How to Use the RAG Model for Code Search and Retrieval

The system consists of two main components: **RAG_UI** and **RAGUpdateV4.py**

1. RAG_UI.py – This file is run to **start the UI** for querying and interacting with the RAG model. An **Embedding Prompt** and an **Instruction Prompt** must be entered. The **Embedding Prompt** should hold keywords related to functions. The **Instruction Prompt** should hold higher level, more experimental instructions. See the usage examples for more information. **RAG_UI.py** holds abstracted UI code and references to **RAGSearchV4.py** which holds most of the core logic. Run this file, and a locally hosted ip will appear in the terminal. Navigating to this ip will bring you to the RAG model UI.

2. RAGUpdateV4.py - This file is run to **update embedded indexes**. These are just small databases of embedding vectors. This file pulls all C# functions from files within a directory. This code also takes in a ".gitignore" directory, and it will not process files included in ".gitignore". An embedding model turns these functions into embeddings. Embeddings are just vectors, or a list of numbers used to find similarity. With this embedding model, the embeddings are 512 dimensions. We then collect all these embeddings into an embedding database called **Pinecone.** When we query for the code, we are just turning the query into an embedding, and we are searching Pinecone for embedded functions most similar to this query.

Usage Examples

Example 1

Background: You are being asked to find an initialization bug inside of the pick and place babservice codebase that you have never seen before. You can't ask higher level engineers for help finding the general location of the code, so you must find where this code exists on your own. Instead of aimlessly searching, we can use this RAG model to pull the top x most likely functions. We use the embedding query for keywords that will find related functions. We use the instruction query to get further assistance in manipulating this code.

Embedding Query: "Initialization pnp"

Instruction Query: "Where is the code that initializes the pnp tool? Do you see any possible bugs?"

Function Output: Here we have the correct functions listed with relevant explanations in this screenshot. Other similar/related functions appear. We can set the number of returned matched functions as well. Here it is set to the default size of 5:

🔍 Top Matched Functions

#	Function Name
1	`PathIdHelper.cs::LoadPnP()`
2	`PathIdHelper.cs::ConfigurePnP()`
3	`PID.cs::Initialize()`
4	`BabPickAndPlace.cs::FollowPickAndPlacePath()`
5	`Robot.cs::Initialize()`

📜 Function Details

1. `PathIdHelper.cs::LoadPnP()`

📂 **File Path:** `C:\Users\ASLS\source\asls\bab\BioAssemblyBot.Core\Calibration\Pathing\PathIdHelper.cs`

Comment:

