tecture and is simple to implement, has the downside of requiring the agent to constantly be running, incurring significant computational costs.

In order to increase the system's flexibility while minimizing cost, we propose a method by which SAGE can autonomously program conditional routines by enabling it to write python code that implements the condition checking logic. This method is summarized in Figure 6. Two tools are introduced to support this functionality: a condition code writing tool and a condition polling tool. The SAGE agent queries the condition code writing tool to write the necessary code, then registers this code with the condition polling tool, which runs it periodically. Along with the condition checking code, it also registers a description of the action that must be taken when the condition is met. Once the code returns "True", the polling process triggers a second execution of the SAGE agent with the command registered with it by the first execution.

The implementation of the condition code writing tool is complicated by the same challenge as the device interaction tool – the requirement to inject API details into the query that generates the code. We also overcome this challenge in a similar fashion by creating an agent which uses the device interaction planner tool and the API documentation retrieval tool. However, instead of a the device attribute retrieval and command execution tools, the condition code writing tool agent has access to a code execution tool which allows it to test its code. This tool also stores the code it has run in memory, so that it can be referred to by the name of the function. Similarly to the device attribute retrieval and command execution tools, the code execution tool handles exceptions by returning their messages to the to the code writing tool agent, facilitating recovery from faulty code.

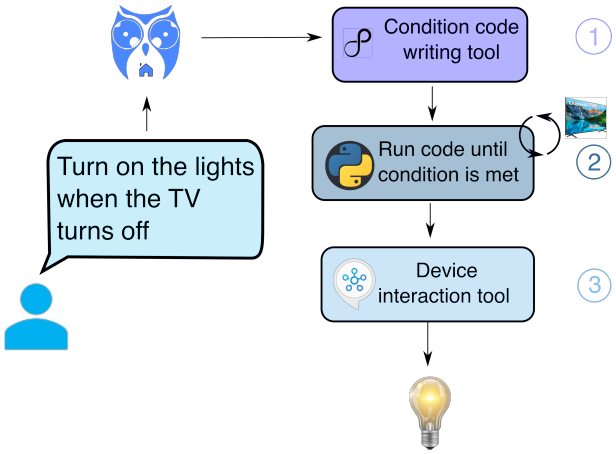

Figure 6: A summary of the persistent command handling mechanism. **1:** After receiving the user request, the SAGE agent extracts the condition ("is the TV off?") and uses the condition code writing tool to write code to check this condition. The tool registers this code in memory and responds with the name of the function (is_tv_off). **2:** The SAGE agent registers the function is_tv_off with the condition polling tool, along with the command reflecting the action to take once the condition is detected "turn the lights on". At this point the SAGE agent finishes executing. The condition polling tool periodically runs the is_tv_off function. Once the function outputs True, the condition polling tool triggers the SAGE agent again with the registered action "turn the lights on." Note that the agent will only be triggered when the status of the action transitions from False to True to avoid rerunning the agent the entire time the TV is off. **3:** The agent begins executing with the command "turn the lights on" and turns on the lights using the device interaction tool.

5.3.1 Working with other APIs.

The device interaction tool need only be capable of interaction with the SmartThings API, since interactions with other APIs (e.g. a weather API) can be handled by the top level agent through other tools. However, the code writer tool must have knowledge of all APIs required to check the condition. Luckily, high-performance LLMs such as GPT-4 are sometimes capable of writing correct code without reference material, especially for simple, popular APIs. However, we feel that the capability of the code-writing approach would be greatly expanded through the addition of a "code search" style functionality that would allow the code-writing agent to interactively access documentation for thousands of open