# AI Clone API Documentation

This document provides comprehensive documentation for the AI Clone API, which enables the creation and management of AI experts with domain-specific knowledge using OpenAI’s Assistant API and vector stores.

## Table of Contents

1. [Introduction](#introduction)
2. [Base URL](#base-url)
3. [Authentication](#authentication)
4. [API Models](#api-models)
5. [API Endpoints](#api-endpoints)
   * [Domain Management](#domain-management)
   * [Expert Management](#expert-management)
   * [Vector Store Management](#vector-store-management)
   * [Document Management](#document-management)
   * [Memory Management](#memory-management)
   * [OpenAI Assistant Integration](#openai-assistant-integration)
   * [Query and Chat](#query-and-chat)
6. [Error Handling](#error-handling)
7. [Examples](#examples)

## Introduction

The AI Clone API allows you to create and manage AI experts with domain-specific knowledge. The system uses OpenAI’s Assistant API and vector stores to provide contextual responses based on the expert’s knowledge domain.

Key concepts: - **Domains**: Knowledge areas that contain documents and information - **Experts**: Personas with specific knowledge in domains - **Vector Stores**: Storage for document embeddings used for retrieval - **Memory Types**: Different levels of knowledge access (llm, domain, expert, client)

## Base URL

http://localhost:8000/api

## Authentication

Authentication details should be provided here. The API currently uses Supabase for database operations.

## API Models

### Core Models

#### Expert

{  
 "name": "string",  
 "domain": "string",  
 "context": "string"  
}

#### ExpertCreate

{  
 "name": "string",  
 "domain": "string",  
 "context": "string",  
 "use\_default\_domain\_knowledge": true  
}

#### ExpertResponse

{  
 "id": "uuid",  
 "name": "string",  
 "domain": "string",  
 "context": "string"  
}

#### DomainCreate

{  
 "domain\_name": "string"  
}

### Vector Store Models

#### UpdateVectorStoreRequest

{  
 "domain\_name": "string (optional)",  
 "expert\_name": "string (optional)",  
 "document\_urls": {  
 "document\_name1": "url1",  
 "document\_name2": "url2"  
 }  
}

#### VectorStoreQuery

{  
 "domain\_name": "string (optional)",  
 "expert\_name": "string (optional)",  
 "client\_name": "string (optional)",  
 "owner": "string (optional)" // 'domain', 'expert', or 'client'  
}

### Document Models

#### AddFilesToDomainVectorCreate

{  
 "domain\_name": "string",  
 "document\_urls": {  
 "document\_name1": "url1",  
 "document\_name2": "url2"  
 }  
}

#### UpdateFilesToDomainVectorCreate

{  
 "domain\_name": "string",  
 "document\_urls": {  
 "document\_name1": "url1",  
 "document\_name2": "url2"  
 },  
 "append\_files": true  
}

#### AddFilesToExpertVectorCreate

{  
 "expert\_name": "string",  
 "document\_urls": {  
 "document\_name1": "url1",  
 "document\_name2": "url2"  
 },  
 "use\_for\_specific\_client": false,  
 "client\_name": "string (optional)"  
}

#### UpdateFilesToExpertVectorCreate

{  
 "expert\_name": "string",  
 "document\_urls": {  
 "document\_name1": "url1",  
 "document\_name2": "url2"  
 },  
 "use\_for\_specific\_client": false,  
 "client\_name": "string (optional)",  
 "append\_files": true  
}

### Query Models

#### QueryRequest

{  
 "query": "string",  
 "expert\_name": "string",  
 "memory\_type": "string", // Options: "llm", "domain", "expert", "client"  
 "client\_name": "string (optional)"  
}

#### QueryResponse

{  
 "response": {  
 "text": "string",  
 "citations": [  
 {  
 "quote": "string",  
 "source": "string"  
 }  
 ]  
 }  
}

### OpenAI Assistant Models

#### CreateAssistantRequest

{  
 "expert\_name": "string",  
 "memory\_type": "string", // Options: "llm", "domain", "expert", "client"  
 "client\_name": "string (optional)",  
 "model": "string" // Default: "gpt-4o"  
}

#### CreateThreadRequest

{  
 "expert\_name": "string",  
 "memory\_type": "string", // Options: "llm", "domain", "expert", "client"  
 "client\_name": "string (optional)"  
}

#### AddMessageRequest

{  
 "thread\_id": "string",  
 "content": "string",  
 "role": "string" // Default: "user"  
}

#### RunThreadRequest

{  
 "thread\_id": "string",  
 "assistant\_id": "string"  
}

## API Endpoints

### Domain Management

#### Create Domain

POST /domains

Creates a new domain with a custom name and initializes a default vector store.

**Request Body**: DomainCreate

**Response**:

{  
 "domain\_name": "string",  
 "vector\_id": "string",  
 "message": "string"  
}

#### Get All Domains

GET /domains

Returns a list of all domains.

**Response**:

[  
 {  
 "domain\_name": "string",  
 "expert\_names": ["string"]  
 }  
]

#### Get Expert Domain

GET /expert\_domain/{expert\_name}

Gets the domain name for a specific expert.

