7

Model Selection and Prompting Strategies in Resource Constrained Environments for LLM-based Robotic System

Toms Eduards Zinars, Oskars Vismanis, Peteris Racinskis, Janis Arents, and Modris Greitans

Institute of Electronics and Computer Science (EDI), Latvia

Abstract

Large Language Models (LLMs) are increasingly becoming adopted for various applications in information processing and content generation. Coupled with the diverse availability of model weights, quantization, and fine-tuning, it is desirable to find the best-performing LLM within a certain memory budget. Prompting strategy plays a similarly major role in the quality of generation, with various methods existing that attempt to induce desired behaviour, requiring a major time investment to develop. As new models with better performance to memory ratio become available, it may be tempting to implement them into an existing system for potential performance improvements. In this work we explore the process of changing weights and note that weights from larger models or of different quantization precision are unable to replace the original model without modifications to the prompt contents, which in turn implies complications in developing modular, weight agnostic systems.

Keywords: large language models (LLM), natural language processing (NLP), high-level planning (HLP), robotics.

7.1 Introduction and Background

The interest to employ Large Language Models for various tasks has grown over the past couple of years [1]. Among these are applications in robotic systems, including tasks such as high-level planning [2], low-level control [3] and semantic map creation [4], but most of these approaches rely on top-of-the-line LLMs provided as a cloud service [5]. To increase the autonomy of such systems, the inference process must be brought as close as possible to the physical robotic agent. On-device LLMs for smartphones have received official support from Google with Gemini Nano 2 for Android [6] and from Apple with Apple Intelligence for MacOS ecosystem [7], a major step for on the edge deployment.

Other LLM developers have published their model weights with relatively open licenses, though such models are typically less capable than the ones offered exclusively through the cloud [8]. One of the main characteristics of an LLM is its size, which is measured in billions of parameters ("LLM 7B" denotes a model with 7 billion parameters). Currently, graphics cards (GPU) serve as the primary means of deploying these models, as they offer faster inference speeds than processor (CPU) based solutions [9]. Because of the standard 16-bit precision used for LLM weights even a relatively small model of 7B parameters (GPT-3, the "first" LLM, had 175B [10]) requires at least 14GB+ of VRAM, which currently is only available on the upper end of consumer grade GPUs [11]. Various Post-Training Quantization (PTQ) methods are available that can successfully reduce the precision down to 4-bits per weight while still retaining most of the performance [12], crucial for deploying as close to the edge as possible.

The autoregressive nature of decoder-only transformer architecture that underpins most LLMs [1] means that they generate the next (sub-)word based on all the input text before it according to the defined sampling parameters. As such the contents that are fed into the LLM are of high importance and has led to the emergence of the discipline of "Prompt Engineering" - the process of developing the textual content of the prompt to invoke the desired behaviour in the generation. Determining a prompting strategy for a particular task includes not only deciding on the contents of the prompt but also considerations such as if the task would be better executed over several separate prompts, a method known as "prompt chaining" [13], and what patterns [14] and methods [15] to employ for each prompt.

Creating an LLM-based high-level planner in a local context for a robotic system places several limitations on model selection which in turn heavily

influences the prompting strategy. The ideal solution would provide a valid plan in the fastest time within the smallest amount of VRAM spending least amount of energy. As response times are a major consideration, even though GPUs are an additional power consuming component, currently it is not viable to create the system without one, and even the smallest models can strain and quickly deplete batteries of purely edge systems like smartphones [16]. Because the context window (the maximum amount of text a model can process at once) also contributes to memory usage, it may be necessary to reduce it, which in turn limits the number of examples that can be given and how complex of an instruction can be provided.

In this work we test how practical it is to change the underlying model of an already developed LLM-based system as a way to gain improvements. To this end we take our Natural Language Processing-High Level Planning (NLP-HLP) module for a mobile manipulator system [17] that uses an LLM for inference and replace it with different quantization precision weights and from different model families to see if the prompts carry over between different weights. A brief overview of LLMs, prompting strategies and LLM use in robotics is given in chapter 1.2. We create a validation set for each of our prompting steps and compare how well each set of weights processes these test questions, with chapter 1.3 elaborating on the experimental setup. In chapter 1.4 we display the results from our tests that were carried out on the set of different precisions and on the set of different models. In chapter 1.5 we give our conclusions and outlook for further development.

7.2 Related Work

LLM-based systems, regardless of use case, are built on top of various prompting strategies, which in turn rely on quality LLM weights to perform the generation. The landscape of LLMs has developed rapidly since early 2023, when “LLaMA” model weights were leaked to the public [18], beginning a wave of interest in local deployment of LLMs that as of October 2024 has resulted in over 140 thousand “Text Generation” weights of various model families, finetunes and their quantizations being uploaded to the public Hugging Face model library [19].

