

Milestone 2 Progress Evaluation

AI Task Agent Using More Flexible Generic Tools via Model Context Protocol (MCP)

- 1. Project title, names and email addresses of team members (CSE members first)

AI Task Agent Using More Flexible Generic Tools via Model Context Protocol (MCP)

Team Members

Jadon Duff, Computer Science (jduff2024@my.fit.edu)
Jacob Ewasko, Computer Science (jewasko2024@my.fit.edu)
Ioannis Limniatis, Computer Science (ilimniatis2023@my.fit.edu)
Jacob Lebckuecher, Software Engineering (jlebkuecher2023@my.fit.edu)

- 2. Faculty Advisor: name and email address

Eraldo Ribeiro, College of Engineering and Science: Department of Electrical Engineering and Computer Science (eribeiro@fit.edu)

- 3. Client (or user): name and affiliation (department at FIT, company name, ...)

Doug Armstrong, District Directorate Chief - Response of The United States Coast Guard Auxiliary

- 4. Progress of current Milestone (progress matrix)

Task	Completion %	Jadon	Jacob E	Jacob L	Ioannis	To Do
1. Implement, test, and demo Prototype for Task Execution using Generic MCP Tools, without advanced security features implemented (feature 1).	100%	45%	25%	15%	15%	None
2. Implement, test, and demo Prototype AI Agent, with full connection to Prototype Task Execution using Generic MCP Tools (feature 1).	100%	30%	30%	20%	20%	None
3. Implement, test, and demo POC RAG memory, with basic documentation tested and linked to Prototype AI Agent (feature 2).	100%	40%	20%	20%	20%	None

4. Implement, test, and demo Prototype for Versatile Deployment with Widespread Hardware Compatibility, with the system functional to current development on a new install on a new system (feature 3).	100%	25%	25%	25%	25%	None
---	------	-----	-----	-----	-----	------

5. Discussion (at least a paragraph) of each accomplished task (and obstacles) for the current Milestone:
 - Task 1 (Implement, test, and demo Prototype for Task Execution using Generic MCP Tools, without advanced security features implemented):

The prototype for task execution using Generic MCP Tools consisted of various tools that assist the model in performing tasks. The MCP Tools that were implemented are executing commands, executing Python, launching applications, getting processes, taking a screenshot or viewing images, clicking/scrolling/typing, delay, the Brain feature (remembering, learning, and forgetting information from the user across sessions), interacting with the browser, CRUD (create, read, update, delete) file operations, searching the Internet, and getting environment information. Then team members referred to the Requirements Document to construct tests and demo the tools, which revealed what requirements were fulfilled.

One obstacle with constructing MCP tools was that some models are able to use MCP tools while others cannot. This was more found from the smaller models rather than the larger ones. The solution to this was to find higher parameter models to use. Another obstacle that came up was with the Brain feature where, even though told in the system prompt, did not automatically use the remember tool.

- Task 2 (Implement, test, and demo Prototype AI Agent, with full connection to Prototype Task Execution using Generic MCP Tools):

The prototype for the AI Agent with full connection to the MCP tools featured formulating the system prompt, selecting an applicable Large Language Model (LLM) for offline usage for each team member being optimized for computer specs, setting up llama.cpp (application that hosts the LLM), setting up the Agentic computer (Ubuntu-based system running in a Docker container (virtual OS but not hardware)), linking the frontend with the backend Agentic system, and setting up the computer to allow the AI Agent to use it. Then, basic tests were created for the LLM as MCP tools were developed. Tests were also created for the Agentic system with the limited set of tools.

One obstacle with the prototype for the AI Agent was selecting applicable LLM's for each team member's computer, which all had different specifications. Some models performed well but slow, fast but dumb, and some even ran out of memory and crashed computers. However,

through this trial-and-error phase, suitable LLM's were found. Another obstacle was debating how to spread the work across the group, whether that be 2 members working on frontend while the others worked on backend, or all working on the backend. We as a group concluded that using the built-in frontend with llama.cpp with a proxy was the best way to go forward to ramp up our production.

- Task 3 Implement, test, and demo POC RAG memory, with basic documentation tested and linked to Prototype AI Agent:

For the current milestone, task 3 was one later task that was tackled. Only after setting up the system and getting the AI Task Agent minimally working would we begin work on the RAG POC. During the 3rdnd week of the milestone, Jadon pivoted from testing and began working on implementing a prototype for the RAG system. The RAG POC initialized the embedding model EmbeddingGemma2. They used both live and synthetic documents and created embeddings. After that, the embedding model was queried to the document within the RAG database, using 'supersonic flight' as an example query. Sections relevant to the example query were returned, verifying the POC worked. As of right now, we have a small POC of the RAG system using the EmbeddingGemma2 model that can be built upon in the next milestone.

