POKT AI Lab

Report II

Authors:
Nicolas Aguirre (nicolas@poktscan.com)
Ramiro Rodríguez Colmeiro (ramiro@poktscan.com)
POKTscan Data Science Team

April 30, 2024

Disclaimer

This work was partially funded by Pocket Scan Technologies LLC, a company operating on the Pocket Network.

Changelog

v1.0	2024-05-01	First version.

Table of Contents

1 Progress Overview

During the month of April, the socket achieved the following milestones:

- Helped in the creation of the *POKT Square RAG Agent*[1].
- Finished the architectural design of the asynchronic testing procedure (the Pocket machine learning (ML) Test-Bench).
- Merged issues on the Test-Bench code:
  - Template code for Temporal IO on Python [2].
  - Created docker-compose files for deploying the test-bench for development [3].
  - Combined the task of storing Hugging Face (HF) datasets with the Sampler to create a single Temporal App for both [4].
  - Added code for saving and retrieving tasks from MongoDB [5].
  - Updated the readmes with final software architecture [6].
  - Added basic requester code [7].
- A lot of code was also merged code without a matching issue (due to premature stage of the project), it includes:
  - Creating a proper logger function for both Go and Python that rungs OK with temporal.
  - Added initial *Sampler* workflow code in Python.
  - Cleaned all Readmes for better understanding of the repository.
  - Make the code more independent from Language Model Evaluation Harness (LMEH).
  - Added packages for Pocket RPC and MongoDB connection handling in Go.

[1] https://github.com/pokt-scan/pokt-square
[2] https://github.com/pokt-scan/pocket-ml-testbench/issues/9
[3] https://github.com/pokt-scan/pocket-ml-testbench/issues/12
[4] https://github.com/pokt-scan/pocket-ml-testbench/issues/18
[5] https://github.com/pokt-scan/pocket-ml-testbench/issues/17
[6] https://github.com/pokt-scan/pocket-ml-testbench/issues/14
[7] https://github.com/pokt-scan/pocket-ml-testbench/issues/11

2 Language Models - Benchmarks and Leaderboards

> I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be.
>
> *William Thomson, 1st Baron Kelvin*

In many situations in our lives, we face the problem of measuring. Sometimes is the temperature of the oven, sometimes it is the quality of a large language model (LLM). It is often said that if you can't measure something, you can't improve or control it, which is mostly true, if you miss the right temperature you will ruin your meal and if you don't know how an LLM behaves you will get nonsense from it. However, depending on the problem at hand, measuring something can also become a problem. If we are designing ovens and we focus solely on controlling the oven temperature and forget that food needs to fit in you will get a wonderful temperature control that wont help you make a cake, or in the case of LLMs, you will get outstanding scores that wont actually mean anything. This problem was conceptualized by Charles Goodhart, "When a measure becomes a target, it ceases to be a good measure" [19], meaning that over focusing on some measurements results in failure. In the engineering world live in a constant fight between these two concepts, trying to measure something and trying not to get blinded by the resulting metrics. In the case of ovens we are somewhat above this problem and we know whats important to build them and make great food, on the case of LLMs, we are not there yet.

Through the time, and more specifically in the field of engineering, having a Common Task Framework (CTF) helped set the objectives and measure the improvements of a given field, let it be machine translation, hand-written character recognition, or any task. A CTF is normally composed of [6]:

1. A public dataset, feature measurements and labels.

2. A set of participants, engaged in the task of predicting labels.
3. A scoring referee, to which competitors can submit their work for evaluation.

The field of ML grew very used to these kinds of setups to track and guide the advancement of research, and many important competitions were created following these patterns, for example using EMNIST [5] for hand-written number recognition or SuperGLUE [20] for text understanding.

As we know, the LLMs are ML models and as such the way that we measure them is by mean of CTFs. An special quality of the LLMs is that they are not used for a single thing and these makes them a little harder to measure. For example, in the past it was enough for a vision model to achieve a great score in the MNIST dataset to revolutionize the field of ML, like AlexNet [9] did. Today LLMs need to past several of these tests to create impact. It's not enough for LLMs to excel at a single CTF, a LLM needs to be good at a large variety of them. Now the question is: Which of them? There are many CTFs from where to choose, testing all of them is expensive and making sense of all the generated numbers is difficult. To solve this the scientific community came up with collections of CTFs that are organized into *leaderboards* or *benchmarks*, being the most publicly known the Holistic Evaluation of Language Models (HELM) [10] and LMEH [7].

