of mitochondria?";

var chatHistory = new ChatHistory();

chatHistory.AddSystemMessage("You are a helpful biology assistant.");

chatHistory.AddUserMessage(prompt);

var response = await chatService.GetChatMessageContentAsync(chatHistory);

Console.WriteLine(response.Content);

**✅ Final Output**

You now have a working RAG pipeline:

* Embedding generation via Azure OpenAI
* Storage and retrieval using PostgreSQL + pgvector
* Contextual response from GPT-4.5 or GPT-5 via Semantic Kernel

Would you like me to help you turn this into a Web API or add chunking and indexing for large documents?