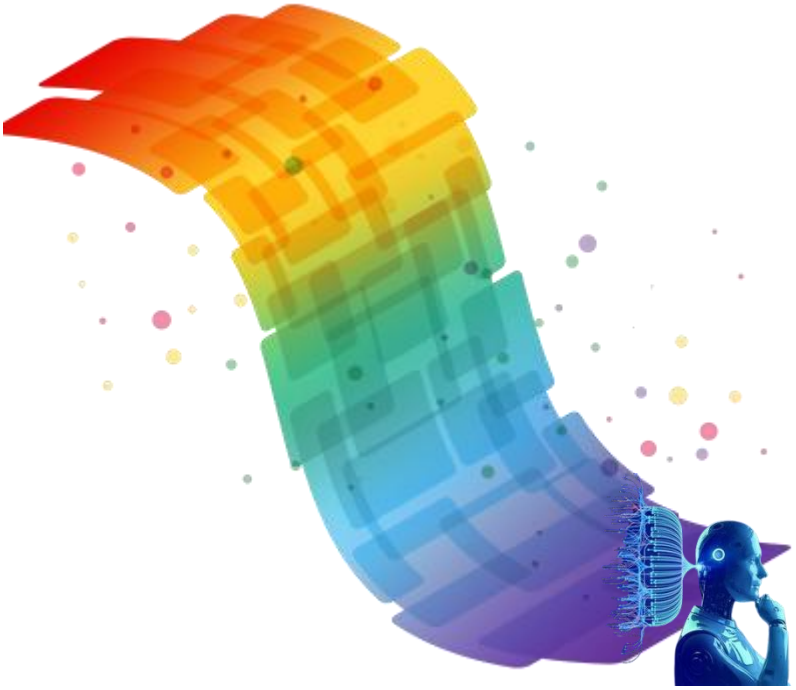

HyperLab Project

First Steps in AI Model Training: a case study

Francesco Carraro

Sergio Fonte

1. Introduction

The *HyperLab* project, funded in 2023, initially aimed at developing a mobile/web application for data analysis related to the *SLab* laboratory data: a laboratory located @ INAF-IAPS in Rome whose main goal is to work systematically on solids as analogues for planetary surfaces. Several instruments can prepare samples by cutting minerals and powdering at different grain sizes and then analyze them with microscopes and stereoscopes.

Over time, *HyperLab* has evolved into a more complex and generic system designed to provide the research community with a versatile tool that goes beyond its initial objectives. This tool is intended to be simple and modern, catering to a broader range of research needs.

The initial phase of the *HyperLab* project focused on creating a user-friendly interface for researchers to analyze and visualize data from the SLab laboratory. This application provided essential features such as data import, processing, and visualization, which significantly improved the efficiency of data analysis tasks.

The *HyperLab* team then decided to expand the scope of the project to develop a more robust and flexible system. The new version of the application incorporates advanced data analysis techniques and a couple of machine learning algorithms.

The *HyperLab* system is designed to be intuitive and accessible, ensuring that researchers can easily adopt and utilize its features without requiring extensive training. Its modern architecture allows for seamless updates and scalability, making it adaptable to future advancements in research technology. By providing a generic tool that addresses the diverse needs of the research community, *HyperLab* aims to foster collaboration and innovation, ultimately contributing to the advancement of scientific knowledge.

N.B.

Although this technical note uses a very linear structure to explain the reasoning, the definition of the problems to be addressed, and the steps followed to achieve the result, the practical experience was much less linear and cumbersome. The lack of experience and knowledge in the field of training and local use of AI models initially led to exploring a series of paths that turned out to be wrong and mixing useful experiments with useless ones before putting the pieces together correctly and arriving at a linear and fruitful path.

2. Study Phase and Definition of a Use Case for Implementation

The objective of this project is to take full advantage of the powerful AI models developed by leading technology companies to significantly accelerate the creation of a functional product from scratch. The team's small size makes it imperative to prioritize efficiency and pragmatism, avoiding the pitfalls of an overly academic approach that might entail a prolonged phase of theoretical study before transitioning to practical implementation. Modern AI models, crafted by these major tech players, offer exceptional learning and computational capabilities, the result of intensive research and development by highly specialized teams of engineers. These experts continuously push the boundaries of what is possible in AI, producing models that are not only increasingly sophisticated but also optimized for a wide range of applications.