At this point, we can see that in order to measure the quality of a LLM is not a straightforward thing. Lots of discretional choices are made in the process of selecting a benchmark to use (like, what I want to know about the LLM, or what is its application). Also, the selected benchmark is built upon many assumptions around how the different CTFs reflect the various characteristics of a LLM. Finally, each of these CTFs are constructed on datasets that have other assumption or limitations (like, they are built from whats available and not form what they need sample to measure something [17]).

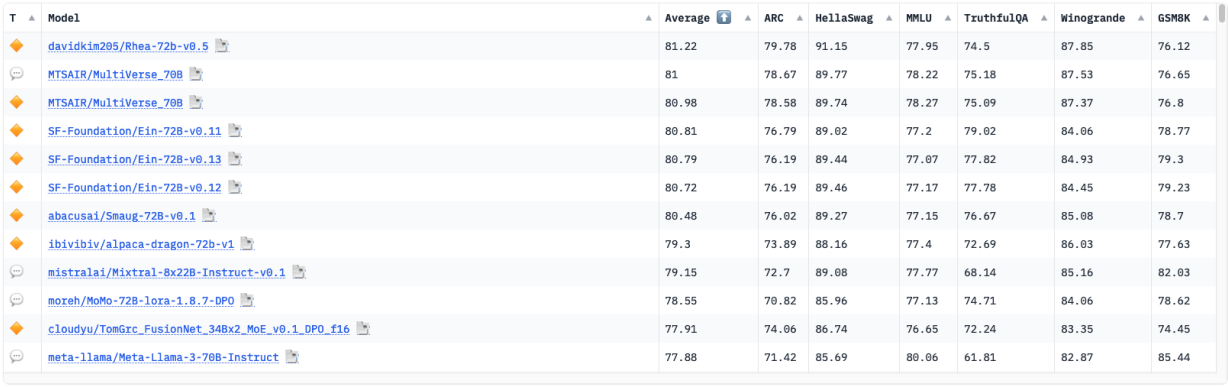
We can see how measuring a LLM is far more complicated that measuring the oven temperature, and how fragile and opinionated is the construction of the benchmarks or leaderboards, as opposed to setting the right temperature to bake a cake [8]. However, as Lord Kelvin points out, we need to measure something in order to know something even when it means that we are accepting lots of assumptions.

2.1 Language Model Evaluation Harness

Today there are a large number of benchmarks that can be implemented to describe the quality of a LLM. Their number grows constantly, so we wont try to mention all of them. In

[8] We acknowledge our lack of experience in baking cakes and apologize for this oversimplification.

this section we will focus on a single one of them, the LMEH. We choose to use this framework due to be the one used to compute the HF Open LLM leaderboard [1], shown in figure 1. This leaderboard is usually referenced when comparing different language models, and since the Pocket Network will be composed of mostly open models, it is an easy choice that will provide the Pocket Machine Learning Test-Bench (MLTB) with a direct comparison.



T	Model	Average ⬆️	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K
🔶	davidkim205/Rhea-72b-v0.5	81.22	79.78	91.15	77.95	74.5	87.85	76.12
💬	MTSAIR/MultiVerse_70B	81	78.67	89.77	78.22	75.18	87.53	76.65
🔶	MTSAIR/MultiVerse_70B	80.98	78.58	89.74	78.27	75.09	87.37	76.8
🔶	SF-Foundation/Ein-72B-v0.11	80.81	76.79	89.02	77.2	79.02	84.06	78.77
🔶	SF-Foundation/Ein-72B-v0.13	80.79	76.19	89.44	77.07	77.82	84.93	79.3
🔶	SF-Foundation/Ein-72B-v0.12	80.72	76.19	89.46	77.17	77.78	84.45	79.23
🔶	abacusai/Smaug-72B-v0.1	80.48	76.02	89.27	77.15	76.67	85.08	78.7
🔶	ibivibiv/alpaca-dragon-72b-v1	79.3	73.89	88.16	77.4	72.69	86.03	77.63
💬	mistralai/Mixtral-8x22B-Instruct-v0.1	79.15	72.7	89.08	77.77	68.14	85.16	82.03
💬	moreh/MoMo-72B-lora-1.8.7-DPO	78.55	70.82	85.96	77.13	74.71	84.06	78.62
🔶	cloudyu/TomGrc_FusionNet_34Bx2_MoE_v0.1_DPO_f16	77.91	74.06	86.74	76.65	72.24	83.35	74.45
💬	meta-llama/Meta-Llama-3-70B-Instruct	77.88	71.42	85.69	80.06	61.81	82.87	85.44

