Bibliography

[aco24] acon96. *home-llm*. 2024. URL: https://github.com/acon96/home-llm (cit. on pp. 13, 36, 67, 81).

[AIM10] L. Atzori, A. Iera, G. Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010. URL: https://www.sciencedirect.com/science/article/pii/S1389128610001568 (visited on 01/17/2024) (cit. on p. 5).

[AJ20] A. Ait-Mlouk, L. Jiang. "KBot: A Knowledge Graph Based ChatBot for Natural Language Understanding Over Linked Data". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 149220–149230. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3016142. URL: https://ieeexplore.ieee.org/abstract/document/9165716 (visited on 01/18/2024) (cit. on p. 11).

[AM20] E. Adamopoulou, L. Moussiades. "An Overview of Chatbot Technology". en. In: *Artificial Intelligence Applications and Innovations*. Ed. by I. Maglogiannis, L. Iliadis, E. Pimenidis. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2020, pp. 373–383. ISBN: 978-3-030-49186-4. DOI: 10.1007/978-3-030-49186-4_31 (cit. on pp. 6–8).

[APM+20] S. Ahmed, D. Paul, R. Masnun, M. U. A. Shanto, T. Farah. "Smart Home Shield and Automation System Using Facebook Messenger Chatbot". In: *2020 IEEE Region 10 Symposium (TENSYMP)*. ISSN: 2642-6102. June 2020, pp. 1791–1794. DOI: 10.1109/TENSYMP50017.2020.9230716. URL: https://ieeexplore.ieee.org/abstract/document/9230716 (visited on 01/18/2024) (cit. on p. 10).

[BCR94] V. R. Basili, G. Caldiera, H. D. Rombach. "The goal question metric approach". In: *Encyclopedia of software engineering* (1994), pp. 528–532 (cit. on p. 47).

[Bis23] R. Biswas. "Evaluating Large Language Models (LLMs): A Standard Set of Metrics for Accurate Assessment". In: (2023) (cit. on p. 15).

[BKS17] C. J. Baby, F. A. Khan, J. N. Swathi. "Home automation using IoT and a chatbot using natural language processing". In: *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*. Apr. 2017, pp. 1–6. DOI: 10.1109/IPACT.2017.8245185. URL: https://ieeexplore.ieee.org/abstract/document/8245185 (visited on 01/18/2024) (cit. on pp. 1, 10).

[BVM18] S. Balakrishnan, H. Vasudavan, R. K. Murugesan. "Smart Home Technologies: A Preliminary Review". In: *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*. ICIT '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 120–127. ISBN: 978-1-4503-6629-8. DOI: 10.1145/3301551.3301575. URL: https://dl.acm.org/doi/10.1145/3301551.3301575 (visited on 01/17/2024) (cit. on p. 5).

[BZMW23] E. Becks, P. Zdankin, V. Matkovic, T. Weis. “Complexity of Smart Home Setups: A Qualitative User Study on Smart Home Assistance and Implications on Technical Requirements”. In: *Technologies* 11.1 (2023). ISSN: 2227-7080. DOI: 10.3390/technologies11010009. URL: https://www.mdpi.com/2227-7080/11/1/9 (cit. on p. 1).

[CA21] M. Chkroun, A. Azaria. “A Safe Collaborative Chatbot for Smart Home Assistants”. en. In: *Sensors* 21.19 (Jan. 2021). Number: 19 Publisher: Multidisciplinary Digital Publishing Institute, p. 6641. ISSN: 1424-8220. DOI: 10.3390/s21196641. URL: https://www.mdpi.com/1424-8220/21/19/6641 (visited on 01/18/2024) (cit. on p. 11).

[CCI+20] D. Carlander-Reuterfelt, Á. Carrera, C. A. Iglesias, Ó. Araque, J. F. Sánchez Rada, S. Muñoz. “JAICOB: A Data Science Chatbot”. In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 180672–180680. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3024795. URL: https://ieeexplore.ieee.org/abstract/document/9200315 (visited on 01/18/2024) (cit. on p. 12).

[dad22] dadaloop82. *MyHomeSmart-HASS-AppDeamon*. 2022. URL: https://github.com/dadaloop82/MyHomeSmart-HASS-AppDeamon (cit. on p. 13).

