POKT AI Lab

Report I

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

April 1, 2024

Disclaimer

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

Changelog

v1.0	2024-04-01	First version.

Table of Contents

1 Progress Overview

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

- Released a Local-Net proof of concept for the Pocket Network (Morse)[1]. The Local-Net features two machine learning (ML) services: large language models (LLMs) and Diffusion Models (text to image generation).
- Updated *pocket-core* and *pocket-localnet* repos (POKTscan forks) to support ML relays:
  - The default mesh config timed-out on ml calls (both LLMs and Diffusers).
  - The pocket core had a limit of 1 MB incoming RPC sizes, a parameter was added to make this configurable and enforced in a more modular fashion.
  - The relayer app of *pocket-localnet* was updated to allow response dumping and custom relays (this will only be used for the proof of concept).
- Released guides to deploy ML models:
  - LLMs using vLLM backend and OpenAI API compatibility[2].
  - Diffusers using a custom backend that looks to be like Stable Diffusion API (needs more work)[3].
- Started work on benchmarking process:
  - Conceptual designed almost settled.
  - Started to work on *lm-eval-harness* implementation.

[1] https://github.com/pokt-scan/pocket-ml-testbench/tree/main/morse-localnet-poc
[2] https://github.com/pokt-scan/pocket-ml-testbench/tree/main/model-deployment/llm
[3] https://github.com/pokt-scan/pocket-ml-testbench/tree/main/model-deployment/diffusers

2 Machine Learning Models

ML is a wide term that conflates many different kinds of mathematical models. In this section we pretend to present their main distinctions in terms of the methods used to test or train them. This distinction is very important to the Pocket Network, since one of the main challenges of the protocol is to asses the correctness of the models inference.

For those not familiar with the terminology, the terms artificial intelligence (AI) and ML are not interchangeable. The first one refers to systems that perform tasks that require human intelligence and the second one is a subset of AI which contains algorithms that are capable of learning from data. The later is the correct one to use in the context of the models served by the Pocket Network.

2.1 Supervised Models - Ground Truth

Supervised models are the subset of ML models that are trained on tasks that have a specific goal. For instance, an algorithm that tries to predict the movements of the financial market is a supervised model. That model has a specific task, given some inputs (any kind, lets say previous prices and market volumes), predict if an asset will increase or decrease its value. During training the model will be presented with known pairs of inputs-outputs and the model output will be compared to the objective using a hard metric like quadratic distance. The performance of these models is measured in the same way that they are trained, only that for testing the model is given sets of inputs that it has never seen before.

In the case of supervised models the ground truth is always present, or it will be revealed at some point in time. In our financial market movements example, we just need to wait and see what happens. So, if we want to test the quality of these models there is a mathematical way to do it and the data is or will be available.

These kinds of models are easy to test on production and their quality can be measured easily on the Pocket Network, we just need to feed them test samples and check the output error, or even create simple majority errors and punish outliers [4].

Examples of these models are:

- Linear Regression (predict prices of an asset).
- Classification (by any technique, like neural networks or support vector machines).
- Decision Trees (credit risk assessments).

[4] The Allora network has a great lite paper on a blockchain based implementation for these kind of models, see [10].

- Convolutional Neural Networks (image segmentation)

2.2 Unsupervised Models - Structure Discovery

Another type of ML models are those models that do not need a human annotated dataset or a absolute truth value to be trained. The training sets are only composed of inputs to the network, examples with no tags or questions without answers, and hence the training is done in an unsupervised fashion, no metric is *directly* guiding the output of the model.

These kind of models are normally used to discover patterns in data (clustering), like dividing large groups of data. An example of these kind of models are the natural language embeddings[5], these models take as input a string of characters and produce an output that is a codified vector of floats. The output of the model is actually a point in a multidimensional space, and its neighbors (other points close to it) are projections of other strings that have the same meaning or talk about the same subject. The natural language embeddings (like ROBERTA [15]) are used in retrieval augmented generation (RAG) systems [11], when these RAGs need to find context for a query they first pass the input string to one of these models and retrieve its codification and then they use a vector database to find other strings in the vicinity of the input's code. Then they read the data in the neighboring codes and produce an output based on those.

