Programming Language

Based on development considerations, the most suitable foundation for the system seems to be the Python programming language. Python offers a wide range of libraries, tools, and frameworks specifically designed to support machine learning and large-data applications. Its extensive work field makes it a top option to handle the complex requirements of the system, particularly in managing and processing large volumes of variable data.

Even more, Python's large adoption in both academic and industrial projects contributes to its accessibility. Though might be considered a slower processing compared to other programming languages, it's a nice to-go option since it is known for its ease of use, making it approachable for experimentation in this iterative development. This allows us to prototype, test, and refine the system implementation efficiently while ensuring the system aligns with the overall objective and technical constraints.

Frameworks and Tools

First of all, **Pandas** is a well-known and widely used library that offers the flexibility and functionality necessary for efficient data preprocessing and ingestion. Its intuitive and data structures make it easier to handle large and variable datasets. Furthermore, Pandas integrates with key machine learning frameworks like TensorFlow and PyTorch, making it a reliable and practical choice for system implementation. Handling very large variable datasets may initially seem

challenging, but this obstacle is manageable thanks to the support of complementary libraries like **NumPy** or some other mentioned later, which establish a solid foundation for numerical computations and data manipulation.

Moving forward into machine learning frameworks and pipeline development, **scikit-learn** stands out due to its reputation for high performance in building, evaluating, and training machine learning models. For more advanced experimentation with model tuning, frameworks like **PyTorch**, combined with parameter optimization libraries such as **Optuna**, provide a powerful and flexible environment. Since the system involves processing extensive variable data, the ability to customize parameter settings is crucial for achieving refined model optimization. These tools also support concepts related to uncertainty quantification, chaos and entropy, which can enhance the robustness of machine learning models when dealing with complex, variable data.

Given the need for large-scale and batch inference, it is needed to explore tools capable of intensive training and distributed data processing. In this context, specialized frameworks such as **PySpark** offer strong capabilities for handling big data workflows, including efficient querying, ingestion, and transformation. This makes PySpark an optimal addition to the toolkit, enabling scalable machine learning pipelines that can leverage diverse data sources referenced in the original project specifications.

In addition to environment management and experiment tracking, tools like **PostgreSQL** for database management and **MLflow** for experiment tracking provide essential functionalities to monitor model performance and maintain version control. These platforms allow for the definition of visual metrics, detection of performance degradation, and systematic recording of model versions, which together support transparency and reproducibility throughout the development lifecycle.

Finally, to facilitate result visualization and decision analysis, front-end technologies such as **React** combined with interactive visualization libraries like **Plotly** offer effective solutions. These tools enable the creation of dynamic and user-friendly interfaces to present insights, which might be put under further discussion later at the end of implementation if detailed and visual-ease results are required, on contrary possible discard.

Considerations on possible upcoming implementation

Our system architecture is designed using a layered, pipeline-oriented approach, where each technology serves a clearly defined role and interacts with others through structured stages.

For data preparation, we rely on Pandas and NumPy within a Pipeline pattern, where each processing stage is implemented as a deterministic, composable step. This ensures reproducibility across experiments and makes it straightforward to replay the exact same processing chain during cycles. Knowing the dataset size

might be large, we propose switching to PySpark using the Strategy pattern. This allows us to maintain the same processing flow while substituting the underlying execution engine to support distributed processing and scalability, without requiring changes to the logic itself.

When it comes to model training, a combination of scikit-learn, PyTorch, and Optuna, structured with possible implementation of the Factory and Builder patterns allowing different algorithms to be plugged in interchangeably. Parameter tuning is handled by Optuna, and resulting models are tracked and versioned as artifacts. We hope this proposal enables easy experimentation, replacement, or extension of models without impacting downstream components.

To maintain full traceability and reproducibility across the ML lifecycle, we use MLflow, which applies the Memento pattern at the system level. MLflow captures and snapshots model parameters and performance metrics, allowing any past system state to be reconstructed when needed.

During distributed training or batch inference, particularly when leveraging PySpark, we found about the Map–Reduce pattern. Data is partitioned into manageable chunks, the same preprocessing and model logic is applied independently to each partition, and the results are subsequently aggregated. This approach might allow us to come over large datasets while preserving consistency with single-machine logic.

For data storage and access, we use PostgreSQL under a Repository pattern. Application logic interacts with the database through abstract, intention-revealing methods, rather than directly querying raw tables. This abstraction provides a clean, stable boundary between logic and persistence, making easier testing and potential backend adjustments.