7.2.1 Local Large Language Models

Large Language Models as a distinct category emerged after the release of GPT-3 [10]. The initial focus was placed on training ever larger models

[20], as parameter count correlates with both performance and the concept of "emergent abilities" - at a certain scale, models become able to perform tasks that with fewer parameters they could not, thought [21] has questioned how "emergent" these abilities really are. As larger models are more resource intensive to both train and run, to the benefit of local deployment the focus shifted away from scale of the model to the scale and quality of training, such as the focus of the compact "Phi" model series [22]. The increase of scale can be demonstrated by comparing GPT-3's training of 300 billion tokens versus the training of "Llama 3" series models of up to 15 trillion [23] and "Qwen2.5" with 18 trillion [24], with both model families providing weight sizes that can be comfortably deployed locally. Besides requiring more memory, larger weights tend to be slower, which is a problem that "mixture of experts" (MoE) models try to overcome by mixing the knowledge of a larger with the speed of a smaller model, as done with [25]. This approach has limited relevancy for use in edge computing applications, as the memory requires all the parameters to be loaded in memory while actively only using a smaller portion of them at any one moment.

To truly place these models on the edge, quantization of weight precision is necessary to maximally reduce the memory requirements while still maintaining most of the performance [12]. Several methods for this process have emerged with often different priorities, but two common inference and quantization frameworks are the speed oriented, GPU-only ExLlamaV2 [26] with its EXL2 format and the more general llama.cpp [27] with GGUF format weights that also heavily invest in CPU-only and hybrid inference set ups.

7.2.2 Prompting

Prompting is the primary means of interacting with the LLM as the textual content is the basis upon which new information is generated. Developing a prompt that induces the desired behaviour is a sizable part of deploying these LLM-based systems. A variety of methods have been developed by the LLM research and open-source community to improve the generation quality [15]. Two noteworthy aspects that often get explored include determining what kind information should go into a prompt and how much LLM calls per an input should be done.

A common approach is known as "few-shot" or "n-shot" prompting [10], where "n" designates how many examples are used in the prompt to condition generation towards a particular output format. Another approach is to have the model first generate additional information about the input, such as the

"Chain-of-Thought" ("CoT") [28] method, where each example includes an intermediate reasoning step that is generated before the final answer. Various prompting patterns are highlighted in [14] which can be used to invoke specific behaviour in the models, such as having the model be the one asking questions.

LLMs generate their answers auto-regressively one new token at a time, with all the previous tokens affecting the probability of the next. If the model gets distracted and begins generating "incorrect" data, it is unlikely to recover [29]. Various methods attempt to mitigate this by generating several outputs, like it is done with "Self-Consistency" [30], which picks the most common answer amongst the outputs. "Least-to-Most" [31] prompting seeks to decompose a problem into subproblems and solves them sequentially. "Tree-of-Thought" [32] for each decomposition step generates several possible paths that are explored with search algorithms.

"Chain prompting" [13] is another fundamental skill utilized by many methods and systems, as it allows creating complex interactions by breaking up any task into several specialized prompts, which also allows creation of more modular systems.

7.2.3 LLM in Robotics

In general, the versatility of LLMs has been explored in wildly differing robotics roles. An implementation sceptical of LLMs ability to plan was explored in [33], where the model was used to translate natural language inputs into the structured planning language, which is then passed to a traditional planner. LLM as a planner was integrated into [2], where it generates the whole plan which is then verified for geometric feasibility and performs replanning if necessary. Implementation from [34] uses the LLM to rate a list of predefined actions on their probability as the next action needed for plan completion, while [35] uses an LLM to provide closed loop feedback (including dialog). LLMs have also been used to generate code directly for robot control, as shown in [36], while [3] has the LLM generate rewards for low-level control functions. LLMs have also been used for mapping, such as iteratively searching over a graph as presented in [37] or as in [4] where the LLM is used to embed semantic information inside the map itself, making it searchable through language.

All the examples provided above rely on using some of the largest models available at the time, which helps showcase the performance of the method. However, in order to create an autonomous robotic system that doesn't rely

on a constant internet connection, local LLM application, especially of the smaller sized models, is of interest. Methods that attempt to rely on a smaller LLM as the primary inference engine are less known. In [38] a LLM is combined with a vision encoder to detect and reason about manipulation task failures, while [39] uses the LLM to guide the learning process of new long-horizon tasks. In [40] an LLM is one of the two main models tested for generation planning steps that are then evaluated based on feasibility and reward, whereas [41] notes the lower performance of open LLMs, but still invest in fine-tuning one to act as communication and planning manager for cooperative agents.

These methods rely on what now would be considered outdated models, given the rapid development of openly distributed LLMs over 2023 and 2024 [42], so it seems as if switching to newer models should provide an improvement in performance and make these smaller weights more enticing to use. However, a big part of performance comes from the contents of the prompt which are usually developed to maximize the quality of results out of specific weights. If the LLM-based system consists of a lot of complicated prompts and the new model does not immediately outperform the old, it may require practically rebuilding the whole system. Therefore, we want to explore how well prompts developed for one model perform on different models and quantization precisions.