While implementing the RAG POC, we ran into some obstacles. For one, the version of llama.cpp that was installed on our systems did not support the embedding model. Fortunately, the fix was simply setting up llama.cpp's latest build on our systems and progress continued as usual. Aside from that obstacle, progress on the RAG POC was steady.

- Task 4 Implement, test, and demo Prototype for Versatile Deployment with Widespread Hardware Compatibility, with the system functional to current development on a new install on a new system:

For the current milestone, task 4 was one of the first tasks we tackled. Around a week after development, the team met and attempted to set up the software on each team member's systems. We managed to implement and deploy the AI Task Agent on both Windows and Mac systems, with specialized commands for running and stopping the software from their respective operating systems. Linux support is currently in scope and was discussed but was not pursued in this milestone due to other focuses taking priority.

One of the main obstacles for this task was installation time and size. Initially, the system installed relatively fast with little space needed. However, Jadon spent some time tweaking the pre-installed packages for Ubuntu as it would significantly increase the operations that the AI Task Agent could do, however it came at a cost. Now installing the system took almost an hour and 36GB of space. So, Jadon managed to prune the packages during installation and after the changes it took around 15 minutes and 3.6 GB of total space.

6. Discussion (at least a paragraph) of contribution of each team member to the current Milestone:

○ Jadon Duff:

1. Task 1: Contributions include the development, implementation, testing, and demo of MCP tools for the following: executing commands and executing Python, launching applications, getting processes, taking a screenshot or viewing images, clicking/scrolling/typing, delay, the Brain feature (remembering, learning, and forgetting information from the user across sessions), CRUD (create, read, update, delete) file operations, searching the Internet, and getting environment information. These MCP tools were developed and then integrated with the Agentic system allowing the AI to perform all the prior listed operations. Contributions also include mapping general computer tasks to a static list of dynamic tools that would allow an AI assistant to control a computer to complete tasks. Also, contribution includes setting up the MCP server architecture and ensuring it functioned correctly on MacOS devices. Also tested the MCP tools and frequently sent screenshots of live tests to the group's Discord chat to update group members with progress. The majority of development and implementation was contributed by this team member.
2. Task 2: Contributions include formulating the system prompt and selecting an applicable Large Language Model (LLM) for offline usage for each team member, optimized for computer specs. Additionally, set up llama.cpp, which is the application that hosts the LLM, and linked the frontend with the backend Agentic system. Next, the group member created basic tests for the LLM as MCP tools were developed, and tested the Agentic system with the limited set of tools. Finally, this team member set up the Agentic computer, which is an Ubuntu-based system running in a Docker container (virtual OS but not hardware), and set up the computer to allow the AI Agent to use it.
3. Task 3: Contributions include developing the POC RAG system using the EmbeddingGemma2 model. Obstacles were encountered; namely, the version of llama.cpp installed on team member computers did not have support for this embedding model. So, this team member manually set up llama.cpp's latest build, fixing the issue. This team member initialized the embedding model and used both live (not specific to client) and synthetic documents and created embeddings. The embedding model was then queried to the document databased using supersonic flight as an example query, and relevant sections from the RAG database were returned, ensuring that the POC functioned correctly.
4. Task 4: Contributions include setting up the versatile deployment, including a refactor. This includes a Makefile that allows the application to be run via Make commands, on MacOS (another team member contributed to the Windows startup). Contributions also include setting up the deployment architecture, which consisted of identifying 3 ports the application will run on and which parts run where. This team member set up the Web Application and LLM Hosting on port 6573, and configured

the llama-proxy to ensure the team-member selected GUI does not encounter CORS errors. The team member then selected port 6574 for the MCP server and port 6575 for the Agent's computer, and configured the Docker container to work with these, and expose to the local host. Finally, the repo skeleton and structure was contributed, including the /transfer folder (allows secure, controlled ingress and egress from the container), the /brain folder (allows Brain files to be stored, persisting LLM-selected context between sessions), and /scratch folder (allows the LLM to work with temporary files easily). /src was chosen as the source code directory, which contains /assets, the Dockerfile, MCP server, and startup scripts. /scripts include 4 executable scripts the Makefile calls, and this group member contributed 2 of the 4 executables (ones relevant to MacOS).

