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}. From this, Figure \ref{fig:abstractNetwork} depicts a high-level layout of the project’s core components. The components have been split into two categories: \textit{online} and \textit{offline}. Online describes inference being performed directly in response to a request from the user. Offline describes generating inference results on batches of data, independent of the front-end.

* The online components responsible for emulating the MLaaS model are where most of the discussion of this dissertation occurs. 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. The result is then passed through the \textit{client} to the \textit{server}. In the \textit{inference} component, the data is 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 during implementation and evaluation. The \textit{testing} component refers to developing and refining the inference algorithms used to extract moving objects. The \textit{evaluation} component encompasses evaluating the application, including both inference and networking.

Figure \ref{fig:abstractInference} provides 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. A layer that may be particularly relevant to this investigation is 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 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 interacts with. It allows the encryption scheme and inference method to be configured, and videos to be uploaded.
* 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 scheme can be easily substituted.
* 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.

# 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 overlaps between diagrams exist, they have been separated into three figures for clarity.