AKI Detection System Design Document

Objective

In this project, we aim to deploy an acute kidney injury (AKI) detection system which deals with real-time HL7 messages sent from hospital systems such as the patient administration system (PAS) and the laboratory information management system (LIMS) via the MLLP protocol. The system alerts the clinical response team via the pager management system if the AKI detection system predicts AKI.

Motivation

This machine learning system aims to integrate multiple hospital systems relating to AKI, enabling real-time monitoring and early detection of AKI. The goal is to reduce the time taken for patients to receive critical medical care in the event of AKI, improving the effectiveness of medical treatment and the efficiency of hospital systems.

Goals

- Deploy an AKI predictive model in a real-time hospital environment.
- Receive HL7/MLLP messages from PAS and LIMS, making AKI predictions before paging the pager management system in event of positive predictions.
- Achieve <3 seconds of latency between receiving a blood test result and paging the clinical response team if the AKI has been predicted.
- Throughput of up to 140 messages in a day.
- System stop/start functionality.
- Database persistence, to preserve patient information in event of interruptions.

Non goals

- Record model prediction history for patients.
- Explain the AKI prediction.
- State model confidence in its AKI prediction.
- Notify the medical team in the event of negative AKI classification.
- Preserve patient personal information.
- Provide an interface for medical practitioners to retrieve historic blood test results.
- Provide medical recommendations based on AKI prediction.

Outline solution

Our solution involves a system which listens to real-time HL7 messages from the PAS and LIMS. The system extracts patient data relevant to AKI predictions with threading, stores or updates patient data in a database, and pulls information from the database for model inference when a new blood test for a patient has been received. If AKI is detected, the system sends a post request to the pager management system to notify the medical team of AKI.

Background

AKI (Acute Kidney Injury) is a serious medical condition, and early intervention is necessary. South Riverside Hospital would like to deploy a predictive model for acute kidney injury. Hence, multiple hospital systems can be integrated and real-time monitoring becomes possible.

Currently, there are three existing systems relevant to this project. The first one is the patient administration system, or PAS, which takes patients as the primary objects and a unique id will be assigned to each patient. The second is laboratory information management system, or LIMS, which records the results of diagnostic tests. Here, we focus only on creatinine tests. For both systems, messages are sent in HL7 format, wrapped in the MLLP protocol. The third system is the pager management system, which is responsible for notifying the clinical response team. When the AKI detection system identifies a case of acute kidney injury, it should send an alert to the clinical response team via HTTP request within a 3-second limit.

This system we design will integrate the hospital's existing infrastructure and ultimately improve the efficiency of South Riverside Hospital's AKI treatment workflows.

Design overview

Database System

The data coming from the PAS and LIMS systems will be stored in a MongoDB database, which is NoSQL and document-oriented. MongoDB stores data as documents within collections. Each document is a flexible, JSON-like data structure which can store complex and hierarchical data.

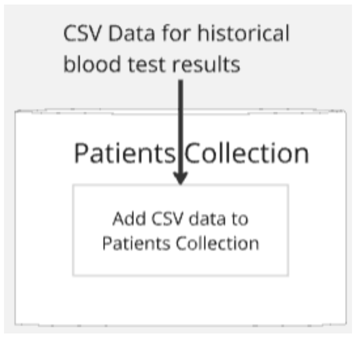
Structure of documents

Table 1: Table showing the key-value pairs in each document within the Patients Collection.

Name of key in document	**What is the meaning of the value?**	**MongoDB datatype of value**	**Example value**
MRN	Medical Record Number (Unique patient identifier)	String	"478237423"
DOB	Patient's date of birth	ISODate	ISODate("1984-02-03T 00:00:00Z")
SEX	Patient's biological sex	String	"F"
NUM_TESTS	Number of creatinine blood tests previously conducted for this patient	NumberInt	3
MEAN_AKI	Mean of all creatinine test results	Double	101.1
LATEST_AKI	Latest creatinine test result	Double	97.8

Each message from the LIMS provides information about individual test results; this information is preprocessed before being added to the database. This preprocessing means that the database only stores the information necessary for the model to predict whether the patient has acute kidney injury. Each patient's information will be stored in a document within the Patient's Collection. Each patient's document will include the six keys listed in Table 1, which also provides the meaning and datatype of each key's value. The date of birth is not converted to age, as that way, the patient's age does not have to be updated in the database every time their age changes. Information on next of kin, hospital arrival/discharge times, and non-creatinine test results is excluded from the database as it is not needed to predict acute kidney injury.