7.3 Experimental Setup

After a model is chosen, a lot of time is invested in developing the prompt contents such that they invoke the desired generation. Prompting performance is typically framed as being mainly affected by model size and the model training data from which the various abilities are believed to emerge from [20], [28]. As new models with better performance to memory ratios emerge it can be worth exploring them as a potential avenue of upgrade for the system. Another avenue might be to increase the available memory and move to a higher precision quantization, which in theory should also result in improvements [12]. However, this would mainly be practical if the prompt contents can be retained for the new weights. If the validation shows no improvements, it then implies that in certain applications model weights are not “hot swappable”, as the prompt contents need to be rewritten which reduces the systems overall modularity.

7.3.1 System Description

The experiment is based on a previously developed HLP system, which utilizes *Llama 3 8B Instruct Q4_K_M* [43] weights, so the prompts have been developed in tandem with these specific weights. The HLP is designed to fulfil the planning phase for a mobile manipulator system (MMS) performing actions in a vaguely structured indoor environment. In the MMS's low-level planner, a set of action primitives is defined, that describes the system's abilities in a general manner or as highly adaptive functions [17]. The user inputs a NL command that is sent to the HLP. The HLP then generates a plan based on the MMS's set of abilities. This system consists of a prompt chain with 4 steps split into 3 stages: Filtering ("categorization" and "mobility"), Structuring ("preplanning") and Planning ("planning"):

- The "categorization" step determines the type of the incoming request between 4 categories: "mobile manipulation", "question answering", "casual discussion", "any other query". The rest of the system is only interested in "mobile manipulation". The prompt is 4-shot – containing examples for all categories.
- Because the currently used demonstrator is a static industrial arm, the "mobility" step is used to determine in a "True/False" fashion if moving between working positions is required or not. Once the demonstrator is

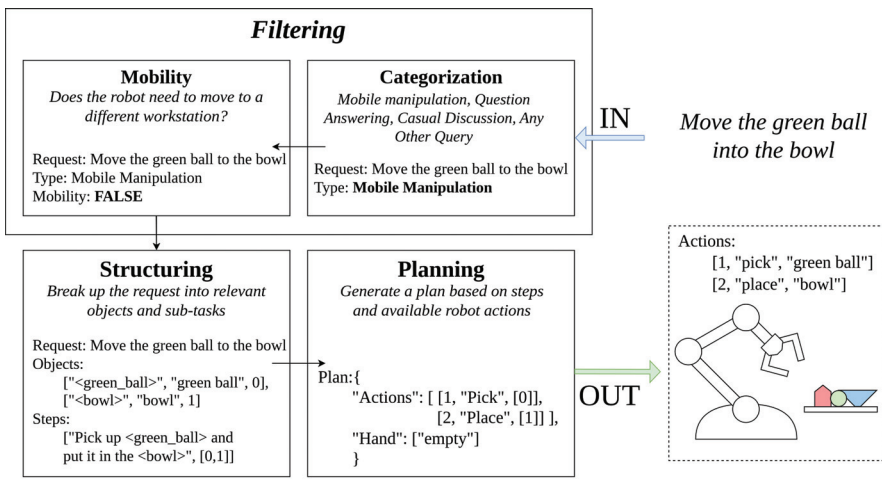

Figure 7.1 Scematic of the MMS first demonstrator's HLP system.

transferred to a mobile platform, this step can simply be taken out. This prompt is zero-shot and is the shortest of the ones tested.

- The "preplanning" performs two tasks: creating a description (consisting of a marker, original text and a unique ID) for each object relevant to the request, formatted as "*[“<object_name>”, “object name”, 0]*"; breaking up the request into smaller sub-tasks that can be planned for individually - "*[“Pick up <large_cactus>”, [1]], [“Place <large_cactus> on <small_tile>”, [0]]*". The prompt uses 3 examples of increasing difficulty.
- The "planning" step creates a plan for each subtask that consists of assigning relevant action primitives to respective objects in a reasonable order. The input consists of the sub-task text, a list of objects and the status of the hand (*[“empty”]* or *[“<obj>”, “obj, 0]”*). An input "*put the <box> on the <table>*" should output a plan like "*[[“pick”, [0]], [“place”, [1]]]*". This is the most complex task of the set as it requires some form of reasoning, which smaller models are much less capable of [28]. This step employs CoT style step-by-step generation of step-by-step plan, where it is asked to first restate the task, check its hand status and then generate the step: "*1. Pick up <obj>. My Hand: <obj>*". It also relies on 3 examples with increasing difficulty.

By defining standardized input and output formats for each step, it is possible to integrate them together in a prompt chain in such a way that the intermediate product can be processed by traditional programming. This also allows the creation of a testing set for each step that is necessary to both validate and track the performance of the weights and prompts, though an alternative for tasks that are hard to standardize is to use another LLM to act as a judge [44].

7.3.2 Testing process

The validation process rates the generated output as follows: if the output could not be successfully processed by the script functions, then it is rated as "-1"; if the output could be processed but was wrong or unexpected it is rated as a "0"; if the output was processed and aligned with the expected result, it is rated with "1" and considered acceptable. Scoring only takes into consideration expected results and no malus is applied for failed generations. The validation set was created specifically for this test, however it is relatively simple and small.

The initial validation set for all tasks consist of 25 test inputs for each output type. The "categorization" set consists of 25 requests for each of the categories, to test how well the model can identify the correct category. "Mobility" consists of separate sets for "True" and "False". Both "preplanning" and "planning" consist of 25 each. "Preplanning" is tested on "objects" (if the right objects were generated), "object length" (if no extra objects were created) and "Steps" (if the expected number of sub-tasks were generated). "Planning" is evaluated on "Plan" (if the expected plan was generated) and "Hand" (if the correct status of the hand was registered). In total 11 positions are evaluated across the current implementation of the system, for a total of 275 points.

7.3.3 Model selection

A selection of different models using Q4_K_M GGUF quantization and alternate quantization precisions for the "Llama 3 8B Instruct" model are put through the validation system to see if there is transferability of the designed prompts in an application with strictly defined input/output formats.

GGUF quantization method was chosen for its versatility to be deployed for CPU, GPU or hybrid inference, however only GPU inference has been used for the development and testing of the system. The chosen precisions are: "Q8_0", "Q6_K", "Q5_K_M", "Q5_K_S", "Q3_K_L", "IQ3_XXS", "IQ2_M" and "IQ2_XXS" [43], [45] , which roughly cover the range of available precisions.

The choice of models for testing was based on their size (able to fit into a single consumer-grade GPU) and their status as weights released directly by the developers. Third party finetunes were not considered for testing. Overall, the selected models represent a size range from 1B to 15B parameters.

Several benchmarks exist that attempt to quantify and compare the performance of different LLM models. The Open LLM Leaderboard [47] combines 6 such benchmarks to provide thorough evaluation of open-source models, while Berkeley Function Calling Leaderboard (BFCL) [48] specializes in testing model ability on tool use or function calling, which is a skillset comparable to the tasks expected of the planner. A different kind of benchmark is Chatbot Arena [8], which lets users compare two models and rate which they believe is better, but as such can be susceptible to user preference, "Style Control" is used to help mitigate it. These sorts of benchmarks play a role in informing decisions about which models to choose and create expectations about the model relative performance.

7.3.4 Testing environment

The testing is performed on *Ubuntu 22.04* using an *Nvidia GTX 3060 12GB* graphics card, running *CUDA version 12.3* and *Python 3.11.5*. The models were loaded using llama.cpp [27] with the *llama-cpp-python* [53] wrapper, which uses GGUF format weights and serves as a quantization method. The used maximum context window is fixed at 2048 tokens across all models. Sampling parameters include setting "temperature" to "0.0" (greedy sampling for most deterministic generation), which also removes random seeds as a factor in sampling. Despite this, tests were run at least 3 times with different seeds to verify no change between generations.

7.4 Results

With "Llama 3 8B Inst Q4_K_M" serving as the base line, two sets of weights were tested against it. The first set was used to determine if it is possible to shift the system to a different quantization precision either to free up memory by going to a lower precision or to improve performance by going to a higher one. The second test looked at the viability of replacing the LLM with a different model series all together. At temperature "0.0" no changes were noticed between the generations in either batch.

7.4.1 Differences between quantization precisions

Results from the first test shown in figure 7.2 visualize how 3- and 2-bit precisions rapidly degrade in quality, which aligns with general understanding of their behaviour [12], as smaller models incur more loss at lower precision. 4-bit and higher precisions however underperform expectations as none of them manage to noticeably improve upon the default weights – Q4_K_S (smaller than default) match results (245 points), but both models make different mistakes. The larger "Q6_K" and "Q8_0" weights score lower (237 and 242 respectively), while in theory these models should maintain more of the original non-quantized performance. "Q4_K_M", "Q4_K_S" and "Q5_K_M" were the only weights to not suffer from any completely failed generations.

Comparing the specific outputs of "Q4_K_S" and "Q4_K_M" for "mobility" tests, both models make 3 mistakes, but only 1 is different between them. The questions that are failed by both are "Find the basketball" for "False" and "Move the chairs" for "True", which might point to semantic ambiguity

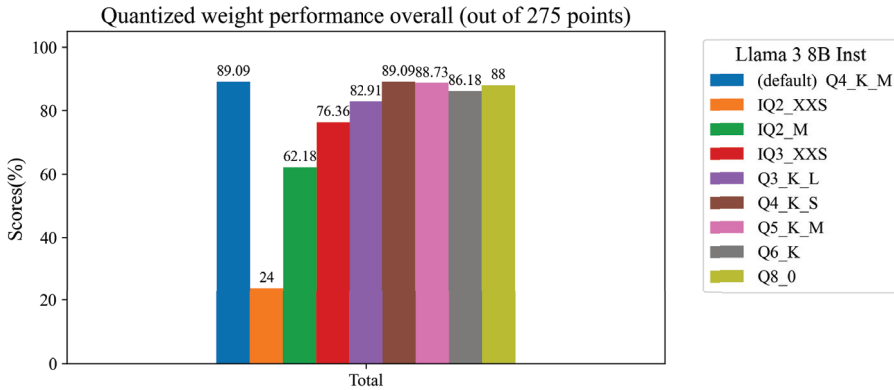

Figure 7.2 Correctly passed tests for quantization precision comparison.

in the validation set. "Q4_K_M" failed on "Flip the book to the next page" for "False" while "Q4_K_S" failed on "Carry this chocolate cake for me" for "True". Such minor differences can be a cause of concern, as it can mean that a new model could pass the validation test but then fail on tasks the previous model had no problem doing. This implies the need for prompting methods that can minimize such differences as well as potentially maintain a history of previously performed actions that a promising model can be further tested on.

7.4.2 Differences between models

During the second test, as seen in figure 7.3, none of the tested models were able to outperform the default model, despite several of them being larger in size and having a higher score on the Open LLM and BFCL benchmarks (Table 7.1). The further development of Llama 3 - "Llama 3.1 8B Instruct" - seems to perform about as well as "Gemma 2 9B instruct", while "Qwen2.5 14B Instruct" takes second place behind the default model. Model size plays a noticeable role as the larger models performed better, though not to the degree it would be expected. The prompting template also did not seem to factor in, considering that "Gemma 2" uses a "user/model" style prompting versus "Llama 3" that uses "system/user/model", which effects how prompts are structured. Overall, a similar conclusion can be drawn to what was seen with the quantization precision – prompts are unlikely to carry over to smaller models, but larger models do not guarantee superior results despite technically better performance.

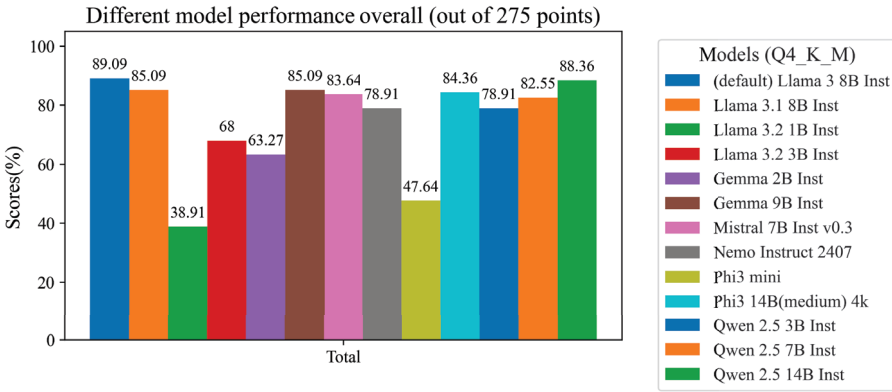

Figure 7.3 Correctly passed tests by various LLMs.

Table 7.1 Model Selection (all models using GGUF type Q4_K_M, all weights used in tests sourced from **[46]**

Model Name	**Parameters (in Billions)**	**OpenLLM score [47]**	**BFCL Overall Acc [48]**	**Chatbot Arena score (Style Controlled)** [8]
Llama 3 8B Instruct [23]	8.03	23.91	34.32	1143
Llama 3.1 8B Instruct [23]	8.03	27.77	50.87	1133
Llama 3.2 1B Instruct [49]	1.24	13.76	20.59	1045
Llama 3.2 3B Instruct [49]	3.21	23.85	46.92	1094
Gemma 2 2B Instruct [50]	2.61	17.05	22.38	1116
Gemma 2 9B Instruct [50]	9.24	28.86	51.59	1179
Mistral 7B Instruct v0.3 [51]	7.25	19.11	-	-
Nemo-Instruct-2407 [52]	12.2	23.53	42.56*	-
Phi 3.5 mini instruct [22]	3.82	27.4	-	-
Phi 3 medium 4K instruct [22]	14.0	32.67	-	1118
Qwen 2.5 3B Instruct [24]	3.09	21.03	47.09	-
Qwen 2.5 7B Instruct [24]	7.62	26.87	53.69	-
Qwen 2.5 14B Instruct [24]	14.8	32.11	57.68	-

* - value taken from "Open-Mistral-Nemo-2407"

Looking at individual categories, certain model performance can vary between different tasks drastically. As seen in figure 7.4, "Gemma 2 9B Instruct" has very strong results when it comes to identifying the status of the robot's hand after the conclusion of the plan but has some of the weakest results when it comes to the actual planning. Further investigation reveals that the model struggles with placing the correct item IDs in the respective actions, a critical failure with no discernible cause.

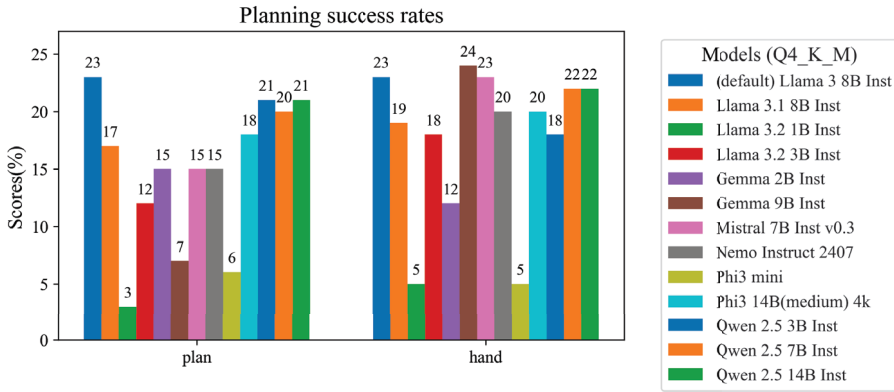

Figure 7.4 Planning success rates for the various models.

7.4.3 Result comparison to VRAM usage

As the available memory budget is an important metric to consider in the choice of models, the maximum amount of VRAM used during testing was noted for all weights tested and is relative to the model size in parameters. VRAM usage peaks during the "plan" step, as it has the longest instruction prompt.

Figure 7.5 shows how the larger models that require more memory seem to plateau and perform on par with the middle of road models, meanwhile the performance of the sub-7B models is more scattered, where the largest of this group ("Phi3.5 mini Q4_K_M) shows the second worst results and is outperformed by the almost twice as small "Gemma 2B Inst Q4_K_M". Its benchmark scores seen in Table 1.1 do not correlate well with real results seen in figure 7.3 and figure 7.4. Further investigation reveals complete failure of the "mobility" step, with which other sub-7B models did not struggle as much. An example comparison for the "mobility" question "shuffle the deck of cards", "Llama 3.2 3B inst Q4_K_M" answers with "{*"Mobile": False*}", while "Phi3.5 mini Q4_K_M" answers with "'{*"Mobile": True*}\n*The action of shuffling a deck requires moving the cards, which implies that you would need to leave your current workstation (the table) where they are placed. Therefore, this task necessitates mobility beyond just manipulating objects on-table without leaving it entirely.*" A model being too verbose is a problem from performance perspective, as even if the model can generate the correct answer, the additional time spent generation unnecessary tokens slows the overall execution time of the system. It also should be noted that all the

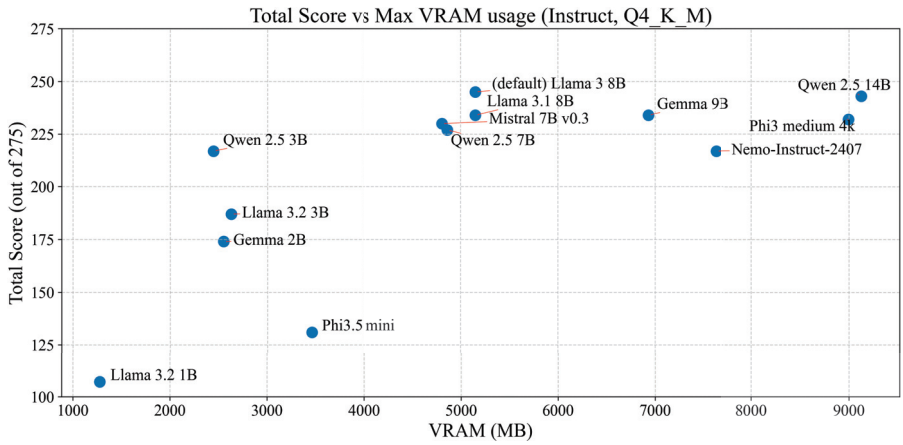

Figure 7.5 VRAM usage of the tested models.

text generated after the answer has no contribution to it, rather the model is "justifying" why it printed the answer it did.

7.4.4 Result comparison to Benchmark performance

Comparing the test scores against the BFCL benchmark results, it can be seen in figure 7.6 that the models with higher benchmark scores show better results overall, while the default model massively over performs. The "Qwen2.5 3B Inst Q4_K_M" performs well on the "planning" tasks which most other small models struggle with which is reflected in the BFCL results, however on further inspection of the generated outputs, it seems that all "Qwen2.5" models have a problem with following the defined structure of "CoT" reasoning,

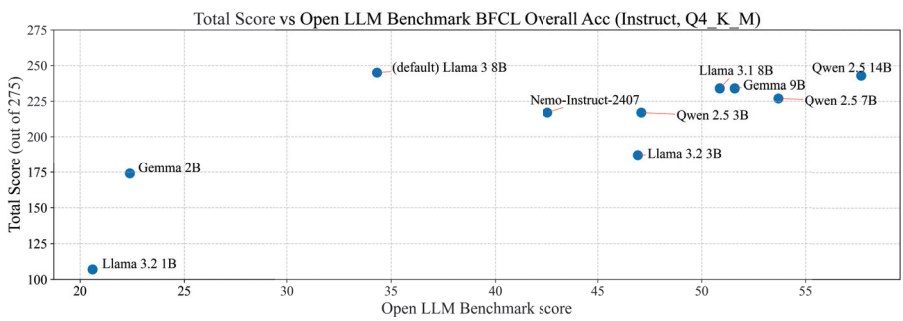

Figure 7.6 Testing results relative to model performance on the BFCL.

typically ignoring the "My hand: <object>" part, which all the other models, even the ones that fail most of the plans, do maintain, which might imply a different training approach from the other model families.

7.5 Conclusions

Development of an LLM-based system in a resource constrained environment faces many challenges. Besides the obvious limitations on model size, a general problem that such systems may encounter is the over-specializations of prompts for certain applications to not only a specific model (Llama 3 8B Instruct), but to specific weights (Q4_K_M precision) which complicates the adaptation of the system to different models. Tests performed showed that larger models with higher rated benchmark performance were unable to surpass the performance of the default model, falling short. Perhaps more concerning is the fact that the models that almost matched the performance typically did so by making a different set of mistakes. A similar trend was seen between different quantization precisions – lower precisions lose performance, yet going higher does not seemingly improve it. While larger precisions did show better results than larger models, considering that both would require rewrites of the prompt content, what should be the criteria used to decide where to invest time in? As locally testing all the various models and their finetunes is simply not viable, trusted benchmarks are a critical aspect when it comes to deciding on what weights to spend development time. But it seems that certain prompt content may favour one model over another.

This implies the need to solve two problems. The first problem is to eliminate the variance that comes from weights making minor but different mistakes, which requires developing the existing system further with more advanced prompting methods, however the main drawback will be the increase of execution times, which at some point may no longer be practical for a robotic system. Second problem relates to weight "onboarding" to an existing LLM-based system by a more elaborate validation system that is capable of rating new weights without bias for specific weights. Both changes should strive to reduce the need for rewriting of prompts and to improve trust in the reliability of LLMs to perform these tasks.

More research is needed to better understand the deployment of the more compact LLM models in robotic systems and on the edge in general, but this impression may simply be the result of the research cycle, given how recent many of the innovations and performance jumps in local LLMs have been.

Acknowledgements

This work was supported by Chips Joint Undertaking EdgeAI project. The project EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No 101097300.

References

[1] W. X. Zhao et al., "A Survey of Large Language Models," arXiv, arXiv:2303.18223 [cs.CL], 2023.

[2] K. Lin et al., "Text2Motion: from Natural Language Instructions to Feasible Plans," Autonomous Robots, vol. 47, no. 8, pp. 1345–1365, Nov. 2023, doi: 10.1007/s10514-023-10131-7.

[3] W. Yu et al., "Language to Rewards for Robotic Skill Synthesis," arXiv, arXiv:2306.08647 [cs.RO], 2023.

[4] Q. Gu et al., "ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning," arXiv, arXiv:2309.16650 [cs.RO], 2023.

[5] Y. Hu et al., "Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis," arXiv, arXiv:2312.08782 [cs.RO], 2023.

[6] T. Darra, "Gemini Nano is now available on Android via experimental access," Android Developers Blog: Oct. 17, 2024. [Online]. Accessed: Oct. 17, 2024. Available: https://android-developers.googleblog.com/2024/10/gemini-nano-experimental-access-available-on-android.html

[7] "Introducing Apple's On-Device and Server Foundation Models," Apple Machine Learning Research, Jun. 10, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://machinelearning.apple.com/research/introducing-apple-foundation-models

[8] W.-L. Chiang et al., "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference," arXiv, arXiv:2403.04132 [cs.AI], 2024.

[9] "GPU inference," Hugging Face, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/docs/transformers/perf_infer_gpu_one

[10] T. Brown et al., "Language Models are Few-Shot Learners," arXiv, arXiv:2005.14165 [cs.CL], 2020.

[11] X. Dai, "GPU-Benchmarks-on-LLM-Inference" GitHub, 2023. Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/XiongjieDai/GPU-Benchmarks-on-LLM-Inference

[12] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," arXiv, arXiv:2210.17323 [cs.LG], 2023.
[13] T. Wu, M. Terry, and C. J. Cai, "AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts," in Proc. 2022 CHI Conf. Hum. Factors Comput. Syst., New Orleans, LA, USA, 2022, pp. 385:1-385:22, doi: 10.1145/3491102.3517582.
[14] J. White et al., "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," arXiv, arXiv:2302.11382 [cs.SE], 2023.
[15] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications," arXiv, arXiv:2402.07927 [cs.AI], 2024.
[16] S. Laskaridis, K. Katevas, L. Minto, and H. Haddadi, "MELTing point: Mobile Evaluation of Language Transformers," arXiv, arXiv:2403.12844 [cs.LG], 2024.
[17] T. E. Zinars, O. Vismanis, P. Racinskis, J. Arents, and M. Greitans, "Natural language conditioned planning of complex robotics tasks," in Advancing Edge Artificial Intelligence, pp. 131–151, River Publishers, 2024. e-ISBN 9781003478713, doi: 10.1201/9781003478713-6.
[18] A. Hern, "TechScape: Will Meta's massive leak democratise AI – and at what cost?," The Guardian, Mar. 07, 2023. Accessed: Oct. 17, 2024. [Online]. Available: https://www.theguardian.com/technology/2023/mar/07/techscape-meta-leak-llama-chatgpt-ai-crossroad
[19] "Models," Hugging Face. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/models
[20] J. Wei et al., "Emergent Abilities of Large Language Models," arXiv, arXiv:2206.07682 [cs.CL], 2022.
[21] R. Schaeffer, B. Miranda, and S. Koyejo, "Are Emergent Abilities of Large Language Models a Mirage?," arXiv, arXiv:2304.15004 [cs.AI], 2023.
[22] M. Abdin et al., "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone," arXiv, arXiv:2404.14219 [cs.CL], 2024.
[23] A. Dubey et al., "The Llama 3 Herd of Models," arXiv, arXiv:2407.21783 [cs.AI], 2024.
[24] QwenLM, "Qwen2.5," GitHub, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/QwenLM/Qwen2.5
[25] A. Q. Jiang et al., "Mixtral of Experts," arXiv, arXiv:2401.04088 [cs.LG], 2024.

[26] turboderp, "turboderp/exllamav2," GitHub, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/turboderp/exllamav2
[27] G. Gerganov, "llama.cpp," GitHub, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/ggerganov/llama.cpp
[28] J. Wei et al., "Chain of Thought Prompting Elicits Reasoning in Large Language Models," arXiv, arXiv:2201.11903 [cs.CL], 2022.
[29] L. Huang et al., "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," ACM Trans. Inf. Syst., Nov. 2024, doi: 10.1145/3703155.
[30] X. Wang et al., "Self-Consistency Improves Chain of Thought Reasoning in Language Models," arXiv, arXiv:2203.11171 [cs.CL], 2022.
[31] D. Zhou et al., "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models," arXiv, arXiv:2205.10625 [cs.AI], 2022.
[32] S. Yao et al., "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," arXiv, arXiv:2305.10601 [cs.CL], 2023.
[33] B. Liu et al., "LLM+P: Empowering Large Language Models with Optimal Planning Proficiency," arXiv, arXiv:2304.11477 [cs.AI], 2023.
[34] M. Ahn et al., "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," arXiv, arXiv:2204.01691 [cs.RO], 2022.
[35] W. Huang et al., "Inner Monologue: Embodied Reasoning through Planning with Language Models," arXiv, arXiv:2207.05608 [cs.RO], 2022.
[36] J. Liang et al., "Code as Policies: Language Model Programs for Embodied Control," arXiv, arXiv:2209.07753 [cs.RO], 2022.
[37] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning," arXiv, arXiv:2307.06135 [cs.AI], 2023.
[38] J. Duan et al., "AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation," arXiv, arXiv:2410.00371 [cs.RO], 2024.
[39] J. Zhang et al., "Bootstrap Your Own Skills: Learning to Solve New Tasks with Large Language Model Guidance," arXiv, arXiv:2310.10021 [cs.RO], 2023.
[40] R. Hazra, P. Zuidberg, and D. R. Luc, "SayCanPay: Heuristic Planning with Large Language Models using Learnable Domain Knowledge," arXiv, arXiv:2308.12682 [cs.AI], 2023.
[41] F. Joublin et al., "CoPAL: Corrective Planning of Robot Actions with Large Language Models," arXiv, arXiv:2310.07263 [cs.RO], 2023.

[42] H. Naveed et al., "A Comprehensive Overview of Large Language Models," arXiv, arXiv:2307.06435 [cs.CL], 2023.
[43] Bartowski, "bartowski/Meta-Llama-3-8B-Instruct-GGUF," Hugging Face, 2024. Accessed: Oct. 18, 2024. [Online]. Available: https://huggingface.co/bartowski/Meta-Llama-3-8B-Instruct-GGUF
[44] L. Zheng et al., "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," arXiv, arXiv:2306.05685 [cs.CL], 2023.
[45] "GGUF," Hugging Face, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/docs/hub/gguf
[46] Bartowski, "bartowski (Bartowski)," Huggingface.co, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/bartowski
[47] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, and T. Wolf, "Open LLM leaderboard v2," Hugging Face, 2024. Accessed: Jan. 20, 2025. [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard
[48] F. Yan et al., "Berkeley Function Calling Leaderboard," Accessed: Jan. 20, 2025. [Online]. Available: https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html
[49] Meta AI, "Llama 3.2: Revolutionizing edge AI and vision with open, customizable models," Meta.com, Sep. 25, 2024. Accessed: Oct. 18, 2024. [Online]. Available: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/
[50] Gemma Team, "Gemma 2: Improving Open Language Models at a Practical Size," arXiv, arXiv:2408:00118 [cs.CL], 2024.
[51] A. Q. Jiang et al., "Mistral 7B," arXiv, arXiv:2310.06825 [cs.CL], 2023.
[52] Mistral AI, "Mistral NeMo," Mistral AI, Jul. 18, 2024. Accessed: Jan. 20, 2025. [Online]. Available: https://mistral.ai/news/mistral-nemo/
[53] Andrei, "Llama Python Bindings for llama.cpp," GitHub. Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/abetlen/llama-cpp-python