Unsupervised models like encoders used for RAGs can be the easiest models to support in the Pocket Network, they are like blockchains in a sense. A given query should always return *exactly* the same output, so a simple majority voting or challenge mechanism can be used to prove the data integrity. Other models that perform free-structure discovery (like clustering) could require further analysis (like overlapping metrics).

Examples of these models are:

- Encoding (create embeddings for vector databases).
- Clustering (discover communities inside a social networks).
- Outliers detection (fraud analysis).

2.3 Generative Models

The last category of ML models that we will present here is the one that contains the models that are trained to sample from a given probabilistic distribution. This group of models is not created by looking at how the model is trained, in fact they can be unsupervised or super-

[5] Although in general these models then do supervised fine-tuning, in order to simplify and not go into details, we decided to place text embedding in this category, since in general its first training phase is unsupervised.

vised models (to some extent). What make these models special is that they are trained to reproduce an arbitrary distribution of data that is represented in the train dataset, like human faces or Shakespeare books. The training process of these models generally have no hardcoded target metric, they often rely on adversarial training o reinforcement learning. In the adversarial training [7], the model that is generating the outputs is guided by an other model that is trained at the same time to discriminate between generated samples and real dataset samples, leading to a Nash-Equilibrium [16] and a (hopefully) effective generator model. On the reinforcement learning setup [18], a human is used in the loop to evaluate or generate preference scores of a series of model outputs, then the model is tuned to follow these preferences using a reward system. In either case, there is no exact score for a given output.

The most known generative models uses are the text-to-image and chatbots applications, examples of these are Stable Diffusion (SD) and ChatGPT respectively. When we use these models we do not seek for a fixed output, instead, we look for a creative output or an output that is built following some guide, like "an image of a cat on a keyboard" or "Create an article that talks about the different kinds of machine learning models". Theres is not an unique answer to this commands and two completely different model outputs can be just as valid.

Measuring the effectiveness of these models on the tasks that they perform is not simple, in fact it is an open problem. Nevertheless these models are the ones that have the higher demand and are the ones that are expected to be deployed in the Pocket Network. As we commented before, there is no single metric that can correctly score all the nuances of the generated output. The only way to asses the correctness of one of these models is though multiple step measurements of different kinds of generated outputs. In practice, the common approach is to use benchmarks that include several metrics that observe the different aspects of the generated samples.

Examples of these models are:

- Text-to-Image (create images that follow a given description).
- Language Models (complete an string using the most probable words).
- Text-to-Video.
- Text-to-Audio.

3 The Pocket Network Tech

In this section we will comment on deploying a model in the Pocket Network Morse (V0). We will be focusing on the existing capabilities and what we can do without breaking the blockchain consensus. While we would love to extend on the Pocket Network capabilities in the future, we want to remain as realistic as possible in our first moves.

The Pocket Network is a protocol for decentralized remote procedure call (RPC) infrastructure that relies on crypto-economic incentives to match compute and data suppliers to consumers in a trust-less fashion. The permission-less and decentralized nature of the supply side of the network makes it vulnerable to bad actors that can provide low quality services to the consumer. This fact limits and shapes what we can do in Morse.

3.1 What we CANNOT do on Morse

The HTTP specification do not limit request sizes, but some services used by node runners will limit this to 100 MB, like Cloudflare [13]. So, models that require large payloads will be discarded (like video processing), unless we create an off chain solution like placing results in temporary locations.

Morse does not have the ability of opening streams between the requester and the services. This limits its ability to handle calls like the ones used for chat bots where the text is present to the user *on the fly* or stream video frames for processing.

