On-Device LLMs for Home Assistant: Dual Role in Intent Detection and Response Generation

Rune Birkmose[*] **Nathan Mørkeberg Reece**[*] **Esben Hofstedt Norvin**[*]
Johannes Bjerva **Mike Zhang**[†]
Aalborg University, Denmark
[*]`{rbirkm20, nreece20, enorvi20}@student.aau.dk` [†]`jjz@cs.aau.dk`

Abstract

This paper investigates whether Large Language Models (LLMs), fine-tuned on synthetic but domain-representative data, can perform the twofold task of (i) slot and intent detection and (ii) natural language response generation for a smart home assistant, while running solely on resource-limited, CPU-only edge hardware. We fine-tune LLMs to produce both JSON action calls and text responses. Our experiments show that 16-bit and 8-bit quantized variants preserve high accuracy on slot and intent detection and maintain strong semantic coherence in generated text, while the 4-bit model, while retaining generative fluency, suffers a noticeable drop in device-service classification accuracy. Further evaluations on noisy human (non-synthetic) prompts and out-of-domain intents confirm the models' generalization ability, obtaining around 80–86% accuracy. While the average inference time is 5–6 seconds per query—acceptable for one-shot commands but suboptimal for multi-turn dialogue—our results affirm that an on-device LLM can effectively unify command interpretation and flexible response generation for home automation without relying on specialized hardware.

1 Introduction

Smart home technologies and IoT devices have proliferated in recent years, with an expected rise from 16.6 billion to 18.8 billion connected devices by the end of 2024 (IoT Analytics, 2024). Major providers like Amazon, Google, and Apple typically handle speech recognition and intent detection on cloud servers, which raises user concerns about privacy, data ownership, and reliance on proprietary ecosystems (BBC News, 2025). Conventional solutions for home assistants often rely on specialized, domain-specific classifiers for slot and intent detection (SID), paired with templated system responses. While these approaches can be efficient, they can also be rigid, sometimes requiring precisely phrased user inputs and yielding repetitive or unpersonalized answers.

Recent developments in on-device computing—coupled with improvements in model compression and quantization (Liang et al., 2021; Gholami et al., 2022; Lang et al., 2024)—have paved the way for smaller yet still capable language models to run on commodity hardware. These models offer privacy benefits and allow customizable local inference with reduced latency. However, deploying a capable model under strict memory and computational constraints remains challenging. Large-scale Transformer-based language models (Vaswani et al., 2017), and especially LLMs (Touvron et al., 2023; Dubey et al., 2024; Bai et al., 2023; Yang et al., 2024; Groeneveld et al., 2024), have demonstrated remarkable proficiency in tasks ranging from question answering to text generation (Arora et al., 2024; Yin et al., 2024), yet typically demand substantial hardware resources, restricting them to cloud-based services or large compute clusters.

This paper explores whether a smaller, fine-tuned LLM can provide two capabilities essential to a home assistant—accurate recognition of *what* users want (i.e., slot and intent detection), and *natural* textual responses—while running entirely on an edge device with limited CPU and memory. By unifying these tasks into one end-to-end system, we eliminate the need for separate domain-specific classification modules and templated responses, focusing on efficiency, robust language understanding, and strict correctness in JSON action output.

Additionally, we move away from classic SID datasets and other general spoken language understanding benchmarks. Instead, we investigate whether LLMs can be directly applied to digital assistant software. To this end, we take the open-

[*]The authors contributed equally to this work.

source Home Assistant software[1] as our gold standard for evaluation, targeting real-world device-service pairs and actionable JSON outputs.

Contributions. Our contributions can be summarized as follows:[2] ① We show that a 0.5B LLM can be fine-tuned to already jointly handle SID *and* response generation with high accuracy. ② By quantizing the model (from 16-bit to 8-bit and 4-bit), we quantify trade-offs between memory usage, accuracy, and generative fluency on CPU-only edge hardware. ③ We evaluate the approach on synthetic data, human queries, and out-of-domain tasks, confirming robust generalization.