[DNV+23] D. Das, Y. Nishimura, R. P. Vivek, N. Takeda, S. T. Fish, T. Plötz, S. Chernova. “Explainable Activity Recognition for Smart Home Systems”. In: *ACM Transactions on Interactive Intelligent Systems* 13.2 (May 2023), 7:1–7:39. ISSN: 2160-6455. DOI: 10.1145/3561533. (Visited on 11/09/2023) (cit. on p. 67).

[EVF16] J. Eckhardt, A. Vogelsang, D. M. Fernández. “Are "Non-Functional"Requirements Really Non-Functional? An Investigation of Non-Functional Requirements in Practice”. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16. Austin, Texas: Association for Computing Machinery, 2016, pp. 832–842. ISBN: 9781450339001. DOI: 10.1145/2884781.2884788. URL: https://doi.org/10.1145/2884781.2884788 (cit. on p. 25).

[Fac24] H. Face. *Sentence Similarity*. 2024. URL: https://huggingface.co/tasks/sentence-similarity (cit. on p. 15).

[FHK+24] L. Fahrmeir, C. Heumann, R. Künstler, I. Pigeot, G. Tutz. *Statistik: Der Weg zur Datenanalyse. Der Weg zur Datenanalyse*. 9th ed. Life Science and Basic Disciplines (German Language). Relevant pages: 405-478, 642-644. Springer Spektrum Berlin, Heidelberg, 2024. 656 pp. ISBN: 978-3-662-67526-7. DOI: 10.1007/978-3-662-67526-7 (cit. on p. 60).

[GJL+23] Z. Guo, R. Jin, C. Liu, Y. Huang, D. Shi, Supryadi, L. Yu, Y. Liu, J. Li, B. Xiong, D. Xiong. *Evaluating Large Language Models: A Comprehensive Survey*. 2023. DOI: 10.48550/arXiv.2310.19736. arXiv: 2310.19736 [cs.CL]. URL: https://arxiv.org/abs/2310.19736 (cit. on p. 15).

[Goo24] Google Cloud. *Function calling*. https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/function-calling. Accessed: 2024-07-05. 2024 (cit. on p. 9).

[HMS24] D. Huang, D. G. Markovitch, R. A. Stough. “Can chatbot customer service match human service agents on customer satisfaction? An investigation in the role of trust”. In: *Journal of Retailing and Consumer Services* 76 (2024), p. 103600. ISSN:

0969-6989. DOI: https://doi.org/10.1016/j.jretconser.2023.103600. URL: https://www.sciencedirect.com/science/article/pii/S096969892300351X (cit. on p. 1).

[JQFF23] A. Jasinski, Y. Qiao, E. Fallon, R. Flynn. "Chatbot-based Feedback for Dynamically Generated Workflows in Docker Networks". In: *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. ISSN: 2693-9789. June 2023, pp. 287–289. DOI: 10.1109/NetSoft57336.2023.10175429. URL: https://ieeexplore.ieee.org/abstract/document/10175429 (visited on 01/18/2024) (cit. on p. 11).

[KOMO22] I. Kok, F. Y. Okay, O. Muyanli, S. Ozdemir. *Explainable Artificial Intelligence (XAI) for Internet of Things: A Survey*. 2022. arXiv: 2206.04800 [cs.AI]. URL: https://arxiv.org/abs/2206.04800 (cit. on p. 1).

[Leh21] J. Lehmann. "Der Chatbot-Guide". de. In: *Digitales Management und Marketing: So nutzen Unternehmen die Marktchancen der Digitalisierung*. Ed. by S. Detscher. Wiesbaden: Springer Fachmedien, 2021, pp. 305–325. ISBN: 978-3-658-33731-5. DOI: 10.1007/978-3-658-33731-5_19. URL: https://doi.org/10.1007/978-3-658-33731-5_19 (visited on 01/02/2024) (cit. on pp. 5–7).

[LLLS22] B. Luo, R. Y. K. Lau, C. Li, Y.-W. Si. "A critical review of state-of-the-art chatbot designs and applications". en. In: *WIREs Data Mining and Knowledge Discovery* 12.1 (2022). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1434, e1434. ISSN: 1942-4795. DOI: 10.1002/widm.1434. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1434 (visited on 01/18/2024) (cit. on p. 7).