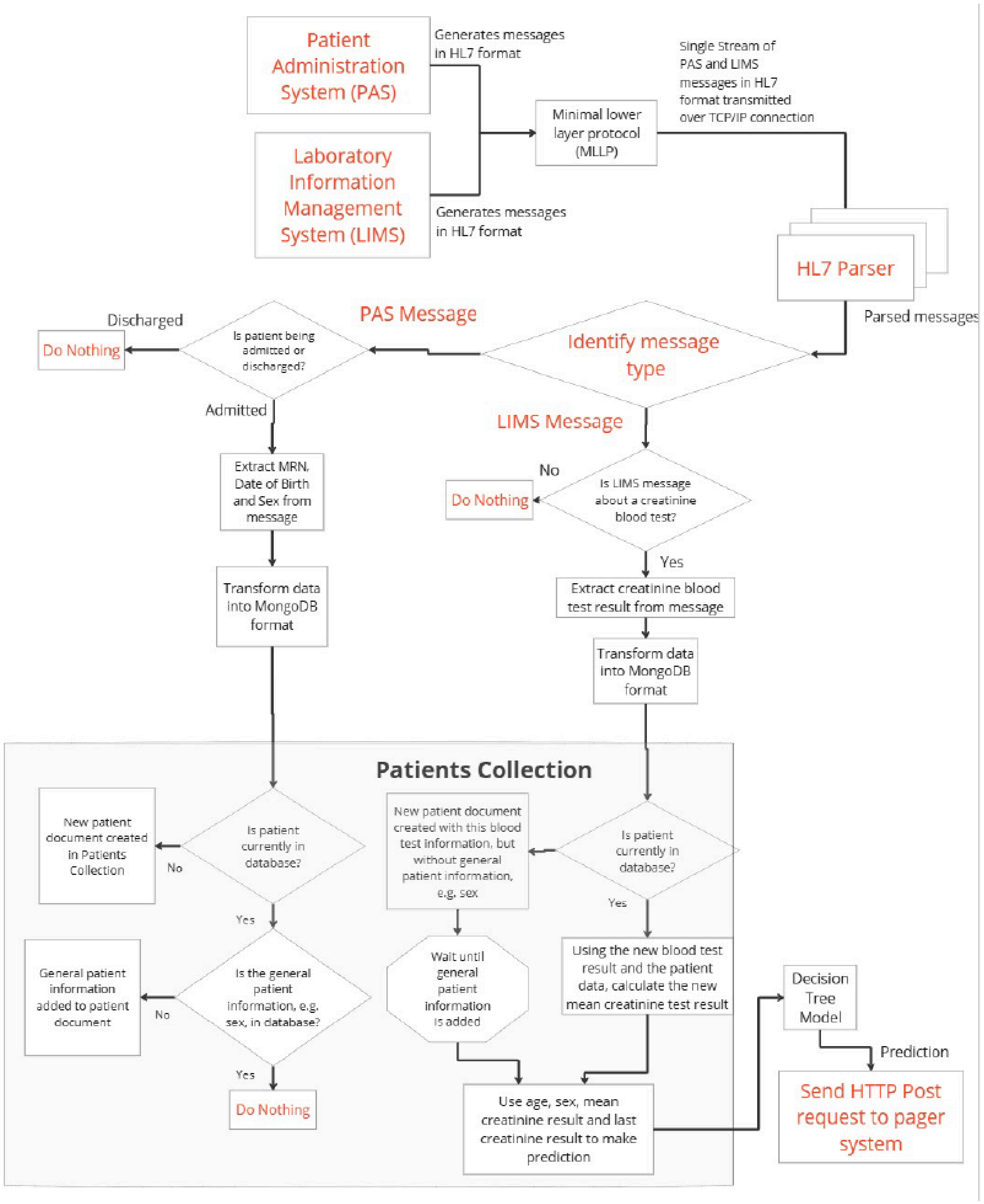
Build Stage


CSV Data for historical blood test results
Patients Collection
Add CSV data to Patients Collection

