



Tartu University

Faculty of Science and Technology

Institute of Technology

Dorian Lamouille

**Benchmarking and Deploying Local Language Models for
Social Educational Robots using Edge Devices**

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisors:

Farnaz Baksh, MSc
Matevž B. Zorec, MSc

Tartu 2025

Abstract

Deploying Educational Language Model on Low-Resource Social Robot

This thesis investigates how a local language model pipeline can support the Robot Study Companion, an affordable open-source educational social robot under severe computational constraints. It addresses the challenge of delivering accurate, engaging learning assistance on edge hardware by benchmarking twenty-five recent open-source language models on a Raspberry Pi 4, 5 versus a gaming laptop, using performance, energy, and pedagogical metrics. The study evaluates inference speed, resource consumption, and teaching effectiveness, and identifies clear trade-offs between model size, responsiveness, and instructional quality.

Building on these findings, the thesis proposes and implements a three-tier hybrid architecture that combines caching, lightweight rephrasing models, and a higher-capacity reasoning model to balance accuracy with real-time interaction. An exploratory user study further examines student perceptions of written and spoken interactions with the system. The results demonstrate that carefully selected small and hybrid models, combined with architectural optimization, can provide effective, privacy-preserving learning support on local and low-resource robotic platforms.

CERCS: T125 Automation, robotics, control engineering; P176 Artificial Intelligence.

Keywords: Robotic Study Companion (RSC), Language Model (LM), Raspberry Pi (RPi), Edge-device, Teaching Model

Resüme

Haridusliku keelemudeli rakendamine vähete ressurssidega sotsiaalrobotis

Käesolevas väitekirjas uuritakse, kuidas kohalik keelemudeli torustik saab toetada Robot Study Companionit, mis on taskukohane avatud lähtekoodiga hariduslik sotsiaalrobot, millel on ranged arvutusvõimsuse piirangud. Töö käsitleb väljakutset pakkuda täpset ja kaasahaaravat õppimisabi piiratud ääreseadmete abil, võrreldes 25 hiljutist avatud lähtekoodiga keelemudelit Raspberry Pi 4-l, Raspberry Pi 5-l ja suurema jõudlusega sülearvutil, kasutades jõudluse, energiatarbe ja pedagoogilisi mõõdikuid. Uuringus hinnatakse järeluste tegemise kiirust, ressursitarbimist ja õpetamise efektiivsust ning tuvastatakse selged kompromissid mudeli suuruse, reageerimisvõime ja õpetamise kvaliteedi vahel.

Nende tulemuste põhjal pakub lõputöö välja ja rakendab kolmeastmelise hübriidarkitektuuri, mis ühendab vahemällu salvestamise, kergekaalulised ümbersõnastamismudelid ja suurema võimsusega järelusmudeli, et tasakaalustada täpsust ja reaajas suhtlemist. Täiendav kasutajauuring analüüsib õpilaste arusaamu kirjalikust ja suulisest suhtlusest süsteemiga. Tulemused näitavad, et hoolikalt valitud väikesed ja hübriidmudelid koos arhitektuuri optimeerimisega võivad pakkuda tõhusat ja privaatsust säilitavat õppetoetust kohalikel ning vähete ressurssidega robotikaplatvormidel.

CERCS: T125 Automatiseerimine, robotika, juhtimistehnika P176 Tehisintellekt

Märksõnad: Robotõpikaaslane (RSC), keelemudel, Raspberry Pi (RPi), servaseade, õpetamismudel

Table of Contents

- Abstract 2
- Resümee..... 3
- List of Figures 6
- List of Tables 7
- 1 Introduction 9
 - 1.1 Problem Statement 9
 - 1.2 Objectives and Roadmap 10
- 2 State of the Art 10
 - 2.1 LLM with Low computational requirements suitable for edge deployment 11
 - 2.2 Voice assistants and Chatbots..... 12
 - 2.3 LLM Architectures 12
- 3 Requirement 13
- 4 Methodology 14
 - 4.1 Research Methods 14
 - 4.2 Limitations and possibilities 14
 - 4.2.1 Local Limitation: 14
 - 4.2.2 Memory Limitations:..... 15
 - 4.2.3 Computation Limitations: 15
 - 4.2.4 Language Limitation:..... 16
 - 4.2.5 Personalization possibilities: 16
 - 4.2.6 Model possibilities:..... 16
 - 4.3 Local LM Benchmarking 16
 - 4.3.1 The models 16
 - 4.3.2 Metrics..... 18
 - 4.3.3 Benchmarking method 19
 - 4.3.4 Benchmarking on other devices 21
 - 4.3.5 Benchmarking method update 21
 - 4.4 Finetuning and quantization 22
 - 4.5 The Three-Tier Architecture..... 23
 - 4.6 User Model interaction experiment 24
 - 4.6.1 Metrics..... 24

| | | |
|-------|--|----|
| 4.6.2 | Methodology | 25 |
| 4.6.3 | Experiment Procedure | 25 |
| 4.6.4 | Data collection procedure | 26 |
| 5 | Results..... | 28 |
| 5.1 | Benchmarking Results for the Raspberry Pi 4 Model B | 28 |
| 5.1.1 | Small model results (model with fewer than 2 billion parameters) | 28 |
| 5.1.2 | Big models results (model with more than 2 billion parameters)..... | 32 |
| 5.1.3 | final model choice | 37 |
| 5.2 | Performance comparison of Pi4, Pi5, RTX4060 | 38 |
| 5.3 | User Model Interaction Experiment Results..... | 39 |
| 5.3.1 | Analysis of the experimental results on a text basis | 40 |
| 5.3.2 | Comparison of text and speech | 40 |
| 5.3.3 | Verbal feedback from participants..... | 40 |
| 5.4 | Results and proposed architecture | 40 |
| 5.4.1 | architecture | 40 |
| 5.4.2 | Initialization | 41 |
| 5.4.3 | Real-time use | 42 |
| 6 | Analysis and Discussion | 43 |
| 6.1 | Challenges | 43 |
| 6.1.1 | Benchmark challenge..... | 43 |
| 6.1.2 | Sample Size for the User Experiment | 44 |
| 6.2 | Future Work | 44 |
| 7 | Conclusion | 45 |
| | Bibliography | 47 |

List of Figures

| | |
|---|----|
| Figure 1 Example of CUQ..... | 24 |
| Figure 2 Token Per Seconds (models less than 2 billion parameters) | 28 |
| Figure 3 Response length (model less than 2 billion parameters) | 29 |
| Figure 4 Inference time (models less than 2 billion parameters) | 30 |
| Figure 5 Average IOPS (models with fewer than 2 billion parameters) | 30 |
| Figure 6: Teaching effectiveness (models with fewer than 2 billion parameters) | 31 |
| Figure 7 Token Per Joule (model less than 2 billion parameters) | 31 |
| Figure 8 MMLU overall scores (models with fewer than 2 billion parameters)..... | 32 |
| Figure 9 Token Per Second (models more than 2 billion parameters)..... | 32 |
| Figure 10 First question TTFT (Model with more than 2 billion parameters) | 33 |
| Figure 11 Average TTFT of the last 9 questions (Model with more than 2 billion parameters) | 33 |
| Figure 12 Average Response Length (models with more than 2 billion parameters)..... | 34 |
| Figure 13 Inference time (model with more than 2 billion parameters) | 34 |
| Figure 14 Average IOPS (model with more than 2 billion parameters) | 35 |
| Figure 15: Teaching effectiveness (model with more than 2 billion parameters)..... | 35 |
| Figure 16 Token Efficiency (model with more than 2 billion parameters) | 36 |
| Figure 17 MMLU overall score (models with more than 2 billion parameters)..... | 36 |
| Figure 18 MMLU scores of Qwen3 0.6B..... | 37 |
| Figure 19 MMLU scores of Granite 4 Tiny Hybrid | 37 |
| Figure 20 Comparison of the devices on TPS and Inference Time for Gemma3 :1b | 38 |
| Figure 21 Comparison of the devices on TPS and Inference Time for Granite4:tiny-h..... | 39 |
| Figure 22: architecture of the actual LocalLM of the RSC..... | 41 |
| Figure 23 flowchart of the Initialisation of the RSC questions..... | 41 |
| Figure 24: Flowchart of the architecture during the use | 42 |

List of Tables

| | |
|--|----|
| Table 1 Requirement for Study companion [1] | 13 |
| Table 2: Performance comparison between Raspberry Pi 4 B and other devices | 15 |
| Table 3: Language Models chosen for benchmarking..... | 17 |
| Table 4: Table of metrics for the benchmarking of the models | 18 |
| Table 5 Table of Benchmark Questions | 20 |
| Table 6: Feeling Model Question Metrics | 25 |
| Table 7 Experiment procedure | 26 |

Abbreviations

| | |
|-------------|--|
| RSC | Robot Study Companion |
| LLM | Large Language Model |
| LM | Language Model |
| NLP | Natural language processing |
| MOE | Mixture of Experts |
| SML | Small Language Model |
| AI | Artificial Intelligence |
| LM | Language Model |
| API | Application Programming Interface |
| GPU | Graphics Processing Unit |
| TPS | Token Per Second |
| TTFT | Time To First Token |
| RAG | Retrieval augmented generation |
| RPi | Raspberry Pi |
| QAT | Quantization aware training |
| PTQ | Post Training Quantization |
| RPI | Raspberry PI |
| HPC | High Performance Computing |
| CUQ | Chatbot Usability Questionnaire |
| AI | Artificial Intelligence |
| IOPS | Input/Output Operations Per Second |
| MMLU | Massive Multitask Language Understanding |

1 Introduction

Social-educational robots have gained increasing popularity and are being integrated into the higher education field. But there remains a need for a versatile, open-source, low-cost and accessible robot. Having reproducible, low-cost, and open-source robots is perfectly suitable for educational and research purposes in restricted environments. The Robot Study Companion (RSC) aims to address this gap and enhance accessibility in learning environments [1]. Its primary purpose is to provide interactive and multimodal learning support, thereby creating a more engaging learning environment. Open-source projects aim to provide modularity and facilitate sharing of improvements, designs and all possible applications within the project.

Initial studies indicate strong user acceptance of the RSC, with students appreciating its effectiveness and ease of use [1], [2]. RSC is a promising tool designed to enhance educational experience by providing interactive, customizable, and effective learning support.

The RSC project aims to build an adaptable robot that can fit every student and their learning style. Having an adaptable robot is important and can significantly enhance student engagement and performance. For instance, a study utilizing an adaptive response-selection algorithm demonstrated that measures of task engagement were higher with adaptive robots compared to non-adaptive ones [3][4][5]

Integrating Large Language Models (LLMs) via a common API represents one approach to bring this adaptability to the RSC. However, it brings some concern, especially in terms of privacy. The increasing collection and potential misuse of personal data by LLMs have raised significant privacy issues, necessitating the development of privacy-preserving techniques and local language models.[6], [7], [8]

Considering these factors, this thesis aims to explore possibilities to implement an LLM in the RSC while providing a safe and engaging environment for students. The goal is to identify viable locally deployed language models and architecture candidates for the RSC project. Additionally, this thesis proposes solutions for subsequent production integration.

1.1 Problem Statement

Providing efficient and accurate information support to students using limited computational resources is challenging due to the latency of the hardware and the difficulty of integrating optimal LMs (Language Model) in it. In the development of an affordable, open-source, and reproducible social robot, we end up facing some hurdles to reach the expected performance for high-level studies using limited hardware (RPi 4). A primary challenge stems from the huge size of such models, which often require internet connectivity, as the computational resources needed to run them locally are typically insufficient. In contrast, many research groups develop smaller, more specialized versions of these models that can operate on a local device. However, this approach introduces alternative limitations, such as increased hallucinations, which result in inaccurate information. For example, Gemma2 has, on HaluEval and TruthfulQA benchmarks, 63.9% of hallucination for the 27 billion parameters model, which increases to 73.6% for the 9 billion parameters and 79.0% for the 2 billion parameters [9]. Running a model

locally can also lead to performance issues, such as increased response latency, again due to limited computational resources.

1.2 Objectives and Roadmap

The main objective of this thesis is to find and create a suitable software pipeline that will allow the Robot Study Companion (RSC) to answer questions from students on specific topics. This full pipeline will lay the foundation for using the RSC offline with efficiency and improved privacy. This work directly contributes to RSC design requirements 4, 5, and 6 [1].

RO1: To benchmark and compare a list of open-source local language models on an edge device (Raspberry Pi 4 model B) by measuring latency, computational use and accuracy in the scope of the study.

RO2: To design an architecture for the RSC that enables fast, accurate, and engaging answers for specific topics on edge devices with limited compute.

RO3: To measure the perceived usability of the software architecture with the selected local language model through a controlled, informal user survey.

2 State of the Art

In the last few years, social robots interacting with humans have improved and brought numerous possibilities to develop a performant robot teaching assistant or intelligent tutor. In this educational context, a wide range of types of robots emerged, such as humanoids or more desktop ergonomic robots. Their range of uses varied notably from simple social interaction and presence to more complex and helpful responding. Additionally, since COVID-19, the number of online courses has been steadily increasing, demanding for something that can fulfill the missing social interaction need for students [10]. On the other hand, most social robots tend to be expensive and not accessible for students, which limits their use even if the interaction and learning experience with a teaching robot assistant can lead to potentially better results in terms of academic performance [11].

LLM research keeps improving and developing rapidly, bringing bigger, more complex, and often more accurate models. While the majority of open source LLMs can be found on sites like HuggingFace [12], most state of the art models remain closed source and often require internet access. Local and open-source models are bringing some advantages for the purpose of being a teacher assistant which include privacy and the ability to run without an internet connection. Most of these open-source models come in different sizes, lowering the size of the model lower computational needs for this one in most cases. However, this comes with a reduction in general accuracy and a higher probability of hallucination for the model [9].

This section examines the state-of-the-art research on deploying local language models on edge devices for educational robotics applications. The review focuses on three main areas: LLM with low computational requirements suitable for edge deployment; Voice assistants and chatbots; and LLM architecture.

2.1 LLM with Low computational requirements suitable for edge deployment

Identifying a suitable LM for edge devices presents a significant challenge. Several different methods with distinct advantages and drawbacks need to be used. Finding the balance between lowering accuracy and using a better device with more computational power is the key and needs to be adapted for each usage case. This subsection explores the various techniques used to reduce the size and computational requirements of LMs.

Quantization represents one approach to address deployment challenges on edge devices. Quantization is the discretization of model parameters (weights, activations) to reduce memory and computation [13]. This method involves reducing the precision of the weight matrices from, for example, 16 floating-point to 2-bit, 4-bit, or 8-bit quantization. However, as quantization levels decrease, model accuracy diminishes. The model sacrifices precision but gains inference speed and memory efficiency. Lower-bit quantization, such as 2-bit or 4-bit, achieves substantial efficiency improvements, ranging from 4× to 60× gains [14]. Nevertheless, this comes with a loss of accuracy of x2 or x3 on the NUBIA (NeUral Based Interchangeability Assessor for Text Generation) score [15], an automatic evaluation metric for text generation. Quantization can be applied to any language model, but some of them, like Phy3B are more resistant to quantization [14] and maintain higher accuracy under aggressive quantization.

Pruning represents another well-used method. Pruning is a method used to compress the size of a neural network model by removing neurons that contribute minimally to the architecture and do not have a significant influence on model performance (e.g. accuracy) [16]. Neuron removal can be achieved using structured or unstructured pruning [17]. These two methods aim to reduce the model size and the memory footprint by suppressing layers or neurons within layers.

Using Knowledge distillation can significantly improve accuracy on specific domains for small language models. [18] Knowledge distillation involves creating high-quality domain-specific data using a bigger LLM and incorporating this high-quality data into the SLM. When trained using adapt and distill, smaller models can surpass Bert LM on specific subjects by up to 4 points on ACL-ARC, which is a dataset used to classify citations in scientific papers and work[19], while achieving inference speed that is five times faster than their (bigger, more complex) counterparts [20]. This approach works precisely on very specific domains and is a suitable strategy for developing low-computation experts in a given domain.

The Mixture of Experts (MoE) architecture is another approach that can drastically reduce computing requirements. MoE is composed of multiple smaller models that are selectively activated to lower the computational resources used. This architecture divides the input space into specialized subspaces, with each specialized expert dealing with a specific task. Each of the expert answers is then gathered with a gating network that determines the optimal combination of individual expert answers [21]. This method substantially reduces the calculation resources required to answer with accuracy. However, it requires considerably

more memory, as you need several small expert models that combined will end up taking a lot of memory space. Mixtral is a good example of a large mixture of experts that can answer questions particularly precise by selecting which expert will be most appropriate [22].