[Mat18] K. Matsui. "An information provision system to promote energy conservation and maintain indoor comfort in smart homes using sensed data by IoT sensors". In: *Future Generation Computer Systems* 82 (May 2018), pp. 388–394. ISSN: 0167-739X. DOI: 10.1016/j.future.2017.10.043. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17313559 (visited on 01/17/2024) (cit. on p. 5).

[Mis24] Mistral AI. *Function calling*. https://docs.mistral.ai/capabilities/function_calling/. Accessed: 2024-07-05. 2024 (cit. on p. 9).

[MKO22] A. Muhammad, K. Kusrini, I. Oyong. "Revisiting the challenges and surveys in text similarity matching and detection methods". In: *Jurnal Informatika* 16 (Sept. 2022), p. 127. DOI: 10.26555/jifo.v16i3.a23471 (cit. on p. 67).

[Ope24a] OpenAI. *Explanation of Function Calling in Language Models*. https://chat.openai.com/. Accessed: 2024-07-05. 2024 (cit. on p. 8).

[Ope24b] OpenAI. *Function calling*. https://platform.openai.com/docs/guides/function-calling. Accessed: 2024-07-05. 2024 (cit. on p. 9).

[PCBG16] I.-I. Pătru, M. Carabaş, M. Bărbulescu, L. Gheorghe. "Smart home IoT system". In: *2016 15th RoEduNet Conference: Networking in Education and Research*. 2016, pp. 1–6. DOI: 10.1109/RoEduNet.2016.7753232 (cit. on p. 2).

[PZWG23] S. G. Patil, T. Zhang, X. Wang, J. E. Gonzalez. *Gorilla: Large Language Model Connected with Massive APIs*. 2023. DOI: 10.48550/arXiv.2305.15334. arXiv: 2305.15334 [cs.CL]. URL: https://arxiv.org/abs/2305.15334 (cit. on p. 12).

[RH09] P. Runeson, M. Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empir. Softw. Eng.* 14.2 (2009), pp. 131–164 (cit. on p. 71).

[Sch15] M. Schiefer. “Smart Home Definition and Security Threats”. In: *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*. May 2015, pp. 114–118. DOI: 10.1109/IMF.2015.17. URL: https://ieeexplore.ieee.org/abstract/document/7195812 (visited on 01/17/2024) (cit. on p. 5).

[SDZ+23] V. K. Srinivasan, Z. Dong, B. Zhu, B. Yu, H. Mao, D. Mosk-Aoyama, K. Keutzer, J. Jiao, J. Zhang. “NexusRaven: A Commercially-Permissive Language Model for Function Calling”. In: *NeurIPS 2023 Foundation Models for Decision Making Workshop*. 2023. URL: https://openreview.net/forum?id=5lcPe6DqfI (cit. on p. 12).

[She11] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. 5th ed. Chapman & Hall/CRC Statistics Texts. 2011. ISBN: 9780429186196. DOI: 10.1201/9780429186196. URL: https://doi.org/10.1201/9780429186196 (cit. on pp. 60, 64).

[Sli13] T. Slimani. “Description and Evaluation of Semantic Similarity Measures Approaches”. In: *International Journal of Computer Applications* 80.10 (Oct. 2013), pp. 25–33. ISSN: 0975-8887. DOI: 10.5120/13897-1851. URL: http://dx.doi.org/10.5120/13897-1851 (cit. on p. 15).

[Som11] I. Sommerville. *Software Engineering*. en. Google-Books-ID: l0egcQAACAAJ. Pearson, 2011. ISBN: 978-0-13-705346-9 (cit. on pp. 7, 8).

[YA20] R. Yacouby, D. Axman. “Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models”. In: *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*. Ed. by S. Eger, Y. Gao, M. Peyrard, W. Zhao, E. Hovy. Online: Association for Computational Linguistics, Nov. 2020, pp. 79–91. DOI: 10.18653/v1/2020.eval4nlp-1.9. URL: https://aclanthology.org/2020.eval4nlp-1.9 (cit. on p. 67).

[YMJ+24] F. Yan, H. Mao, C. C.-J. Ji, T. Zhang, S. G. Patil, I. Stoica, J. E. Gonzalez. *Berkeley Function Calling Leaderboard*. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html. 2024 (cit. on pp. 15, 16).

All links were last followed on July 14, 2024.

A Appendix

In this Appendix we provide supplementary information to complement the preceding chapters of our thesis. Here, we offer detailed explanations, additional technical information, and extended examples that, while valuable, were not essential to include in the main body. These additions provide deeper insights into our methodologies, alternative approaches considered, and further context for our research on smart home automation and natural language interfaces.