- Jacob E:

1. Task 1: Contributions include building the initial browser MCP tool. This tool used the playwright python library, which given a url, would give the ARIA snapshot and visible controls on the website. This tool was developed and then initially tested in Postman, which gave this group member initial feedback on the tool before letting an LLM attempt to use the tool, and a greater chance to catch bugs. Once the bugs were dealt with and the desired output from the tool was achieved, then the tool was integrated with the Agentic system, which is where the tool was tested to see if the agent can better navigate webpages given this new context. This was then extended to multiple other browser tools by another group member to give the agent more sophisticated commands. Another contribution includes rediscussing with the group on how to implement the email MCP tool after previous "hello world" demos with potential implementations of this. Then once tests were completed by other group members, this group member reviewed and accepted correctly done tests.
2. Task 2: Contributions include reviewing several local LLM's performance on a Windows laptop with 8 GB of VRAM. Given this VRAM constraint, this included testing models that fit within VRAM and models that overflowed from VRAM into RAM. Models with 7 billion parameters and below fit inside of VRAM, which made the models run quickly, generating at a speed of around 50 tokens per second, while models over 7 billion parameters overflowed into RAM, such as a 27 billion parameter model, which still ran, but only produced at a rate of around 2 tokens per second. Contributions also include initially revealing to other group members that llama.cpp comes with their own frontend, which was then implemented by other group members later on. Then once tests were completed by other group members, this group member reviewed and accepted correctly done tests.
3. Task 3: Contributions include talking with group members on how we should specifically implement the RAG embedding model later on, as it is only a proof of concept currently.
4. Task 4: Contributions include building the Windows startup architecture linked up with a Makefile through batch files. The batch files accomplish

the same job as the MacOS shell files. The first batch file started up the server and docker container, essentially starting up AI Task Agent. The second batch file shuts down the server and docker container. This has to be run at the end of each AI Task Agent session to ensure the server is still not running. Then once tests were completed by other group members, this group member reviewed and accepted correctly done tests.

- Jacob L:

1. Task 1: Contributions include creating a variety of unique test cases to test the currently implemented MCP tools and comparing the Agent's performance to our Software Requirements Specification Document. These ranged from installing a web browser, installing LUA on the AI Task Agent's system, getting system information, monitor running processes, getting computer performance metrics, testing various UI functionalities, and more. These test cases were then traced back to the requirements, and after reviewing with another team member, found to have verified 31 of the software requirements of the 153 in total.
2. Task 2: Contributions include creating a variety of unique test cases to test the current limits and performance of the AI Agent and comparing the Agent's performance for Prototype Task Execution to our Software Requirements Specification Document. Tests include writing and executing a basic program, installing the LUA programming language, checking if it comes pre-installed with Python, Java, C, C++, and Assembly, and more. After initial review with the team, we found that these tests verified a chunk of the OTS requirements.
3. Task 3: Contributions include discussing RAG implementation during weekly meetings and understanding how it would affect the overall system. Assistance in designing the RAG architecture and defining its implementation specifications within the overall system. Watching demos of the RAG system and keeping up with new progress when released.
4. Task 4: Contributions include installing, building, running, and stopping the containerized agent within the docker using the terminal commands respective to the user's operating system after updates. Ensured that the AI Task Agent correctly installed and did it in a reasonable timeframe while using little space. Testing a variety of Agents on the AI Task Agent as well as gauging their performance and intelligence. Opened discussions for potential Linux support. Modifying the docker makefile to support automatic opening of the browser upon startup.

- Ioannis:

1. Contributions include setting up and ensuring that various tools included such as Unsloth were viable and working within the VM. Executing, testing, and demoing features such as user response, code and software generation, general intelligence amongst various models. Contributions also include comparing and updating our requirements matrix to the current fix of what our AI Task Agent can accomplish, ensuring the most accurate data of what our agent has accomplished so far.