Figure 1: Screenshot of the Hugging Face Open LLM leaderboard (https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard) taken on 2024-04-25.

The LMEH is composed of many different CTFs built upon many datasets [9] and each of them with one or more associated metrics. The pair of dataset-metrics makes up a *task*, and a leaderboard is composed of many tasks. The one that we want to reproduce initially for the Pocket Network is the Open LLM leaderboard from HF which is based on many paris of task-dataset, organized in 6 groups:

- AI2 Reasoning Challenge [3] : Natural questions from grade-school science created for testing humans.
- HellaSwag [21] : Sentence completion tasks that are trivial to humans, designed to test model's commonsense.
- MMLU [8] : Multiple choice questions on more than 57 tasks, including elementary mathematics, US history, computer science, law, and more.
- TruthfulQA [12] : Questions designed to test common human misconceptions on 38 categories, including health, law, finance and politics.
- Winogrande [18] : A set of 44000 expert-created questions to test commonsense reasoning, created via crowdsourcing.

[9] LMEH supports ≈90 datasets at the time of writing

- GSM8k [4] : A set of 8500 grade school math problems written in a linguistically diverse way.

The calculation of the scores in each of these areas varies depending on the task, but there are three main ways of getting an score (for these or any other tasks):

- Accuracy: The model is given a prompt and the generated text is compared to the expected answer and the number of correct matches is used. The comparison for a correct answer can be using an exact match or near matches, for example, if you ask "who was the first person in the moon?" the answer should be "Neil Armstrong", or "Neil Alden Armstrong" or "Armstrong, Neil A." and so on. Also, used in most multiple choise tests, the answer of the LLM can be chosen by presenting each question-answer pair to the model and selecting that with the highest generation probability [10].
- Perplexity: In this case the "surprise" of the model is calculated for each answer. Given a prompt, the probability that the model would produce that prompt is calculated. A higher perplexity means that the model is less likely to produce a given prompt. So, if a given question-answer pair has a high perplexity it means that the model is not confident on this answer (usually used to test the correct answer).
- Others: There are also metrics specific to compare models outputs, like BLEU [14] or ROUGE [11] that will provide a value of how much text pieces relate (useful to compare summarization tasks).

In conclusion, a leaderboard like the one in HF is nothing more than a collection of metrics that were calculated based on known tasks (datasets), so, in order to reproduce it we only need to execute a set of tasks on the nodes in the Pocket Network and reconstruct the leaderboard using the LMEH framework (which is not so easy, as we will see section 3).

2.2 Leaderboards Weaknesses

Until now we described the how a LLM is measured and what are the accepted ways of grouping and presenting the scores. However there is a big issue around all this subject that is not often talk about, which is the validity of them.

The selected tasks used in the HF leaderboard are not justified using a developed taxonomy like the one proposed in HELM, they seem to respond to simpler motivations, focused on providing enough task diversity (covering, language, reasoning, knowledge and math) using the available and relevant datasets that are interesting to the community and relevant to real-world use cases. This last point is a key issue, the relevance of them to real-world use

[10] by means of the tokens probabilities, also calculated by the LLM.

cases.

When we described the metrics used in the HF leaderboard we talked about 6 groups of tasks, each of them specialized in different areas of interest. If we go deeper into the selected tasks, we will find that instead of 6 metrics (the columns shown in figure 1), around 60 metrics are calculated and then they are averaged into categories (for example, for the MMLU score, the score for law knowledge is averaged with math). Moreover, many people only look at the average, which is an average of averages of wildly different things. Now the question seems obvious, what does this mean? are these leaderboards even useful for something?

