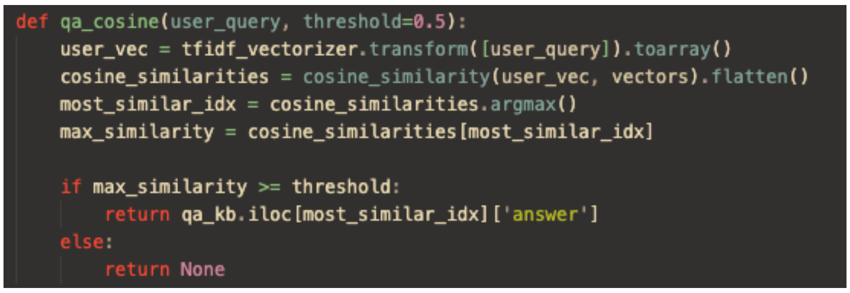
General explanations of the system and its goals	The chatbot can provide answers to the user’s questions on a variety of minerals based upon their characteristics, as well general questions associated with minerals. The system also provides the ability to classify images provided from the user, providing information on what mineral it is likely to be. The user can also interact with the chatbot with the use of a microphone, in which is translated to text. Furthermore, the chatbot can output text-too-speech audio output. If the chatbot does not recognize the user’s input / query, the chatbot will fallback to use of interpreting the user’s input through cosine similarity of the tf/idf values of the sentence in question. The chatbot also uses a knowledge base, in which can be used to determine if a statement provided by the user is correct or false. Two knowledge base files are present, with one being utilised with fuzzy logic & another KB acting as a fallback, with statements with just words / phrases. The fuzzy logic knowledge base associates’ numerical values, in which allows the program to determine the output. For example, if the moss hardness of a mineral is greater than 5 for instance, the chatbot will determine the mineral as ‘hard’. Additionally, the user has the ability to add additional statements to the knowledge base, as long as the statements do not contradict the two corresponding knowledge bases. Image classification is implemented utilising tensorflow & keras respectively in order to identify the user’s provided picture of a mineral, in which the chatbots output provides the name of the mineral with the highest similarity.

The system requirements, i.e., the list of what the system should do/have from a user's perspective	Regarding system requirements, the chatbot is required to answer questions relating to minerals and their associated characteristics, as well as basic queries surrounding minerals such as definitions to characteristics. In order for the user to interact with the chatbot, the user will submit queries utilising text input. Assuming the user's query does not get triggered via the predetermined commands, the chatbot will provide the most suitable reply. Furthermore, the user can interact with the chatbot using their voice, in which is translated to text and inputted to the chatbot as normal. If the user chooses to interact with their voice, the chatbot will provide TTS audio output of the answer. If the text input is a query to classify an image, the chatbot will prompt the user to enter the image's directory. Assuming the directory is valid, the chatbot will attempt to identify the mineral provided by the user.
The employed AI techniques, and the explanation of program codes and the supplied files.	**Natural language processing** The mineral Bot utilises natural language processing methods in order to process the user's questions and match them with the corresponding pre-stored knowledge. Comparing the user's queries with the QA knowledge base, TF-IDF vectorisation is used to convert the knowledge bank & the user's queries into vectors. In order to identify the most appropriate response, cosine similarity is utilised to measure the similarity between the user's question and the QA KB contents. This is achieved via calculating the cosine angle between two vectors, with values closer to 1 indicating higher similarity.



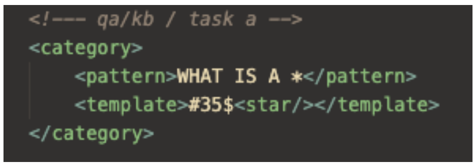
```python
def qa_cosine(user_query, threshold=0.5):
    user_vec = tfidf_vectorizer.transform([user_query]).toarray()
    cosine_similarities = cosine_similarity(user_vec, vectors).flatten()
    most_similar_idx = cosine_similarities.argmax()
    max_similarity = cosine_similarities[most_similar_idx]

    if max_similarity >= threshold:
        return qa_kb.iloc[most_similar_idx]['answer']
    else:
        return None
```

The function **qa_cosine** takes the user's input & the minimum acceptable threshold value. It then vectorises the user's question & compares it to the vectorised QA KB contents. Assuming the threshold value is appropriate, the highest value question is stored in the vector as **most_similar_idx** and the corresponding answer is returned from the vector.

AIML

To appropriately identify and route the user's input to the desired functionality, AIML is used. **mineralbot-logic.xml** stores all the relevant AIML patterns.