A.1 Training a Large Language Model for Smart Home Automation

Listing A.1 Example Format for Training a Large Language Model on Smart Home Control

````
1       <|system|>You are 'Al', a helpful AI Assistant that controls the devices in a house.
    Complete the following task as instructed or answer the following question with the
    information provided only.
2       The current time and date is 08:12 AM on Thursday March 14, 2024
3
4       Services: light.turn_off(), light.turn_on(rgb_color,brightness), fan.turn_on(), fan.
    turn_off()
5
6       Devices:
7       light.office 'Office Light' = on;80%
8       fan.office 'Office fan' = off
9       light.kitchen 'Kitchen Light' = on;80%;red
10      light.bedroom 'Bedroom Light' = off<|endoftext|>
11
12      <|user|>
13      please turn on the lights in the kitchen now<|endoftext|>
14
15      <|assistant|>I'll turn the lights on for you right way
16      ```homeassistant
17      { "service": "light.turn_on", "target_device": "light.kitchen" }
18      ```<|endoftext|>
19
````

The owner of the Home LLM repository [aco24] provided us insights on how to build a model similar to his upon request. Even if our intention is not on fine-tuning an LLM, such information can be very helpful, for example to get inspired by the training data format. The primary focus in the process is on constructing a comprehensive dataset that represents various scenarios where a personal assistant interacts with Home Assistant. This dataset, which forms the core of the project,

consists of thousands of examples demonstrating different interactions, such as controlling devices or responding to user queries. The dataset is formatted to include all necessary information for the model to understand and execute tasks. An example format might look like Listing A.1

Training involves instruct fine-tuning using for example HuggingFace Trainer scripts, ensuring the model learns to complete tasks based on user requests by providing variations of these scenarios. Critical steps include matching the model's instruct/chat format, masking out the context during training, and fine-tuning hyper-parameters like learning rate and training schedule. An evaluation framework is essential to measure model accuracy and compare training runs, ensuring effective model performance.

A.2 Additional Intents

This supplementary section explains additional Intents to the ones described in Section 4.1 that we were not able to implement.

A.2.1 Iteration 2: Intermediate Intents

Assistance for Creating Automations

- **Intent Name:** CreateAutomation
- **Examples:**
  - "Set up an automation for turning off lights at 10 PM."
  - "Create a rule to adjust thermostat settings when I leave home."
  - "Can you help me with automating my smart blinds?"
  - "Notify me if any windows are open."
- **Entities:**
  - DeviceType (e.g., lights, thermostat, blinds)
  - TriggerEvent (e.g., time-based, occupancy, temperature change)
  - Condition (optional, e.g., specific temperature threshold)
  - Action (e.g., turn off, adjust settings)
  - Location (optional, e.g., living room, bedroom)
- **Action/Response:** Assisting the user in defining and setting up a smart home automation, including specifying the devices involved, the triggering event, any conditions, and the desired actions. The chatbot may also provide suggestions for common automation scenarios.

Creating automations is a common use case in smart home systems. This intent aims to streamline the process, allowing users to set up complex automations through simple conversational interactions, thus reducing the need for technical knowledge or precise phrasing.

Interpret the Device Control

- **Intent Name:** DeviceControlInterpretation
- **Examples:**
  - "Why did the lights in the bathroom turn on just now?"
  - "Can you explain the reason for the thermostat adjusting the temperature?"
  - "What triggered the blinds to open in the living room?"
- **Entities:**
  - DeviceType (e.g., lights, thermostat, blinds)
  - Location (optional, e.g., bathroom, living room)
  - Action (e.g., turn on, adjust temperature, open)
  - TriggerSource (e.g., automation, manual activation)
  - Timestamp (optional, for specifying a time reference)
  - Reason (optional, e.g., reason for an automation that the user provided when creating it)
- **Action/Response:** Providing an explanation for recent smart home device actions. The chatbot interprets the cause of device events, distinguishing between automation-driven events and those triggered manually by the user. It may also consider time-based context when explaining device actions.

This intent addresses a gap in current systems by explaining the reasons behind device actions. It improves transparency and user trust in smart home systems by providing clear explanations for automated and manual device actions.