The world of ML faces this problem since a while, it seem to have fallen victim of its own thirst for achieving the next State of the Art (SOTA) and revive the history of AlexNet. This subject is really vast and we wont pretend to cover it here [11], we only want to highlight the current limitations and how the Pocket Network can help.

A key missing piece in relating scores to actual usefulness is the feedback from users, not only prompt-based feedback like the one we give the various LLM chat bots when they kindly ask for it, but market feedback. Going back to the example of ovens, how did we got so good at building ovens? We could argue that it was in part a selection process where those ovens that did not have the correct balance between temperature control and cavity size were not selected by buyers and they disappeared. The buyer probably were unaware of the many metrics that a particular oven had and just selected the oven that fitted their needs. In the same spirit we can believe that the Pocket Network could be the missing piece between the leaderboards and the users, the free market place were models come to find users and users could freely change from one model to another until they find the best one. This usage data, correlated with a correlated leaderboard (a set of CTFs) can be of great utility to guide the development of new models tailored for real-world use cases. In the end we might finally get a LLM as reliable as our good old oven.

[11] For a comprehensive read on this subject we recommend the following works: [13] [2] [17]

3 Pocket Machine Learning Test-Bench

The MLTB is an off-chain element of the Pocket Network whose objective is to provide online tracking of the ML nodes of the network. It can also be thought as an specialized and early version of a Pocket Network *Watcher* [12]. Initially the MLTB will be focusing on LLM metrics, but it is designed to be agnostic of the type of service to evaluate.

3.1 Overview of the solution

This first version of the MLTB is focused on implementing the LMEH framework, specifically to be able to reproduce the tasks used to create the HF Open LLM leaderboard. Recreating this leaderboard for Pocket Network's nodes is not straightforward, as there are important differences between a self-hosted or centralized LLM service and the behavior of the nodes in the network. The measurement process should be able to handle these situations:

- **Service availability**: A Pocket Node is not always available to any app in the network, the random session construction means that a given node to be tested could not be reachable at a given time. Also, a node can be down due to connection issues or being a faulty node.
- **Time windows**: When an app and a node are paired, the connection only last one hour [13]. While this time might seem large, the fact that it is finite means that at any point the node can refuse to continue answering our queries if the time is up.
- **Identity/quality changes**: A node is not forced to provide a fixed model, as Quality of Service (QoS) is expected to change over time (improve, adapt to consumers choices). This means that a node's metrics should be tracked over time and not just instantly and never repeated.

These conditions shaped the test-bench design, leading us to adopt an asynchronic approach to the problem. The solution that we are proposing is shown in figure 2, where we can see that the process is articulated using 4 different modules (shown in orange), a database for the tests data or datasets (shown in blue) and coordination services (shown in black).

[12] While this is likely to change, a watcher is described here: https://github.com/pokt-network/pocket-network-protocol/blob/main/utility/README.md#33-fisherman-protocol

[13] On Morse, where sessions are 4 blocks, each of 15 minutes.