Failing to leverage their groundbreaking work would not only result in a substantial waste of time and resources but would also limit the team's ability to capitalize on the cutting-edge advancements already made available. Instead, by integrating these pre-built models, the team can focus its efforts on adapting and customizing the solutions to meet specific project requirements, thereby speeding up the development cycle while maintaining high quality. This approach enables small teams to effectively compete in the AI space, tapping into the collective expertise of some of the world's most advanced tech organizations to achieve ambitious goals in a fraction of the time

2.1 Identifying challenges

The identified problems are as follows:

1. Where and how to find pre-trained models that can be further trained.
2. Feasibility study on running these models, if available, on a local PC (can they only run on large servers?): in this regard, the knowledge acquired from reading articles about *SLM* (*Small Language Models*), which are small in size compared to *LLM* (*Large Language Models*) that require significant computational resources typical of specialized company server farms, has been reused.
3. In the case of available models, which one to choose and by what criteria?
4. Investigation of any development environments prepared for this type of task.

When a technical feasibility is established, a critical question arises: which case study should be selected as the starting point?

Within the scope of the *HyperLab* project, an initial idea involved identifying spectral bands from spectra. However, this approach presents significant challenges related to the scientific aspect of the problem, compounded by a lack of prior expertise in the field.

As a result, it was decided to begin with a case study where the primary difficulty lies solely in the lack of familiarity with the methodologies for training a model, rather than confronting complex scientific barriers.

Staying within the *HyperLab* framework, the proposed focus shifted to a task that involves enabling users to create queries for spectrum searches using natural language. This natural language input would be automatically converted by the model into SQL queries. Such a feature would be particularly valuable for constructing complex queries that require multiple parameters. This approach not only provides a more user-friendly interface but also represents a practical entry point into training AI models, as it concentrates on the technical implementation and the development of model training pipelines without being hindered by domain-specific scientific obstacles.

To face this kind of challenge we decided to look among the LLM models, whose nature is suitable for understanding text input and answering with a text response.

3. How to tackle challenges

After completing the study phase and identifying the key challenges to be addressed, the focus shifted to practical implementation and the effort to acquire the necessary knowledge to outline a clear path forward. The primary approach chosen was to engage directly with the AI tools currently available on the market. These tools have been in use for over a year and a half, both in professional and personal contexts, and have proven to be highly effective on numerous occasions due to the quality of the suggestions they provide.

Based on personal experience, these AI tools have essentially replaced traditional search engine-based research, offering a more intuitive, efficient, and tailored way of finding relevant information. This practical engagement with advanced AI not only accelerates learning but also helps build expertise in leveraging these technologies effectively for real-world applications.

3.1 Found answers

The suggestions that emerged during numerous discussions over the course of several days, following the numbering of the questions in the previous paragraph, are as follows:

1) There exists a global community dedicated to the release and sharing of AI models intended to be made public and almost always available for download. This allows individuals to either use the models directly or perform specific fine-tuning and training tasks. This community is Hugging Face (https://huggingface.co/), a well-known platform where users can search for the AI model they need by leveraging existing categorizations. From there, users can proceed to review detailed documentation and technical specifications to identify the most suitable candidate for their use case.
   However, this process can be exceedingly time-consuming and often frustrating due to the sheer number of models available on the platform. The abundance of options, combined with the need to assess each model's compatibility and potential performance, often makes manual selection a daunting task. In this regard, the assistance of an AI to streamline the selection process has proven invaluable, significantly reducing the time required to pinpoint the appropriate model while ensuring that the chosen solution aligns with the project's specific requirements. This approach highlights the practical application of AI even in facilitating the adoption of other AI tools, emphasizing its utility in accelerating workflows and optimizing decision-making.
2) AI models can also be utilized on local computers, provided that the hardware requirements are compatible. Unlike LLMs (Large Language Models), which often have substantial hardware demands that are beyond the reach of most users, SLMs (Small Language Models) are more accessible and can be deployed on commonly available machines, albeit those of mid to high performance.
   There are dedicated tools, such as LM Studio (https://lmstudio.ai/), specifically designed to allow users to download a model locally and start using it immediately in chat mode. These tools often offer the option to define a context that the model can use as a reference for generating responses, making interactions more relevant and tailored to specific use cases.

A similar functionality can also be achieved programmatically by using the corresponding Python library, bypassing the graphical interface altogether. This approach provides greater flexibility for developers and technical users, enabling deeper integration and customization of AI capabilities within their workflows.

3) Once it was determined that it is feasible to run a model locally on a PC, the focus shifted to selecting the most suitable model to carry out the intended training and usage objectives. Specifically, this involves training the model to generate SQL queries from natural language sentences. The recommended category is that of *transformer* models, which are specifically designed for tasks involving text understanding, processing, and generation. A brief explanation about *transformers* models is provided in section 3.2) An overview of Transformers models.
4) The global AI community has largely converged on the use of Python as the primary programming language. Over the past several years, Python has seen remarkable growth, becoming a go-to choice due to its simplicity, versatility, and its wealth of pre-built libraries. These libraries are specifically designed to facilitate the use of AI models, both during the training phase and in production environments. This made Python an almost inevitable choice for many practitioners and organizations.
   However, Python is not without its challenges:
   a. **Compatibility between Python 2 and Python 3**: Introduced in 2008, Python 3 is not backward compatible with Python 2, which created significant hurdles for projects that needed to transition. Although Python 2 was officially deprecated in 2020, numerous legacy projects continue to rely on it.
   b. **Support for older versions**: Older Python versions, such as 3.6 and earlier, no longer receive security updates or bug fixes. This lack of support introduces potential risks for projects that have not been updated to newer versions.
   c. **Performance**: While recent versions of Python, such as Python 3.12, have improved performance over earlier releases, some legacy applications may still face slower execution speeds if not optimized properly.

Despite these challenges, Python remains the dominant choice due to its extensive ecosystem, ease of use, and adaptability to a wide range of AI tasks, making it an indispensable tool for professionals in the field.

3.2 An overview of Transformers models

Generative artificial intelligence models based on transformers represent one of the most advanced and groundbreaking technologies in the field of AI. These models are particularly prominent in tasks such as natural language processing (NLP), machine translation, text generation, and even computer vision. Their versatility and ability to handle diverse applications have made them an integral part of modern AI development.

How Do Transformers Work? Transformer models are built on an architecture introduced in the seminal 2017 paper *"Attention Is All You Need."* This innovative design incorporates key concepts that set transformers apart from earlier neural network architectures:

1. **Attention Mechanism**: At the core of transformer architecture lies the attention mechanism, which allows the model to focus on the most relevant parts of an input while processing information. For example, in a sentence, the model can identify and emphasize key words or phrases that carry the most meaning. The most widely used version of this mechanism is *self-attention*, which enables the model to examine the input as a whole and understand the relationships between various components, enhancing its contextual understanding.

2. **Modular Block Structure**: Transformers are composed of a series of modular, repeatable blocks. Each block contains two essential components:

    - An attention layer, which applies the self-attention mechanism to prioritize important information.

    - Fully connected neural networks (feed-forward networks), which process and transform the information further. This modular design makes transformers highly scalable and adaptable, allowing them to handle large datasets and complex tasks efficiently.

    By combining these elements, transformers have revolutionized the way AI models process and generate information, setting a new standard for performance and accuracy across a wide range of applications

3. **Encoder and Decoder** types:
    - The encoder processes the initial input (e.g., a sentence in one language).
    - The decoder generates the output (e.g., the same sentence translated into another language).

    Some models, such as BERT, utilize only the encoder, focusing on tasks like understanding and analyzing text. Other models, such as GPT, rely solely on the decoder, excelling in tasks like generating coherent and contextually appropriate text. This division highlights the flexibility of transformer-based architectures, enabling them to be tailored for either comprehension-focused tasks or generative ones, depending on the specific application requirements.

4. **Pre-trained and Fine-tuned Models**: Transformer models undergo a two-step process to maximize their effectiveness. First, they are pre-trained on enormous datasets to build a foundational understanding of language patterns, semantics, and context. This broad training phase equips the model with general linguistic capabilities. Following this, the model is fine-tuned using smaller, task-specific datasets to specialize in particular applications. For example, GPT (Generative Pre-trained Transformer) is fine-tuned to excel in text generation tasks, while BERT (Bidirectional Encoder Representations from Transformers) is specifically tailored for tasks involving text comprehension, such as question answering and language understanding.
    Practical Applications: Transformers power a wide range of technologies that we interact with daily, including:

a. **Virtual Assistants**: Used in applications like Alexa, Siri, and Google Assistant, enabling efficient and natural user interactions.
b. **Machine Translation**: Supporting automatic translation across languages with enhanced fluency and accuracy.
c. **Creative Content Generation**: Producing original content such as poetry, articles, and stories, often indistinguishable from human-written material.
d. **Sentiment Analysis**: Assessing emotional tone and opinions in social media posts, reviews, and other textual data to identify public sentiment.
e. **Chatbots and Semantic Search**: Improving conversational agents and search engines to understand user queries deeply and deliver contextually accurate responses.

Through their ability to process and generate high-quality text and other data, transformer models have become indispensable tools in modern AI, enabling advancements across numerous domains.

4. Implementing the Model Training Process

4.1 Model selection

The selected model, from among those recommended, is Google's *T5-small*. Both the original LLM and this derived SLM are designed to address tasks related to text comprehension and transformation, making them ideal for the intended applications. The decision to use this particular model was primarily driven by its low resource requirements, which are optimal for an initial study phase where computational efficiency is a key consideration.

The *T5-small* model is a scaled-down version of *T5-base*, containing approximately 60 million parameters, significantly fewer than its larger counterpart. This reduction in parameters translates to lower demands on hardware resources, allowing the model to run efficiently even on mid-range machines. Such characteristics make *T5-small* particularly suitable for environments where access to high-performance computing infrastructure is limited, while still retaining sufficient capability for effective experimentation and learning during the study phase.

Requirements:

VRAM Requirements: For inference alone, the model requires approximately 220 MB of VRAM when utilizing precision formats such as float16 or bfloat16. However, during the training phase, the memory demand increases, with around 850 MB of VRAM needed when using the Adam optimizer, a commonly applied method for efficient gradient-based optimization.

RAM Requirements: It is recommended to have at least 8 GB of RAM available to adequately support the model and handle the associated processes. This ensures a smooth operation, particularly when running multiple tasks or working with larger datasets.

GPU Requirements: A mid-range GPU with CUDA support, such as the NVIDIA GTX 1060 or a comparable model, is sufficient for inference tasks. However, for training purposes, which demand higher computational power, a more recent and robust GPU will provide better performance and efficiency, reducing processing time.

CPU Requirements: A modern multi-core CPU is well-suited for preprocessing tasks, including data preparation and model pipeline management. While not as critical as the GPU in terms of performance, a good CPU complements the overall system setup and ensures efficient handling of auxiliary operations.

This hardware configuration strikes a balance between resource efficiency and functionality, making it ideal for lightweight experimentation and smaller-scale projects while ensuring flexibility for more advanced tasks.

4.2 Setting-up for Python

The second step involved setting up the development environment. Given the experimental nature of the project and the desire to avoid installing a multitude of libraries that could unnecessarily burden the workspace, the decision was made to use the well-established technique of containerization.

This approach offers several notable advantages:

- **Portability**: Containers encapsulate the application along with all its dependencies, making it executable on any platform that supports container technology (such as Docker) without requiring complex configuration.
- **Isolation**: Each container operates in a self-contained environment, ensuring that issues arising from one application do not affect others running on the same system.
- **Efficiency**: Containers are lightweight compared to virtual machines, as they share the operating system kernel, resulting in faster performance and reduced resource overhead.
- **Scalability**: Containers enable the rapid deployment and termination of multiple instances of an application, allowing for effective management of load spikes.
- **Consistency**: Containers support the development and testing of applications in environments identical to production, reducing common discrepancies often summarized as "it works on my machine but not in production."