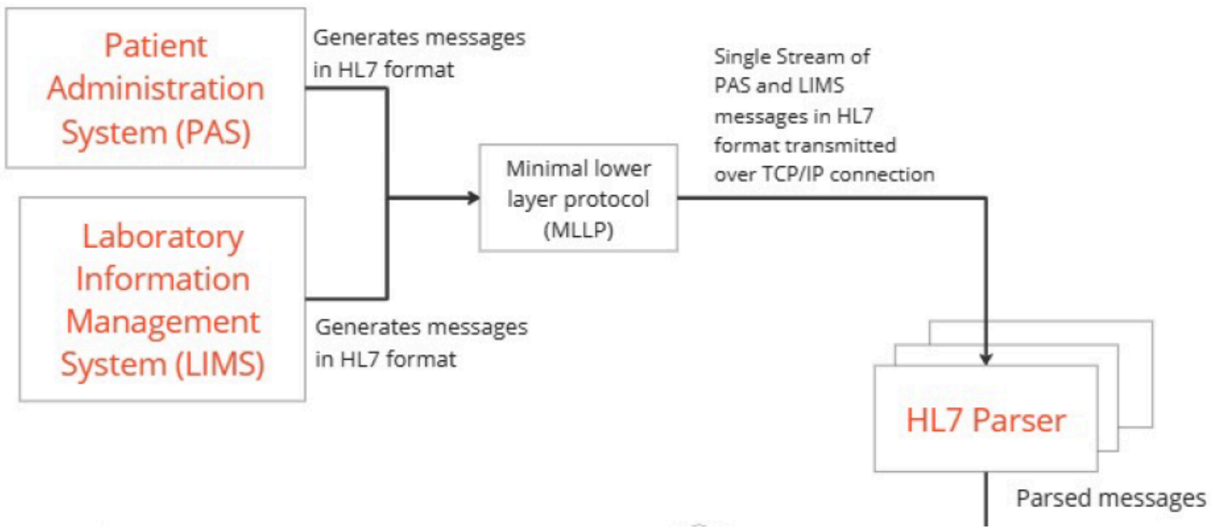
Initially, the CSV data containing the historical blood test results for all patients will be processed and added to the Patients Collection.

Inference Pipeline

Overall Diagram

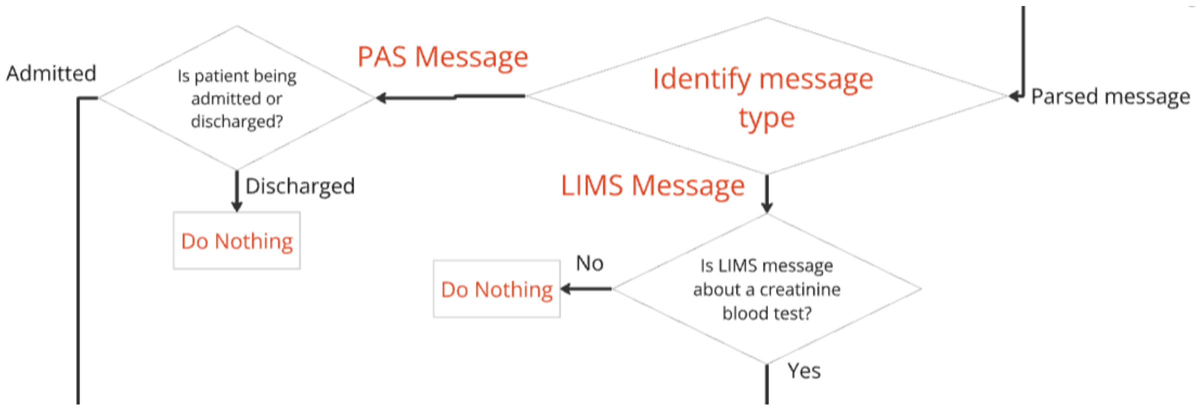

Patient Administration System (PAS)
Generates messages in HL7 format
Laboratory Information Management System (LIMS)
Generates messages in HL7 format
Minimal lower layer protocol (MLLP)
Single Stream of PAS and LIMS messages in HL7 format transmitted over TCP/IP connection
HL7 Parser
Parsed messages
Identify message type
PAS Message
LIMS Message
Is patient being admitted or discharged?
Discharged
Do Nothing
Admitted
Extract MRN, Date of Birth and Sex from message
Transform data into MongoDB format
Is LIMS message about a creatinine blood test?
No
Do Nothing
Yes
Extract creatinine blood test result from message
Transform data into MongoDB format
Patients Collection
Is patient currently in database?
No
New patient document created in Patients Collection
Yes
Is the general patient information, e.g. sex, in database?
No
General patient information added to patient document
Yes
Do Nothing
Is patient currently in database?
New patient document created with this blood test information, but without general patient information, e.g. sex
Wait until general patient information is added
Yes
Using the new blood test result and the patient data, calculate the new mean creatinine test result
Use age, sex, mean creatinine result and last creatinine result to make prediction
Decision Tree Model
Prediction
Send HTTP Post request to pager system

