APIs. There are several projects pursuing this kind of capability (e.g. [28, 29]), and we believe that the integration of these ideas into SAGE is a promising avenue for future research.

5.4 Human interaction tool

The human interaction tool allows SAGE to ask the user questions, which it usually uses to clarify intent. Empirically, we have found that the introduction of the user interaction tool can cause the agent to become over-cautious, using this tool over other data sources to reduce uncertainty. We use prompt engineering in SAGE to encourage it not to over-use the tool, but how to best trade-off personalization, human interaction, and risk aversion is topic of active development.

6 Evaluation

To evaluate SAGE, we created a dataset of 43 test tasks. Test tasks are implemented by initializing device states and memories, running the home assistant with an input command, then evaluating whether the device state was modified appropriately. For tasks which involve answering user questions, the test code checks for the presence of specific sub-strings in the answer. These results of these tests are binary (pass/fail). To gain a better sense of the reasons for failure, we also manually analyze the results of the tests.

We classify the test cases according to five types of challenge that are difficult for existing systems to handle. Each test belongs to one or more of these categories. The categories are:

1. **Personalization**: Forming a response that it tailored to the user.

2. **Intent resolution**: Understanding unstructured commands and drawing logical conclusions.

3. **Device resolution**: Identifying the target device based on natural language description.

4. **Persistence**: Handling commands that require persistent monitoring of system states.

5. **Command chaining**: Parsing a complex command that consists of multiple instructions, breaking it into actionable parts and executing each parts in a coherent manner.

The test cases and their categories are summarized in Table 2. They are designed to be fully automated, without a human in the loop. For this reason, we disable the human interaction tool during testing.

6.1 Baselines

To contextualize SAGE's performance, we compare our method to two LLM-based baselines on our test tasks. The first method, called *one prompt*, involves creating a single prompt comprised of the states of all the devices and the command, and asking the LLM to generate updated states in response to the user query. The full device state, serialized to JSON format, exceeds GPT-4s token limit (8000 tokens), so we manually selected the parts of the device state involved in the tests. In addition, the model was asked to output the changes that need to be made, not the full new state.

The second baseline, called "Sasha," implements the pipeline described in [7], with some modifications. The original pipeline in [7] consists of 5 pipeline states – clarifying, filtering, planning, feedback, and execution. The clarifying and feedback stages required human intervention, and were thus not compatible with our fully automated testing framework, so they were removed in our implementation. Additionally, this pipeline distinguishes between "sensors" and actionable devices, allowing the pipeline to output sensor-based trigger-action pairs to handle persistent commands. This requires the manual definition of triggers, which our testing framework does not provide, since SAGE is able to generate its own triggers by writing code. As such, our implementation of Sasha does not include the trigger concept, and is therefore unable to handle persistent commands.

Both of these baselines are handicapped in that they are not able to access as much information as SAGE (e.g. user preferences, photos of the devices,