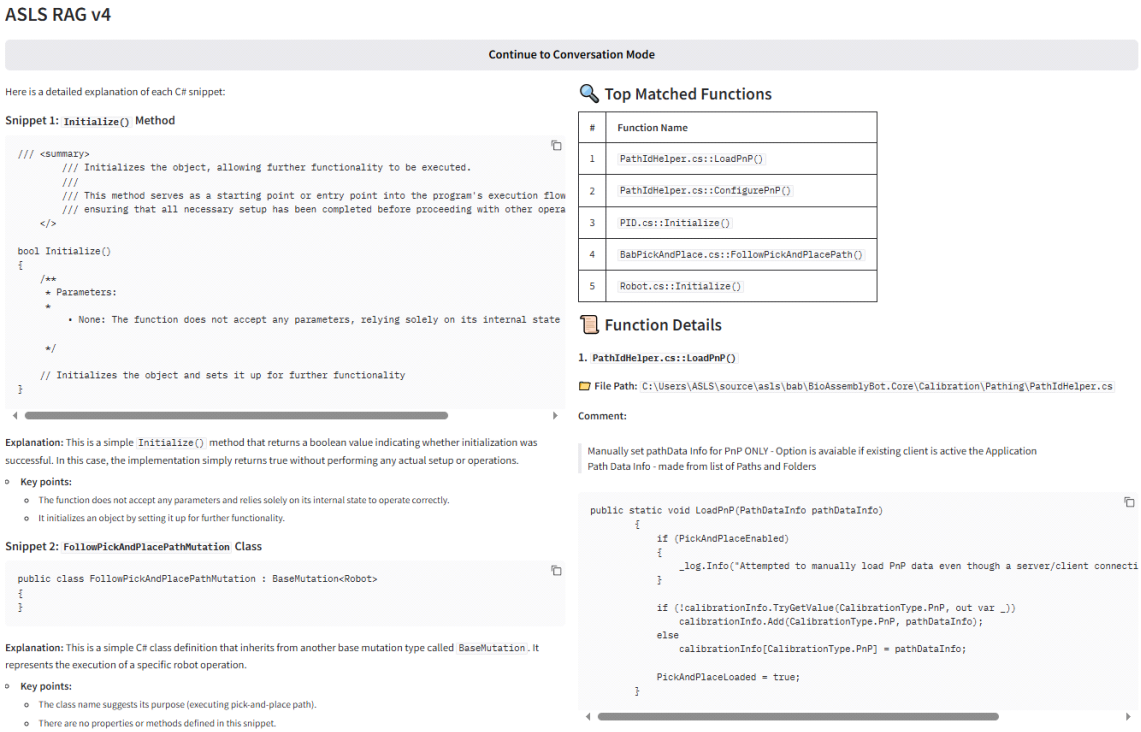
> Manually set pathData Info for PnP ONLY - Option is avaiable if existing client is active the Application
> Path Data Info - made from list of Paths and Folders

```
public static void LoadPnP(PathDataInfo pathDataInfo)
        {
            if (PickAndPlaceEnabled)
            {
                _log.Info("Attempted to manually load PnP data even though a server/client connecti
            }

            if (!calibrationInfo.TryGetValue(CalibrationType.PnP, out var _))
                calibrationInfo.Add(CalibrationType.PnP, pathDataInfo);
            else
                calibrationInfo[CalibrationType.PnP] = pathDataInfo;

            PickAndPlaceLoaded = true;
        }
```

Full Output: Notice that the full answer to the **instruction prompt** is getting a bit lost. We can continue to conversation mode to ask further questions to refine the results and build more relevant memory. This part can be significantly improved with better hardware and further software development (See **Q&A** at the bottom). Clearing the memory can help the accuracy sometimes as the entire conversation history is sent as a prompt to Ollama 3.2. The addition of agentic supervision would be extremely helpful to solve this problem.



Example 2

Background: You don't understand part of the pathing serialization code in babservice. Higher level engineers are busy, and you need assistance from an LLM to decipher what is happening without googling aimlessly. The problem is that you can't put your proprietary

code into OpenAI's infrastructure because we can't expose the code. We can use the RAG model to solve this issue.

Embedding Query: "pathing serialization"