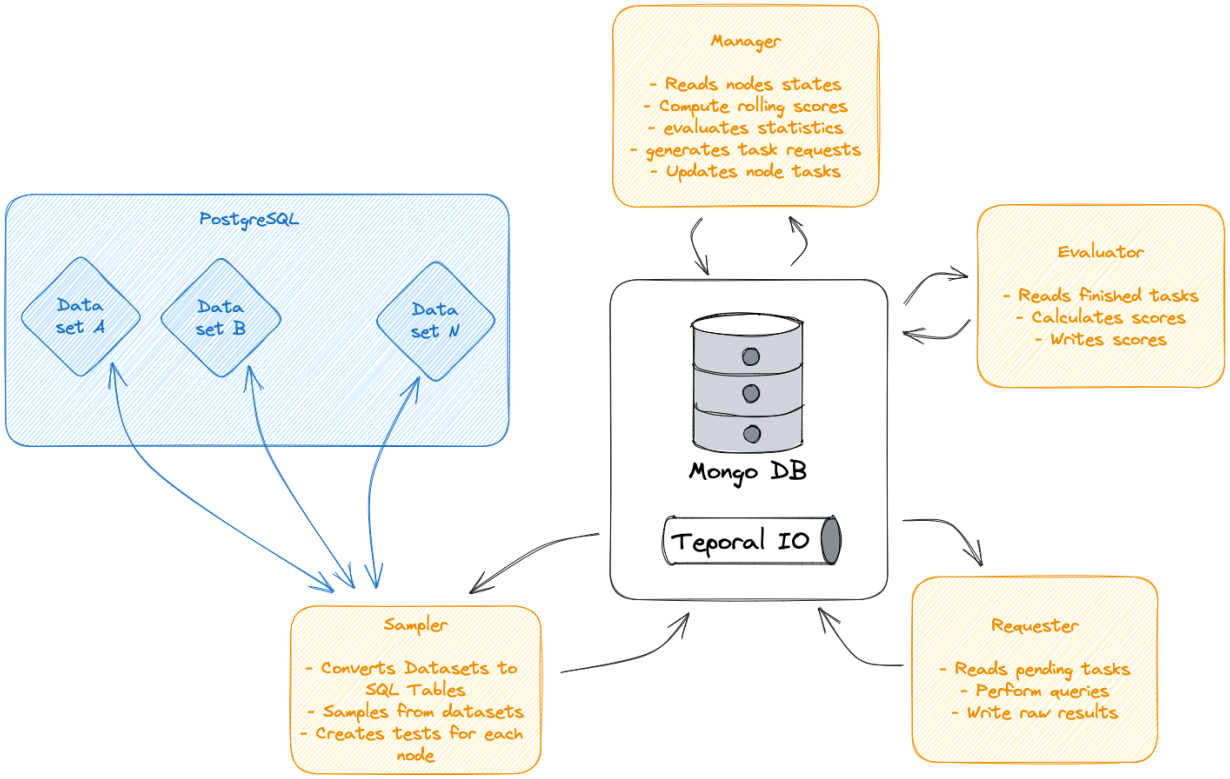


Figure 2: Overview of the different elements composing the Pocket's MLTB.

Each of the MLTB four main blocks work together to track the selected CTFs (or *tasks*) scores of each of the nodes in the Pocket Network. Each of the main blocks do the following:

- **Manager:** This block is in charge of keeping account of all the nodes in the networks and the scores associated with the tracked CTFs. It will periodically refresh the list of nodes that are staked to a given Pocket Network service, review each of them and look for missing or outdated scores, remove old data, include new test results (if available from the evaluator) and request new tests (following a given strategy). Also, it is responsible of generating the average scores and other information that is later exposed to the public (i.e. generate the metrics needed to reproduce a leaderboard).
- **Sampler:** This block is in charge of handling the actual data that is used to generate each sample from a given CTF. The manager gives the sampler an indication of which node to test, on which task and how many samples, with this data the sampler collects the required information and creates a generic call request for the Pocket Network. This call request is stored in a database where it waits until it can be executed. The sampler is also responsible of tracking datasets and keeping them available in a fast-access database. Each time a new CTF is requested, a new collection will be written in the datasets database.

- **Requester:** All the pending requests, generated by the sampler, are consumed by this block. Its job is to interact with the Pocket Network, implementing retries, timeouts and apps management. It controls a series of apps that are used to make relays, it periodically checks if any of these apps is paired (in session) with a node that has pending requests and execute the requests when possible. After fulfilling a requests it writes the raw results back in the database and commands the evaluator to finish the measurement cycle.
- **Evaluator:** This is the final block of the measurement process, it takes the task information and the node's response and finishes the task requirement. For each task it will create a result (a value of a metric specific to the task at hand) and write it back to the database.

The coordination of all these blocks is done using Temporal IO [14] where each block translates into a different work queue. This framework was chosen due to its robustness and scaling capabilities. As the Pocket Network grows, the number of nodes and tasks to test will grow too, in order to keep fresh track of all nodes, the measurement process needs to be easily parallelized and modular.

The data between the workers and also the data that composes the node's profiles (used to create the leaderboard) is managed using MongoDB [15] because it offer us the possibility of adding new fields to the records without re-writing the whole database. This is needed when a new CTF is added to a collection of nodes that are already in the test rotation.

To record the datasets of the different CTF we chose to use PostgreSQL [16] for ease of integration with common ML development tools such as Torch [15] and HF.