Message Parsing


Patient Administration System (PAS)
Generates messages in HL7 format
Laboratory Information Management System (LIMS)
Generates messages in HL7 format
Minimal lower layer protocol (MLLP)
Single Stream of PAS and LIMS messages in HL7 format transmitted over TCP/IP connection
HL7 Parser
Parsed messages

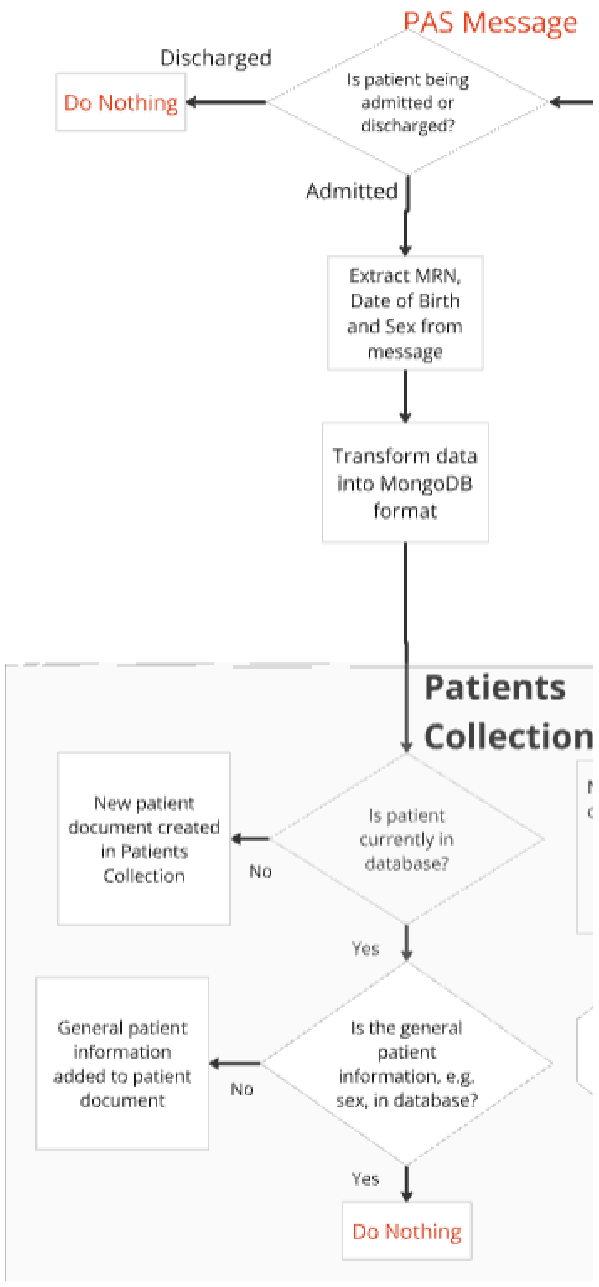
A HL7 parser, e.g. HL7apy, is used to parse the stream of PAS and LIMS messages. Multiple threads are used to parse multiple messages simultaneously and improve latency.