For the last years, every major actor in the LLM field has aimed to create bigger models with extended context length to enhance efficiency. For example, the ChatGPT 4 turbo version has a context length of 128 000 which is 31x more than ChatGPT 3.5 (with a context length of 4096 tokens). However, increasing context length requires models with more computational power. Furthermore, sometimes increasing the context windows can reduce accuracy for certain tasks [23]. A possibility to solve this issue is to bring a side memory to the model, like MemGPT [24]. This memory, stored in a database, contains external documents and information that transforms the model into a more domain-specific agent. It maintains a summary of the past conversations, including the most relevant information, and updates it when necessary. This information is stored in a vector database, and then, using the similarity between the prompt and the database, the important part will be injected into the prompt, allowing the model to generate more precise and accurate responses.

2.2 Voice assistants and Chatbots

Recently, thanks to the development and the rise of AI voice generators such as ElevenLabs [25] or Artlist [26], AI assistants have become increasingly realistic. These realistic and accessible voices enhance the social and engaging quality of human-machine interaction. Nonetheless, such applications require API access and, therefore, internet connectivity. There are still some open-source models that are available on Hugging Face [12] which can be fine-tuned locally and perform well for commonly used languages. These models still show some limitations in expressing emotions and encounter difficulties processing smaller languages such as Estonian. The recent emergence of N8N [27] brings accessibility to a wide range of AI agents that can be easily equipped with tools such as voice synthesis and phone calling tools, which bring easy access to an AI voice assistant for non-expert users.

2.3 LLM Architectures

Transformers represent the most commonly used architecture in LLMs. The transformer architecture comes from the paper 'Attention is all you need' [28]. This makes the model rely solely on a self-attention mechanism that weighs the importance of different elements in sequential data, allowing the model to recognize complex patterns. The architecture is composed of an encoder and a decoder. Most popular LLMs such as Gemini3[29], Gpt5[30], Claude 4.5[31], are based on the transformer architecture.

However, alternative architectures exist and bring other advantages. The Mamba [32] architecture is based on a state space model, which allows it to capture longer dependencies than transformers. Mamba2[33] is an advanced version of Mamba that introduces space state duality, thereby increasing efficiency during multi-token processing.

Some models try to combine these architectures to create hybrid models that will handle longer context at low cost while maintaining high accuracy. Zamba [34] is a model composed of a backbone of mamba block with a single transformer block at the end, enabling it to be

fast and low cost while maintaining efficiency. Jamba[35] integrates Mamba layers between transformer blocks and incorporates MoE components inside his architecture. The best-known hybrid model is Granite 4.0 [36], which features one transformer block per nine Mamba blocks. Each hybrid model is trying to find the perfect balance between the accuracy of the transformer and the low-cost efficiency of the Mamba transformer, especially at a long context window.

3 Requirement

The requirements for this thesis must align with the requirements of the environment the model is going to be integrated. The RSC already has established requirements, and integrating a language model into it addresses some of these. Additionally, more specific requirements must be defined to narrow the RSC requirements to more suitable and specific Language model requirements.

The RSC already has six main requirements defined by Baksh et al [1]. The development of the Local language model pipeline will fall under the specific requirement number 4: *“The RSC should provide Adaptable Learning Support.”* Developing an adaptable software architecture around a language model fulfills the requirement for adaptable learning support. This approach can also motivate students and bring real social interaction. This LM can also be highly versatile, adapting to various subjects and different kinds of teachers' requirements.

Table 1 Requirement for Study companion [1]

| | | |
|-------|---|---|
| Req 4 | The RSC should provide Adaptable Learning Support | The RSC should analyse student performance, adjust responses based on learning styles, recommend customized study plans, and apply gamification to boost motivation. This approach may enhance engagement and support individual learning needs in diverse educational contexts. The RSC should personalise learning in line with students' and teachers' perspectives. The RSC should support a variety of subjects or courses to provide host academic institutions curricula or programs. |
|-------|---|---|

To better address requirement 4, some specific requirements for the Local language model component have been defined:

Req1. The local language model should be adaptable and personalized for the user.

As a purpose of bringing the best learning experience, the user should have access to a lot of possibilities to personalize the experience to their needs. The local language model needs to be able to remember a few personal information, from simple names to his learning style. It should also be fully adaptable to every possible course that the user wants to learn.

Req2. The local language model should be built around an education and learning perspective

The local language model has only one main objective: to be the best learning support for the user. It should not bring any additional features if it does not increase the learning experience of the user.

Req3. The local language model should provide a safe learning environment

To provide a good learning environment, the local language model should be equipped with a guardrail to avoid any disrespectful or hateful behavior. It should also provide a safe personal information storage policy. The personal data should not be shared with any output sources and respect the private lives of the user.

4 Methodology

To develop a fully adaptable learning pipeline with a local language model, four main steps must be completed: research existing solutions for implementing LLM on edge devices. Combine the best solutions/models and compare them. Assess the chosen model with user feedback. Build the final pipeline around the best solution.

4.1 Research Methods

Research on LLMs for edge devices reveals two main problems: insufficient inference and potential hallucination of the models. Given the rapid evolution of the NLP field, a paper that was written one year ago may be outdated with the frequent emergence of new models and solutions.

Furthermore, most existing research about building a chatbot on edge devices includes access to the internet and use of global API from major companies such as OpenAI [30] or Google [37]. The primary research gap in this field involves building a local, personalized, and efficient chatbot on an edge device.

4.2 Limitations and possibilities

Based on the literature [14], [38], [39] regarding LLM deployment on edge devices and analyzing the requirements of the RSC, several limitations need to be set up to choose and develop a performant local LM. These constraints enable focus on the main aspect necessary for the chatbot integration into the RSC.

4.2.1 Local Limitation:

The RSC is designed to operate without access to the internet following initial setup. This restrains the selection to locally deployed LM. API based LLMs such as ChatGPT or Claude [40] cannot be used. This limitation necessitates the use of an open-source model that can be downloaded and implemented locally. This constraint introduces two additional limitations (Memory and computational limitations) linked to the fact that it needs to be run locally.

4.2.2 Memory Limitations:

Given that the Raspberry Pi (RPI) 4B has only 8GB of RAM, the size of the model that can fit on it is limited. Additional memory must be allocated for system tools, pipeline operation around the model, and hardware control (speakers, light, panel).

Based on preliminary research, the first assumption is that the biggest size that fits on the RPI 4B is around 3 to 4 billion parameters models (roughly equivalent to 4GB). Specific architectures (Mamba) or methods (big quantization) could extend this limit to 7 billion parameters, but it requires some testing to ensure the good functioning of the RPI.

4.2.3 Computation Limitations:

Initially, the RPi was designed to be a learning tool for teaching the basics of programming and electronics to kids. This had then been taken over by people for its low price and capacity to achieve a lot of IoT or robotic tasks. However, its original design did not anticipate the possibility of running local LM at an acceptable speed. For comparison, the latest GPU of Nvidia (RTX 5090 [41]) is 7,000 times faster than the RPi 4 B. Computational limitations represent one of the most significant constraints for the RPi 4. For comparison, a gaming computer can execute models 10 to 30 times faster than the RPi 4 (Table 2).

Table 2: Performance comparison between Raspberry Pi 4 B and other devices

| Device | Computer Performance (FP32 equivalent) | Performance compared to PI4B | Potential size of model (Billions of parameters) |
|---|--|------------------------------|--|
| Raspberry Pi 4 model B | 15 GFlops | 1x | 1 - 4 |
| Raspberry Pi 5 | 36.6 GFlops | 2.4x | 2 - 7 |
| iPhone 15 Pro | 1 TFlop | 67 x | 2 - 4 |
| Samsung Galaxy S22 | 1TFlop | 67 | 2 - 4 |
| Desktop computer (Intel Core I7-13700k) | 2-3 TFlops | 134-201 x | 7 – 13 |
| Gaming computer (RTX 4090) | 80 Tflops | 5 333 x | 13 – 70 |
| Latest GPU (RTX 5090) | 105 Tflops | 7000 x | 70 - 150 |

4.2.4 Language Limitation:

Multilingual models are uncommon for low-parameter models. Training a fully operational Multilingual model requires a lot of data for each language, which results in a heavy model. That is why the multilingual models that are lower than 7 billion parameters focus on the “most used” languages such as English, Spanish, Chinese, French, Russian, and German. This limits the use of those models for specific cases, such as language learning for less-known languages, such as Estonian. Fine-tuning for specific language can reduce this gap, but it does not fully solve the inaccuracy in the answers. That will lead to serious problems in the learning process of the student.

4.2.5 Personalization possibilities:

Given the limited context windows of small models, alternative methods need to be found to bring more information or personalization to the local LM is important. Retrieval Augmented Generation (RAG) represents the most effective and used approach. RAG System enables bringing into the context windows the most relevant learning material and personal information. This allows us to provide personalized answers using a limited number of tokens. Because the model used will be open source, each user of the RSC can fine-tune the model on their own for a specific topic.

4.2.6 Model possibilities:

Looking at the latest open-source models, most of them are concentrated on the Hugging Face platform. All selected models should fit on the RPi 4B and will be tested on several metrics to identify the optimal candidate. The main open-source models are available and easy to use using the Ollama library. This library is up to date and includes a fine-tuned and quantized version of recent models. This makes it easy to perform a benchmark on it.

4.3 Local LM Benchmarking

The wide variety of available models can be confusing, and a comprehensive benchmark to determine the best models on edge devices is limited. By creating a full pipeline that can benchmark every new model for edge devices, being up to date with the flow of new models coming every month will be possible.

Defining the most suitable option for our model implies testing a large number of models available, on metrics such as teaching effectiveness and resource consumption. This benchmark evaluates models’ abilities to provide teaching help in constrained environments.

4.3.1 The models

Numerous models are available, and the benchmarking needs to be narrowed to the one that can fit the purpose of the RSC. Several constraints were established to choose a variety of models for the benchmarking.

The Number of parameters needs to be lower than or equal to 8 billion.

As described in the 4.2 section, for our use, we should restrict the parameters of the models to less than 8B (4.4GB of RAM used for Mistral 7Billions for example). To better compare the models and for the architecture of the pipeline that will be explained later in section 4.5, the

size comparison will be divided into two distinct groups. The ‘Small’ models group should be restricted to models with less than 2 billion (1.6GB of RAM used for Granite4 hybrid 1Billion parameters for example), and the ‘Big’ model group should include models that range between 2 and 8 billion parameters.

The choice of models should cover a wide range of sizes

To ensure an efficient and comprehensive benchmark, the variety of model size should be as wide as possible (it should range between 0 and 8 billion parameters).

The choice of models should cover a wide range of architecture

Various approaches and architectures must be evaluated to see the advantages and drawbacks of these approaches.

The models need to be open source

Given local deployment requirements, Open-source models are the only models that can be deployed on the RPi. This excludes all the API models, such as Chat-GPT or Claude as well as the Ollama API models.

The models need to be recent (less than 1–2 years old)

This benchmark should focus on the most recent and performative models. With this in mind, the choices of the models for testing have been made around October 2025, by the end of this thesis, therefore, some new, better models might already beat the current chosen one.

The models should be available on the Ollama framework

The Ollama framework regroups the main models available and facilitates deployment. Although Hugging Face offers a wider range of models, a fast and efficient way to test them has priority. Ollama also supports hybrid models such as the Granite hybrid.

After a deeper search on the Ollama library, a total of twenty-five models were selected for the benchmarking, ranging from 270 million parameters to 8 billion parameters. This list features classical transformers, hybrid models (mamba-transformers), and reasoning models. Some models are already using different quantization techniques that will be taken into consideration for the result analysis.

Table 3: Language Models chosen for benchmarking

| Name | Model parameters (Billion) | Ollama Name | Quantization | Real Size (GB) |
|-------------|----------------------------|-------------------|--------------|----------------|
| Gemma3 | 0.27 | gemma3:270m | Q8 | 0.291 |
| Qwen3 | 0,6 | qwen3:0.6b | Q4_K_M | 0.522 |
| Tinyllama | 1,1 | tinyllama: latest | Q4_0 | 0.637 |
| Deepseek R1 | 1,5 | deepseek-r1:1.5b | Q4_K_M | 1.1 |
| Qwen3 | 1,7 | qwen3:1.7b | Q4_K_M | 1.4 |
| Gemma 3 | 1 | gemma3:1b | Q4_K_M | 0.815 |
| Llama 3,2 | 1 | llama3.2:1b | Q8_0 | 1.3 |
| Granite4 | 1 | granite4:1b | BF16 | 3.3 |
| Granite4-h | 1 | granite4:1b-h | Q8_0 | 1.6 |

| | | | | |
|---------------------|-----|--------------------------|--------|-----|
| Falcon3 | 1 | falcon3:1b | Q8_0 | 1.8 |
| Exaone deep | 2,7 | exaone-deep:2.4b | Q4_K_M | 1.6 |
| Phi3 | 3,8 | phi3:3.8b | Q4_0 | 2.2 |
| Phi4 mini-reasoning | 3,8 | phi4-mini-reasoning:3.8b | Q4_K_M | 3.2 |
| Llama 3,2 | 3 | llama3.2:3b | Q4_K_M | 2.0 |
| Granite4 | 3 | granite4:3b | Q4_K_M | 2.1 |
| Granite4-h | 3 | granite4:3b-h | Q4_K_M | 1.9 |
| Cogito | 3 | cogito:3b | Q4_K_M | 2.2 |
| Falcon3 | 3 | falcon3:3b | Q4_K_M | 2.0 |
| Gemma 3 | 4 | gemma3:4b | Q4_K_M | 3.3 |
| Qwen3 | 4 | qwen3:4b | Q4_K_M | 2.5 |
| Nemotron-mini | 4 | nemotron-mini:4b | Q4_K_M | 2.7 |
| Mistral | 7 | mistral:7b | Q4_K_M | 4.4 |
| Granite4-tiny-h | 7 | granite4:tiny-h | Q4_K_M | 4.2 |
| Deepseek R1 | 7 | deepseek-r1:7b | Q4_K_M | 4.7 |
| Cogito | 8 | cogito:8b | Q4_K_M | 4.9 |

4.3.2 Metrics

The performance of the different models needs to be assessed on two different types of metrics: hardware metrics and accuracy metrics. This will assess the capacity of the models to achieve their teaching goal while being on an edge device such as the Raspberry Pi 4 model B.

Table 4: Table of metrics for the benchmarking of the models

| Metric | Description | Type of metrics |
|---|---|-----------------|
| Token Per Second (TPS) | Number of tokens generated by the model in one second (this is an average based on the total time while generating tokens divided by the total number of tokens). | Hardware |
| Inference time | Total time spent on the generation of one output (time between the query being input to the model until the last token) | Hardware |
| Time To First Token (TTFT) | Time spent to generate the first token of the output. | Hardware |
| Response length | Total number of tokens for an output | Hardware |
| IOPS | Input and Output memory operations | Hardware |
| Token Per Joule (TPJ) | Number of tokens generated by the model for one joule consumed | Hardware |
| Teaching effectiveness | Rating of the output based on teaching criteria on a scale of 1 to 10 (Judged by a bigger LLM: gpt-4.0-mini) | Accuracy |
| Massive Multitask Language Understanding (MMLU) | Benchmark to assess the general knowledge of a model in several topics) | Accuracy |

What is MMLU?

MMLU (Massive Multitask Language Understanding) [42] is a benchmark used to test specific knowledge in a wide variety of categories. It highlights the strengths and weaknesses of the different models in certain categories. MMLU was developed to replace and improve the already existing GLUE benchmark [43], which was not efficient enough and did not reflect the actual weaknesses of the models. This benchmark is composed of 57 categories from high school subjects to formal logic and more abstract domains such as business ethics. The benchmark contains 15908 questions spread across these categories. Each question presents four answers (A, B, C, D), avoiding judging on token precision but focusing more on pure knowledge and binary answers (right or wrong). This benchmark asks questions in a really specific way and expects a really specific output. This means it also highlights models that have good instruction following capacities. This is perfect for the use of the RSC, making MMLU one of the most important metrics of the benchmark. The way the questions are built makes the randomness to 25 % of accuracy. If a model reaches this level, it has almost no knowledge in this field and has a greater tendency to guess and hallucinate rather than to give an accurate answer from its training data.

Domain experts achieve an average of 89% of good answers, making it a good target and a good potential comparison point. Therefore, for limited models such as the one used in this experiment, reaching 50% accuracy represents a strong performance.