3.2 In-depth workings

To keep the language concise, we opted to follow the LMEH nomenclature to describe different parts of a CTF. Each job in the MLTB, an end-to-end cycle through all the apps, is a partial execution of a CTF consisting of:

- **Task:** The main element of the job, it is composed of all the information and metadata required to produce a valid Pocket Network request for a particular CTF on a given node, including:
  - Reference ID: A unique identifier of the job.

[14] https://temporal.io/
[15] https://www.mongodb.com/
[16] https://www.postgresql.org/

  - Requester Args: Set of arguments to be used by the requester to make the call like: node address, service, method, path.
  - Framework: A high order grouping (or framework name) like LMEH or HELM.
  - Task: The name of the CTF to test, like *mmlu_high_school_macroeconomics*"
  - Quantity: The amount of tests to do, a job run commonly tests only a sub-set of all the samples available in the CTF.
  - Blacklist: The sample's indexes that were already performed and should not be part of futures jobs. This is used to not repeat part of the CTF on successive calls.

- **Instance:** After being processed by the sampler, each task will produce an *instance*. An instance is a set of calls to a Pocket Network node, it is composed of:
  - Reference ID: A reference to the originating task.
  - Task Reference ID: A reference to the task that originated this instance.
  - Node address: Node to be tested.
  - Service: The service that is going to be called.
  - Query list: A list of queries to perform. The length of the list will vary, as some tasks require multiple queries to the model in order to be solved. Each of the elements in this list will have all the data required to make a successful call in the Pocket Network (like *method* and *path* to call) via Task Reference ID and the reference IDs of the prompts.

- **Prompt:** This is the actual content of the request that is going to be executed against the node. Since prompts can be lengthy, we opted for a separated collection to hold this data. Each element consists of a reference ID to the corresponding instance and the payload to be sent.

- **Response:** The raw result of a prompt. The outputs of the model as answer to the given prompts are stored without any change into a collection for later analysis. It also has a reference ID to the corresponding prompt.

- **Result:** This is the actual numerical value of a metric that was applied to a finished instance. This value is created by recovering the task metadata and performing the specific calculations over the responses. Ot is also referenced to the original task via an ID.

Each of these elements of the CTF are in fact different collections in the Mongo database and are also the communication paths between the different blocks in the MLTB. A closer look of the internal workings of the test-bench is presented in figure 3.