2 Related Work

Slot and Intent Detection. Traditional approaches to spoken language understanding (SLU) often treat SID separately using domain-specific classification or sequence tagging approaches (Zhang and Wang, 2016; Wang et al., 2018; Weld et al., 2022; Qin et al., 2021; Pham et al., 2023). More recent transformer-based solutions unify both tasks, leveraging contextual embeddings to improve performance (Castellucci et al., 2019; van der Goot et al., 2021; Stoica et al., 2021; Arora et al., 2024) with models like BERT (Devlin et al., 2019). However, many of these solutions still presume tailored sequence labeling datasets or full-size transformer backends. Our work aligns with the shift to more expressive transformer models for SLU, but we push inference to a local environment while also adding dynamic text generation.

Running LLMs on Edge Devices. While training large-scale LLMs remains computationally expensive, numerous works explore strategies for *deploying* them on edge hardware. Haris et al. (2024) propose FPGA-based accelerators to reduce memory overhead for LLM inference. Zhang et al. (2024) distribute an LLM across multiple low-power devices to increase throughput. An empirical footprint study by Dhar et al. (2024) shows that even 7B-parameter models can strain embedded hardware if not sufficiently compressed. Our approach uses a much smaller LLM (0.5B–1.5B parameters) plus weight quantization, showing that near-commodity devices with 8GB RAM can handle both intent classification and text generation if the domain is sufficiently specialized.

[1]`https://github.com/home-assistant/core`

[2]We release all our code and models at `https://github.com/Run396/P9`.

Partition	Train	Test	Total
Classification	23,372	5,843	29,215
LLM	33,361	2,435	35,796

Table 1: **Aggregated Train/Test Splits.** For the classification baseline, 20% of the original training set was used as test data (after removing multi-intent samples). The LLM used the full synthetic data; 2,435 remain as test.

3 Methodology

Our goal is to integrate two core functionalities of a home assistant into a single model:

- **Slot and Intent Detection:** The model outputs a valid JSON object that maps to a desired service (intent) and device (slot) pair:

```
{"service": "light.turn_on",
  "device": "light.living_room",
  "assistant": "Sure, turning on
    on the living room light."}
```

- **Natural Language Generation:** The model also produces a textual response confirming or elaborating on its action, as can be seen in the example above. The text can then be propagated to, e.g., a text-to-speech model.

Traditional classifiers only handle device-service classification and do not produce any text. For user-facing text, the baseline approach would rely on templated responses.

3.1 Data and Pre-processing

To the best of our knowledge, there is no existing human-curated dataset specifically for the Home Assistant software. Thus, we rely on synthetic data. We use a publicly available synthetic dataset (acon96, 2024), which consists of 35,840 synthetic examples designed to mimic Home Assistant commands. Each instance consists of:

- A **User Prompt:** e.g., *"Turn on the kitchen light"*, *"Set the thermostat to 22 degrees"*.
- One **Valid JSON Action**, containing the `service` and `device` fields corresponding to Home Assistant calls.
- A **Natural Language Response:** e.g., a paraphrase or affirmation of the action taken.

Model	Accuracy	BERTScore
Baselines		
SVC Classifier		
Service	76.6	—
Device	45.4	—
DistilBERT		
Service	98.8	—
Device	47.9	—
Qwen2.5-0.5B (16-bit)	98.8	0.84
Qwen2.5-0.5B (8-bit)	98.4	0.79
Qwen2.5-0.5B (4-bit)	81.7	0.88
Qwen2.5-1.5B (16-bit)	96.9	0.84
Qwen2.5-1.5B (8-bit)	96.5	0.83
Qwen2.5-1.5B (4-bit)	90.7	0.82

Table 2: **Slot/Intent Detection and NLG Results on Synthetic Test Data.** Accuracy is based on exact JSON match. BERTScore measures semantic similarity of the generated text vs. gold reference.