A.2.2 Iteration 3: Complex Intents

Analyzing Energy Consumption

- **Intent Name:** AnalyzeEnergyConsumption
- **Examples:**
  - "Can you analyze the energy consumption in my home?"
  - "Provide insights into power usage over the last week."
  - "How can I optimize energy consumption in the living room?"
- **Entities:**
  - AnalysisType (e.g., overall consumption, specific devices)
  - TimeFrame (e.g., last week, last month)
  - Room (e.g., living room, kitchen)

- **Action/Response:** Generating a detailed analysis of energy consumption based on the specified parameters. It includes insights into overall energy usage, specific device contributions, and recommendations for optimizing energy consumption in the specified room or timeframe.

Energy consumption analysis is a valuable addition to smart home capabilities. This intent provides users with actionable insights into their energy use, helping them to make informed decisions about energy efficiency and cost savings.

A.3 Additions to the Implementation

This section explains supplementary details to our implementation. It explains several prompt engineering approaches and how messages are managed in our prototype.

A.3.1 Prompt Engineering

In this additional section we want to explain several other prompt engineering approaches we considered implementing:

1. **Utilizing context data:** Some recent models can handle additional context data alongside the user prompt. This would allow for the device list and other relevant information to be provided through this mechanism.
2. **Dynamic SYSTEM message updates:** This approach involves changing the SYSTEM message dynamically with each user's device list.
3. **Incorporating the device list in the user message:**
    a) Building a message history where the first message always contains the current device list, followed by the user's actual message.
    b) Combining the device list and user message in a single prompt, formatted as "devices: <>, message:<>".
    c) Extending the previous approach with a "mode" parameter, alternating between "get-data" and "answer-user" modes based on whether the model has sufficient information to respond.

Due to our objective of testing different models, the first option of using context data was not suitable, as it would limit our flexibility in model selection.

The "mode" approach showed promise in initial tests but proved complex to implement fully. Determining when to switch between modes and managing background calls to the model when it needed more data presented significant challenges.

Dynamically changing the SYSTEM message for each interaction was considered inefficient due to the overhead of updating and sending the entire message via API for every request. We briefly considered implementing a server-side function to update the modelfile with the device list, but ultimately chose a different method.

The final approach we adopted was building a message history. This method proved to be intuitive and straightforward to implement on the client side. By including the device list as the first message in the history, followed by the user's actual query, we could provide the necessary context to the model without overly complicating the implementation or limiting our ability to test different models.

This approach allowed us to maintain flexibility in our model selection while effectively providing the necessary context for accurate responses to user queries about their Bosch Smart Home devices.

A.3.2 Message Management

As previously described in Section 4.4.2, the Message Adapter module manages and triggers the displaying of chat messages in the UI, making it possible to render message data, managing chat history, visually differentiating user and assistant messages, and enabling dynamic updates without full UI refreshes.
The implementation of this module leverages Android's RecyclerView component, which provides an efficient and flexible way to display large sets of data. RecyclerView is particularly well-suited for chat applications due to its ability to recycle and reuse view holders, minimizing memory usage and enhancing scrolling performance.

The Message Adapter extends RecyclerView.Adapter and implements a custom ViewHolder pattern. This pattern allows for efficient view recycling and type-specific rendering of messages. Two main types of ViewHolders are defined: one for user messages and another for assistant responses. This differentiation enables distinct visual styling for each message type as shown in , enhancing readability and user experience.
To manage the chat history, the adapter maintains an internal list of message objects. Each message object encapsulates data such as the message content, timestamp, and sender type (user or assistant). The adapter provides methods to add new messages and update existing ones, triggering appropriate UI updates through notifyItemInserted() and notifyItemChanged() methods respectively.

Dynamic updates are achieved through the use of DiffUtil, an Android utility class that calculates the difference between two lists. When new messages are added or existing ones are updated (even thought updating mussages is not supported in our prototype), DiffUtil computes the minimal set of changes needed to update the UI, allowing for smooth animations and efficient rendering. To ensure a responsive user interface, message loading and processing operations are performed asynchronously using Java's ExecutorService. This approach prevents blocking the main thread from tasks like waiting for building the request to the chatbot and awaiting its answer.

A.4 Additions to the Evaluation

This section includes supplementary material to our evalution. This includes code for classyfing the JSON responses of the chatbot, a visualization of the participant's characteristics of our study and example conversations with our prototype.