2. Task 2: Contributions include discussing and plausible frontend usable to the given agent. Demonstrating and testing how llama.cpp can be comfortably set up on each laptop. Tested how the model executed on its HTML page and how it can apply and execute given software code, asking it to generate small, generic python code for execution. Including tests using various languages such as Java as well. Gauged general intelligence in how the given model would respond and act upon a given task.
3. Task 3: Contributions include documenting and analyzing how the current iteration of the agent responds and completes the given requirements matrix we have built for it. Analyzing how well the architecture we've developed is building up and understanding our current limitations of the model. Tested the model's ability to digest the given data. Identified the general next step for the model's data and memory caching.
4. Task 4: Contributions include testing, building, and running the containerized agent within the docker via running commands in VSCode. Moved and ensured that the environment-specific MCP settings were working on the host's PC. Showed to the team the agent building and working properly, also tested building on various agents. Built, stopped, ran the agent to ensure that it worked properly.

7. Plan for the next Milestone (task matrix): Milestone 2

<i>Task</i>	<i>Jadon</i>	<i>Jacob E</i>	<i>Jacob L</i>	<i>Ioannis</i>
1. Implement, test, and demo <u>MVP</u> for Task Execution using Generic MCP Tools, without advanced security features implemented (feature 1).	Implement/test/demo Model Context Protocol tools to meet at minimum <u>100%</u> of the ATA-FTE requirements. This team member will be primarily responsible for continuing leading development, and adding MCP tools as the team sees fit. However, all team members are responsible for all work.	Implement/test/demo Model Context Protocol tools to meet at minimum <u>100%</u> of the ATA-FTE requirements. This team member will be primarily responsible for optimizing browser and Internet tools for the AI and developing the email system. However, all team members are responsible for all work.	Implement/test/demo Model Context Protocol tools to meet at minimum <u>100%</u> of the ATA-FTE requirements. This team member will be primarily responsible for fixing bugs in the existing tools, and modifying tools based on feedback from the AI itself. Also, implementing performance metrics and clipboard MCP tools. However, all team members are responsible for all work.	Implement/test/demo Model Context Protocol tools to meet at minimum <u>100%</u> of the ATA-FTE requirements. This team member will be primarily responsible for optimizing token usage, adding token security checks within MCP tools, and developing scheduled task MCP tools. However, all team members are responsible for all work.
2. Implement, test, and demo <u>MVP</u> AI Agent, with full connection to Prototype Task Execution using Generic MCP Tools (feature 1).	Implement/test/demo Agentic System that meets at minimum <u>50%</u> of <u>all requirements</u> . This team member will be primarily responsible for optimizing system prompt and all text provided to the LLM, and performing tests for requirements. However, all team members are responsible for all work.	Implement/test/demo Agentic System that meets at minimum <u>50%</u> of <u>all requirements</u> . This team member will be primarily responsible for implementing authorization to the agentic system and subsystems. However, all team members are responsible for all work.	Implement/test/demo Agentic System that meets at minimum <u>50%</u> of <u>all requirements</u> . This team member will be primarily responsible for handling audio ingestion for the Agentic system. However, all team members are responsible for all work.	Implement/test/demo Agentic System that meets at minimum <u>50%</u> of <u>all requirements</u> . This team member will be primarily responsible for handling max token and any memory management this team member sees fit. However, all team members are responsible for all work.
3. Implement, test, and demo <u>Prototype</u> RAG memory,	Implement/test/demo RAG subsystem with <u>33%</u> of the FAI requirements met.	Implement/test/demo RAG subsystem with <u>33%</u> of the FAI requirements met.	Implement/test/demo RAG subsystem with <u>33%</u> of the FAI requirements met.	Implement/test/demo RAG subsystem with <u>33%</u> of the FAI requirements met.

with basic documentation tested and linked to Prototype AI Agent (feature 2).	This team member will be primarily responsible for enabling agentic access to the RAG system, after it is implemented. However, all team members are responsible for all work.	This team member will be primarily responsible for integrating the RAG system with the current architecture, exposing to the LLM (with Jacob L). However, all team members are responsible for all work.	This team member will be primarily responsible for integrating the RAG system with the current architecture, exposing to the LLM (with Jacob E). However, all team members are responsible for all work.	This team member will be primarily responsible for performing tests of the RAG system.
4. Implement, test, and demo MVP for Versatile Deployment with Widespread Hardware Compatibility, with the system functional to current development on a new install on a new system (feature 3).	<p>Implement/test/demo Versatile Deployment system with 50% of the FDP requirements met.</p> <p>This team member will be primarily responsible for maintaining the architecture this team member implemented from the previous Milestone, optimizing Docker container build and adding any relevant additional tools, and developing the Tether system, which allows computers to use other machines for LLM access, and adding any other deployment mechanisms the team member sees fit. However, all team members are responsible for all work.</p>	<p>Implement/test/demo Versatile Deployment system with 50% of the FDP requirements met.</p> <p>This team member will be primarily responsible for investigating request interception via proxy. Development of a POC or leading a discussion detailing why the current methodology works best will count as equal contribution to development for this Task (with Ioannis). However, all team members are responsible for all work.</p>	<p>Implement/test/demo Versatile Deployment system with 50% of the FDP requirements met.</p> <p>This team member will be primarily responsible for optimizing the deployment and running of the system, and will create a POC for automated installation. Team member will develop for Windows and give guidance on how to create a similar system on MacOS. This team member will optimize the Makefile and startup scripts. However, all team members are responsible for all work.</p>	<p>Implement/test/demo Versatile Deployment system with 50% of the FDP requirements met.</p> <p>This team member will be primarily responsible for investigating request interception via proxy. Development of a POC or leading a discussion detailing why the current methodology works best will count as equal contribution to development for this Task (with Jacob E). However, all team members are responsible for all work.</p>