The Pocket Network do not have code to challenge or validate on-chain the execution of certain model or if the data is actually correct. The quality of service sampling mechanics, like Watchers, will be only available in Shannon upgrade (and not since day zero). This limits what we can do to check models being staked and the plan is to delegate the checking to off-chain actors and gateways. The combination of these agents will filter adversarial nodes and they are expected to get no work (similar to whats done for blockchain nodes).

Distributed inference and/or multi-model inference (like mixing of experts, chain of thoughts, agents, etc) is out of scope for Morse. Morse can serve RPC requests, it does not have the ability to do intelligent routing or provide responses that are combinations of several models responses. Such service (and many more) can only be provided by gateways, which are off-chain entities and as such it falls outside the scope of this document.

3.2 What we CAN do on Morse

We can do inference of arbitrary models, that can be called using simple HTTP requests. This is a very large group of models that includes some of the most known applications like:

- Language Models: Used for summarization, bots, text analysis, etc.
- Image Generation Models: Images of up to 100 MB can be used without problem, current open text-to-image models produce images of 1024×1024 pixels, which can be safely encoded in less than 10MB.
- Image Editing Models: Related to previous point, these models take up to $3\times$ the request size but also are under 100MB.
- Optical Character Recognition: Also limited by payload size, but not too limited. These models are the ones used to recognize and locate text on images.
- Embedding Models: Any kind of model, a common use of these models is to do text embedding for vector databases (used in RAGs).
- Many more probably...

4 Services to Explore and Deploy

From all the models that the Pocket Network can support there are two that are the most known to the public and can provide the best exposure: the language models and the image generation models. These kind of models are the ones that require the largest compute power and hence are more difficult to be run locally. Also other web3 projects have not been able to provide permission-less and decentralized inference on these kinds of models to the date, giving us the first mover's advantage.

A third interesting option can be the text embeddings, which are part of the RAG pipelines, however these are very specific and easy to deploy normally.

4.1 Large Language Models

The LLMs can be deployed in infrastructure with or without graphic processing unitss (GPUs), however the speed of the inference on GPUs is much higher than in CPU (on equivalent hardware costs). An ever increasing list of solutions exisits to run the LLMs, to name a few:

- Llama.cpp: https://github.com/ggerganov/llama.cpp
- Ollama: https://ollama.com/
- GPT4All: https://github.com/nomic-ai/gpt4all
- TGI: https://github.com/huggingface/text-generation-inference
- vLLM: https://github.com/vllm-project/vllm

The efficiency of the model is dependent on your hardware, we provide a solution using vLLM since its license terms allows the commercial use (as opposed to TGI) and that it provides an OpenAI compatible API out of the box.

While the underlying engine (vLLM, Llama.cpp, etc.) is irrelevant for the Pocket Network, the way that the user interact with the model is very important. Being a distributed network, the requester cannot adapt to the node that is providing the service at any time, the protocol must settle on an API specification for all LLM models.

We argue that following OpenAI [1] is the best option, since it is the leading company in the generative AI market to the date [2] and it provides full documentation of its API [6].

4.2 Image Generation Models

The second kind of models that is interesting to see deployed, is the image generation or text-to-image models. These models, just as LLMs, require GPUs for faster model inference, however the options for running these models at scale are limited. During our research we were not able to find an existing project focused on serving inference for these models. The best solution seems to be to default Hugging Face's Diffusers library [4], built on top of PyTorch [17].

Using the Diffusers library gives us great liberty on how to deploy the models but it gives us no API to access the model, something required by the Pocket Network. The API solution that we provide was created ad-hoc and based on the SD API [3] which provided enough details and is a leading company in the text-to-image inference market.

While the provided API is functional and based on SD API, there are important differences. First the API is a work in progress that would need load testing. Also, the implementation differs from the reference API in how the image payload is handled. In the SD API, the images are transferred from and to the client using a temporary link. In our implementation, the image is part of the request payload, where it is encoded as an string. This way, there is no need for off-chain communication between actors.

4.3 Text Embeddings

The text embeddings are models that map text to a n-dimensional vector. The idea is such that phrases with similar semantic meaning are vector represented in nearby regions of space. These models are used to populate vector databases which in turn are part of the popular RAGs systems. As opposed to LLMs and diffusers, running these models does not require GPU for acceptable performance (at low scale), meaning that they are much more

[6] https://github.com/openai/openai-openapi?tab=readme-ov-file

compatible with the Pocket Network's suppliers hardware. Also, the data integrity can be checked using similar checks as blockchains (for example, majority voting).

There is one difficulty with running these models, and it is selecting the model to be whitelisted. The text embedding models are not interchangeable, meaning that if a vector database is populated using a given model, then the model cannot be changed afterwards. This is like a chicken and egg problem, should we stake a model and let demand adapt or wait for demand to ask for a given model ?

Due to this open question we did not provide code to set-up text embeddings, however as soon as we select one the implementation can be done immediately as it wont be harder to process than an LLM call (from protocol perspective).

5 Language Models Services

In this section we will comment on the specific problems of deploying LLMs in the Pocket Network. As we have seen before, the LLMs are generative models and as such they are inherently difficult to measure. On top of this difficulty, there is the popularity of LLMs that leads to an explosion in open models from where to choose and the variations of these models that are released at an unprecedented pace. We will try to explain these difficulties as simple as possible, and lay out some concepts that will guide other iterations of this group.

5.1 Same-Model Variations

The LLMs are popular ML models that are used in chat bots, assistants, community analysis and many other applications. The popularity and impressive performance of these models lead to the creation of several variations of LLMs that are open-source, among the most popular we can find:

- Bloom[7]: An open-access 176B parameters model.
- Llama-2[8]: A family of 12 medium and small size models, ranging from 7B to 70B parameters, developed by Meta.
- Mistral and Mixtral[9]: A family of 5 small models (7B parameters) with capacity comparable to Llama-2 13B or higher (specially the Mixtral models).
- Yi[10]: A family of 14 models with performance higher than Llama-2 (when measured at same model size) and trained on multilingual corpus (great in Chinese).

In this small list, that does not pretend to be a complete representation, we can already count 32 variants of open source models that can be deployed in the Pocket Network, each of them with their specific costs and capabilities. Deploying each of them in their own service ID (formerly known as chain-ID), will require to increase the number of services by approximately $\sim 65\%$, which will impact in the block size as more unique sessions will be created.

However, 32 is not the real cardinality of the open LLM models that can be whitelisted, a more precise number would have to include all variations of each of these models. By *variations* we mean all modifications to the base models that are not fine-tuning. These variations include numerical precision reduction techniques like Round To Nearest, using Float16 or BFloat16 [9], and also the compression of the model's weights, like GPTQ [5], AWQ [14],

[7] https://huggingface.co/bigscience/bloom
[8] https://huggingface.co/meta-llama
[9] https://huggingface.co/mistralai
[10] https://huggingface.co/01-ai

APTQ [8] all in 8 or 4 bits. The models that are processed by these techniques are still in the same quality range as their base models, but are much easier to deploy (in terms of hardware). Only counting these variations we would need to multiply the cardinality by $\times 8$, totaling 256 variations an number that is already $\times 5$ the number of services in the current network. If we then add the fine tuned models (like CodeLlama), the number becomes impossible to follow.

As we have shown, having a single service ID for each unique LLM model is not practical, the number of nodes per service will become too sparse and the number of applications required to consume relays will also be too large. In conclusion, the supply side will be sparse and probably collapse to a few service IDs, but this collapse wont be in terms of model quality, only in supply dominance which means that outside model runners wont be encouraged to join, as there is no reason to comply with the Pocket Network models when you have a running pipeline based on other models.

Even if we decide to ignore all of the previous problems and opt for a "family" based service staking, i.e. using a subset of the 32 model families, we wont be able to enforce it correctly. Suppose that we whitelist a service ID that is called "Llama-2 13B", then we would expect that only this model family will be staked there. So, we might think that we can do a honest majority check for model compliance, sadly this will never work. A node runner can set up a Llama-2 13B on *Float32*, another use *AWQ* compression and a third use a *GPTQ* compression, all three will be in compliance with the service ID, but the honest majority will depend on how many nodes each of them have in a session. Some sessions will penalize the AWQ and GPTQ nodes, while others the Float32 and GPTQ ones. Moreover, a fourth node runner could stake lots of nodes with a Mistral model and the honest majority will be signaling that all Llama-13B models are offending the Llama-13B service ID. Finally if the gateway decides to use a trusted source to compare responses to (like self-deploying a model to check), the only effect that this will have is that instead of restricting the service ID to a given family they will restrict it to a given model specification (numerical precision) that is not publicly known.

In conclusion there is no way to enforce a model without overly restricting the models and losing external model providers in the process.

5.2 The Black Box Problem

As we argue in the previous section, there is no way to practically enforce the model family in the Pocket Network, and hence to us the LLMs are nothing more than *black boxes* language models (LMs). This is a central concept that should guide the development of the LLM offering of the Pocket Network. The LM black boxes should only comply with two things to be staked:

1. Respond to OpenAI API standards.
2. Return generated text and requested metrics such as:
    - tokens (including# prompted, completed and total tokens),
    - tokens log probabilities,
    - tokens' bytes,

Any model that is able to do that should be welcomed in the Pocket Network, regardless if behind it there is a BLOOM model or a real human (make sure he/she types fast enough).

Now the question is how we should divide them in service IDs, such that the Pocket Network is not advertising impossible things and that we are not over- or under-paying model providers. The middle ground between having ≥ 256 service IDs and having a single service ID comes from looking at the problem from an other angle, the model capabilities side.

As we have mentioned before, the number of parameters does not define completely the model capabilities, so using the number of parameters as a driver of reward is not correct. A better approach is to use the perceived quality of the LM. The perceived quality is nothing more than the result of a series of metrics of the model, for example the Language Model Evaluation Harness (LMEH) framework proposed by EleutherAI [6] or the Holistic Evaluation of Language Models (HELM) proposed by Stanford [12]. Both LMEH and HELM can provide a trustworthy measure of the black box ML models. We will leave the details of how to divide the models using this tools for later, in the meantime is enough to say that we can take the average of the different metric scores and set up a series of thresholds. For example we can whitelist the following services [11]:

- **Base Quality LMs** (reward $\times 1$) : Models with an average score below 55.0% (but above 30.0%), includes many known models, like:
    - 54.96% : mistralai/Mistral-7B-Instruct-v0.1
    - 54.91% : meta-llama/Llama-2-13b-chat-hf

[11] Using https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard as reference for values, accessed on 25/03/2024

- **Middle Quality LMs** (reward $\times 4$) : Middle level, many of the best open model are here, such as:
  - 72.62% : mistralai/Mixtral-8x7B-Instruct-v0.1
  - 67.87% : meta-llama/Llama-2-70b-hf
  - 65.32% : 01-ai/Yi-34B-Chat
- **High Quality LMs** (reward $\times 12$) : High end models, above 80% average. Near GPT4 level (that should be $\sim 84\%$).

This approach (as any other in Morse) requires an off-chain element to enforce the model quality. For blockchains, this element is inside the gateways operators node selection logic, however for LMs this element cannot remain hidden. The complexity of measuring these models is high and certain tasks require models that achieve certain thresholds in some of these metrics, so, in order to enable the wider community to consume the Pocket Network LMs, it is imperative to provide basic information about the nature of the nodes.

This is the reason why this socket is working to create an off-chain benchmark tool that can be used to create an open benchmark of the staked nodes with clear references to other offers (such as OpenAI or Gemini APIs).

6 Future Work

In the following moths we will be focusing mostly on technical aspects of the network, specifically preparing everything to enable the correct deployment of LLMs. Our priority is to have part of LMEH [6] working, at least with a single metric. To reach this milestone the following things need to be covered:

- Finalization of the asynchronic evaluation architecture that requires the Pocket Network. Up to this point we have a rough sketch of how it will work, the next report should include a full explanation of its architecture.
- Adaptation of LMEH code to run asyncronically using the proposed architecture.
- Analyze the amount of relays required to cover a test for a single node and the expected refresh rate of the metrics.

On the following months we will also be studying the problem of correctly dividing the LMs in the Pocket Network. We believe that the basic approach of using an average of metrics is not correct and does not provide enough clarity to users that are not versed in the topic. Providing the leaderboard with more meaningful (higher order) metrics will help us understand what the users want to consume and how to guide the on-chain incentives to align to their needs.

Also, there is a very interesting problem of dataset inclusion and adversarial gaming that will begin to develop as we deploy the first models and off-chain metrics. For example, after having LMEH working using a given set of datasets, a node runner can try to game the leaderboard by learning the datasets being used, so we might implement new datasets or HELM procedures. At this point, the decision of which dataset to implement should not be arbitrary, it should be the one that provides best value to the ecosystem. We expect to be playing this game for a while until the system evolves enough such that gaming it is more expensive than just comply, at that point the Pocket Network will become real public good for the ML community.

List of Acronyms

AI	artificial intelligence
GPU	graphic processing units
HELM	Holistic Evaluation of Language Models
LLM	large language model
LM	language model
LMEH	Language Model Evaluation Harness
ML	machine learning
RAG	retrieval augmented generation
RPC	remote procedure call
SD	Stable Diffusion

Reference List

[1] Open AI. *LLM - API Reference*. Accessed 22-03-2024. URL: https://platform.openai.com/docs/api-reference.

[2] IOT Analytics. *The leading generative AI companies*. Accessed 22-03-2024. URL: https://iot-analytics.com/leading-generative-ai-companies/.

[3] Stable Diffusion. *API Reference*. Accessed 22-03-2024. URL: https://stablediffusionapi.com/docs/.

[4] Hugging Face. *Diffusers*. Accessed 22-03-2024. URL: https://huggingface.co/docs/diffusers/index.

[5] Elias Frantar et al. "Gptq: Accurate post-training quantization for generative pre-trained transformers". In: *arXiv preprint arXiv:2210.17323* (2022).

[6] 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.

[7] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[8] Ziyi Guan et al. "APTQ: Attention-aware Post-Training Mixed-Precision Quantization for Large Language Models". In: *arXiv preprint arXiv:2402.14866* (2024).

[9] Dhiraj Kalamkar et al. "A study of BFLOAT16 for deep learning training". In: *arXiv preprint arXiv:1905.12322* (2019).

[10] Diederik Kruijssen et al. *Allora: a Self-Improving, Decentralized Machine Intelligence Network*. 2024. URL: https://litepaper.assets.allora.network/allora-litepaper.pdf.

[11] Huayang Li et al. "A survey on retrieval-augmented text generation". In: *arXiv preprint arXiv:2202.01110* (2022).

[12] 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).

[13] *Limits · Cloudflare Workers docs — developers.cloudflare.com*. Accessed 22-03-2024. URL: https://developers.cloudflare.com/workers/platform/limits/#request-limits.

[14] Ji Lin et al. "Awq: Activation-aware weight quantization for llm compression and acceleration". In: *arXiv preprint arXiv:2306.00978* (2023).

[15] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[16] John F Nash Jr. “Equilibrium points in n-person games”. In: *Proceedings of the national academy of sciences* 36.1 (1950), pp. 48–49.

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

[18] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.