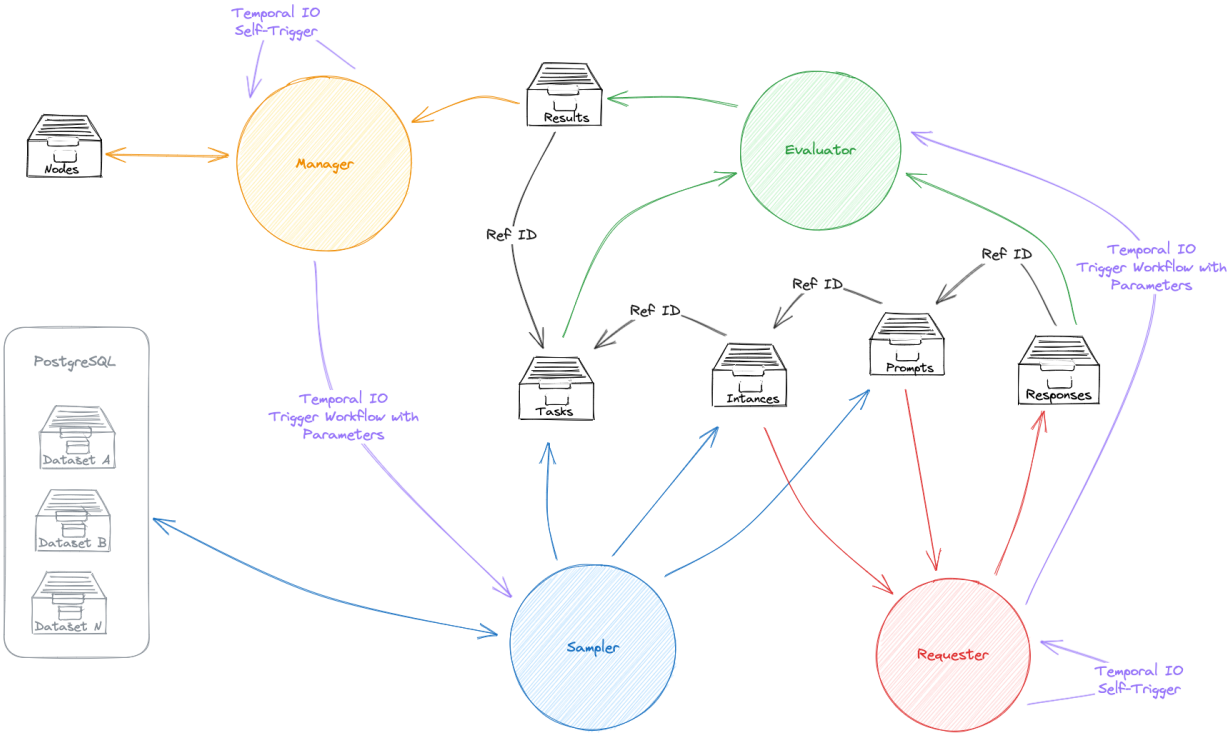


Figure 3: Detail of the Pocket's MLTB internal working, including databases relations to execution of work queues, triggering signals (in violet) and collections references for task reconstruction.

The different workers' queues of the MLTB (shown in colors: orange, blue, red and green) are controlled by the Temporal IO triggers (shown in violet). The manager and requester are the two processes controlling the workflow and executing periodically (cadency depends on configurations), they are responsible on executing the sampler and evaluator workflows respectively. The MongoDB collections are shown in black, and each collection corresponds to a part of the CTF job being executed (described above). The PostgreSQL databases are shown in grey and they only interact with the sampler workflow.

A MLTB job lifecicle has the following steps:

1. At a given time, the manager will inspect the network nodes, find a missing task score and execute the sampler workflow with a given job config.

2. The sampler will create a task definition and save it to the *Tasks* collection, along with its instance and prompts data in their respective collections: *Instances* and *Prompts*. All these entries are referenced using unique job IDs.

3. Later, when the requester checks for sessions and finds the target node in a controlled session, it will start the prompt process. This process begins by reconstructing the prompt from the *Instances* and *Prompts* collections. After obtaining a response it will write it to the *Responses* collections and trigger the execution of the evaluator workflow for this task.
4. The evaluator will finally recreate the CTF job by means of the task ID and all referencing IDs, calculate the necessary metrics and write the results in the *Results* collection.
5. The job is closed in a subsequent manager workflow execution that will read the evaluator's metrics written to the *Results* collection, clear the *Tasks* entry (and referenced IDs) and update the *Nodes* collection entry.

3.3 Leaderboard Data

The data required to recreate a leaderboard using the MLTB data is found on the *Nodes* collection. This collection has one entry per node and service [17] that tracks the following data:

- Node address
- Service
- Last seen height (Pocket Network's block number)
- Last seen time (ISO date)
- Tasks : A vector of all the tracked CTFs, containing:
  - Framework : Name of the evaluation framework.
  - Task : Name of the task or CTF.
  - Mean score : Mean value of the tracked scores.
  - Std. score : Standard deviation of the tracked scores.
  - Median score : Median value of the tracked scores.
  - Num. samples : Number of processed samples in the configured time frame.
  - Scores samples : Score of each of the samples tested in the configured time frame.
  - Times : ISO date at which each of the samples was obtained.

[17] On Shanon update there probably be a single service per node allowed, making this database have a single entry per node.

The collection is not created in the form of a leaderboard, it is not ordered by any kind of score, nor it calculates global scores (i.e. averages) between different tasks. However producing a replicate of any leaderboard using the provided data is trivial (its merely a grouping and averaging of tasks scores).

It's important to note that the scores calculated for each task are volatile, meaning that they are expected to change (due to normal variability in sampling strategy or node performance changes through time). They represent the instantaneous score assigned to a given task based on a number of samples that were obtained in a fixed time frame. The length of the buffer and the time frame are two configurable values that depend on different things:

1. **Metric buffer length**: It determines the accuracy of the task assigned score, normally with 50 samples it is enough to provide good approximates [16]. This length also defines the time it takes the system to produce a complete run over a node, the larger the buffer, the longer it takes to be filled.
2. **Score time frame**: Since the system is designed to evaluate models that we cannot guarantee to remain the same through time, a given time frame must be given for the samples. This time frame defines a time to live for the samples in the scores buffers, when a sample surpass this time it is erased and a process to sample a new one is created. This value is configurable, the shorter the time, the fastest the test cadence, we expect this value to be in the order of days.