Model	CPU	T/Q (s)	Load (s)
Baselines			
SVC Classifier	4	<1	—
DistilBERT	4	<1	—
Qwen2.5-0.5B (16-bit)	4	6.25	±3.2
Qwen2.5-1.5B (16-bit)	4	10.81	±5.6
Qwen2.5-0.5B (8-bit)	4	5.50	±3.2
Qwen2.5-1.5B (8-bit)	4	10.32	±5.6
Qwen2.5-0.5B (16-bit)	2	8.49	±5.6
Qwen2.5-1.5B (16-bit)	2	17.72	±5.6
Qwen2.5-0.5B (8-bit)	2	7.89	±5.6
Qwen2.5-1.5B (8-bit)	2	16.11	±5.6

Table 3: **Computation Time.** Mean time per query (T/Q) across 500 samples under different CPU core counts and quantization levels. Load time is model initialization.

A full example can be found in Figure 1 (Appendix A). We stratify the dataset, maintaining the inherent imbalance (some device types and services appear more frequently). There are 38 service labels and 858 device labels. We split into training and test sets as shown in Table 1. The final training set for the LLM includes ∼33k examples, and we set aside 2,435 synthetic samples for evaluation. Note that for the classification-based baselines, we split up the train and test set to separately predict `service` and `device` instead of as one prediction, ending up with double the test data (5,843 samples; excluding multi-intent examples). The input consists of only the user message and leave the system message out. A more detailed distribution of the data can be found in Table 6 (Appendix B).

3.2 Models

Baseline Classifiers. We train a Linear SVC from Scikit-Learn (Pedregosa et al., 2011) on TF-IDF features of the user prompt. The classifier outputs a concatenated device-service pair, which is then wrapped in JSON. Additionally, we fine-tune DistilBERT (Sanh et al., 2019) for classification. We use the transformers library (Wolf et al., 2020) for fine-tuning. We train for 1 epoch using a learning rate of $3{\times}10^{-4}$ with the AdamW optimizer, and a batch size of 64 on a NVIDIA A10 (24GB) GPUs. Both models have no generative capability, so user-facing text is templated.

Small Large Language Models. We train using a chat-style format with user–assistant pairs. We primarily use the Qwen2.5-0.5B-Instruct model and the Qwen2.5-1.5B-Instruct (Yang et al., 2024). We fine-tune both models for one epoch with a batch size of 4, using the AdamW optimizer at a learning rate of 2×10^{-5} with a cosine scheduler. The maximum sequence length is set to 2,048 tokens. We use the HuggingFace Transformers library (Wolf et al., 2020) for training on NVIDIA L4 (24 GB) GPUs.

Quantization. After fine-tuning and having the original 16-bit model, we produce two quantized versions of each model: NF8 and NF4 (Dettmers et al., 2024), using bitsandbytes.[3] This allows us to compare accuracy, generative quality, and inference speed under varying memory constraints.

3.3 Evaluation

Slot-Intent Detection Accuracy. SID must be correct with near-exact string matching, as JSON calls are consumed downstream by the home automation system. We thus parse the model output for the `service` and `device` fields; if they match the gold annotation exactly, it is counted as correct. Any mismatch or invalid JSON results in an error.

For the classification task, instead, we separately predict `service` and `device` using the same classification model and take the average accuracy.

Text Generation Quality. For the natural language responses using the LLMs, we compare each generated response to the reference using BERTScore (Zhang et al., 2020).

[3]See `https://github.com/bitsandbytes-foundation/bitsandbytes`. We also use double quantization.

Model	Accuracy	BERTScore
Qwen2.5-0.5B	80.0	0.76
Qwen2.5-1.5B	86.7	0.74

Table 4: **Results Out-of-Domain Queries.** Accuracy and BERTScore over 60 OOD samples.

Model	Accuracy	BERTScore
Qwen2.5-0.5B	84.0	0.68
Qwen2.5-1.5B	86.4	0.66

Table 5: **Results Human-Generated Queries.** Accuracy and BERTScore over 81 real-user queries.

Inference Environment. We simulate a CPU-only setup on an 8 GB RAM device with up to four CPU cores. We measure average inference time on a 500-sample subset, varying both quantization level and the number of CPU cores.