This benchmark is available in the deepeval Python library [44]. This framework enables benchmark evaluation and topic-specific testing. No prompt is needed using this library, it is already implemented to format the answers of the models in the way expected (A, B, C, D). However, this automatic prompt can be a problem for small models as it requires them to follow a complex prompt. This should lead in our experiment to some poor results for models with fewer parameters than 1 billion. Nevertheless, the main objective involves testing global knowledge of 'big' models as they are going to be the ones handling the majority of queries.

As processing MMLU is really time and resource-consuming, focusing only on the specific topics we want our models to excel in is important. To accelerate the process, the benchmark will be done on the HPC cluster of Tartu University. This will be performed on a single Tesla A100 40GB.

For the topics, our RSC should be adapted to a university environment, have some global knowledge, and global logic. That's why the MMLU benchmark will only be performed on 6 topics:

Formal logic, global facts, college computer science, college mathematics, marketing and high school macro-economics.

4.3.3 Benchmarking method

Model performance was evaluated using a predefined set of ten questions assessing their ability to provide teaching assistance to potential students. On each question, each metric will be recorded and stored in a CSV file and a JSON file that will be stored in GitHub [45]. At the end, all the data will be gathered for comparison, visualization and analysis.

Small models struggle with prompts and tend to implement them in their answers, making their answers not representative of their teaching skills. Therefore, they will be separated into two groups. The bigger models (more than 1 billion parameters) will have a prompt, but not the others.

Here is the prompt that is going to be used on every big model before asking the questions:

“You are a teaching assistant, your goal is to provide accurate answers and information in a comprehensible and interesting manner for a university student in the field.”

The questions assess a range of skills necessary to be a good teaching assistant.

General teaching skills: These will be generic questions, and expect the LM to answer for someone without skills or with intermediate skills. It should assess the general knowledge and capabilities of explanation by interacting with the student.

Adaptability: The LM should be able to adapt itself to different learning styles (gamification, quizzes, analogies) and weaknesses (like asking questions and telling the LM that the user has some lacks in a specific part of the topic/questions).

Casual talking: The LM should also be able to handle casual talking with the student to have a social interaction with them in an informal way.

Table 5 Table of Benchmark Questions

| # | Questions | Use |
|----|--|---|
| 1 | Describe gravity in simple terms for someone who doesn't know anything about it. | Simple knowledge testing |
| 2 | Describe gravity for someone who knows about it and wants to know more about it. | Testing of the adaptation to different levels of the user |
| 3 | Find a good analogy to describe gravity to someone. | Adaptability |
| 4 | Give a small quiz (3 questions) about gravity | Capacity to provide good learning support |
| 5 | Do you think I need a big jacket if the weather is 10 degrees | Casual Talking |
| 6 | Give a common myth about gravity and explain why it is false | Capacity to deal with confusion |
| 7 | Explain why a plane can fly even with gravity in simple bullet points | Capacity to simplify |
| 8 | What are the steps to write a master's thesis? | Student guidance |
| 9 | What are the pros and cons of nuclear energy | Critical thinking |
| 10 | I am struggling with C++, where should I start? | Student guidance |

To assess model performance, an external, bigger LLM model will be used (gpt-4o-mini). This model will be used via the OpenAI API and will be given a special prompt to make it rate the teaching skills of the different models.

This prompt is composed of multiple components:

- The purpose (identity - teaching expert that evaluates. Objective – rating from 1 to 10)
- The constraint (consider those criteria to evaluate)
- The question-answer given to evaluate.
- The expected output (JSON with results)
- Safeguard (only output JSON and nothing else)

Full prompt:

“You are an expert educator evaluating the teaching effectiveness of AI responses. Rate the following response on a scale of 1-10 based on its ability to teach effectively. Try to be sure that your rating is strict enough.

Consider these criteria:

- 1. Clarity: Is the explanation clear and easy to understand?*
- 2. Accuracy: Is the information correct?*
- 3. Engagement: Does it engage the learner?*
- 4. Structure: Is it well-organized?*
- 5. Completeness: Does it adequately address the question?*
- 6. Actionable: is it suitable for a talking conversation with a student ?*

QUESTION: {question}

RESPONSE TO EVALUATE: {response}

Provide your rating in this exact JSON format:

```
{{  
  "score": <number 1-10>,  
  "strengths": ["<strength 1>", "<strength 2>"],  
  "weaknesses": ["<weakness 1>", "<weakness 2>"],  
  "brief_justification": "<1-2 sentence explanation>"  
}}
```

Only output the JSON, nothing else. ""

“

4.3.4 Benchmarking on other devices

As the possibility of RSC improvement is still on the table, having a comparison with better micro-controller and hardware will help analyze how much performance increases with hardware improvement.

This benchmark will also be run on Raspberry Pi 5 and a computer with an RTX 4060 using the same process as on the RPi 4 model B.

4.3.5 Benchmarking method update

During the performance of the benchmarking some problems that were not expected arose. The process, metrics, and assessment needed to be modified to bring a good and fair representation of the performance of the different models.

Testing under the same conditions: As the benchmark requires a lot of computing power from the RPi4 model B, the device heats up fast and reaches its maximum temperature. This activates the throttling making the RPi computing slower and therefore can make the

benchmark inaccurate if a model starts already at maximum temperature. To solve that, the RPi needs to have some time to cool and reach the same steady temperature before starting the benchmark on each model. This temperature was set to 60° Celsius to keep the cooling process quite fast. The RPi4 model B takes around 15 minutes between each model.

New metrics added: Some new metrics are going to be added to record the storage metrics. All the reading and writing are recorded and stored. The average IOPS are going to be calculated to see how memory heavy each model is.

Inference time calculation: Firstly, the time for the models with taken using the time python library. This might not be precise enough so in addition, the time will be taken using time.perf_counter directly in the internal clock of the RPi.

MMLU problem and improvement: After performing the first try of the MMLU, some problems come out, some models (such as thinking models or reasoning models (deepseek-r1, qwen3, Exaonedeeep)) have a hard time answering only a letter even when prompted. This comes from their thought process that is done out loud. This led to these models having 0 percent accuracy which does not reflect their actual knowledge. However, it shows that those models will add more complexity and problems when it comes to prompting and filtering the answers to the user in general. To solve this issue, a quick fix, restraining their thought process and excluding it from the answers has been put in place. Each reasoning model has different ways of thinking (different tokens to start and end thinking) so if new reasoning models are being tested with the same code, it might require some minor changes in the thought process handling.

Internal Reasoning models speed problems: Deepseek-r1 and qwen3 also have another unexpected problem, the results showed them way faster than the other deep reasoning models. A problem has been found in the benchmarking code, as deepseek-r1 “think” internally and not out loud, the timer started only at the end of the thinking. This problem leads to inaccurate results especially for TTFT, which leads to an inaccurate inference time and token per second. As this error came late and the models to benchmark are really long (especially for deepseek-R1 7B), the data will stay with this incoherence, but this will be taken into account in the explanation and analysis of the benchmarking.

4.4 Finetuning and quantization

After choosing our best raw models, we will have to have a look at making them as efficient as possible for our use case. For that, we will use two main methods: first, quantization to decrease the memory and computational requirements of our model and keeping most of its accuracy and fine tuning on some teacher data behavior to make him better at teaching some lessons to the students. As no ‘teaching dataset’ exists, it is complicated to perform fine-tuning this way, the closest that it can get is instruction fine-tuning. This will allow the models to be better at following instructions and prompts, and as a consequence stick with the teaching assistant’s purpose they need to follow.

There are two ways to implement quantization to LLM: one during pre-training (QAT) and one post-training (PTQ).

Post-training quantization takes a pre-trained model that usually has a precision of FP32 and reduces the precision of the weights or activations. This can be reduced to FP16, INT8, INT4, INT2. This allows the model to be run using lower memory and computation resources. However, it comes with reduced accuracy. The bigger the quantization, the more the accuracy decreases.

Quantization aware training happened during the training phase. During training, some quantized nodes are added to the models, and the gradient keeps full precision. The loss function is influenced by the quantization, forcing the model to adapt to this quantization and be more robust to this loss of precision. Those 'fake' nodes are suppressed at the end of the training, and then the model is fully quantized.

QAT needs involvement during the training process and requires more computational resources. PTQ is easier to implement and is not really resource consuming. However, QAT is known to have a performance closer to the initial model than the PTQ which makes it better at accuracy for the same level of quantization.

4.5 The Three-Tier Architecture

Balance between speed and accuracy is hard to find especially when the resources are limited. Bringing new and innovative solutions to improve inference time while keeping a good accuracy in the answer is mandatory to provide a good and engaging learning experience.

After reviewing the actual different state of the arts methods to reduce the size and computational necessity of the local Language Model, I concluded that a compromise would have to be made between accuracy and speed. As we are working to make student progress with accurate help, we need to bring accuracy as the main objective. However, to keep the student in a good learning environment, we need to bring him an interactive and responsive robot. To deal with that, a three-tier based model will be used in order to bring most of the time an efficient and fast answer to the students without sacrificing accuracy.

This idea will take the base of the GPTcache method described in this paper[46]. GPTcache uses vectorial storage of old queries or questions from the users that are stored in a cache and can be displayed again if the same question is asked, avoiding the use of the LLM in this case. It was initially designed to reduce the cost of using the API when similar questions are asked to the LLM (like in a chatbot). It also used a similarity between the user question and the ones in the database using semantic search. This method is, however, inefficient for production as the answers from a user with 90 percent accuracy with another question don't have the exact expected answers.

The architecture is inspired by the GPTcache architecture but will implement an intermediate level. This cache will only be used if there is a high accuracy (95 percent or higher). In this intermediate level (from 95 to 80 percent), a smaller model (1B or lower) will take the lead, and using a similar question/answer from the cache and the associated part of the document it refers to will generate a new answer only by rephrasing and making small adjustments to fit the answers to the actual questions. This can be done by using a less powerful LM as it does not require a lot of knowledge and the possibility of hallucinations is low. If the questions are too low, the big model will take the lead and answer slowly to the users. After that using the

GPTcache architecture, these questions will be stored in the vectorial database if the user asks the same question again.

The last problem that needs to be addressed in order to make it an efficient architecture is to fill the cache with some questions and answers before the student can use the RSC. The cache is empty at the beginning of use, making the first user's questions always slow to be answered. To solve this problem, the inference time of answers will be moved to the initiation of the RSC. When a new PDF or document is implemented inside the RSC, the big model will automatically generate a certain number of questions that the user might ask, produce the answers, and store the pair in the cache. Which means that at the first use of the RSC by the user, the cache will already be filled with a lot of questions, and it will use the small model or the cache more often, making the interactions faster and smoother while keeping the accuracy of the bigger model.

4.6 User Model interaction experiment

The model and architecture are built in a user-centered way. The feeling and feedback from the students are really important to improve the interaction and the architecture. Having real students will help gather real questions and general feelings about the answers that the model can give. Getting this evaluation can also help gather some precious information about the behavior of the users in a learning interaction with a Language Model. Also, testing different types of interaction with the model and the potential impact of this difference on the general feeling of the student can be interesting to analyze.

This experiment will have two main objectives:

Objective 1: Get the general feeling and feedback of the students for the chosen model that is going to generate the questions and answers pairs. Also, does this feeling change depending on the interaction (written or spoken).

Objective 2: Get real student questions to better understand how to modify the prompt and threshold to adapt the generation of the question to real users needs.

This experiment will have two main objectives:

4.6.1 Metrics

The metrics retrieved in this experiment are from two distinct perspectives. CUQ metrics are extracted from Ulster University [47] and is designed for measuring the usability of chatbots. The questions from this questionnaire are filtered and modified to fit the user's feelings about the model itself and not the chatbot experiment system (Table 6). Those questions can be answered in a range from strong disagreement to strongly agree (Figure 1).

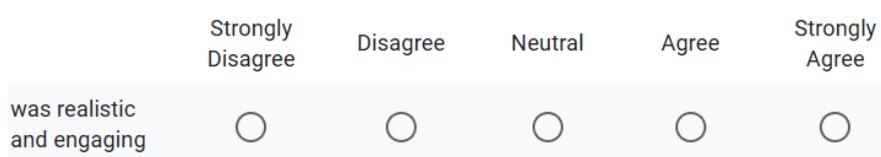


Figure 1 Example of CUQ

Table 6: Feeling Model Question Metrics

| |
|---|
| The model was realistic and engaging. |
| The model seemed too robotic. |
| The model seemed very unfriendly. |
| The model explained the scope and purpose well. |
| The model gave no indication of its purpose (no explanation of who the model is and how it should behave) |
| The model understood me well. |
| The model failed to recognize a lot of my inputs. |
| The model seemed unable to handle any errors. |
| The model responses were useful, appropriate and informative. |
| The model responses were irrelevant. |
| The model was easy to use |
| The model was very complex. |
| The model was easy to navigate. |
| The model would be easy to get confused when using. |
| The model coped well with any errors or mistakes. |

4.6.2 Methodology

A qualitative study will be conducted on a sample of 10 students from Tartu University. Those students are from different fields of study and different levels of study from bachelor’s to PhDs. The topic chosen for this experiment is *LLM and how they work*. The topic is meant to be complex in order to allow the user to find questions and see if the answer of the model actually helped the student or not. This one-page PDF is generated by Chat-GPT 5.1 with the simple prompt: ***generate a one-page PDF about LLM and how they work. This PDF should be understandable by a master student in the machine learning field.*** The PDF is, as expected hard to understand for someone who is not in the field and contains the main component and explanation about the functioning of LLM.

A special code will be made to create the interaction of the student with the model, as well as allowing the storage of the question of student. This code is available on the GitHub repository [48]

The students will be interacting with the model using a computer and a classical QWERTY keyboard. The experiment will be performed in a quiet room to ensure a good understanding of the participant and the model for the voice part.

4.6.3 Experiment Procedure

The experiment procedure will always be the same for each participant and in the same conditions. The total time of the experiment is between 35 and 40 minutes, including 20 minutes alone interacting with the model itself.

Table 7 Experiment procedure

| Phase | Activity | Duration |
|------------------|--|----------------|
| Introduction | <p>Welcome the participants and explain the experiment's purpose, the time it is going to last and how it is going to be performed.</p> <p>Ensure participants understand the confidentiality and consent process, warning that questions are going to be collected in the backend to improve the architecture. Each participant will have to give their consent before starting the experiment. They are allowed to stop the experiment at any moment if any problems occur.</p> | 5 minutes |
| Text Reading | The participant is invited to read the one-page PDF ' <i>How Large Language Models Work.</i> ' The participant is specifically invited not to get stuck on some parts of the text that are too hard for them. | 5 minutes |
| Task Execution | <p>The scenario is explained to the participant '<i>You are revising for an exam on this paper. This model is here to help you ask as many questions as possible to train and revise for your exam.</i>'</p> <p>Phase 1: The participant is interacting with the model on a text-based based, using the terminal. They have to ask at least three questions to the model. The participants can still look at the provided PDF to find the question they want to ask during the experiment. He is invited to read all the answers carefully. When the user is finished with the questions. The CUQ form will be given to him, and he will have to fill it with his personal feelings about the model in a written interaction.</p> <p>Phase 2: The participant is now interacting with the model on a speech-based based, the procedure is the same as in phase 1. They can also reuse the question in phase 1 for this phase. When the user is finished with the questions. The CUQ form will be given to him, and he will have to fill it with his personal feelings about the model in a spoken interaction.</p> | 20 minutes |
| Post-Test Survey | <p>The participant is invited to share his opinions on the model, the thesis and everything he wants to share. He can either give his feedback in the form or by talking with the conductor of the experiment who is going to take note of it.</p> <p>The participant should then fill in personal information, such as level of study or field of study.</p> | 5 – 10 minutes |

4.6.4 Data collection procedure

We need to collect as many real user questions as possible to estimate what kinds of questions students are asking, in what way and how long the questions are in general. This data will help

build a better question generation procedure by improving the prompt via general direction or specific question examples.

All questions will be collected according to what the participant agreed to during the experiment. Each question will be stored anonymously in a CSV and JSON file during the experiment. That data will be stored in a private GitHub repository where only some members of the RSC team have access to. Each question is pushed into the GitHub repository after each user testing to ensure that the data is not getting lost.

5 Results

5.1 Benchmarking Results for the Raspberry Pi 4 Model B

Benchmarking on the RPi4 model B ended up being very time-consuming. To ensure more replicable results, we took the time to cool down the RPi between experiments into account carefully. All aggregated test results are available in multiple formats (JSON, CSV, or in graphs) in the provided GitHub repository [45].