Identifying type of message



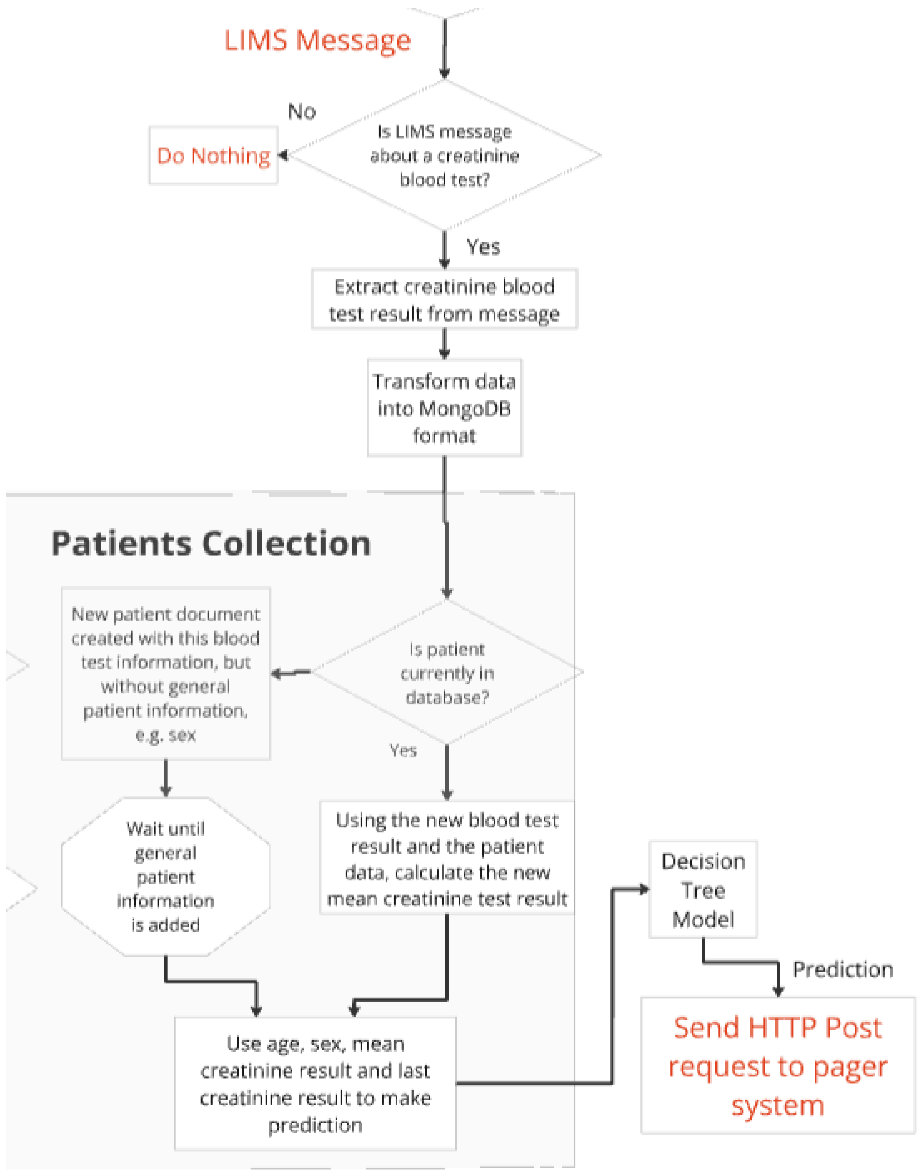
For each parsed message, we identify its type. If it is a PAS message indicating that a new patient is being admitted or it is a LIMS message with new creatinine blood data, then actions are taken (see below). Else if the PAS message indicates the patient is being discharged or the LIMS message contains a different type of test result, then nothing is done. If a patient has been discharged, we do not wish to remove their data from the database in case they are readmitted, in which case information about their previous creatinine levels would be critical.

PAS Messages


PAS Message
Discharged
Do Nothing
Is patient being admitted or discharged?
Admitted
Extract MRN, Date of Birth and Sex from message
Transform data into MongoDB format
Patients Collection
New patient document created in Patients Collection
No
Is patient currently in database?
Yes
General patient information added to patient document
No
Is the general patient information, e.g. sex, in database?
Yes
Do Nothing

If the message is from the PAS, indicating a patient is being admitted, then their MRN, date of birth and sex is extracted from the HL7 message and written to the database. If the patient is not currently in the database, a new document containing their information is added to the Patients Collection. And if they are present in the database, but their general information is not in their document, then this information is added.

LIMS Messages


LIMS Message
No
Do Nothing
Is LIMS message about a creatinine blood test?
Yes
Extract creatinine blood test result from message
Transform data into MongoDB format
Patients Collection
New patient document created with this blood test information, but without general patient information, e.g. sex
Is patient currently in database?
Yes
Wait until general patient information is added
Using the new blood test result and the patient data, calculate the new mean creatinine test result
Decision Tree Model
Prediction
Use age, sex, mean creatinine result and last creatinine result to make prediction
Send HTTP Post request to pager system

If it is a LIMS message with creatinine blood test information, it is identified whether the patient is currently in the database:

- If the patient is not in the database, a new patient document with the blood test result but without the general patient information, e.g. sex, is created. Then, the system waits for the general patient to be added before a prediction regarding the blood test is made.
- If the patient is in the database, a new mean creatinine level is calculated by combining the information about previous blood tests with the new blood test value. Furthermore, the patient's age is calculated from their date of birth. Then this data is passed into the model, which outputs a prediction. This prediction is then sent via a HTTP post request to the hospital's pager system.

