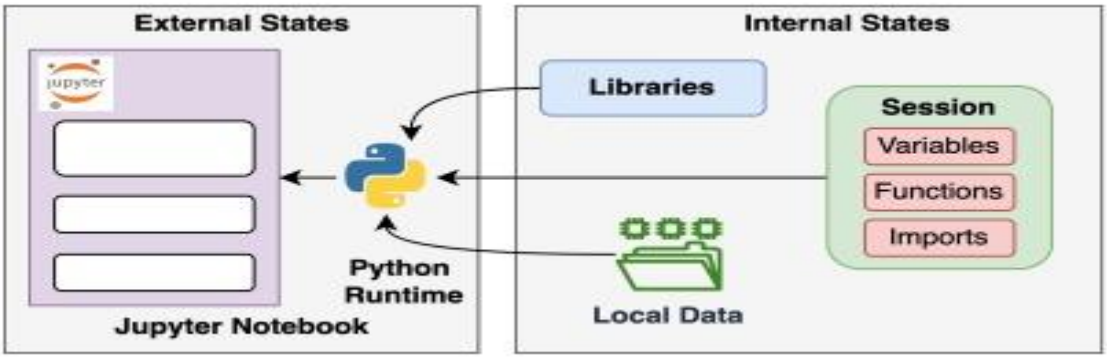
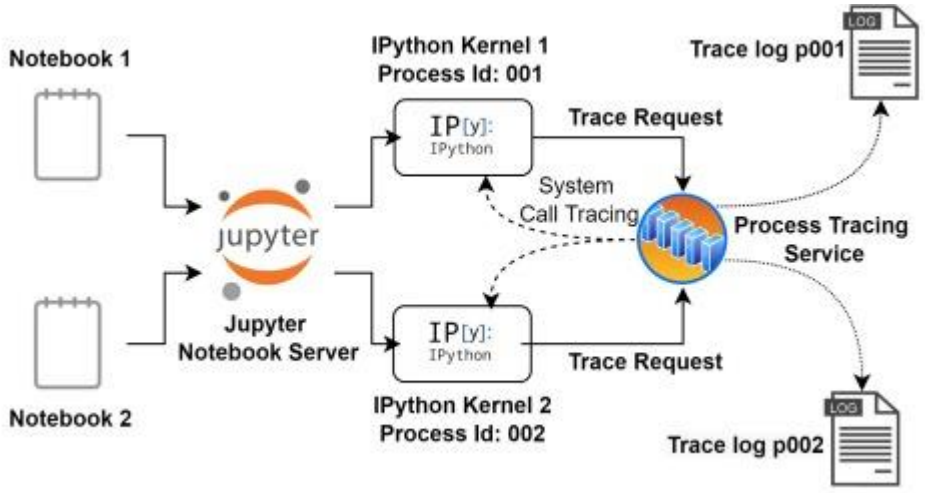
OPEN SOURCE FRAMEWORK

Jupyter Notebooks are a hugely popular tool for creating and narrating computational research projects. They also have enormous potential for producing reproducible scientific research artifacts. Capturing the complete state of a notebook has additional benefits; for example, the notebook execution may be split between local and remote resources, where the latter may have more powerful processing capabilities or store large or access-limited data. When examined in detail, there are several challenges to making notebooks fully reproducible. The notebook code must be completely replicated, as well as the underlying Python runtime environments. More subtle issues arise when replicating referenced data, external library dependencies, and runtime variable states. This paper presents solutions to these problems by utilizing Juptyer's standard extension mechanisms to generate an archivable system state.



Project Jupyter’s notebook format has revolutionized interactive computing and has become ubiquitous among researchers using machine learning techniques and in scientific computing communities. Jupiter’s simple-to-use user interfaces, ease of deployment, rich visualization support, and support for multiple programming languages have nurtured a large user community with a diverse set of use cases, from simple python code execution to complicated
neural network simulations in high-performance computing environments. When code execution and visualization are combined with embedded textual descriptions, the Jupyter computing, and data analysis ecosystem can be viewed as an infrastructure for providing narration for the computing and data life cycle as stories. This storytelling is essential for reproducible science or, even better or re-creatable, or tweakable science.



Jupyter Notebooks are intrinsically sharable, however, some missing components make recreating results in a new context difficult. Five basic elements make restarting a running notebook in a different environment possible:

(1) The code contained within the notebook is transportable.
(2) Python runtimes must be the same.
(3) Recreate the local data that has been referenced.
(4) If external library dependencies are missing, they should be cloned or installed.
(5) Python runtime variables should be reset.
Requirements 1 and 2 are external states that are independent of the notebook's execution. Requirements 3, 4, and 5 are intrinsic to the Python runtime and the order in which the notebook's cells are executed. The replicated environment is not similar to the source environment if any of the above five requirements are not met. We can observe that requirements 1 and 2 are met in most sharing systems, but not requirements 3, 4, and 5.