This section is dedicated to detailing the high-level architecture and design of the project. It will discuss the purpose of each component and the software engineering principles applied to make design choices.

# Framework

The project’s design began with understanding the MLaaS threat model described in §\ref{sec:threatModel}. Figure \ref{fig:abstractNetwork} depicts a high-level layout of the core components based on this framework. The server-side application has been split into two categories: \textit{online} in red and \textit{offline} in blue. Online describes inference being performed directly in response to a request from the client. Offline describes generating inference results on batches of data, independent of the front-end.

* The online components are where most of the discussion of this dissertation occurs. This portion of the application is responsible for emulating the MLaaS model. The \textit{GUI} allows users to select the encryption scheme and inference method used. The user’s video is then passed to the \textit{encryption} component, responsible for encrypting data using the selected scheme – either CKKS or MeKKS in the current implementation. The resulting encrypted data is then passed through the \textit{client} to the \textit{server}. In the \textit{inference} component, the received data is privately analysed, and a video containing only the moving objects is returned to the \textit{client} via the \textit{server}. The \textit{decryption} component must then decrypt the inference results and the video played for the user.
* The offline components are intended for use before the application is deployed. The framework’s \textit{testing} component refers to developing and refining the inference algorithms used to extract moving objects. The \textit{evaluation} component encompasses the process of evaluating the application, including both inference performance and client-server activity.

Figure \ref{fig:abstractInference} provides a deeper insight into the composition of the inference component. The scope of this project only considers the layers above encryption primitives. However, it is important to note that lower layers of abstraction exist. In particular, a layer that may be particularly relevant to the investigation begun by this dissertation is the hardware implementation. Hardware modifications could potentially impact the application's performance considerably, both positively and negatively. For example, accelerators, such as GPUs, could be used to perform cryptographic operations [BADAWI]. Equally, the hardware used in current surveillance implementations may produce weaker results.

# Software Interface

An overview of the project’s repository is given in Figure \ref{fig:filetree}. The project was written to clearly distinguish the layers depicted in Figure \ref{fig:abstraction}. The object-oriented approach to design allowed separate components to be implemented independently. As well as aiding comprehension, this architecture was chosen to minimise interaction across abstraction layers, and make the project straightforward to expand with, for example, more HE schemes or inference methods.

The application can be split into four layers of abstraction, from the high-level interface to the low-level implementation.

* The highest level is the graphical user interface that the user directly interacts with. It allows the user to configure the encryption scheme and inference method used by the server, and upload and receive videos.
* The next layer contains the networking functionality of the application. Managed by the \texttt{connection} files in both the \texttt{client} and \texttt{server} packages, this layer is responsible for passing any data between the client and the server.
* The third layer establishes the API for the cryptographic principles. The HE functionality required by the application is contained within this layer so that the particular scheme in use can be substituted without changes to the above layers.
* The lowest level contains the cryptographic primitives. Contained within the \texttt{lib} folder, the libraries \texttt{Seal-Python} and \texttt{MeKKS} contain the implementations of these primitives so that the preceding levels may use them.

# Class Structure

This section provides a more thorough insight into the project’s structure. Figure \ref{fig:clientUML} details the arrangement of the client-side, Figure \ref{fig:serverUML} contains the classes composing the server-side, and Figure \ref{fig:mekksUML} depicts the structure of the MeKKS library. While there is overlap in these class diagrams, they have been separated into three figures for the sake of clarity.