8. Discussion (at least a paragraph) of each planned task for the next Milestone or Discussion of at least 5 "Lessons Learned" if this is for Milestone 6
 - Task 1: 1-2 PARAGRAPHS

This planned task Implement, test, and demo MVP for Task Execution using Generic MCP Tools features completing at minimum 100 percent of the ATA-FTE Requirements. 100 percent of the Functional Task Execution requirements are completing all 39 requirements in this section. This contains optimizing browser and Internet tools for the AI, developing the email system, optimizing token usage, adding token security checks within MCP tools, developing scheduled task MCP tools, implementing performance metrics and clipboard MCP tools, fixing bugs in the existing tools, modifying tools based on feedback from the AI itself, and adding MCP tools as the team sees fit.

- Task 2: 1-2 PARAGRAPHS

This planned task Implement, test, and demo MVP AI Agent, with full connection to Prototype Task Execution using Generic MCP Tools features completing at minimum 50 percent of all requirements listed in the Requirements document. 50 percent of all requirements are completing 66 total requirements. This contains optimizing system prompt and all text provided to the LLM, implementing authorization to the agentic system and subsystems, handling audio ingestion for the Agentic system, handling max token and any memory management this team member sees fit, and performing tests for requirements.

- Task 3: 1-2 PARAGRAPHS

This planned task Implement, test, and demo Prototype RAG memory, will meet at minimum 33 percent of the ATA – FAI requirements. We will expand the RAG system so that the AI Agent can better digest user-provided documentation, including videos, pdfs, TXT, MD, DOCX, and retrieve the most relevant content. This includes integrating the current RAG system with the current architecture and exposing and allowing the LLM to make use of it as a MCP tool. Tests will be performed in completing the requirements given.

- Task 4: 1-2 PARAGRAPHS

This planned task, Implement, test, and demo MVP for Versatile Deployment with widespread and capable Hardware Capability. At a minimum, 50 percent of the ATA-FDP requirements will be fulfilled. That is approximately 10 of the 20 we laid out. The work includes maintaining deployment architecture, optimizing the Docker container and the time it takes to build, and developing the tether system to allow computers to use other machines for LLM access. The Makefile will be optimized, and startup scripts will be created for automated installation. Tests will be performed on a fresh install on each of the teams' computers.

9. Date(s) of meeting(s) with Client during the current milestone
 - 9/30/2026 (sent milestone 1 deliverables and client said they will review)
 - 10/14/2026 (sent update)
10. Client feedback on the current milestone

- Client has suggestions to help with development
 - 1. Recommends Claude for accelerated development
 - 2. OpenAI models may not work best for our development
 - No additional feedback on Milestone 1
 - Feedback remains same as Planning phase regarding deliverables
11. Date(s) of meeting(s) with Faculty Advisor during the current milestone
- 10/08/2026, 10/09/2026, 10/13/2026, 10/20/2026
12. Faculty Advisor feedback on each task for the current Milestone
- Task 1: Met with team and is happy with current progress. Believes team is working hard on this task and will be successful. Additionally, the team could connect the AI agent's tool-calling loop (or VLA model) to a virtual robot inside a simulation engine.
 - Task 2: Maybe the team could consider adding/using a voice-based frontend such as Pipecat. Same as above, team has completed this task well.
 - Task 3: Same as above, team has completed this task well.
 - Task 4: Same as above, team has completed this task well.

Faculty Advisor Signature: _____ Date: _____