Exported raw data from the RPi benchmarking required additional processing, which made them more readable and analyzable. This was performed using a separate script that generated graphs for each metric across both models (lower than 2 billion parameters and between 2 and 8 billion parameters)

5.1.1 Small model results (model with fewer than 2 billion parameters)

It is important to take into consideration some important distinctions in this ‘small’ model section. Some models occupy more disk space than others despite having an equivalent number of model parameters. This is mainly due to the difference in quantization between them. Additionally, certain models, such as granite-1B, are in fact more around 1.6 billion parameters but are described as 1B for consistency with their hybrid counterpart, Granite-1B-h. As said before, some reasoning models were not recorded correctly, leading to incorrect data for the hardware metric. These models may appear faster than they are in reality, and this applies to Deepseek-R1 and Qwen3, for example. Regarding TTFT, each model must be loaded before the first prompt. Therefore, initial TTFT is higher than subsequent TTFT. The graph will be split into the initial TTFT and the average TTFT for the 9 other questions.

Token per second: As expected, small models perform better in token generation. Gemma3 with 270 million parameters achieved 8.7 tokens per second on the RPi 4B. Falcon3 performed below average compared to the other 1 billion parameter models. Granite4 is well behind due to its large size (3.3GB). On the other hand, TinyLlama (1.1 billion parameters) is almost twice as fast as the similarly sized models while also being really light (637 MB). The two larger models (deepseek-r1:1.5B and qwen3-1.7B) of this category seems to perform well and even better than some 1B models but that is again due to their reasoning not taken into account.



Figure 2 Token Per Seconds (models less than 2 billion parameters)

TTFT: Considering the TTFT Granite4 1 billion is by far the slowest model due to its massive size. Surprisingly, granite4 1 billion hybrid demonstrates one of the fastest TTFT on the first question when it needs to get loaded but the second worst for average TTFT on the other questions. As expected, smaller models load faster and generate the first token more quickly than larger counterpart.

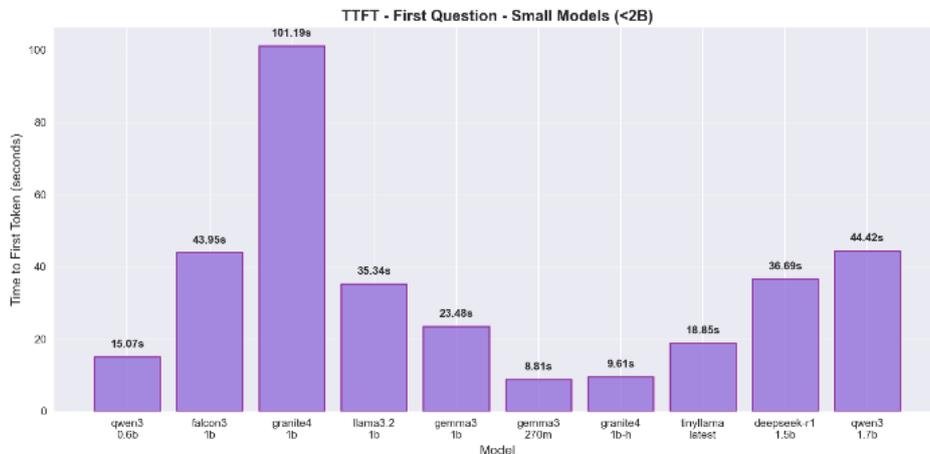


Figure 4 First question TTFT (Model lower than 2 billion parameters)

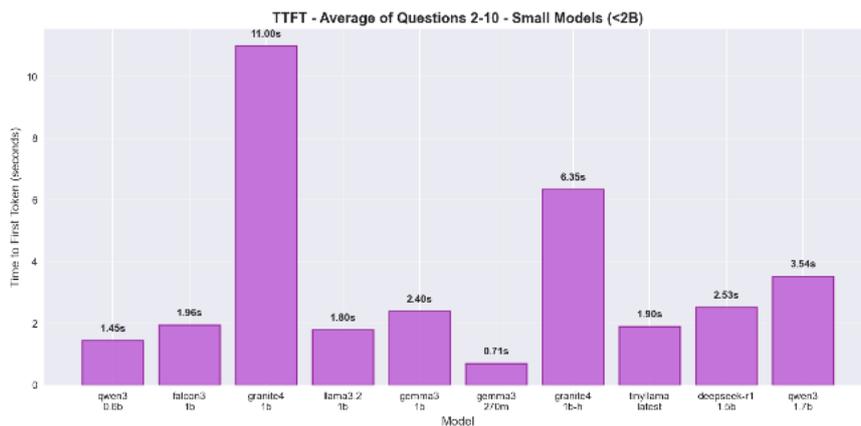


Figure 5 Average TTFT of the last 9 questions (Model lower than 2 billion parameters)

Response length: Considering the response length, Gemma is the model generating the most characters and consequently tokens up to 2814. In this analysis, Deepseek-r1 and Qwen appear relatively low because their internal thinking characters are excluded. Including reasoning processes, they would be at least twice as high as gemma3:1B. Granite4:1B exhibits the shortest response length. Response length must be taken into account as it directly influences total inference time.

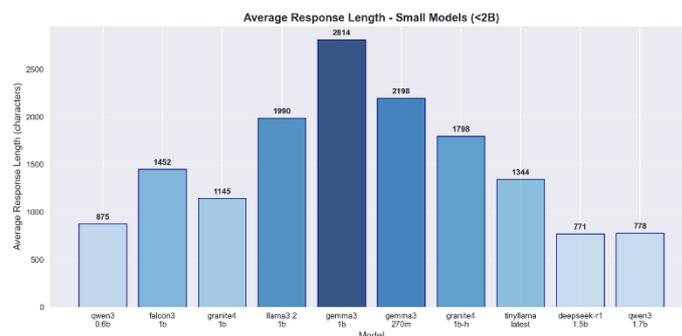


Figure 3 Response length (model less than 2 billion parameters)

Inference time: Considering total inference time, gemma3:270m and Tynylama achieve the lowest inference time (1-2 seconds). Tynylama also demonstrates remarkable consistency. Other models range in average from 2 to 4 seconds of inference time, the Granite family being the slowest (excluding reasoning models).

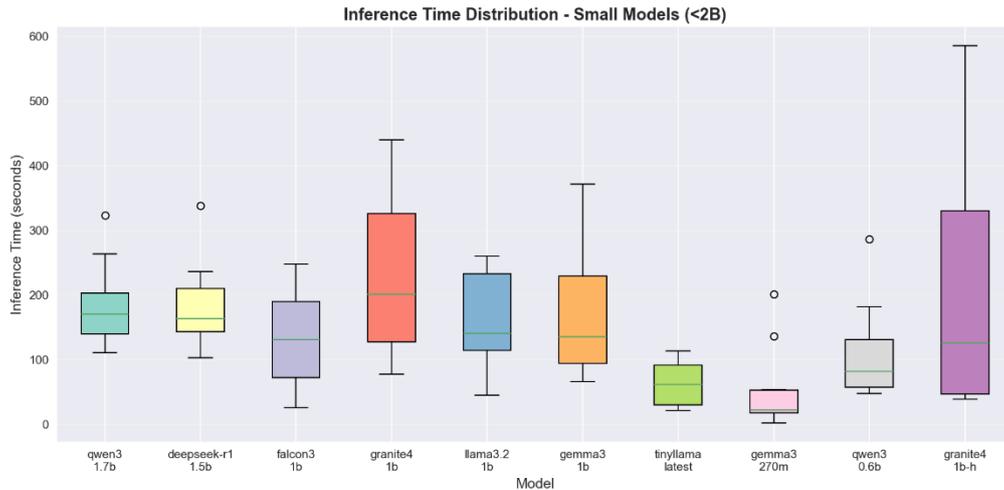


Figure 4 Inference time (models less than 2 billion parameters)

IOPS: Similar to TTFT analysis, IOPS metrics differentiate between cold start values (first question) and warm state values (average across questions 2 to 10). For small models almost every model averages 0.1 operations per second except for llama3.2 and granite4 1b-h which achieve 2.41 and 2.22 average operations per second for the warm IOPS.

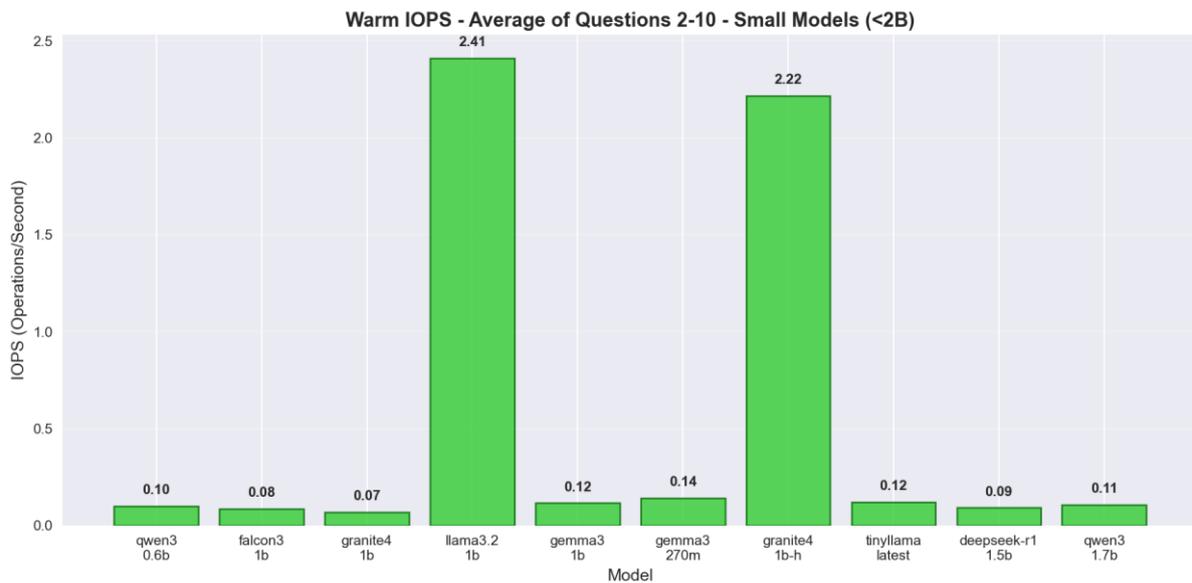


Figure 5 Average IOPS (models with fewer than 2 billion parameters)

Teaching effectiveness: Concerning the effectiveness of teaching, Gemma's 1 billion parameters seem to be above the others. Falcon3, granite4, llama 3.2 and Qwen 1.7 billion range between 7.4 and 8 while Gemma 270m, tinyllama and deepseek-R1 score approximately 6.

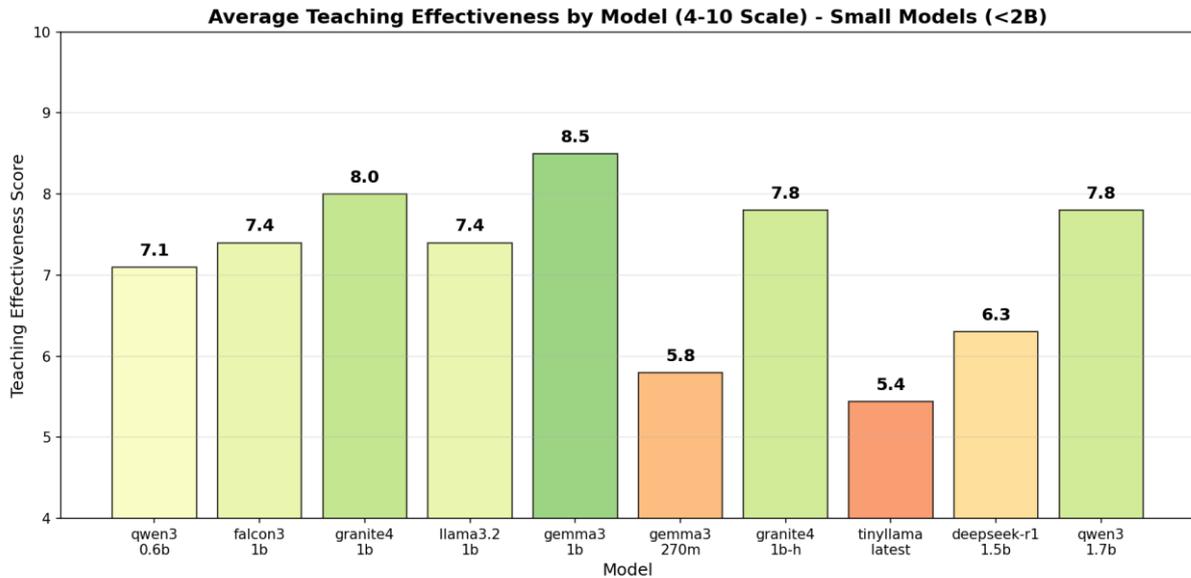


Figure 6: Teaching effectiveness (models with fewer than 2 billion parameters)

Token per Joule: The most energy efficient model is by far Gemma 270 million, due to its compact size and low computational requirements. Tinyllama, Qwen 0.6, deepseek-r1 and gemma3 1b all exceed 1 token per joule. The Granite4 family shows the highest energy consumption.

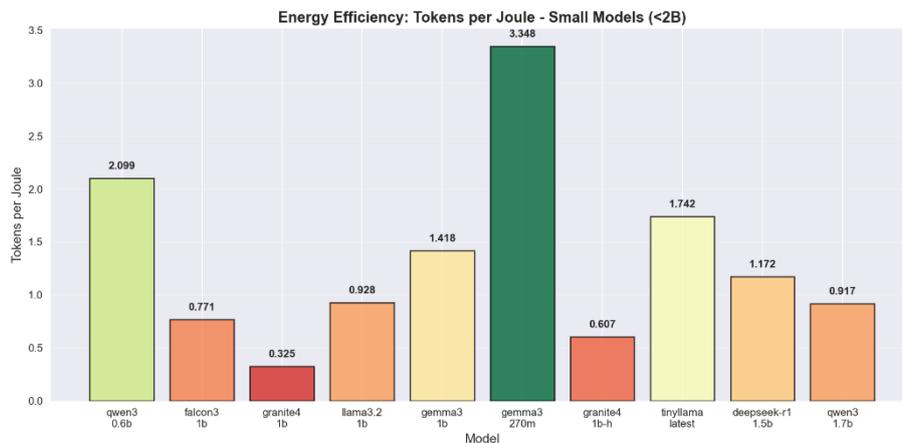


Figure 7 Token Per Joule (model less than 2 billion parameters)

MMLU: Most small models failed to reach the random baseline performance. This is mainly due to the difficulty for small models to understand and follow complex prompts. Qwen, with his ability to handle reasoning tasks, is really good at that benchmark, reaching 0.411 on the 0.6B variant. High performing models in this category include reasoning models (deepseek-r1 and Qwen), but surprisingly granite4 achieves comparable results despite not having a reasoning architecture, primarily due to its big size. Granite4 1B, gemma3:1B are the two models that range around the randomness baseline which is still better than some smaller models such as the llama3.2 and tinylama.

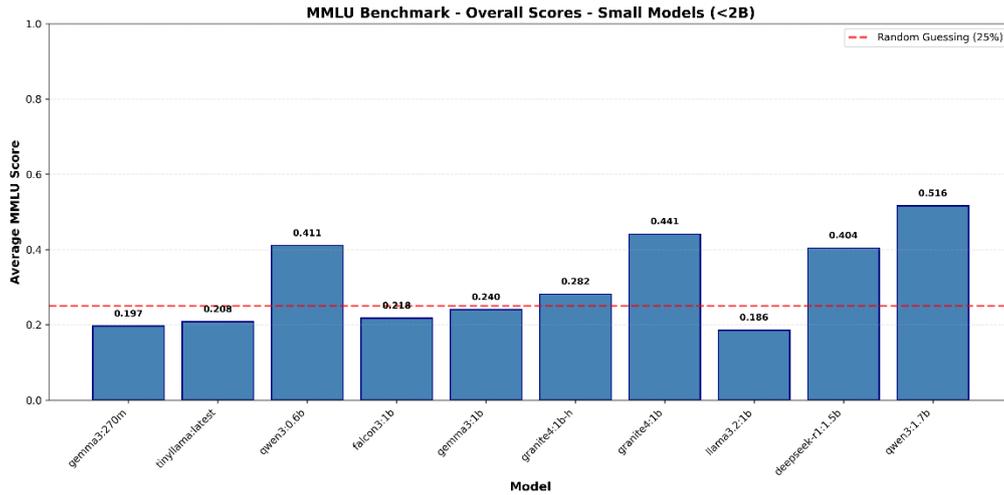


Figure 8 MMLU overall scores (models with fewer than 2 billion parameters)

5.1.2 Big models results (model with more than 2 billion parameters)

Token per second: Most large models range from 1.04 to 1.53 TPS. The only two models faster are Exaone-deep being one of the lightest models of this category and granite4:tiny-h which excels in TPS thanks to its MoE which uses only 1 billion active parameters despite being 7 billion total parameters.