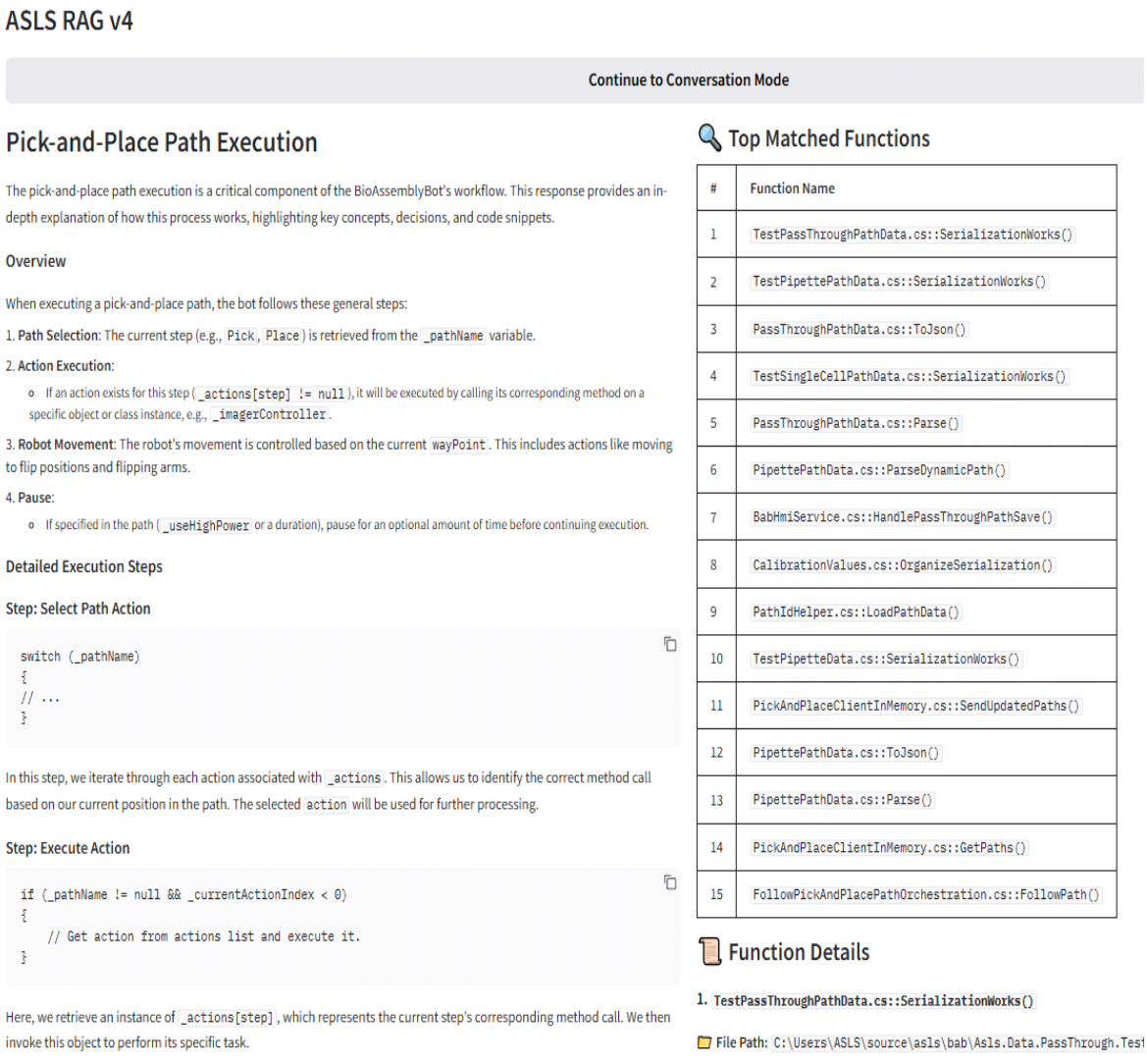
Instruction Query: "How does this serialization logic work? If I need to change the storage format how would I?"

Function Output:

🔍 Top Matched Functions

#	Function Name
1	`TestPassThroughPathData.cs::SerializationWorks()`
2	`TestPipettePathData.cs::SerializationWorks()`
3	`PassThroughPathData.cs::ToJson()`
4	`TestSingleCellPathData.cs::SerializationWorks()`
5	`PassThroughPathData.cs::Parse()`
6	`PipettePathData.cs::ParseDynamicPath()`
7	`BabHmiService.cs::HandlePassThroughPathSave()`
8	`CalibrationValues.cs::OrganizeSerialization()`
9	`PathIdHelper.cs::LoadPathData()`
10	`TestPipetteData.cs::SerializationWorks()`
11	`PickAndPlaceClientInMemory.cs::SendUpdatedPaths()`
12	`PipettePathData.cs::ToJson()`
13	`PipettePathData.cs::Parse()`
14	`PickAndPlaceClientInMemory.cs::GetPaths()`
15	`FollowPickAndPlacePathOrchestration.cs::FollowPath()`

Full Output: (remember DO NOT trust the information blindly. As with all LLMs, take the information as a pointer to what might exist in reality). Here some information is strangely correct, but some may be misleading or incorrect. Another thing to note here is that the **instruction prompt** wasn't fully answered. Hardware limitations may be the main roadblock here (see **Q&A**).



ASLS RAG v4

Continue to Conversation Mode

Pick-and-Place Path Execution

The pick-and-place path execution is a critical component of the BioAssemblyBot's workflow. This response provides an in-depth explanation of how this process works, highlighting key concepts, decisions, and code snippets.

Overview

When executing a pick-and-place path, the bot follows these general steps:

1. **Path Selection**: The current step (e.g., `Pick`, `Place`) is retrieved from the `_pathName` variable.
2. **Action Execution**:
   - If an action exists for this step (`_actions[step] != null`), it will be executed by calling its corresponding method on a specific object or class instance, e.g., `_imagerController`.
3. **Robot Movement**: The robot's movement is controlled based on the current `wayPoint`. This includes actions like moving to flip positions and flipping arms.
4. **Pause**:
   - If specified in the path (`_useHighPower` or a duration), pause for an optional amount of time before continuing execution.

Detailed Execution Steps

Step: Select Path Action

```
switch (_pathName)
{
// ...
}
```

In this step, we iterate through each action associated with `_actions`. This allows us to identify the correct method call based on our current position in the path. The selected `action` will be used for further processing.

Step: Execute Action

```
if (_pathName != null && _currentActionIndex < 0)
{
    // Get action from actions list and execute it.
}
```

Here, we retrieve an instance of `_actions[step]`, which represents the current step's corresponding method call. We then invoke this object to perform its specific task.

Top Matched Functions

#	Function Name
1	`TestPassThroughPathData.cs::SerializationWorks()`
2	`TestPipettePathData.cs::SerializationWorks()`
3	`PassThroughPathData.cs::ToJson()`
4	`TestSingleCellPathData.cs::SerializationWorks()`
5	`PassThroughPathData.cs::Parse()`
6	`PipettePathData.cs::ParseDynamicPath()`
7	`BabHmiService.cs::HandlePassThroughPathSave()`
8	`CalibrationValues.cs::OrganizeSerialization()`
9	`PathIdHelper.cs::LoadPathData()`
10	`TestPipetteData.cs::SerializationWorks()`
11	`PickAndPlaceClientInMemory.cs::SendUpdatedPaths()`
12	`PipettePathData.cs::ToJson()`
13	`PipettePathData.cs::Parse()`
14	`PickAndPlaceClientInMemory.cs::GetPaths()`
15	`FollowPickAndPlacePathOrchestration.cs::FollowPath()`

Function Details

1. **`TestPassThroughPathData.cs::SerializationWorks()`**

File Path: `C:\Users\ASLS\source\asls\bab\Asls.Data.PassThrough.Test`

QUERRYING AN EXISTING INDEX:

Before we start querying, we must make sure our **Pinecone Index Variables** in **RAG_UI.py** and are set correctly. These are separate from the **Pinecone Index Variables** in **RAGUpdateV4.py** because these are used to select which index we are querying rather than which index we are updating/changing.

I have provided some commented **pre-made Pinecone Index Variables** in **RAG_UI.py** for these services: **babservice**, **bioapp**, and **pickandplaceservice**.

If you are intending to query **pickandplaceservice** function, uncomment the indicated values. This is what they should look like:

#PNP SERVICE, PINECONE INDEX VARIABLES:

PINECONE_API_KEY = "xyz123" - replace with your **api key**

INDEX_NAME = "pnp-asls-index" - replace with your **index name**

NAMESPACE = "ns1" - replace with your **namespace**

Overtime these pre-made indexes will become **out of date as new functions are added**, and people will also need to **query different repos/directories**. To solve both of these problems, we are going to have to create/update Pinecone Indexes.

CREATING/UPDATING INDEXES:

Before we start, make sure that **Pinecone Index Variables** in **RAGUpdateV4.py** are set to the index that you would like to update. These are separate from the **Pinecone Index Variables** in **RAG_UI.py** because these are used to select which Index to modify rather than which index to search.

When we **create** a new index, we do this inside Pinecone's website then we update it with the new values. To create a new index, we do the following:

1. **Make a free Pinecone account with an email**

2. **Navigate to Get started->"Generate API key"**. This will be your new API key. There is no sensitive data stored in Pinecone, so don't worry about making this key super secure.

3. **Navigate to Database->Indexes->"Create index"**. Here you can find the INDEX_NAME and the NAMESPACE.

4. **In the code replace values highlighted above**

When we **update** a new index, we just run **RAGUpdateV4.py.** Before we do this, two values must be set correctly. These are the **Pinecone index variables** and the **Directory Strings** that withhold the directory that we would like to scrape.

I have provided some commented **pre-made Pinecone Index Variables** in **RAG_UI.py** for these services: **babservice**, **bioapp**, and **pickandplaceservice**.

If you are intending to update **pickandplaceservice** index, uncomment the indicated values. This is what they look like:

#PNP SERVICE INDEX VARIABLES:

PINECONE_API_KEY = "xyz123" - replace with your **api key**

INDEX_NAME = "pnp-asls-index" - replace with your **index name**

NAMESPACE = "ns1" - replace with your **namespace**

Selecting directory strings correctly is important. Set **CODEBASE_PATH** to a directory that contains all of the files that you would like to query. Set this directory **as low level as possible**, so we don't pull unwanted functions into our index.

This is an example of pulling only the **PickAndPlace Service** code. MOST directories have the ".gitignore" file in the same directory, but this is an example where we must set the ".gitignore" directory one higher.

CODEBASE_PATH = r"C:\Users\ASLS\source\asls\bioapp\bioapp-services\services\Asls.Hardware.PickAndPlace"

GITIGNORE_PATH = r"C:\Users\ASLS\source\asls\bioapp\bioapp-services\.gitignore"

Questions and Answers:

What is a Pinecone Index?

It is just a database of embedding vectors. We make them free on Pinecone (https://www.pinecone.io/). Essentially we are just storing functions as embedding vectors with the local file path as metadata. When we query our Pinecone index, **RAGSearchV4.py** finds the function in the repo stored locally from the file path stored. This way no code is exposed.

Why is my Instruction Prompt not being addressed properly?

Better methods for managing memory need to be implemented. Memory for this model is utilized by sending a giant payload of conversation history to the Llama 3.2 LLM model. Sometimes this can dilute the importance of the **instruction prompt**. Better methods of reprocessing the **instruction prompt** with the given code need to be implemented as well. This best implemented using agentic managers which are AI that can manage each other for different tasks. A "manager" agent would make sure that the other agents are correctly following the task of answering the **instruction prompt** without going into depth on an alternative explanation.

Why don't we just put the entire repo in one big Pinecone Index?

In order to have accurate results, restricting the amount of embeddings that we search is important. For example, there are 8000+ functions in **babservice**. If we are trying to find "pathing code" inside of **bioapp**, it is best to search **bioapp** embeddings, so that we aren't getting results for pathing logic in **babservice**.

What language does this work for?

C# only now. Further development for C++, Python, and QML process was started, but stopped. If one would like to develop support for these, use Libclang to parse C++, AST modules for parsing Python, and Treesitter or QMLparser for parsing QML. Treesitter could be a all in one tool used for all four languages, but this is more complicated than using the four separate technologies in practice.

What other processing tricks do we do?

Each pulled function has its comments pulled and processed with priority. If a function doesn't have comments, an LLM (Ollama 3.2) will generate comments for it. After all functions are pulled and commented, a final answer is generated focusing on the **Instruction Query** and the context of the pulled functions.

Why not use Ollama 3.3?

We do not have enough RAM/VRAM to run Ollama 3.3. Other model choices are inferior, and this is the best we have. Performance for addressing the **instruction prompt** would boost SIGNIFICANTLY if Ollama 3.3 was able to be used especially in the higher parameter models.

Known Bugs

- **- Logic to enter a RAG prompt without restarting needs to be added.**
- Sometimes no output will come. This is probably an Ollama bug as it seems the input is stuck waiting for Ollama. Restart the software and it will work.
- Instruction prompt accuracy and addressing could be better. This is a hardware and development time issue.