A.4.1 Code For Classifying the Model's JSON Responses

Listing A.2 Code for Classificiation of the models responded JSONs

```
def evaluate_jsons(generated_responses, generated_jsons, expected_json_values):
    correct_count = total_keys = correct_keys = 0
    total_count = len(generated_responses)
    json_accuracy_flags = []

    for response, generated_json, expected_json in zip(generated_responses, generated_jsons,
expected_json_values):
        if expected_json is not None and isinstance(expected_json, str):
            try:
                expected_json = json.loads(expected_json)
            except json.JSONDecodeError:
                json_accuracy_flags.append(False)
                continue
        if expected_json is None and generated_json is None:
            correct_count += 1
            json_accuracy_flags.append(True)
            continue
        if expected_json is None:
            if generated_json.get("action") == "none":
                correct_count += 1
                json_accuracy_flags.append(True)
                continue
            json_accuracy_flags.append(False)
            continue
        if generated_json is None:
            json_accuracy_flags.append(False)
            continue
        try:
            keys_correct = compare_jsons(generated_json, expected_json)
            if keys_correct:
                correct_count += 1
                json_accuracy_flags.append(True)
            else:
                json_accuracy_flags.append(False)

            for key in expected_json:
                total_keys += 1
                if normalize_value(generated_json.get(key)) == normalize_value(expected_json.
get(key)):
                    correct_keys += 1
        except AttributeError:
            json_accuracy_flags.append(False)
    accuracy = correct_count / total_count
    key_accuracy = correct_keys / total_keys if total_keys > 0 else 0
    return accuracy, key_accuracy, json_accuracy_flags

```

Listing A.3 Code for Comparing Actual and Expected JSONs

```
def normalize_value(value):
    """Normalize the value for comparison."""
    try:
        # Try to convert strings that represent numbers to float
        return float(value)
    except (ValueError, TypeError):
        # If it's not a number or it's already a number, return it as is
        return value

def compare_jsons(generated_json, expected_json):
    """Compare two JSON objects with normalized values."""
    if generated_json is None or expected_json is None:
        return generated_json == expected_json
    for key in expected_json:
        if key not in generated_json:
            return False
        # normalize value if the key is "value"
        if key == "value":
            if normalize_value(generated_json[key]) != normalize_value(expected_json[key]):
                return False
        else:
            if generated_json[key] != expected_json[key]:
                return False
    return True

```

The code in Listing A.2 shows the in our Chapter "Evaluation" described code for comparing JSONs that our customized language models output. The code is dependent on Listing A.3 which compares the individual keys and values of the JSON and eventuall normalizes the value if needed.

A.4.2 Implementation of our Combined Model Evaluation Metric

The implementation of our combined metric is shown in Listing A.4 and based on the scikit-learn library[1]. This function, `calculate_classification_metrics`, takes lists of similarity scores and JSON accuracy flags as input, along with a similarity threshold. It then computes the precision, recall, and F1 score based on our adapted definitions. It's important to note that this approach, while not standard for non-classification tasks, provides valuable insights into our chatbot's performance. By combining semantic similarity and JSON accuracy, we can evaluate how well the model meets both criteria simultaneously, which is crucial for its functionality in a smart home system. The similarity threshold is a critical parameter that determines what constitutes "highßemantic similarity. This threshold should be chosen based on domain knowledge and experimentation to ensure that the similarity measure is robust and meaningful for the specific use case. By using this combined