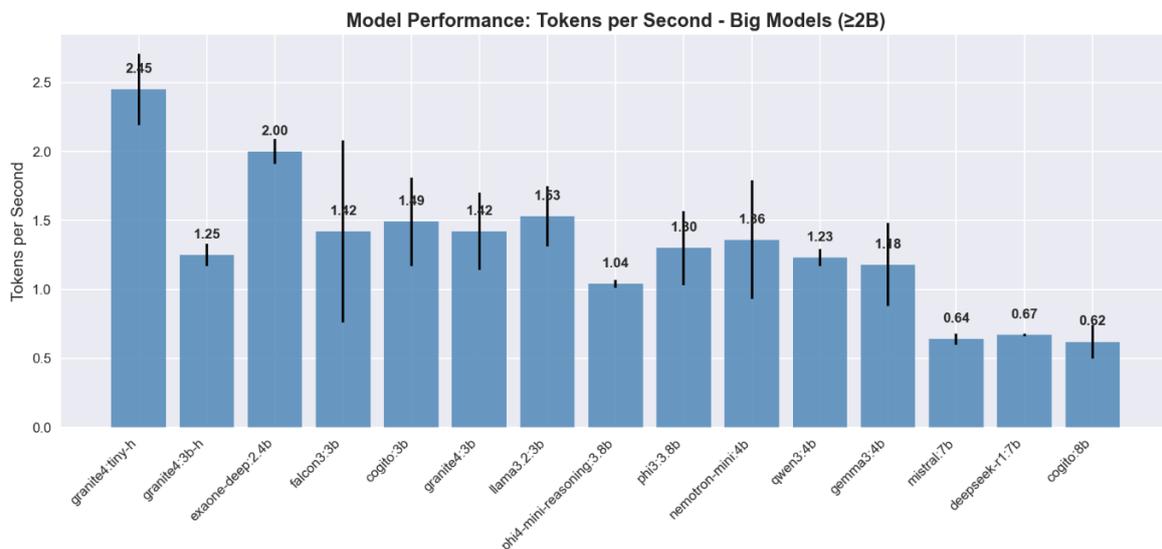


Figure 9 Token Per Second (models more than 2 billion parameters)

TTFT: Cogito shows slow initialization but achieves a TTFT similar to Mistral across the average of the 9 other questions. Both Granite4 models have the lowest initial but the 3b-h variant ends up having the biggest average TTFT which is the same behavior as the 1b-h variant. Most of the 3 billion and 4 billion parameter models average approximately 4 seconds on the TTFT average. Granite4 tiny is at 6.79 seconds in TTFT average which is equivalent to Qwen3 4b and two times faster than the other 7-8 billion parameters models, demonstrating the efficiency of the MoE.

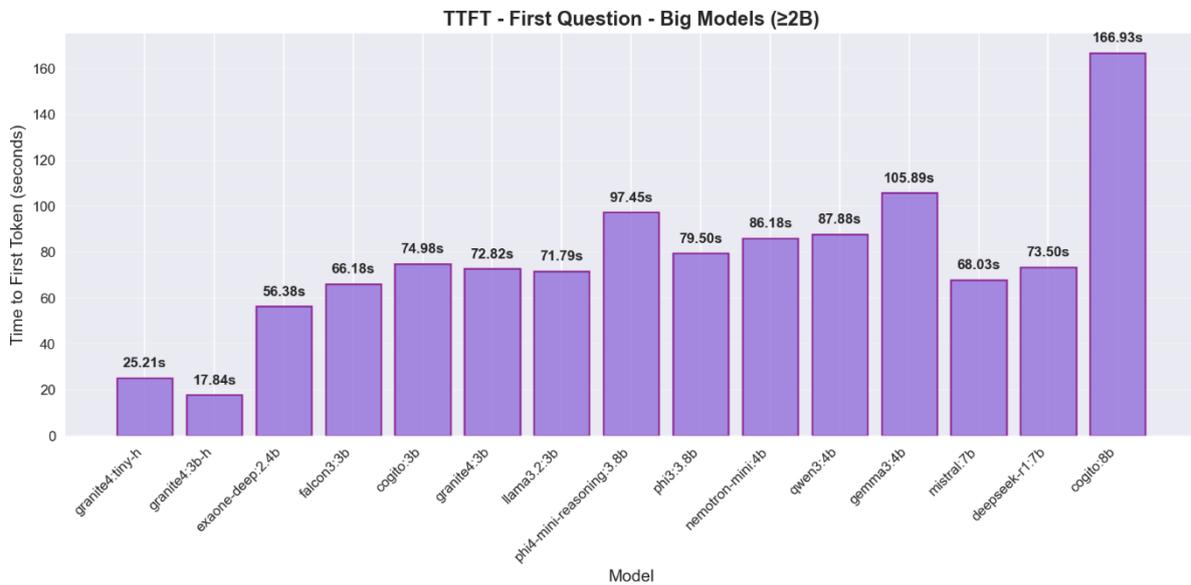


Figure 10 First question TTFT (Model with more than 2 billion parameters)

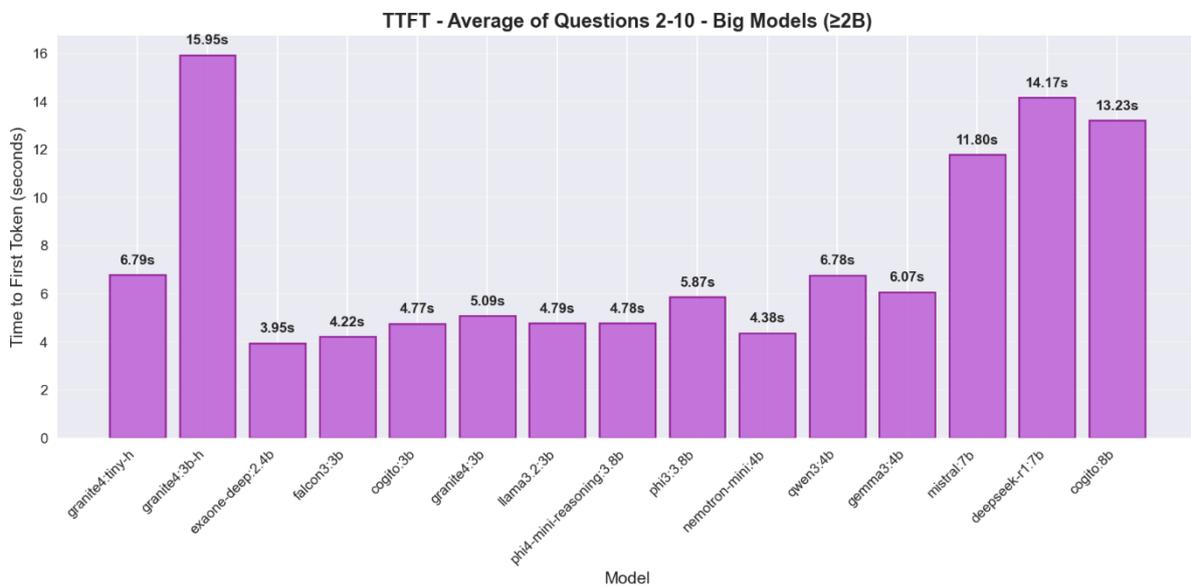


Figure 11 Average TTFT of the last 9 questions (Model with more than 2 billion parameters)

Response length: In this graph, Deepseek-R1 and Qwen are relatively low because their internal thinking characters are excluded, unlike other reasoning models such as Exaone-deep or Phy4 mini reasoning. The Granite 4 family, as well as Mistral, generate the most tokens.

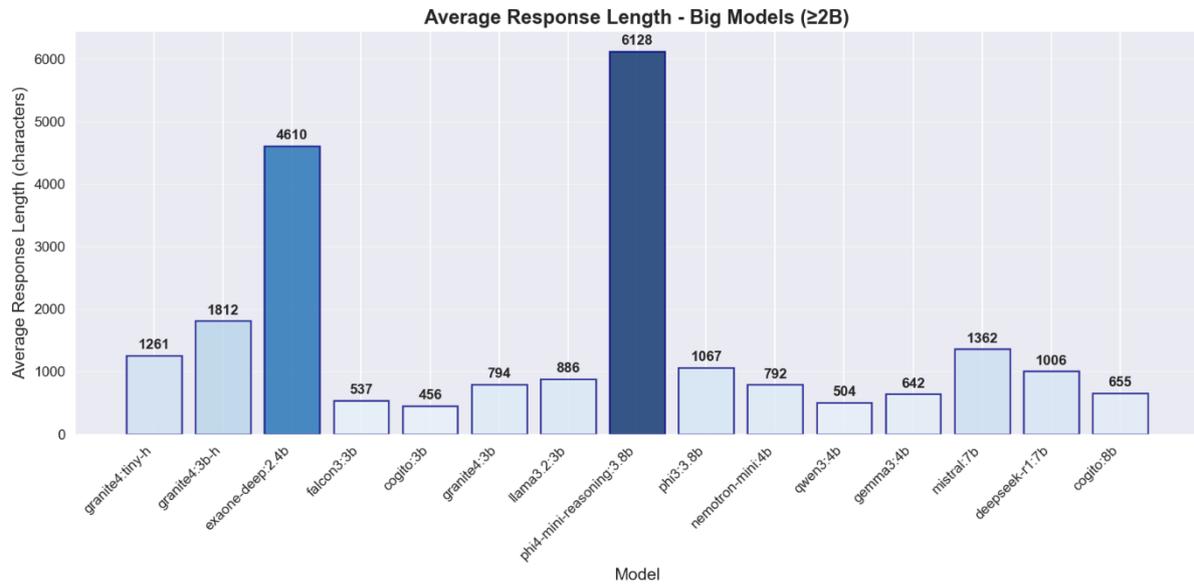


Figure 12 Average Response Length (models with more than 2 billion parameters)

Inference time: Considering total inference time, granite4 tiny, phy3, llama 3.2, granite4 3b, gemma3 and falcon 3 are reaching the lowest inference time. The 7-8B models are all slower except for the granite4 tiny. Surprisingly, granite4 3B-h shows slow performance, even underperforming compared to its non-hybrid counterpart.

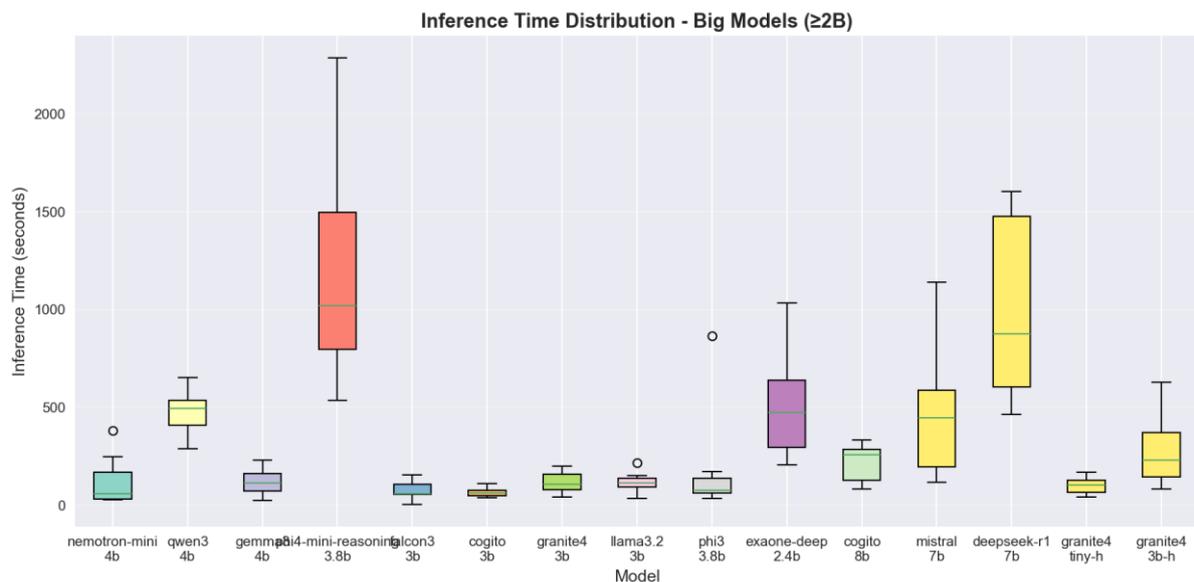


Figure 13 Inference time (model with more than 2 billion parameters)

IOPS: Most large models range from 0.05 and 0.21 average operations per second. Phi3 and Nemotron Mini average approximately 1 operation per second while Falcon3 is at 1.3 and Cogito has the biggest IOPS with 1.57 at warm state.

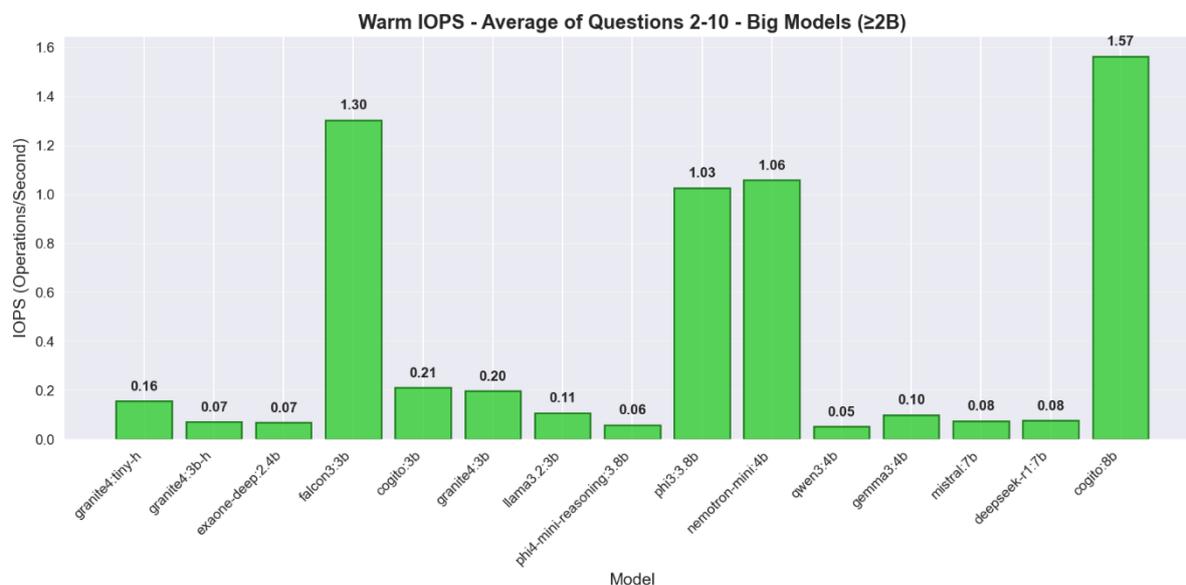


Figure 14 Average IOPS (model with more than 2 billion parameters)

Teaching effectiveness: Deepseek-r1 demonstrates the highest performance in this category. Mistral, Cogito 8B, Granite4 3b-h, Granite 4 Tiny, Exaone-deep, llama3.2, qwen3, and gemma3 are all above 7.5, which is still a good score. Nemotron-mini achieves the lowest score with 6.2/10 despite its reasoning architecture.