A base installation of Python 3 was included in the created container image, with the understanding that additional libraries would need to be installed later to enable the effective use of the selected model. This modular approach ensures flexibility in adapting the environment as the project evolves, while maintaining a lean and efficient workspace during the initial stages

4.3 Generating Training Data for AI Model Development

The initial query chosen for the first test was designed to be simple, incorporating only a single condition specified by the user, specifically the name of the sample to be retrieved. However, a significant challenge quickly emerged, one that is well-known to those approaching AI not as end-users but as trainers: the issue of data generation for training. To ensure that the model operates correctly and achieves satisfactory performance, a substantial amount of data is required during the training phase, allowing the model to learn effectively and adapt to the assigned task.

The specific challenge in this case was to generate a large dataset of sentences, each paired with a corresponding SQL query. This implementation involved several critical tasks: generating a substantial absolute number of unique sentences, ensuring that each sentence referred to a different sample name, and crafting diverse variations of the same natural language request. The goal was to enable the model to interpret various phrasings of user input without requiring users to stick to rigid, pre-defined sentence structures. Such rigidity would undermine the usability of natural language search as a tool.

From the perspective of SQL query generation, the task was relatively straightforward: each sentence was always associated with the same query structure, with only the sample name dynamically updated to match the one mentioned in the corresponding sentence.

With the aid of suggestions provided by the AI tools utilized during development, a foundational Python script was created. This script, starting from just a small set of base sentences, automatically generated 10,000 unique sentences, each featuring a distinct sample name. For every sentence, it

also produced a correctly formatted SQL query, incorporating the appropriate sample name. This automated approach not only accelerated the data generation process but also ensured a diverse and robust training dataset, critical for equipping the model to handle a wide variety of real-world user inputs effectively.

Subsequently, with the support of AI tools, the learning process was structured into several key steps:

- **Reading Input Data**: Each sentence and its corresponding query were meticulously read from the input file, ensuring accurate parsing and preparation of the dataset for subsequent processing.
- **Token Creation**: Tokens were generated to serve as input to the model. Tokens are fundamental units utilized by AI models to process text. During the training of an AI model, input text is divided into tokens, which can be words, subwords, or individual characters, depending on the model's design. These tokens are then transformed into numerical representations (vectors) that the model processes. Tokenizing text facilitates AI in comprehending human language more effectively by structuring data in a format suitable for computation.
- **Model Training and Metric Definition**: The tokens were applied to the model during the actual training process. Specific metrics were employed to evaluate the quality of the training and measure the model's performance. Metrics play a crucial role in assessing the model's learning progress and identifying areas requiring improvement.
- **Creation of Tensors for Model Training**: Tensors were generated as a vital step in preparing data for the model. Tensors, which are multidimensional arrays, serve as primary data structures for AI models, allowing efficient processing of numerical inputs. These tensors represent the input data in a structured and scalable format, enabling the model to perform computations and learn effectively from the training dataset.
- **Execution of the Training Process**: The training phase commenced once the tensors were prepared. This involved applying the data to the model in successive iterations, whereby the model adjusted its internal parameters to minimize errors and enhance performance. The selected metrics guided the training, continuously evaluating the model's progress and ensuring alignment with desired outcomes. This iterative process enabled the model to progressively improve its ability to understand and generate SQL queries based on natural language inputs, leading to a robust and reliable end product.

4.4 Testing the training process

The final step involved testing the model to validate whether providing it with a natural language sentence containing a request to search for data associated with a sample of a specific name would produce an SQL query with the correct sample name as output.

Initially, this step did not deliver the expected results. At first, only 100 sentences, each paired with its corresponding SQL query, were provided as input. Since the query structure remained the same except for the sample name, it was erroneously assumed that a small number of sentence-query

pairs would suffice. The result of this initial test, however, was unexpected—the input sentence was translated into German(!!!) instead of generating the desired query.

Following this, the approach was adjusted to generate a dataset of 10,000 sentences, as described earlier. This adjustment led to the correct generation of the SQL query during testing.

To further evaluate the effectiveness of the training, the model was also tested with an input sentence that differed from those provided during the training process. The results were equally successful, with the model correctly generating the query, demonstrating its ability to generalize and handle variations in natural language input effectively.