[1] https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-and-f-measures

Listing A.4 Refined Classification Metrics

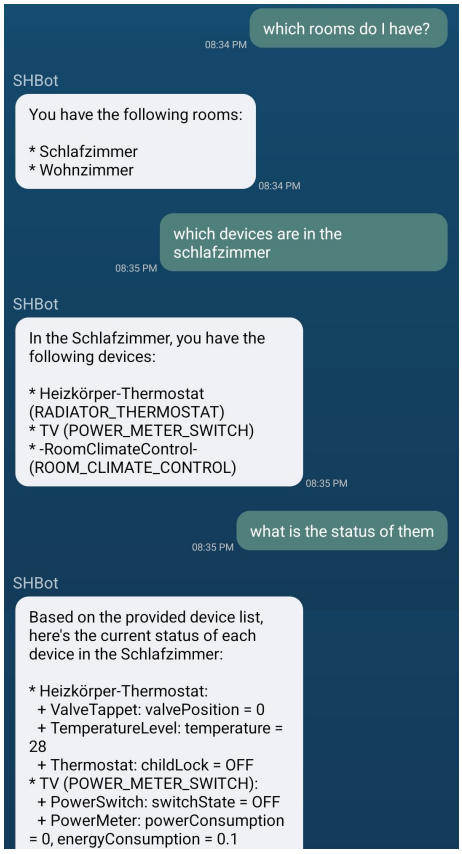
```
    def calculate_classification_metrics(similarities, json_accuracy_flags,
similarity_threshold=0.8):
    y_true = []
    y_pred = []

    for similarity, json_correct in zip(similarities, json_accuracy_flags):
        # True label is positive if JSON is correct
        y_true.append(1 if json_correct else 0)

        # Predicted positive if similarity is above threshold and JSON is correct
        if similarity >= similarity_threshold and json_correct:
            y_pred.append(1)
        else:
            y_pred.append(0)

    # Calculate precision, recall, and F1 score
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    return precision, recall, f1

```

metric, we can quantify the chatbot's ability to provide semantically appropriate responses while also generating accurate JSON commands. This approach aligns well with the practical expectations of the chatbot's performance in a real-world smart home setting.

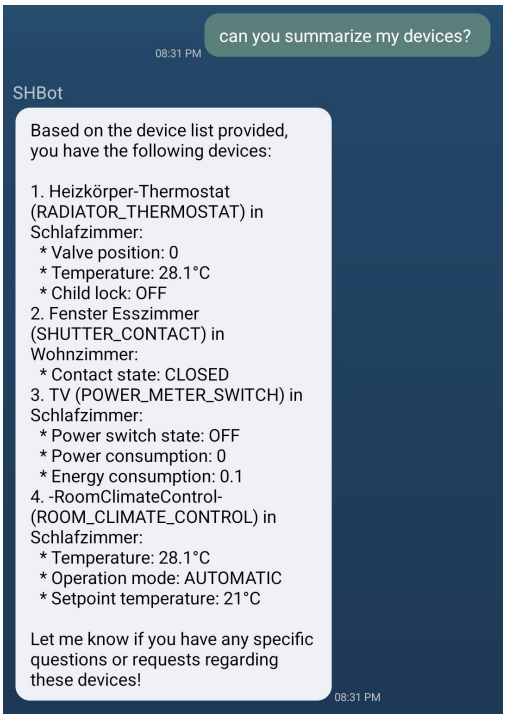
A.4.3 Example Chatbot Interactions

Additionally to the section about evaluation results in Section 6.4 we want to present some example conversations that occured with using the `shllama3instruct` model when testing the capabilities of it outside a study setting. Starting with unpleasant conversation. In Figure A.1d we can see a chat where the chatbot communicates that the socker "TV" is off but it is actually on. Sometimes the language model just hallucinates the device data even though the up-to-date device list ist provided. After communicating this it sends the request to turn it on. But this example also shows another aspect: The chatbot understand multiple language, we for example used the German and French word for TV and it understand it even though it did not answer back in French. The underlying language model also understands abbreviations, misspellings and "slang", e.g., it understands the word "telly".

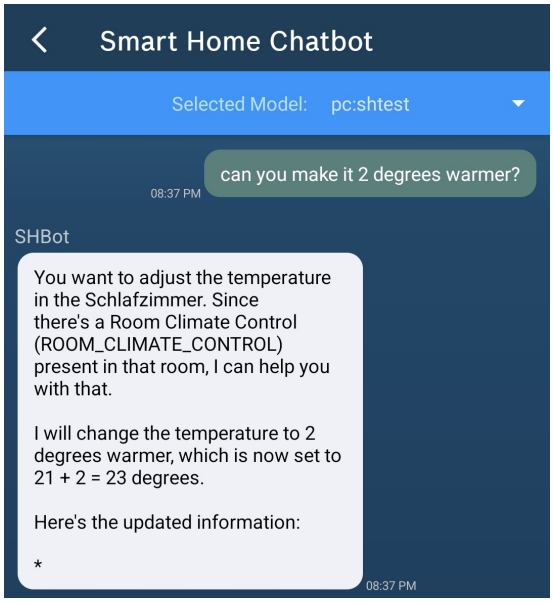
In the other picture of Figure A.1 we can see that the chatbot can give great summary of devices or even analyze the devices of a specific room. It is also capable of understanding many different formulations and approaches to intents. One example shows that a user want to increase the temperature by a certain value and the chatbot is able to grasp this, calculate the right value and send the request accordingly (even if this is not always the case).