4 Results

4.1 Slot and Intent Detection

Table 2 shows the SID performance of both the 0.5B and 1.5B LLMs under various quantization levels, alongside the baseline SVC and DistilBERT. For the 0.5B model, the 16-bit and 8-bit variants reach near-perfect accuracy ($\sim$ 99%). The 4-bit version drops to 81.7%, which is still better than the the SVC baseline (average 61.0% accuracy) and DistilBERT baseline (average 73.4% accuracy).

Interestingly, for the larger 1.5B model, the 16-bit and 8-bit variants achieve 96.9% and 96.5% accuracy, respectively, while the 4-bit version gets 90.7%. Thus, while the smaller 0.5B model actually yields higher raw accuracy in-domain, the 1.5B model remains competitive and in some out-of-domain tests (next section) performs better.

4.2 Natural Language Generation

Although the 4-bit models suffer in SID accuracy, Table 2 shows that the 0.5B 4-bit variant has the highest BERTScore (0.88). This indicates that while it may misclassify device/service fields, the generative text can still be fluent and semantically close to the target. Meanwhile, the 8-bit versions drop in BERTScore for the 0.5B model (0.79) and remain steady for the 1.5B model (0.83). Qualitative samples show that small changes in quantization can shift the style and lexical choices of the generated text.

4.3 Inference Time and Memory

Table 3 summarizes the inference speed across model size, quantization, and CPU core settings. The 8-bit model is only slightly faster than the 16-bit model (5.5 s vs. 6.25 s on 4 cores for the 0.5B). Doubling CPU cores from 2 to 4 reduces latency roughly by half. The 1.5B model takes longer (up to 10–17 s per query), which may be borderline for real-time usage in multi-turn dialogues.

4.4 Out-of-Domain Intents

In Table 4, we evaluate 60 OOD queries that mention either novel device types or services not appearing in the training set. The 0.5B model scores 80.0% accuracy vs. 86.7% for the 1.5B model, with BERTScores of 0.76 and 0.74 respectively. The results suggest that the 1.5B model generalizes somewhat better to unfamiliar domains, though both degrade compared to in-domain performance.

4.5 Human Prompts

Finally, we tested each model on 81 human-written prompts. Ten participants (ages 23–69) contributed typical commands they would issue to a home assistant, including incomplete or ambiguous phrasing. Table 5 shows that the 0.5B model achieves 84.0% accuracy, whereas the 1.5B model is slightly higher at 86.4%. BERTScores are around 0.66–0.68. The gap vs. synthetic data reflects real-user queries with more variation and noisy data.

5 Discussion

Despite near-perfect performance on the synthetic test set, Table 4 and 5 reveal a drop to 80–86% accuracy in real or out-of-domain queries. This discrepancy likely stems from the difficulty of handling spontaneous human phrasing, missing location or device details, and genuinely novel device types. Still, the results surpass the SVC and DistilBERT baseline.

Interestingly, while the 4-bit model can generate fluent natural language responses (often scoring the highest BERTScore in the 0.5B case), its classification accuracy suffers. This underscores that quantizing a model to extreme levels can degrade structured predictions more than open-ended text generation.

Regarding speed, the 1.5B model yields consistent accuracy gains on OOD data but also increases inference time by up to 2–3$\times$. For single-turn commands, 5–6 seconds per query might be acceptable, but multi-turn dialogue would require faster or more efficient strategies. Future work may explore parameter-efficient fine-tuning, context truncation, or advanced quantization (e.g., 8-bit + partial 4-bit layering) to reduce inference times.

6 Conclusion

We present that LLMs can simultaneously perform SID and natural language response generation for a home automation domain. Experiments on an 8GB RAM, CPU-only environment show that 8-bit quantization largely preserves in-domain accuracy (up to 99%) and strong text fluency, while 4-bit introduces significant classification errors despite retaining good generative capability. We further demonstrate promising generalization to human-written prompts and out-of-domain tasks, with accuracy around 80–86%. However, per-query inference times of 5–6 seconds indicate that LLM-based assistants, as implemented here, are not yet ideal for fast multi-turn dialogues on edge devices. Future work can refine these models for faster, more memory-efficient inference, enabling privacy-preserving yet flexible home automation assistants.

Limitations

Our use of synthetic data may limit the diversity of user prompts; while we partially mitigated this with human-written queries, data coverage remains a challenge. The model also relies on structurally valid JSON output. Real-world usage may need fallback logic to handle malformed or incomplete responses. Moreover, we focus on a single domain (home automation); scaling to broader or open-ended tasks likely requires larger models and may degrade performance under CPU-only constraints.

Ethical Considerations

We do not foresee any major ethical issues with this work. The primary domain is home automation, and the dataset is synthetic or user-provided under informed consent. Nonetheless, deploying generative models in user-facing applications requires caution regarding hallucinated or incorrect responses, as well as user data privacy.

Acknowledgments

MZ and JB are supported by the research grant (VIL57392) from VILLUM FONDEN.

References

acon96. 2024. Home Assistant Requests Dataset. `https://huggingface.co/datasets/acon96/Home-Assistant-Requests`. Accessed: 2025.

Gaurav Arora, Shreya Jain, and Srujana Merugu. 2024. Intent detection in the age of llms. *arXiv preprint arXiv:2410.01627*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *ArXiv preprint*, abs/2309.16609.

BBC News. 2025. Apple to pay $95m to settle siri 'listening' lawsuit.

Giuseppe Castellucci, Valentina Bellomaria, Andrea Favalli, and Raniero Romagnoli. 2019. Multi-lingual intent detection and slot filling in a joint bert-based model. *arXiv preprint arXiv:1907.02884*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Nobel Dhar, Bobin Deng, Dan Lo, Xiaofeng Wu, Liang Zhao, and Kun Suo. 2024. An empirical analysis and resource footprint study of deploying large language models on edge devices. *Proceedings of the ACM*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *ArXiv preprint*, abs/2407.21783.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, William Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah Smith, and Hannaneh Hajishirzi. 2024. OLMo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15789–15809, Bangkok, Thailand. Association for Computational Linguistics.

Jude Haris, Rappy Saha, Wenhao Hu, and José Cano. 2024. Designing efficient llm accelerators for edge devices. *arXiv preprint arXiv:2408.00462*.

IoT Analytics. 2024. Number of Connected IoT Devices. `https://iot-analytics.com/number-connected-iot-devices/`. Accessed: 2024-10-16.

Jiedong Lang, Zhehao Guo, and Shuyu Huang. 2024. A comprehensive study on quantization techniques for large language models. *arXiv preprint arXiv:2411.02530*.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Thinh Pham, Chi Tran, and Dat Quoc Nguyen. 2023. MISCA: A joint model for multiple intent detection and slot filling with intent-slot co-attention. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12641–12650, Singapore. Association for Computational Linguistics.

Libo Qin, Tianbao Xie, Wanxiang Che, and Ting Liu. 2021. A survey on spoken language understanding: Recent advances and new frontiers. *arXiv preprint arXiv:2103.03095*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Anda Stoica, Tibor Kadar, Camelia Lemnaru, Rodica Potolea, and Mihaela Dînşoreanu. 2021. Intent detection and slot filling with capsule net architectures for a romanian home assistant. *Sensors*, 21(4):1230.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint*, abs/2307.09288.

Rob van der Goot, Ibrahim Sharaf, Aizhan Imankulova, Ahmet Üstün, Marija Stepanović, Alan Ramponi, Siti Oryza Khairunnisa, Mamoru Komachi, and Barbara Plank. 2021. From masked language modeling to translation: Non-English auxiliary tasks improve zero-shot spoken language understanding. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2479–2497, Online. Association for Computational Linguistics.

A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A Gomez, L Kaiser, and I Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems (NIPS)*.

Yu Wang, Yilin Shen, and Hongxia Jin. 2018. A bi-model based RNN semantic frame parsing model for intent detection and slot filling. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 309–314, New Orleans, Louisiana. Association for Computational Linguistics.