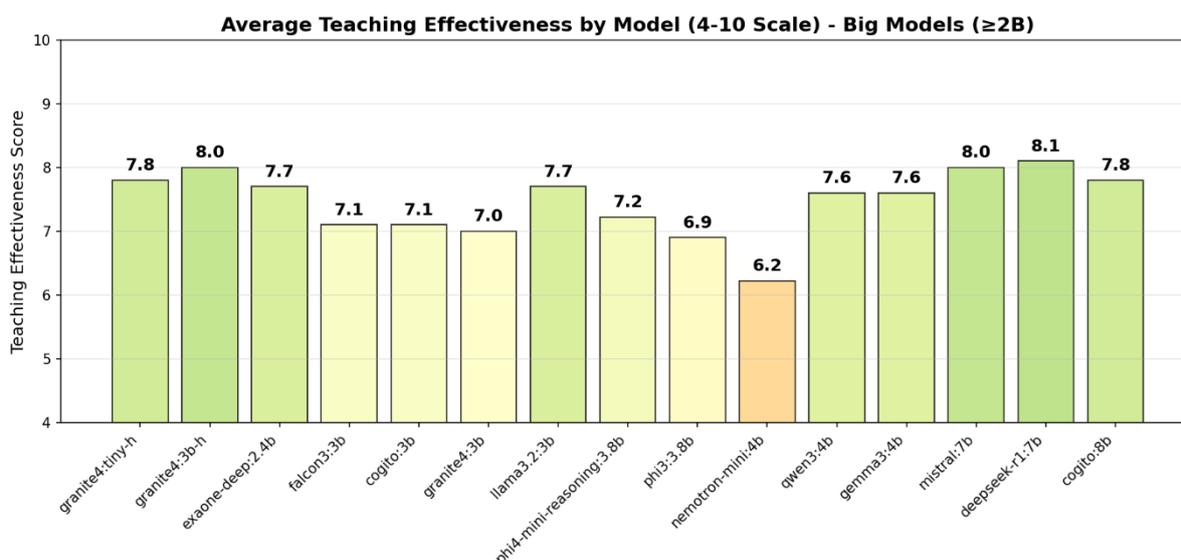


Figure 15: Teaching effectiveness (model with more than 2 billion parameters)

Token per Joule: The most energy efficient model is by far granite4 tiny-h due to its low computational requirements. Other large models are a lot more energy-consuming. Remaining

models are all around 0.5 TPJ except for Exaone deep which reaches 0.73 TPJ and Phi4 mini reasoning which reaches 0.377 TPJ.

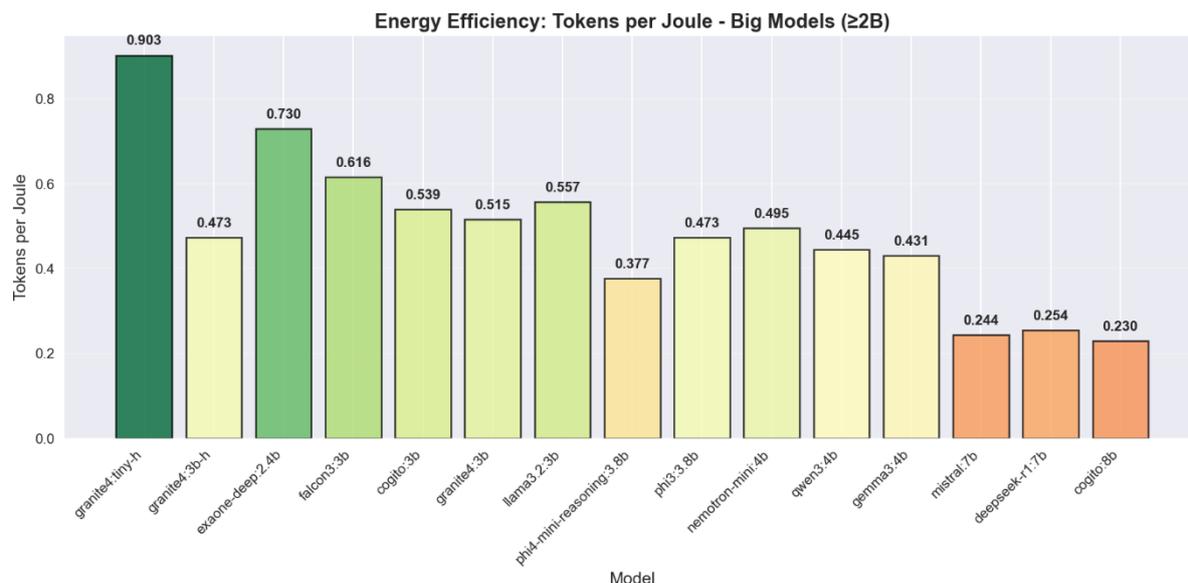


Figure 16 Token Efficiency (model with more than 2 billion parameters)

MMLU: Most of the 3B models struggle to exceed the randomness baseline, excluding the Granite 4 family. Other models are between 0.4 and 0.5 except Phi4 mini reasoning. Granite4 tiny is the one achieving the best score with 49.2% overall.

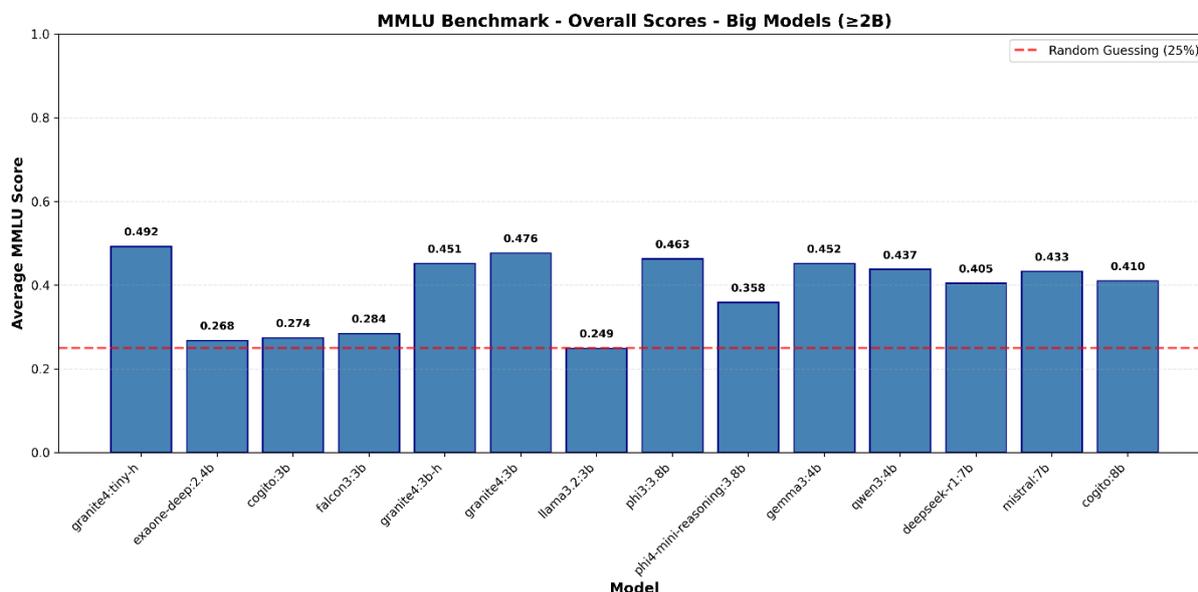


Figure 17 MMLU overall score (models with more than 2 billion parameters)

5.1.3 final model choice

Small model:

Considering the choice of small models, the main point of this model is its ability to follow prompts, its speed and accuracy. Two models really come out, offering different approaches that bring them advantages and drawbacks. For his performance in the MMLU and his ability to follow the prompt and show his knowledge: Qwen3 0.6B.

The second potential candidate would be gemma3:1B, which reaches an acceptable speed and token per second and reach also reaches the best teaching effectiveness scores. It still has a big weakness considering the MMLU score which is equivalent to the random guessing which makes it potentially harder to prompt. This model is supposed to be more direct in its answer than Qwen. It also offers QAT variants that maintain high accuracy for a small size.

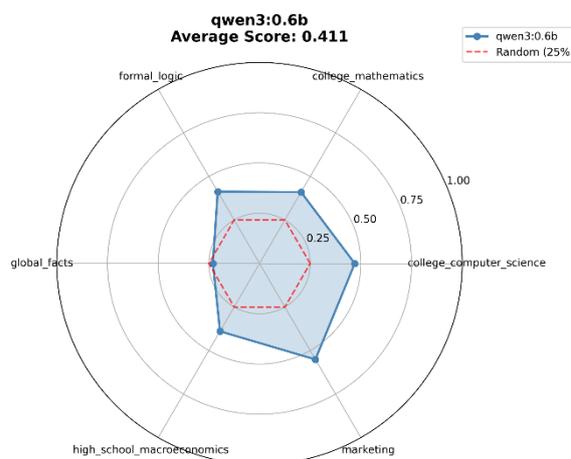


Figure 18 MMLU scores of Qwen3 0.6B

What is Gemma3?

Gemma3 is the last version of the family version of Gemma built by Google DeepMind [49] released on March 12, 2025. These models are designed to be used for applications on single GPUs, smartphones and even smaller ones. Gemma3 now includes some small versions such as 270 million or 1 billion parameters.

All the models of those categories came with their variations that are quantized using QAT. This method allowed better accuracy retention when quantized compared to classic PTQ. This QAT variant includes instruction tuning on Ollama which is a really good advantage for our tier 2 catching model needs.

Big Models:

One model in this category really stands out in speed as well as in general knowledge and MMLU score. This model is Granite 4 Tiny Hybrid. Its MoE architecture enables high inference speed while outperforming mistral7b, deepseek-r1: 7b or cogito8b. He is also way above the other models in TPJ even some models with 1 billion parameters. This model does not excel in teaching effectiveness, but it still achieves a respectable 7.7/10.

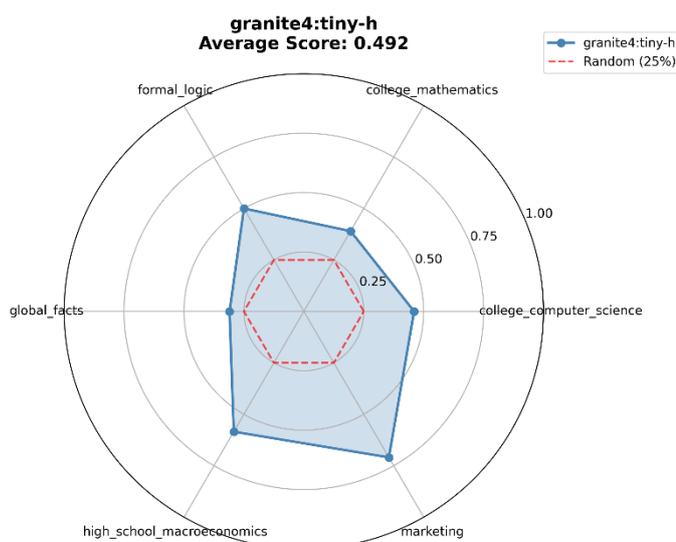


Figure 19 MMLU scores of Granite 4 Tiny Hybrid

What is Granite4.0?

Granite 4.0 represents the latest generation of models developed by IBM [36]. They are open-source models built to be used for edge cases and autonomous applications. It was released at the beginning of October 2025.

They are built using a hybrid architecture featuring mamba blocks and transformer blocks at a ratio of 9:1 (9 mamba for 1 transformer). This hybrid architecture allows for reducing computational requirements while keeping efficient accuracy, making it a great family of models for low computational use.

This hybrid architecture also allows the models to have theoretically an infinite context window which is normally one of the bottlenecks of small transformer models. This is due to the mamba architecture and the use of NoPE (No Positional Encoding).

Another big advantage that the Small and Tiny models have is that they use MoE (Mixture of Experts). This allows them to be even more computationally efficient (Tiny has 7B total parameters by only having 1B active parameters). All these different architectures and adjustments of the models are turned for edge case uses which is why it performs so well in the RPi 4.

5.2 Performance comparison of Pi4, Pi5, RTX4060

Gemma3:1b: Inference speed and TPS increase exponentially as the hardware has more computational power. Total inference time is 2.4 times lower for the Raspberry Pi 5 compared to the Rpi 4 model B. This difference is up to 34.78 times for the computer equipped with the RTX4060.

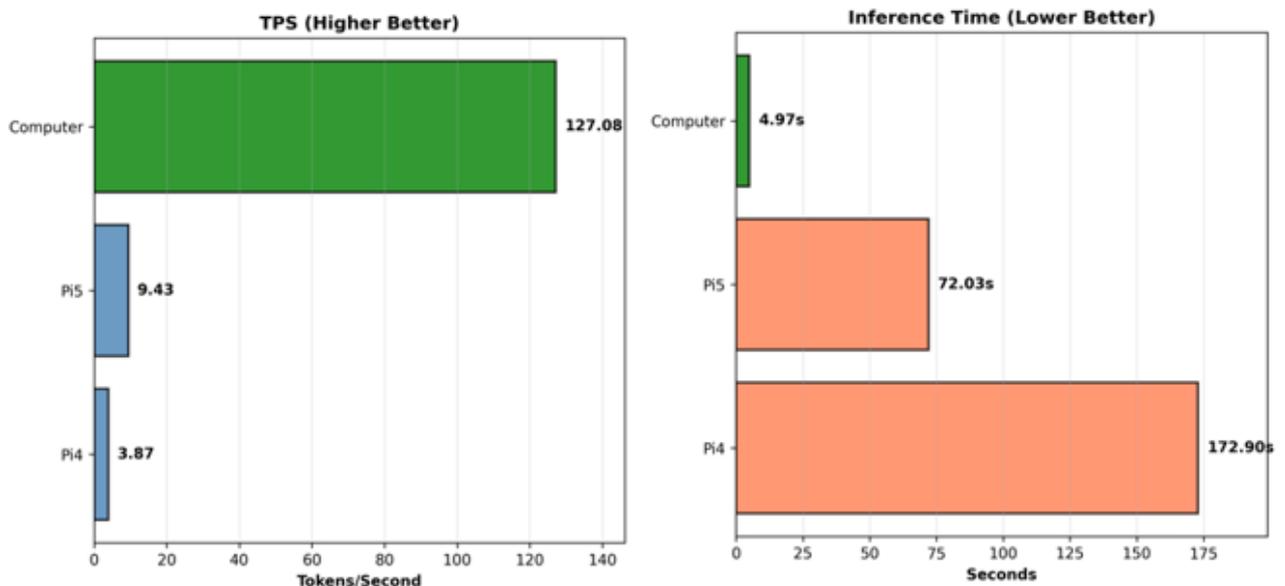


Figure 20 Comparison of the devices on TPS and Inference Time for Gemma3 :1b

Granite 4 Tiny Hybrid: Inference speed and TPS increase exponentially as the hardware has more computational power. Total inference time is 2.8 times lower for the RPi 5 compared to the RPi 4 model B. This difference is up to 25.5 times for the computer equipped with the RTX4060.

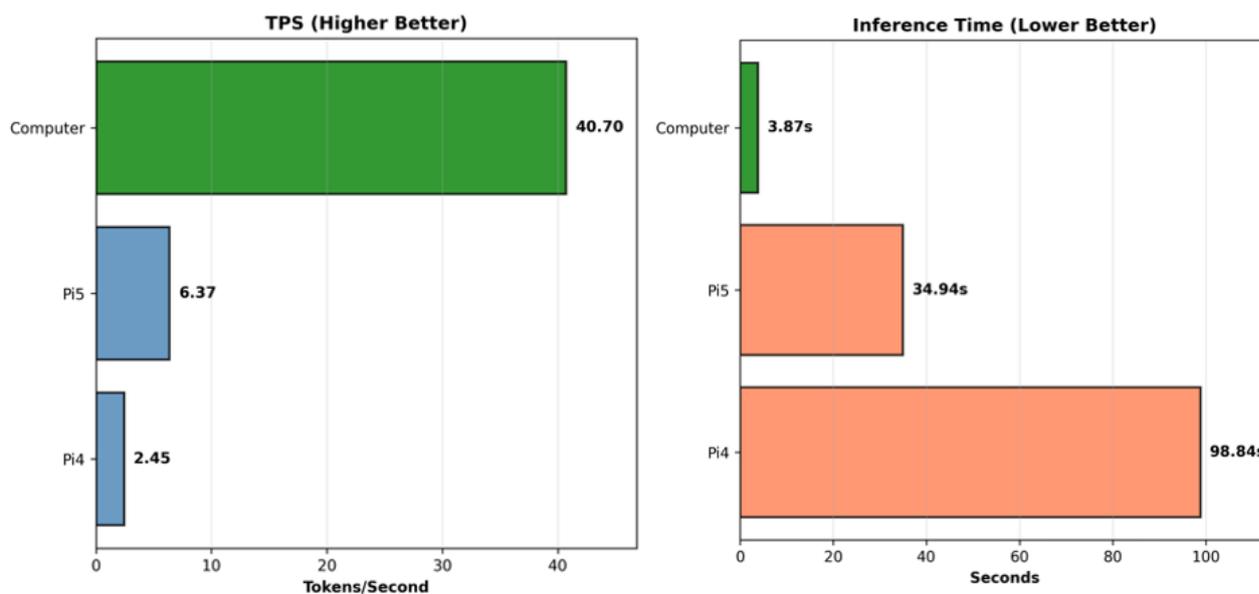


Figure 21 Comparison of the devices on TPS and Inference Time for Granite4:tiny-h

5.3 User Model Interaction Experiment Results

The Experiment was conducted mainly in the Delta building of Tartu University. Ten participants completed the study.

Four out of ten participants were in the computer engineering-related field and three out of ten were in the robotics-related field. The rest of the participants were divided into chemistry, biotechnology and music.

Four participants were bachelor students, four were master students, and two were PhD students.

Six out of ten participants knew a bit about the Language model before the experiment, only two people didn't know anything about it. Two people also knew a lot about the Language model before the experiment.

5.3.1 Analysis of the experimental results on a text basis

On teaching question: Nine out of ten participants found the model realistic and engaging and at least agreed that the model explained the scope and purpose well, understood well the participants, that the responses were useful, appropriate and informative.

On perception: The participants are mixed about the fact that the model seems too robotic. They mostly disagree that the model seemed very unfriendly.

On the use and the model viability: They all disagree that the model failed to recognize output and they mostly disagree that it seemed unable to handle errors. They mostly find the model easy to use, not very complex and it is not easily confusing when using.

5.3.2 Comparison of text and speech

The biggest changes concern the perception of the model. During the speech phase of the experiment, the model is found to be more realistic and engaging. Also, it feels less robotics using the speech interaction.

In the perception of the response and error handling, the answers seem to be more in favor of the model ability to give relevant, useful answers as well as dealing better with errors.

5.3.3 Verbal feedback from participants

In general, the feedback about the model ability to teach is positive, “the answers are clear and accessible” “it really helps understanding the topics”. Therefore, they find the way of interacting with the model a bit too robotics; “He could say hi and bring some more human talking”.

5.4 Results and proposed architecture

5.4.1 architecture

Based on the benchmark done and the prior research, the architecture was built with modules and functions that fulfill the requirements of the RSC.

The final architecture is available in the GitHub repository[50]. All the code of the architecture is built on Python and is composed of multiple components.

Data folder: This folder contains the raw documents that need to be added to the memory of the LM. It also contains a place for storing those files in a vector database, along with the different questions and answers of the cache.

Requirement: You will need ollama downloaded on the RPi and pull the three models needed (granite4:tiny-h, gemma3:1b-it-qat, and any embedding model, for example nomic-embed-text: latest). You will also need the ollama and ChromaDB Python libraries. Other small Python libraries are necessary, and all details are available in the README of the GitHub repository.

Config.py: This file contains all the necessary variables that can be changed to easily modify different components of the architecture.

Modules: The module folder contains all the different functions and parts of the architecture that are going to be used in the main code.

LocalLM.py: This is the main code that will bring all the pieces together and run the architecture.

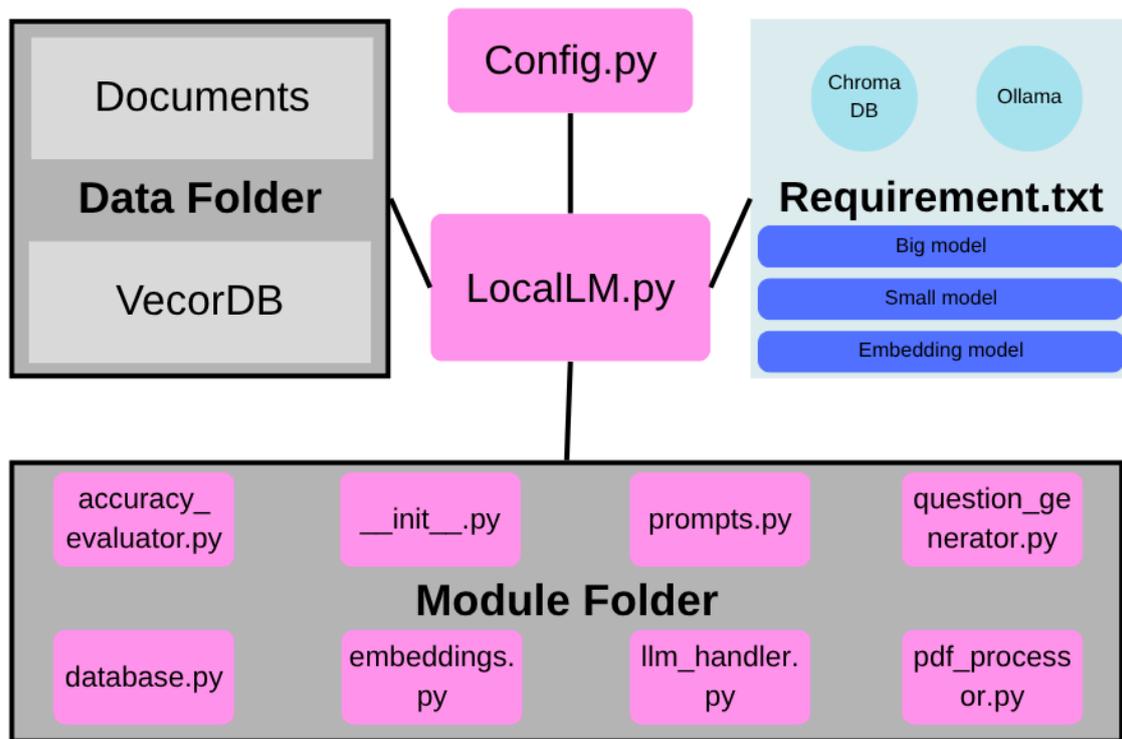


Figure 22: architecture of the actual LocalLM of the RSC

5.4.2 Initialization

At the beginning of the implementation of the different documents into the three-tier, it will take a few hours to process the question and answer generation for the cache. Firstly, all the documents will be embedded and chunked using *pdf_processor.py*, *embeddings.py* and *database.py* to be stored in the vector database.

Once this document is properly stored, the question generation will start. The Big model will be initialized and start generating the questions and answers using *question_generator.py* and the prompt available in *prompt.py*. The number of questions generated can be modified in the *config.py* file. All those questions are going to be stored as cache in the VectorDB.

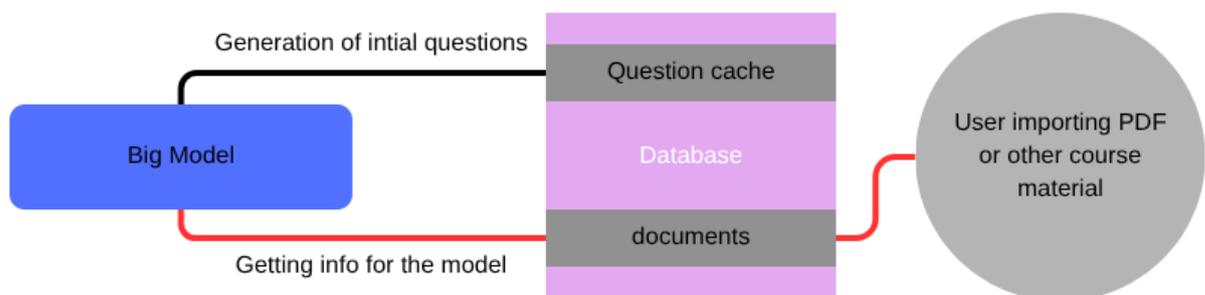


Figure 23 flowchart of the Initialisation of the RSC questions

5.4.3 Real-time use

In use, the proposed architecture compares the user's query or question to the local caching database against stored queries or questions. Then, it will compile and compare semantic similarities against a maximum and minimum threshold defined in the *config.py* file. Afterwards, three scenarios can take place:

Tier 1, almost instant: The similarity is higher than the maximum threshold. The answer associated with a similar question will be delivered directly and really quickly to the user using the cache.

Tier 2, fast: The similarity sits between thresholds. Then, Gemma3:1b-it-qat engages and answers within the user query context. More specifically, it answers the question and answers that are most similar to chunks in the user-submitted document (PDF). The model formulates its answer based on specific prompts from *prompt.py*. Finally, the new question and answer are then stored in the cache for future use.

Tier 3, slow: The similarity is lower than the minimum threshold. Granite4 Tiny engages to answer the user using the most similar chunk and the user query. As in Tier 2, the new question and answer are then stored in the cache for future use.

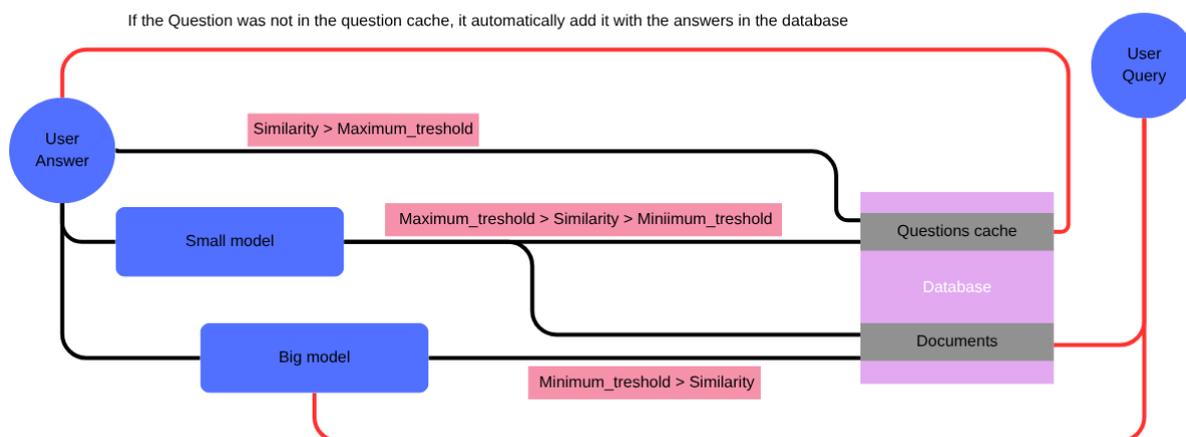


Figure 24: Flowchart of the architecture during the use

6 Analysis and Discussion

Most of the research objectives were achieved. The literature review facilitated a comprehensive analysis of existing approaches for deploying LMs on edge devices. Examining the state of the art of open-source models, their architectural differences, and specificity allows the possibility to choose the widest variety of models for comparison. Comprehensive benchmarking on the RPi 4 model B successfully evaluated model performance across key metrics, identifying optimal candidates for RSC deployment. A three tier architecture was developed based on an existing paper, adapting those methodologies to the specific purpose of the RSC and its constraints. This architecture is able to provide a fast answer to the user and fill the gap between speed and accuracy. This architecture can be improved using the students' data, questions and feedback. All these feelings' data helped to better understand the expectations of the user when conversing with the model and later with the RSC. However, some objectives were not fully achieved, the architecture, even if faster in theory, lacks testing, variable adaptation, and improvement (tier threshold or prompt).

Concerning the different requirements, they are globally met. The model fits the global requirement of the RSC about: 'the RSC should provide adaptable learning support'. Then architecture brings an adaptation to any different kind of courses needed to be learnt by the student. However, the architecture is, for now, not able to store personal learning style, learning expectations, and other information that can make the RSC adapt more to each user independently. Also, using an open-source model locally gives full control over the data from the user, future implementation requires a clear and safe storage system as well as transparent data practices. about it. Selected models already include guardrails to ensure a safe learning environment. However, more tests should be conducted to ensure that no unexpected behavior can harm or provide a negative environment for the student.

For the User evaluation, some valuable results were extracted from it. Results seem to indicate that participants tend to find the model efficient as a teaching assistant and appreciate using it. However, to make it really meaningful, a quantitative study needs to be done. Using a better interface and over a bigger span of time to test the perception of the model's learning ability over time.

6.1 Challenges

6.1.1 Benchmark challenge

Benchmarking presented several challenges. Thermal throttling, identified late in the process, necessitated repeated testing on the RPi. This benchmark is really time-consuming as the RPi 4 spent a lot of time running some models, such as deepseek-R1. After the first iteration, some other models were added, needing to bring another round of benchmarks. A significant problem involved methodological error identified too late in the benchmarking process. The TTFT and inference time measurement for Deepseek-R1 and Qwen3 contain inaccuracies. In fact, the benchmark code was not taking the time at the beginning when the user gives the question, but when it starts generating the first token. As Deepseek is a reasoning model that

'thought' internally, the start of the generation of the first token starts at the end of its thinking, making this processing time not visible in the data. Qwen3 is not properly a reasoning model, but it is built to solve complex reasoning tasks, so it still processes a 'thought' before answering, which is also not recorded in the code.

However, the benchmark still brings a good addition to the RSC framework and will be able to help to easily benchmark a new model using the exact same process. This will facilitate comparison for new models on the RPi 4 or RPi 5.

6.1.2 Sample Size for the User Experiment

The experiment involving users and the model did not involve enough participants. Only ten participants joined the experiment and completed the study. Statistical significance requires larger-scale quantitative evaluation.

Finding participants for the experiments was a challenge. As the experiment needed some time and a specific place, finding participants and performing the experiment was really time-consuming. Having a significant number of participants was not possible within the time constraint of one semester.

6.2 Future Work

In the future, following the integration of architecture on the RSC, some work can be done to further improve the capability of the model and the RSC.

Integration into the RSC: Deploying the proposed architecture into the RSC would require integrating the STT and TTS into the actual architecture.

Tool calling: Implementing tool calling for hand movements and expressions of the RSC on the screen can also bring a more human and natural behavior.

User personalization: Remembering learning style, personal preferences, and inserting them in the prompt of the model can bring a next level of adaptability for the RSC

Data management: With this user adaptation, personal data needs to be stored somewhere. The security of that data should be the priority, and the data retention should be built around the respect of the privacy and the security of the data.

Course feedback: If the RSC with the three-tier architecture is implemented in a few robots and given to the students for a semester, the questions stored in the cache could all be gathered to check the most common one and see where students struggle to bring proper feedback on students' needs. This could help teachers to adapt their courses to students' needs.

Multiple courses at once: Once the RSC and architecture are working, the possibility of extending the database across multiple courses that can be switched easily would be a real improvement. It will allow the students to be able to have one RSC that can help with any of their courses. This would require a way deeper and complex architecture, but it would bring a full experience to the student.

New models testing: Before the end of this thesis, some new LMs came out, and some of them seem quite promising for the RSC, such as the Ministral family.

7 Conclusion

This thesis investigated the feasibility of deploying educational language models on a low-resource social robot under strict computational and energy constraints. By benchmarking twenty-five recent open-source language models across multiple hardware platforms, including the Raspberry Pi 4 and 5, the work systematically analyzed trade-offs between inference speed, resource consumption, and pedagogical quality. The results demonstrate that while small models offer superior responsiveness and energy efficiency, they often struggle with instruction-following and general knowledge, whereas larger models provide higher instructional quality at the cost of increased latency and resource usage.

To address these trade-offs, this thesis proposed and implemented a three-tier hybrid architecture combining caching, lightweight rephrasing models, and a higher-capacity reasoning model. This architecture enables the system to deliver fast responses for common or similar queries while preserving access to more accurate and comprehensive explanations when required. The benchmarking results informed model selection at each tier—identifying Granite4 tiny-h for complex queries and Gemma3:1b-it-qat for rephrasing—and the final pipeline illustrates how architectural design choices can mitigate hardware limitations without relying on external cloud services, thereby improving privacy and offline usability for educational robots.

An exploratory user interaction study further examined student perceptions of both text-based and speech-based interactions with the system. While limited in scale, the findings suggest that locally deployed language models can provide useful and engaging learning support when appropriately constrained and framed. Although further validation is needed to quantify the architecture's performance gains and address measurement limitations identified during benchmarking, this work contributes practical insights into the deployment of local language models for educational robotics and establishes a reproducible foundation for future research on adaptive, privacy-preserving learning systems operating on edge devices.

Acknowledgements

I would like to give a special thanks to my supervisors, Farnaz Baksh and Matevž Borjan Zorec, for their help, support, guidance, and valuable supervision. Thank you for making me feel fully part of the RSC project and for the make session.

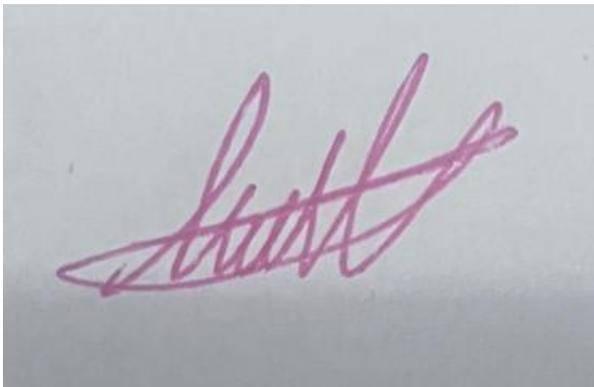
Thank you to Gio and Raiko for proofreading the Estonian part of this thesis.

Big Thanks to my family who supported me throughout my whole master's and through this thesis.

Special thanks to Diima for bringing me into machine learning and making me consider mixing machine learning and robotics for my thesis.

Dorian Lamouille

23.12.2025

A photograph of a handwritten signature in red ink on a light-colored surface. The signature is stylized and cursive, appearing to read 'Dorian Lamouille'.

Bibliography

- [1] F. Baksh, M. B. Zorec, et K. Kruusamäe, « Open-Source Robotic Study Companion with Multimodal Human–Robot Interaction to Improve the Learning Experience of University Students », *Appl. Sci.*, vol. 14, n° 13, p. 5644, janv. 2024, doi: 10.3390/app14135644.
- [2] F. Baksh, I. Jackson, I. Jackson, et M. B. Zorec, « University Students' Acceptance of a Robot Study Companion », in *Robotics in Education*, R. Balogh, D. Obdržálek, et N. Fachantidis, Éd., Cham: Springer Nature Switzerland, 2025, p. 38-50. doi: 10.1007/978-3-031-98762-5_5.
- [3] « Adapting a Teachable Robot's Dialog Responses Using Reinforcement Learning: Cross-Cultural User Study Exploring Effect on Engagement ». Consulté le: 21 décembre 2025. [En ligne]. Disponible sur: <https://dl.acm.org/doi/epdf/10.1145/3743692>
- [4] J. Heinerman, D. Drupsteen, et A. Eiben, *Three-fold Adaptivity in Groups of Robots: The Effect of Social Learning*. 2015. doi: 10.1145/2739480.2754743.
- [5] M. Donnermann, P. Schaper, et B. Lugin, « Investigating Adaptive Robot Tutoring in a Long-Term Interaction in Higher Education », in *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, août 2022, p. 171-178. doi: 10.1109/RO-MAN53752.2022.9900865.
- [6] H. Kibriya, W. Z. Khan, A. Siddiq, et M. K. Khan, « Privacy issues in Large Language Models: A survey », *Comput. Electr. Eng.*, vol. 120, p. 109698, déc. 2024, doi: 10.1016/j.compeleceng.2024.109698.
- [7] B. Huang *et al.*, « FirewaLLM: A Portable Data Protection and Recovery Framework for LLM Services », in *Data Mining and Big Data*, Y. Tan et Y. Shi, Éd., Singapore: Springer Nature, 2024, p. 16-30. doi: 10.1007/978-981-97-0844-4_2.
- [8] S. Li, W. Fu, Y. Zhao, X. Guo, et Y. Jin, « Intelligence In The Fence: Construct A Privacy and Reliable Hardware Design Assistant LLM », in *Proceedings of the Great Lakes Symposium on VLSI 2025*, in GLSVLSI '25. New York, NY, USA: Association for Computing Machinery, juin 2025, p. 659-666. doi: 10.1145/3716368.3735172.
- [9] N. Lamba, S. Tiwari, et M. Gaur, « Investigating Symbolic Triggers of Hallucination in Gemma Models Across HaluEval and TruthfulQA », 9 septembre 2025, *arXiv*: arXiv:2509.09715. doi: 10.48550/arXiv.2509.09715.
- [10] T. de Silva, S. Uthsari, S. Perera, et A. Gunawardana, « Online teaching and Learning post-Covid: An Analysis of LMS Usage and Student Outcomes Following the Pandemic », *Eur. Conf. E-Learn.*, vol. 23, n° 1, p. 365-373, oct. 2024, doi: 10.34190/ecel.23.1.2746.

- [11] M. Donnermann, P. Schaper, et B. Lugin, « Application of Social Robots in Higher Education: A Long-Term Study », *Int. J. Soc. Robot.*, p. 1-16, sept. 2025, doi: 10.1007/s12369-025-01286-7.
- [12] « Text-to-Speech Models – Hugging Face ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: <https://huggingface.co/models>
- [13] K.-U. Carstensen, « Quantification: The View From Natural Language Generation », *Front. Artif. Intell.*, vol. 4, 2021, doi: 10.3389/frai.2021.627177.
- [14] M. Ardakani, J. Malekar, et R. Zand, « LLMPi: Optimizing LLMs for High-Throughput on Raspberry Pi », 2 avril 2025, *arXiv*: arXiv:2504.02118. doi: 10.48550/arXiv.2504.02118.
- [15] H. Kane, M. Y. Kocyigit, A. Abdalla, P. Ajanoh, et M. Coulibali, « NUBIA: NeUral Based Interchangeability Assessor for Text Generation », 1 mai 2020, *arXiv*: arXiv:2004.14667. doi: 10.48550/arXiv.2004.14667.
- [16] B. Liu, Q. Wu, Y. Zhang, et Q. Cao, « Exploiting the Relationship between Pruning Ratio and Compression Effect for Neural Network Model Based on TensorFlow », *Secur. Commun. Netw.*, vol. 2020, p. 1-8, avr. 2020, doi: 10.1155/2020/5218612.
- [17] K. Kamma, S. Inoue, et T. Wada, « Pruning Ratio Optimization with Layer-Wise Pruning Method for Accelerating Convolutional Neural Networks », *IEICE Trans. Inf. Syst.*, vol. E105.D, n° 1, p. 161-169, janv. 2022, doi: 10.1587/transinf.2021EDP7096.
- [18] F. Wang *et al.*, « A Comprehensive Survey of Small Language Models in the Era of Large Language Models: Techniques, Enhancements, Applications, Collaboration with LLMs, and Trustworthiness », *ACM Trans Intell Syst Technol*, sept. 2025, doi: 10.1145/3768165.
- [19] D. Jurgens, S. Kumar, R. Hoover, D. McFarland, et D. Jurafsky, « Measuring the Evolution of a Scientific Field through Citation Frames », *Trans. Assoc. Comput. Linguist.*, vol. 6, p. 391-406, juill. 2018, doi: 10.1162/tacl_a_00028.
- [20] Y. Yao, S. Huang, W. Wang, L. Dong, et F. Wei, « Adapt-and-Distill: Developing Small, Fast and Effective Pretrained Language Models for Domains », 29 juin 2021, *arXiv*: arXiv:2106.13474. doi: 10.48550/arXiv.2106.13474.
- [21] S. Masoudnia et R. Ebrahimpour, « Mixture of experts: A literature survey », *Artif. Intell. Rev.*, vol. 42, n° 2, p. 275-293, 2014, doi: 10.1007/s10462-012-9338-y.
- [22] A. Q. Jiang *et al.*, « Mixtral of Experts », 8 janvier 2024, *arXiv*: arXiv:2401.04088. doi: 10.48550/arXiv.2401.04088.
- [23] Y. Assogba et D. Ren, « Evaluating Long Range Dependency Handling in Code Generation LLMs », 24 juin 2025, *arXiv*: arXiv:2407.21049. doi: 10.48550/arXiv.2407.21049.
- [24] C. Packer *et al.*, « MemGPT: Towards LLMs as Operating Systems », 12 février 2024, *arXiv*: arXiv:2310.08560. doi: 10.48550/arXiv.2310.08560.

- [25] « Free Text to Speech & AI Voice Generator | ElevenLabs ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: https://start.elevenlabs.io/brand/v1?utm_source=google&utm_medium=cpc&utm_campaign=t1_brandsearch_brand_english&utm_id=22886448172&utm_term=elevenslabs&utm_content=brand_-_brand&gad_source=1&gad_campaignid=22886448172&gbraid=0AAAAAp9ksTGftlnXEKVHSu2Ao3JAPwWaY&gclid=CjwKCAjw04HIBhB8EiwA8jGNbYr0plxgbucmEs_A_aM86Pc93QI5p9T2JlBzD2TIA0rzcV_dxmtqVxoC0FYQAvD_BwE
- [26] « AI Voice Over & AI Voice Generator for Creators | Artlist ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: https://artlist.io/voice-over?utm_source=google&utm_medium=cpc&utm_campaign=21312213763&utm_content=162275265549&utm_term=ai%20voice%20generator&keyword=ai%20voice%20generator&ad=699976106808&matchtype=e&device=c&gad_source=1&gad_campaignid=21312213763&gbraid=0AAAAACuwFJOSxvsheTCn21LQekFo2nxVD&gclid=CjwKCAjw04HIBhB8EiwA8jGNbRkIBMzEmdNWxnf3X5Kh-JAYZ4oV0OSJ3VHtF1AlkdqLeql3Sp3axoCqMQQAvD_BwE
- [27] « n8n.io - AI workflow automation tool ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: <https://n8n.io/>
- [28] A. Vaswani *et al.*, « Attention Is All You Need », 2 août 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.
- [29] « Google Gemini », Gemini. Consulté le: 18 décembre 2025. [En ligne]. Disponible sur: <https://gemini.google.com>
- [30] « OpenAI ». Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: <https://openai.com/fr-FR/>
- [31] « Home \ Anthropic ». Consulté le: 18 décembre 2025. [En ligne]. Disponible sur: <https://www.anthropic.com/>
- [32] A. Gu et T. Dao, « Mamba: Linear-Time Sequence Modeling with Selective State Spaces », 31 mai 2024, *arXiv*: arXiv:2312.00752. doi: 10.48550/arXiv.2312.00752.
- [33] T. Dao et A. Gu, « Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality », 31 mai 2024, *arXiv*: arXiv:2405.21060. doi: 10.48550/arXiv.2405.21060.
- [34] P. Glorioso *et al.*, « Zamba: A Compact 7B SSM Hybrid Model », 26 mai 2024, *arXiv*: arXiv:2405.16712. doi: 10.48550/arXiv.2405.16712.
- [35] O. Lieber *et al.*, « Jamba: A Hybrid Transformer-Mamba Language Model », 3 juillet 2024, *arXiv*: arXiv:2403.19887. doi: 10.48550/arXiv.2403.19887.
- [36] « IBM Granite 4.0: Hyper-efficient, High Performance Hybrid Models for Enterprise ». Consulté le: 7 décembre 2025. [En ligne]. Disponible sur:

<https://www.ibm.com/new/announcements/ibm-granite-4-0-hyper-efficient-high-performance-hybrid-models>

[37] « Google ». Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: <https://www.google.com/>

[38] J. Li, C. Xu, L. Jia, F. Wang, C. Zhang, et J. Liu, « EACO-RAG: Towards Distributed Tiered LLM Deployment using Edge-Assisted and Collaborative RAG with Adaptive Knowledge Update », 14 février 2025, *arXiv*: arXiv:2410.20299. doi: 10.48550/arXiv.2410.20299.

[39] K. Sakai, Y. Uehara, et S. Kashihara, « Implementation and Evaluation of LLM-Based Conversational Systems on a Low-Cost Device », in *2024 IEEE Global Humanitarian Technology Conference (GHTC)*, oct. 2024, p. 392-399. doi: 10.1109/GHTC62424.2024.10771565.

[40] « Claude ». Consulté le: 4 novembre 2025. [En ligne]. Disponible sur: <https://claude.ai/chat/c672bec5-4b6e-475f-9250-0b6a1a2ae281>

[41] « Carte graphique GeForce RTX 5090 | NVIDIA ». Consulté le: 4 novembre 2025. [En ligne]. Disponible sur: <https://www.nvidia.com/fr-fr/geforce/graphics-cards/50-series/rtx-5090/>

[42] D. Hendrycks *et al.*, « Measuring Massive Multitask Language Understanding », 12 janvier 2021, *arXiv*: arXiv:2009.03300. doi: 10.48550/arXiv.2009.03300.

[43] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, et S. R. Bowman, « GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding », 22 février 2019, *arXiv*: arXiv:1804.07461. doi: 10.48550/arXiv.1804.07461.

[44] « MMLU | DeepEval - The Open-Source LLM Evaluation Framework ». Consulté le: 2 décembre 2025. [En ligne]. Disponible sur: <https://deepeval.com/docs/benchmarks-mmlu>

[45] « RobotStudyCompanion/Benchmarking_LLM: Benchmarking LLM ». Consulté le: 16 décembre 2025. [En ligne]. Disponible sur: https://github.com/RobotStudyCompanion/Benchmarking_LLM

[46] F. Bang, « GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings », in *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, L. Tan, D. Milajevs, G. Chauhan, J. Gwinnup, et E. Rippeth, Éd., Singapore: Association for Computational Linguistics, déc. 2023, p. 212-218. doi: 10.18653/v1/2023.nlposs-1.24.

[47] « The Chatbot Usability Questionnaire (CUQ) ». Consulté le: 22 décembre 2025. [En ligne]. Disponible sur: <https://www.ulster.ac.uk/research/topic/computer-science/artificial-intelligence/projects/cuq>

[48] « orlandossss/Evaluation_Model_Code ». Consulté le: 21 décembre 2025. [En ligne]. Disponible sur: https://github.com/orlandossss/Evaluation_Model_Code

[49] « Gemma 3 », Google DeepMind. Consulté le: 7 décembre 2025. [En ligne]. Disponible sur: <https://deepmind.google/models/gemma/gemma-3/>

[50] « RobotStudyCompanion/LocalLM_RSC: Code and informations about the local LM of the RSC robot ». Consulté le: 21 décembre 2025. [En ligne]. Disponible sur: https://github.com/RobotStudyCompanion/LocalLM_RSC

Appendices

AI use

During this thesis, for the coding part and the implementation of the benchmark and the three-tier architecture, Claude code was used with the 4.5 Sonnet models. An appendix file has been created for the benchmark. This appendix file is completed by Claude each time a new request is asked of him reporting the query and a summary of the answers. This was implemented a bit after the creation of the repository, and the first part of the code was already created, meaning some of the prompts used are not fully reported.

Code building

The process of creating that code using AI is almost always the same. The architecture was already thought of before as well as the main function. A general main file is created with every module that describes all the functions, their possible input and output as well as the way they are going to be used.

The use of Claude code is step by step. Every module is asked to be created one by one. Claude code has this way of asking for each function creation if the results are as expected, allowing a constant review first and a potential modification of some parts. Claude is always making some unexpected additions to the modules, some of these functions are completely superficial and so not added but sometimes those functions actually bring some interesting tools that were not thought of before but can increase the overall quality of the code.

At the end of each module, a testing file is asked to be created by Claude again and then the results are verified to ensure the good functioning of the module itself and that no unexpected behavior has happened.

Claude has a tendency to add a lot of print to make it easy to read the output of each function and help with debugging. This is useful when processing and creating the code, but it needs to be suppressed for implementation to not slow down the code and make it more readable for potential other users. A 'code cleaning' will be done to ensure good understanding for potential future users who would use the benchmark or the three-tier architecture.

Writing

Grammarly was used during the writing of this thesis to prevent spelling mistakes and improve English writing to a more academic way.

Exemple of the appendix file :

```
## Entry #12
**Date:** 2025-11-30
**Model Used:** Claude Sonnet 4.5 (claude-sonnet-4-5-20250929)

**Prompt:**
> can you add the splitting into 2 categories for the performance_vs_teaching_effectiveness and teaching_effectiveness_score into:
> small models (less than 2B)
> big models (more than 2B)
```

```
>
> Do this modification for @analyse_results_computer.py and @analyze_results_pi4.py

**Response Summary:**
Modified both `analyse_results_computer.py` and `analyze_results_pi4.py` to split teaching effectiveness visualizations into three graph sets based on model size:

**Refactored `plot_teaching_scores()` method:**
- Categorizes models into small (<2B) and big (≥2B) using existing `_parse_model_size()` method
- Generates 3 graph sets: small models only, big models only, and all models combined
- Added console output showing model count per category

**Created two new helper methods:**
1. `_plot_teaching_bar_chart(models, model_scores, output_path, suffix)`: Generates bar charts of average teaching effectiveness scores with category-specific titles and filenames
2. `_plot_performance_vs_teaching(models, output_path, suffix)`: Generates scatter plots correlating tokens-per-second performance with teaching quality scores

**Generated files (6 graphs per analysis):**
- Small models: `teaching_effectiveness_scores_small.png`, `performance_vs_teaching_small.png`
- Big models: `teaching_effectiveness_scores_big.png`, `performance_vs_teaching_big.png`
- All models: `teaching_effectiveness_scores_all.png`, `performance_vs_teaching_all.png`

Each graph includes appropriate category labels in titles for easy identification.

---
```

Benchmark GitHub link:

https://github.com/RobotStudyCompanion/Benchmarking_LLM.git

Evaluation experiment GitHub link:

https://github.com/orlandossss/Evaluation_Model_Code.git

Architecture GitHub link:

https://github.com/RobotStudyCompanion/LocalLM_RSC.git

Non-exclusive licence to reproduce thesis and make thesis public

I, Dorian Lamouille

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

“Deploying Educational Language Model
on Low-Resource Social Robot”

supervised by Farnaz Baksh & Matevž Borjan Zorec

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Dorian Lamouille

22.12.2025