4 Future Work

This moth we focused of building the MLTB code, we defined the main architecture and built the basic code around it. We expect to continue building this code during the next month and be able to produce the first tests on a local network. In order to do so we will have to complete the following tasks:

1. Finish the *Manager* code: Most work to be done is buffer handling. We do not expect to implement elaborated sampling strategies at this pint.
2. Finish the *Sampler* code: Task is advanced we need to finish and test the correct writing of the MongoDB collections.
3. Finish the *Requester* code: Almost complete, working on error handling and test code.
4. Begin and Finish the *Evaluator* code.
5. Create code for proof of concept deployment on local net.

One of the tasks that we worked during the month of April was the *POKT Square RAG Agent*[18]. We will not be performing active development of this product during May due to time and human resources constraints. Nevertheless we will be responsive and provide feedback to any team that wishes to pick up the project in the meantime.

[18] https://github.com/pokt-scan/pokt-square

List of Acronyms

CTF	Common Task Framework
HELM	Holistic Evaluation of Language Models
HF	Hugging Face
LLM	large language model
LMEH	Language Model Evaluation Harness
ML	machine learning
MLTB	Machine Learning Test-Bench
QoS	Quality of Service
SOTA	State of the Art

Reference List

[1] Edward Beeching et al. *Open LLM Leaderboard*. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard. 2023.

[2] François Chollet. "On the measure of intelligence". In: *arXiv preprint arXiv:1911.01547* (2019).

[3] Peter Clark et al. "Think you have solved question answering? try arc, the ai2 reasoning challenge". In: *arXiv preprint arXiv:1803.05457* (2018).

[4] Karl Cobbe et al. "Training verifiers to solve math word problems". In: *arXiv preprint arXiv:2110.14168* (2021).

[5] Gregory Cohen et al. "EMNIST: Extending MNIST to handwritten letters". In: *2017 international joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 2921–2926.

[6] David Donoho. "50 years of data science". In: *Journal of Computational and Graphical Statistics* 26.4 (2017), pp. 745–766.

[7] Leo Gao et al. *A framework for few-shot language model evaluation*. Version v0.4.0. Dec. 2023. DOI: 10.5281/zenodo.10256836. URL: https://zenodo.org/records/10256836.

[8] Dan Hendrycks et al. "Measuring massive multitask language understanding". In: *arXiv preprint arXiv:2009.03300* (2020).

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[10] Percy Liang et al. *Holistic Evaluation of Language Models*. Oct. 1, 2023. DOI: 10.48550/arXiv.2211.09110. arXiv: 2211.09110[cs]. URL: http://arxiv.org/abs/2211.09110 (visited on 03/12/2024).

[11] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[12] Stephanie Lin, Jacob Hilton, and Owain Evans. "Truthfulqa: Measuring how models mimic human falsehoods". In: *arXiv preprint arXiv:2109.07958* (2021).

[13] Made Nindyatama Nityasya et al. "On" Scientific Debt" in NLP: A Case for More Rigour in Language Model Pre-Training Research". In: *arXiv preprint arXiv:2306.02870* (2023).

[14] Kishore Papineni et al. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[15] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[16] Felipe Maia Polo et al. “tinyBenchmarks: evaluating LLMs with fewer examples”. In: *arXiv preprint arXiv:2402.14992* (2024).

[17] Inioluwa Deborah Raji et al. “AI and the everything in the whole wide world benchmark”. In: *arXiv preprint arXiv:2111.15366* (2021).

[18] Keisuke Sakaguchi et al. “An adversarial winograd schema challenge at scale”. In: *arXiv preprint arXiv:1907.10641* (2019).

[19] Marilyn Strathern. “‘Improving ratings’: audit in the British University system”. In: *European review* 5.3 (1997), pp. 305–321.

[20] Alex Wang et al. “Superglue: A stickier benchmark for general-purpose language understanding systems”. In: *Advances in neural information processing systems* 32 (2019).

[21] Rowan Zellers et al. “Hellaswag: Can a machine really finish your sentence?” In: *arXiv preprint arXiv:1905.07830* (2019).