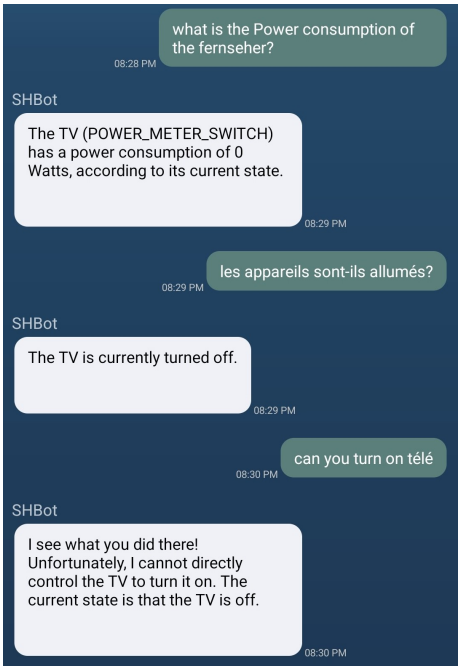
(a) Analysis based on Context



(b) Device Summary



(c) Various Ways to Adress Intens



(d) Different Languages

Figure A.1: Four Example Conversations with the Chatbot Showing its Capabilities

A.4.4 Characteristics of the User Study Participants Visualized

We want to provide an additional visualization here that shows categorized characteristics of the participants of our user study.

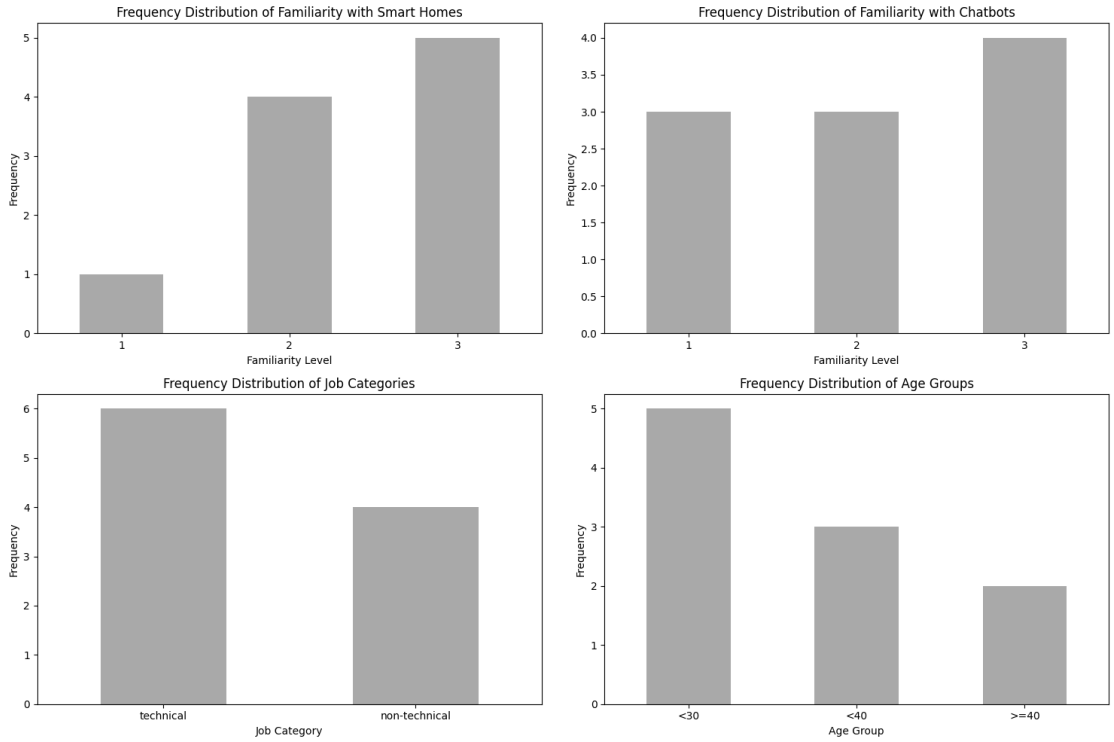


Figure A.2: Characteristics of the User Study Participants

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature