This section will evaluate the implementation from the perspective of practicality in real-world surveillance systems. It will do so through two key aspects: the MLaaS client-server model and the accuracy of inference algorithms.

# Networking

## Data Handling

Before data can be sent across the network, it must be \textit{packed}, and once it is received, it must be \textit{unpacked}. This proved to be a significant bottleneck before transmitting data. During the packing phase, data must be encrypted and serialised before it is transmitted over the network. To make transmission more efficient, a compression stage is added to try and reduce the memory usage of videos. Similarly, in the unpacking phase, data must be decompressed, deserialised, and decrypted to recover the video and inference results.

As described in §\ref{sec:networking}, several methods were investigated to try and reduce this bottleneck. By combining some of these techniques, substantial progress was made in reducing the time the packing and unpacking algorithms took to run. Figure \ref{fig:naivePackingAndUnpackingGraph} provides the running time of a naïve implementation of these algorithms, and Figure \ref{fig:packingAndUnpackingGraph} demonstrates the performance of an optimised implementation. From these charts, Table \ref{tab:packingAndUnpacking} has been derived to highlight the improvement for each category of inference and encryption scheme. Interestingly, the unpacking algorithm can be improved using parallelisation when the CKKS scheme is used, but it will worsen performance when MeKKS is used. This is due to the delays caused by deserialising data that were overcome by implementing directly in Python – although this does make the encryption and decryption functions perform dramatically worse.

While these times may appear slow, it is important to remember that surveillance companies rarely stream all video from a device. Cameras will usually contain multiple sensors to determine when the primary camera should be triggered to conserve battery life. Then, only short clips containing potential events are transferred to the server for inference. Consequently, real-time performance is not required. Although, speed cannot be ignored to ensure users are not notified too late.

## Transmission Times

The other key aspect of the network component of the project is transmitting the data. One fundamental flaw of HE is the memory consumption inflation caused by encrypting data. Consequently, transmission times are much slower than when working with plain video data. In the final application, two main techniques were used to reduce this impact: vectorisation and compression.

For compression, several algorithms were tested – they were compared for both running time and compression ratio – and the algorithm that performed best on CKKS data was selected. The results of these tests are included in Table \ref{tab:compression}. From this, the impact of compressing videos is shown in Figure \ref{fig:compression1}. MeKKS’s more basic implementation means the ciphertext size is constant for all inference methods. Despite this, it is still two orders of magnitude smaller than CKKS data, thanks to the native Python data structures used. Compression is much more significant with CKKS data, almost halving the memory usage for the Gaussian inference, making it a valuable component of the CKKS packing procedure, but largely unnecessary when MeKKS is selected.

Already, this provides good improvements over raw data. However, this can be extended by encrypting rows of video frames as a single ciphertext rather than each pixel distinctly. Figure \ref{fig:compression2} depicts the results of this adaptation.

As a result of the above optimisations, the running times for the client and server are summarised by Figure \ref{fig:clientTimeGraph} and Figure \ref{fig:serverTimeGraph} respectively.

# Inference

Another area of investigation that must be considered when discussing practicality is the performance of inference algorithms. This can be approached from two metrics. Firstly, the \textit{running time} must be considered to evaluate if algorithms will be able to return results in a reasonable amount of time. Secondly, \textit{accuracy} must be analysed to assess the quality of inference results compared to plain inference.

## Running Time

The running time for each inference algorithm varies significantly and is severely impacted by the parameters used to tune the accuracy of each algorithm, as discussed in §\ref{sec:adaptations}. Therefore, for this comparison, the parameters were tuned using the CKKS scheme and kept constant for a fair evaluation when testing the MeKKS scheme. The results are depicted by Figure \ref{fig:inferenceTime}. As expected, the CKKS scheme runs much quicker than the MeKKS scheme due to the more optimised implementation.

## Accuracy

The accuracies of each HE inference algorithm are compared in Figure \ref{fig:accuracy}. While the HE implementations do not exactly match inference in unencrypted data, they are able to produce nearly the same accuracy, producing almost identical results to the human eye. Perhaps surprisingly, MeKKS produces slightly more accurate results with frame differencing, but this is likely due to the random noise induced by HE encryption.

The Moving-MNIST dataset allows accuracy to be easily calculated because it only contains white moving objects on a black background. Therefore, the similarity between the inference result and the original video can be determined by calculating the \textit{sum square difference} according to Equation \ref{eq:sumSquareDiff}.

## Energy Usage

While not a particular focus of this investigation, energy usage is an important factor in determining the practicality of HE in surveillance. This is because the majority of devices being emulated - surveillance cameras or doorbells - are battery-powered. As such, they must be conservative in their energy usage in order to extend battery life as far as possible. Figure \ref{fig:energy} depicts the energy usage of each combination of encryption scheme and inference method for the client and server modules. The absolute value cannot be isolated from background energy usage, so values have been normalised to allow for comparison on a log scale.