Henry Weld, Xiaoqi Huang, Siqu Long, Josiah Poon, and Soyeon Caren Han. 2022. A survey of joint intent detection and slot filling models in natural language understanding. *ACM Computing Surveys*, 55(8):1–38.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

An Yang, Baosong Yang, and Beichen Zhang. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

Shangjian Yin, Peijie Huang, Yuhong Xu, Haojing Huang, and Jiatian Chen. 2024. Do large language models understand multi-intent spoken language? *arXiv preprint arXiv:2403.04481*.

Mingjin Zhang, Jiannong Cao, Xiaoming Shen, and Zeyang Cui. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *arXiv preprint arXiv:2405.14371*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

Xiaodong Zhang and Houfeng Wang. 2016. A joint model of intent determination and slot filling for spoken language understanding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 16, pages 2993–2999.

A Data Examples

We show a data example in Figure 1.

~~~
[
    {
      "from": "system",
      "value": "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.\nServices: cover.close_cover(),
      cover.open_cover(), cover.stop_cover(), cover.toggle(), lock.lock(),
      lock.unlock(), media_player.media_next_track(), media_player.media_pause(),
      media_player.media_play(), media_player.media_play_pause(),
      media_player.media_previous_track(), media_player.media_stop(),
      media_player.toggle(), media_player.turn_off(), media_player.turn_on(),
      media_player.volume_down(), media_player.volume_mute(),
      media_player.volume_up(), switch.toggle(), switch.turn_off(),
      switch.turn_on(), timer.cancel(), timer.pause(), timer.start(duration),
      vacuum.pause(), vacuum.return_to_base(), vacuum.start(), vacuum.stop()
      Devices: media_player.harman_kardon_aura 'Harman Kardon Glass Speaker'
      = standby; vol=0.88\ntimer.kitchen_oven 'Kitchen oven timer' =
      active\nlock.office_cabinet 'Office cabinet lock' = unlocked
      cover.master_bedroom 'Master Bedroom' = closed
      vacuum.hallway_neato 'Hallway path cleaner' = docked
      switch.basement_lights 'Basement Lights Switch' = off"
    },
    {
      "from": "user",
      "value": "reverse the master bedroom blinds"
    },
    {
      "from": "assistant",
      "value": "switching Master Bedroom state as requested\n
      ```homeassistant
         {
          \"service\": \"cover.toggle\",
          \"target_device\": \"cover.master_bedroom\"
        }
      ```"
    }
]
~~~

Figure 1: **Data Example.** In the figure, we show a data example from the acon96 (2024) dataset.

B Data Distribution Detailed

We show a more detailed distribution of the dataset in Table 6.

Class	**Total Dataset**	**Test**
climate.set_fan_mode	1080	0
climate.set_humidity	1080	0
climate.set_hvac_mode	1080	0
climate.set_temperature	1000	0
cover.close	385	35
cover.open	395	40
cover.stop	320	25
cover.toggle	365	25
fan.decrease_speed	360	60
fan.increase_speed	300	40
fan.toggle	390	85
fan.turn_off	390	70
fan.turn_on	405	60
light.toggle	450	90
light.turn_off	2535	600
light.turn_on	11940	150
lock.lock	200	125
lock.unlock	185	125
media_player.media_next_track	55	25
media_player.media_pause	55	25
media_player.media_play	70	25
media_player.media_previous_track	55	25
media_player.media_stop	55	25
media_player.turn_off	25	25
media_player.turn_on	40	40
media_player.volume_down	65	35
media_player.volume_mute	60	30
media_player.volume_up	85	40
switch.toggle	250	50
switch.turn_off	500	175
switch.turn_on	540	165
timer.cancel	600	0
timer.start	600	0
todo.add_item	1560	0
vacuum.pause	15	0
vacuum.return_to_base	150	0
vacuum.start	370	220
vacuum.stop	15	0

Table 6: **Detailed Class Distribution Service.** Total Dataset vs. LLM Test Subset