**Response**:

{  
 "domain\_name": "string"  
}

### Expert Management

#### Create Expert

POST /experts

Creates a new expert with domain and context.

**Request Body**: ExpertCreate

**Response**:

{  
 "id": "uuid",  
 "name": "string",  
 "domain": "string",  
 "context": "string"  
}

#### Get All Experts

GET /experts

Returns a list of all experts.

**Response**:

[  
 {  
 "id": "uuid",  
 "name": "string",  
 "domain": "string",  
 "context": "string"  
 }  
]

#### Get Expert Context

GET /expert\_context/{expert\_name}

Gets the context for a specific expert.

**Response**:

{  
 "context": "string"  
}

#### Update Expert Context

PUT /expert\_context

Updates an expert’s context.

**Request Body**: ExpertUpdate

**Response**:

{  
 "name": "string",  
 "context": "string",  
 "message": "string"  
}

#### Update Expert Persona

POST /expert\_persona  
PUT /expert\_persona

Generates a persona from QA data and updates the expert’s context.

**Request Body**: UpdateExpertPersonaRequest

**Response**:

{  
 "expert\_name": "string",  
 "persona": "string",  
 "message": "string"  
}

### Vector Store Management

#### Create Expert Domain Vector

POST /expert\_domain\_vector

Creates or updates vector IDs for an expert based on domain.

**Request Body**: ExpertVectorCreate

**Response**:

{  
 "expert\_name": "string",  
 "domain\_name": "string",  
 "vector\_id": "string",  
 "message": "string"  
}

#### Update Expert Domain Vector

PUT /expert\_domain\_vector

Updates the preferred vector store ID for an expert.

**Request Body**: ExpertVectorUpdate

**Response**:

{  
 "expert\_name": "string",  
 "domain\_name": "string",  
 "vector\_id": "string",  
 "message": "string"  
}

#### Create Expert Client Vector

POST /expert\_client\_vector

Creates a vector store for an expert with a specific client.

**Request Body**: ExpertClientVectorCreate

**Response**:

{  
 "expert\_name": "string",  
 "client\_name": "string",  
 "vector\_id": "string",  
 "message": "string"  
}

#### Update Vector Store

POST /vectors/update

Updates an existing vector store by adding new documents.

**Request Body**: UpdateVectorStoreRequest

**Response**:

{  
 "status": "string",  
 "message": "string",  
 "vector\_id": "string",  
 "domain\_name": "string",  
 "expert\_name": "string (optional)",  
 "client\_name": "string (optional)",  
 "new\_file\_ids": ["string"],  
 "all\_file\_ids": ["string"],  
 "batch\_id": "string"  
}

#### Delete Vector Memory

DELETE /vectors

Deletes a vector memory based on domain, expert, and/or client name.

**Request Body**: DeleteVectorRequest

**Response**:

{  
 "message": "string",  
 "deleted\_vector\_id": "string"  
}

### Document Management

#### Add Files to Domain Vector

POST /domain\_files

Adds files to a domain’s default vector store.

**Request Body**: AddFilesToDomainVectorCreate

**Response**:

{  
 "domain\_name": "string",  
 "vector\_id": "string",  
 "file\_ids": ["string"],  
 "batch\_id": "string",  
 "status": "string",  
 "message": "string"  
}

#### Update Files to Domain Vector

PUT /domain\_files

Updates files in a domain’s default vector store.

**Request Body**: UpdateFilesToDomainVectorCreate

**Response**:

{  
 "domain\_name": "string",  
 "vector\_id": "string",  
 "file\_ids": ["string"],  
 "batch\_id": "string",  
 "status": "string",  
 "message": "string"  
}

#### Add Files to Domain Vector from Config

POST /domain\_files\_config

Adds files to a domain’s default vector store from a configuration file.

**Request Body**: DomainFilesConfigRequest

**Response**:

{  
 "domain\_name": "string",  
 "vector\_id": "string",  
 "file\_ids": ["string"],  
 "batch\_id": "string",  
 "status": "string",  
 "document\_count": "integer",  
 "message": "string"  
}

#### Add Files to Expert Vector

POST /expert\_files

Adds files to an expert vector store.

**Request Body**: AddFilesToExpertVectorCreate

**Response**:

{  
 "expert\_name": "string",  
 "client\_name": "string (optional)",  
 "vector\_id": "string",  
 "file\_ids": ["string"],  
 "batch\_id": "string",  
 "status": "string",  
 "message": "string"  
}

#### Update Files to Expert Vector

PUT /expert\_files

Updates files in an expert vector store.

**Request Body**: UpdateFilesToExpertVectorCreate

**Response**:

{  
 "expert\_name": "string",  
 "client\_name": "string (optional)",  
 "vector\_id": "string",  
 "file\_ids": ["string"],  
 "all\_file\_ids": ["string"],  
 "batch\_id": "string",  
 "status": "string",  
 "append\_mode": "boolean",  
 "message": "string"  
}

#### Get Documents

GET /documents

Gets documents filtered by domain, expert, and client.

**Query Parameters**: - domain: string (optional) - created\_by: string (optional) - client\_name: string (optional)

**Response**:

[  
 {  
 "id": "uuid",  
 "name": "string",  
 "document\_link": "string",  
 "domain\_name": "string",  
 "created\_by": "string",  
 "client\_name": "string (optional)",  
 "file\_id": "string"  
 }  
]

### Memory Management

#### Initialize Expert Memory

POST /initialize\_expert\_memory

Initializes an expert’s memory by creating domain, adding files, generating persona, and creating expert.

**Request Body**: InitializeExpertMemoryRequest

**Response**:

{  
 "expert\_name": "string",  
 "domain\_name": "string",  
 "status": "string",  
 "message": "string",  
 "results": {  
 "domain": {},  
 "domain\_files": {},  
 "persona": {},  
 "expert": {},  
 "expert\_files": {}  
 }  
}

#### Generate Persona from QA Data

POST /generate\_persona

Generates a persona summary from QA data.

**Request Body**: PersonaGenerationRequest

**Response**:

{  
 "persona": "string",  
 "qa\_pairs\_count": "integer"  
}

### OpenAI Assistant Integration

#### Create Assistant

POST /assistant

Creates an OpenAI Assistant for a specific expert and memory type.

**Request Body**: CreateAssistantRequest

**Response**: CreateAssistantResponse

#### Create Thread

POST /thread

Creates a new thread for conversation.

**Request Body**: CreateThreadRequest

**Response**: CreateThreadResponse

#### Add Message

POST /add\_message

Adds a message to a thread.

**Request Body**: AddMessageRequest

**Response**: AddMessageResponse

#### Run Thread

POST /run\_thread

Runs a thread with an assistant.

**Request Body**: RunThreadRequest

**Response**: RunThreadResponse

#### Get Run Status

POST /get\_run\_status

Gets the status of a run.

**Request Body**: GetRunStatusRequest

**Response**: GetRunStatusResponse

#### Get Thread Messages

POST /get\_thread\_messages

Gets messages from a thread.

**Request Body**: GetThreadMessagesRequest

**Response**: GetThreadMessagesResponse

### Query and Chat

#### Query Expert

POST /query

Queries an expert using their vector index based on the specified memory type.

**Request Body**: QueryRequest

**Response**: QueryResponse

#### Query Expert with Assistant

POST /query\_expert\_with\_assistant

Queries an expert using the OpenAI Assistant API.

**Request Body**: QueryRequest

**Response**:

{  
 "response": "string",  
 "thread\_id": "string",  
 "assistant\_id": "string"  
}

## Error Handling

The API uses standard HTTP status codes:

* 200 OK: Request succeeded
* 400 Bad Request: Invalid request parameters
* 404 Not Found: Resource not found
* 500 Internal Server Error: Server error

Error responses include a detail field with an error message:

{  
 "detail": "Error message"  
}

## Examples

### Creating a Domain and Expert

1. Create a domain:

POST /domains  
{  
 "domain\_name": "AI Education"  
}

1. Create an expert:

POST /experts  
{  
 "name": "AI Expert",  
 "domain": "AI Education",  
 "context": "Expert in artificial intelligence education",  
 "use\_default\_domain\_knowledge": true  
}

1. Add documents to domain:

POST /domain\_files  
{  
 "domain\_name": "AI Education",  
 "document\_urls": {  
 "AI Basics": "https://example.com/ai-basics.pdf",  
 "Machine Learning": "https://example.com/ml-intro.pdf"  
 }  
}

1. Add documents to expert:

POST /expert\_files  
{  
 "expert\_name": "AI Expert",  
 "document\_urls": {  
 "Teaching AI": "https://example.com/teaching-ai.pdf",  
 "AI Curriculum": "https://example.com/ai-curriculum.pdf"  
 }  
}

1. Query the expert:

POST /query  
{  
 "query": "What are the best practices for teaching AI?",  
 "expert\_name": "AI Expert",  
 "memory\_type": "expert"  
}

### Using the OpenAI Assistant API

1. Create an assistant:

POST /assistant  
{  
 "expert\_name": "AI Expert",  
 "memory\_type": "expert",  
 "model": "gpt-4o"  
}

1. Create a thread:

POST /thread  
{  
 "expert\_name": "AI Expert",  
 "memory\_type": "expert"  
}

1. Add a message:

POST /add\_message  
{  
 "thread\_id": "thread\_abc123",  
 "content": "What are the best practices for teaching AI?"  
}

1. Run the thread:

POST /run\_thread  
{  
 "thread\_id": "thread\_abc123",  
 "assistant\_id": "asst\_abc123"  
}

1. Get the run status:

POST /get\_run\_status  
{  
 "thread\_id": "thread\_abc123",  
 "run\_id": "run\_abc123"  
}

1. Get thread messages:

POST /get\_thread\_messages  
{  
 "thread\_id": "thread\_abc123",  
 "limit": 10  
}