Alternatives considered

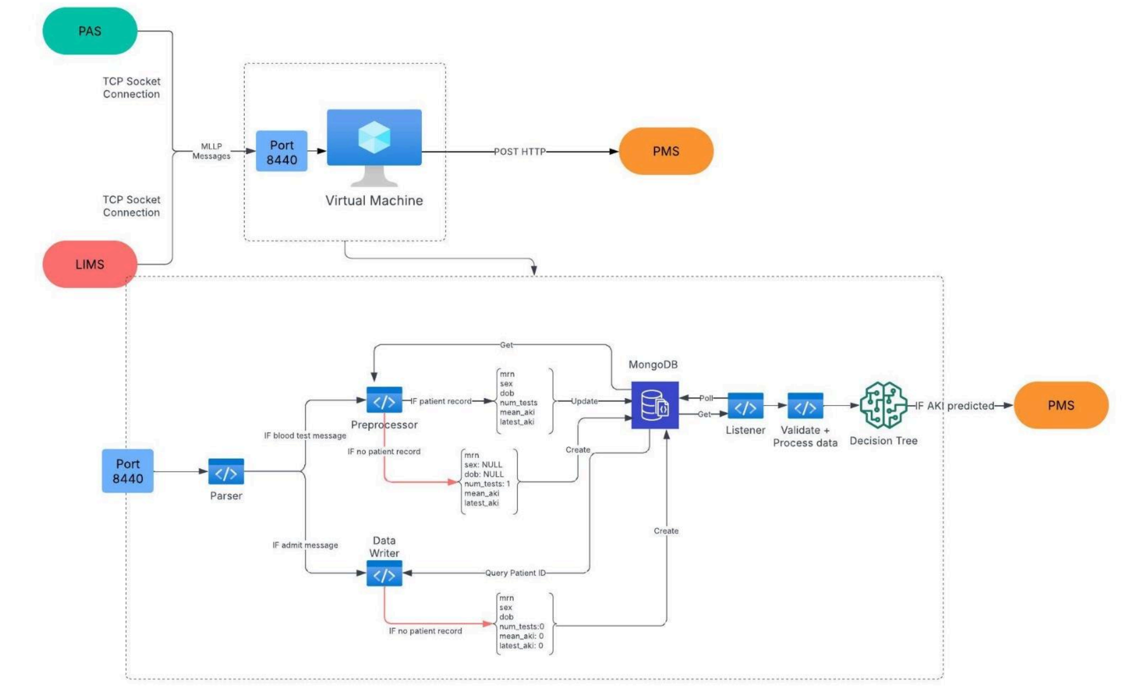
Approach one (rejected)

One considered approach was to utilize a relational database to store processed patient data. While it seemed reasonable at first, we chose instead to use a NoSQL database for a few reasons: Firstly, as the system does not need to compare features across patients, or filter by features, the typical benefits of using a SQL relational database do not apply to our problem. We only need to access one patient's information at a time, and for that, a NoSQL database is sufficient. Secondly, it is more intuitive to store each patient's information separately in its own dictionary, as a patient file. This is precisely the form in which a NoSQL database like MongoDB stores data. Finally, NoSQL databases like MongoDB (when using an indexed lookup) can be extremely fast because they are optimized for direct key-based access, often leveraging in-memory storage. Thus, we chose to use MongoDB as our system's database.

Approach two (rejected)

A second considered approach was to add raw data to the database, and perform preprocessing after a listener detects a new blood test result before feeding it into the model. For instance, in the database, we could simply have an array of historic blood test results for each patient and append a new result to the array whenever a new blood test is received. However, this approach was rejected due to the potential memory and storage constraints of the virtual machine this system is deployed on. It is better to preprocess data before writing it to the database as our current model uses low-dimensional input. The more the system has to handle large amounts of raw data, the more memory and storage usage it requires.

Design details



Our system is implemented on a virtual machine that also acts as an MLLP server to listen to messages from the patient administration system (PAS) and lab information management system (LIMS) on port 8440. Incoming messages are parsed, and discharge messages are ignored. To handle the throughput and latency requirements of the system, we will implement some level of threading for information flow throughout the architecture.

Depending on the message type, a MongoDB NoSQL database is updated accordingly. If the message contains blood test results, preprocessing is performed to update the document corresponding to the patient's MRN. If the message is an admit message, then a new document is created in the database with the patient's key information.

The database contains documents with key-value pairs corresponding mostly to input features to the model. The total number of tests so far is tracked to iteratively update the mean. Note that we do not store age and instead store date of birth, as it is more accurate to compute age at model inference time than patient admission time.

A listener polls for updates to the database, and gets any document which has been updated. A validator checks if all data necessary for the minimal transformation into feature inputs is present. If so, it transforms the data, and feeds the data into the model. Only if the model predicts AKI, is a HTTP post request sent to the pager management system.

In order to accommodate the need to stop and start the system and preserve data, we will set the database's data directory to a persistent disk or volume.

One significant tradeoff of our design is that we do not retain raw blood test results. If the underlying model changes and requires new features, we will have lost historic data that could be used to help predict patient AKI.

Testing

To ensure the smooth running of the system, we have designed the following testing framework, which involves development stage testing (including unit testing, integration testing, performance testing), and post-deployment testing.

1. Unit Testing

These tests are written and run during development.

1.1 HL7 Message Parser

- Tests:
  - Verify correct parsing of PAS messages (Admission & Discharge).
  - Verify correct parsing of LIMS messages (Creatinine blood test results).
  - Handle malformed HL7 messages gracefully (e.g., missing fields, extra delimiters).
  - Validate multi-threaded HL7 processing (ensure concurrent message parsing does not corrupt data).

1.2 Data Preprocessing and Database Operations

- Tests:
  - For the Build Stage, validate that patients' historic data have been successfully added to the database.

1.3 Model Testing

- Tests:
  - Ensure model output matches expected clinical thresholds for AKI detection.

1.4 Pager System Connection

- Tests:
  - Ensure correct HTTP POST request formatting.
  - Simulate network failures to check our system's retry abilities.

2. Integration Testing

Integration tests validate that individual components work together as a whole.

2.1 End-to-End Message Flow

- Simulate a full hospital workflow:
    - Patient Admission → Creatinine Test → AKI Detected → Pager Alert Sent
    - Ensure the entire data pipeline is functioning correctly from input (HL7 messages) to output (Pager API call).

3. Performance Testing

Performance tests ensure that the system meets the hospital's operational constraints.

3.1 Latency Testing

- Send dummy data through the system to ensure that AKI alerts are sent within 3 seconds of receiving a blood test result that yields a positive AKI prediction.
- Measure end-to-end processing time under different message loads.

3.2 Load Testing

- Simulate a spike in HL7 messages (e.g., 300 blood test results within an hour).
- Verify that the system remains stable and processes messages within required latency.

3.3 Scalability Testing

- Test behavior when deployed on a single-core virtual machine with 1GB RAM.
- Test memory behaviour under high volume messages (e.g. 100 messages simultaneously)

4. Post-Deployment Testing

4.1 Automated End-to-End Testing

- Schedule hourly tests to send test HL7 messages and verify that the correct patient alerts are triggered.

4.2 System Health Checks

- Deploy a /health API endpoint to verify database connectivity and that the MLLP server is running

Future work

Model explainability

Model explainability is important as even if a model achieves good performance, healthcare professionals may still be hesitant to use it if its behaviour cannot be analyzed. Furthermore, investigating the behaviour of our model can help identify biases in prediction behaviour.

As we are using a decision tree to make predictions, it is possible to represent the tree as a series of “if-else” rules. These rules can then be interpreted to understand why a particular prediction was made. If the depth of the decision tree is too high, making it difficult to interpret the sequence of decision rules, then the decision tree can be pruned.

Model confidence

Quantifying the confidence in a model prediction is useful because it can influence what next steps are taken. If the model has low confidence in a prediction, it could prompt human guided predictions or for more tests to be performed.

For example, assuming a decision tree is used to make predictions, the confidence in the model prediction is the proportion of the samples in the particular leaf belonging to the predicted class.

Built-in redundancy

We can consider storing raw patient data in a backup database as a means to preserve data if our model needs to be changed and uses different features, or if the existing database information becomes corrupted.