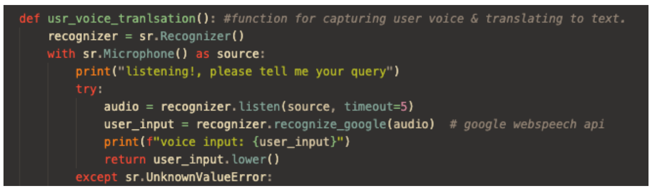


```xml
<!--- qa/kb / task a -->
<category>
    <pattern>WHAT IS A *</pattern>
    <template>#35$<star/></template>
</category>
```

The category above contains the relevant pattern to identify if the user wants to query the chatbot's QA knowledge base, with the * (wildcard) inferring word(s) after the template.

Input & Text-to-Speech

The **SpeechRecognition** library is used to capture & process the user's voice if desired. In order to translate the captured input into text, Google's Web Speech API is used for translation.



```python
def usr_voice_tranlsation(): #function for capturing user voice & translating to text.
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("listening!, please tell me your query")
        try:
            audio = recognizer.listen(source, timeout=5)
            user_input = recognizer.recognize_google(audio)  # google webspeech api
            print(f"voice input: {user_input}")
            return user_input.lower()
        except sr.UnknownValueError:
```

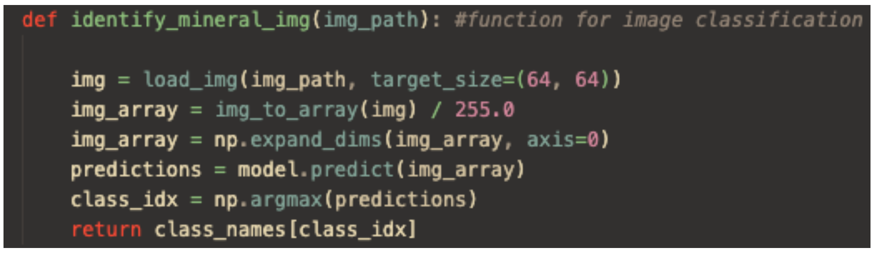
The function **usr_voice_translation** is used to initialise the user's microphone & recognizer & capture the audio feed. The audio feed is then translated back to text via the child function **recognize_google** function / API of the **recognizer**. The corresponding string is returned however, exceptions are caught if the translation through the API is not successful.

Pyttsx3 is used to provide text-to-speech functionality for the chatbot. The engine is initalised upon startup & a simple function (**chatbot_audio**) is used to parse the answer into the child function **say** of the engine to output the response.

Image classification (Hyper parameter tuned)

Image classification utilises a pre-trained **CNN** model in order to identify the user’s provided images. The model in question is **mineral_cnn_tuned.h5,** in which has been trained on a dataset of labelled mineral images (https://www.kaggle.com/datasets/asiedubrempong/minerals-identification-dataset)

TensorFLow/Keras are used to train and load the corresponding model.

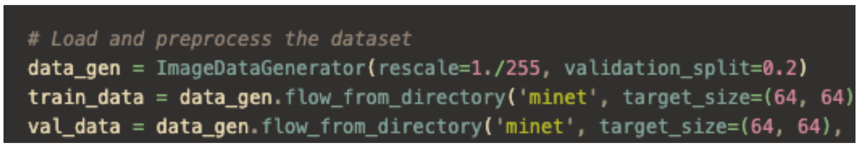


```
def identify_mineral_img(img_path): #function for image classification

    img = load_img(img_path, target_size=(64, 64))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    predictions = model.predict(img_array)
    class_idx = np.argmax(predictions)
    return class_names[class_idx]
```

Function **identify_mineral_img** takes the provided directory path & performs preprocessing such as normalization to prepare the image. The model then predicts the probabilities for each class of mineral. Using a pre-defined dictionary the mineral class is then identified & returned.

cnn_tuned.py contains the relevant code for training / building of the CNN utilised. The dataset used is pre-processed using the **ImageDataGenerator** in order to normalise the pixel values of the associated images. The images are then split into training & validation subsets (80/20 respectively)



```
# Load and preprocess the dataset
data_gen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_data = data_gen.flow_from_directory('minet', target_size=(64, 64)
val_data = data_gen.flow_from_directory('minet', target_size=(64, 64),
```

Keras Tuner is used to identify the best hyperparameters for the tuner. A sequential model is employed, to allow for the CNN architecture to be built within a dynamic manor.

The model uses four key layers, including a **convolutional layer** to extract spatial features from the images, **maxPooling** layers to prevent overfitting through reducing spatial dimensions, **dense layers** to perform classification based on extracted features & finally **dropout layers,** used to prevent overfitting.

In order to ensure the model is trained efficiently with quality validation, **Adam optimizer** & **categorical crossentropy** loss are utilized. These techniques help optimize the training process for ensuring accurate validation.

Since 20 trials are ran, **Best Model Retrieval** (get_best_model()) is used to retrieve the best performing model.

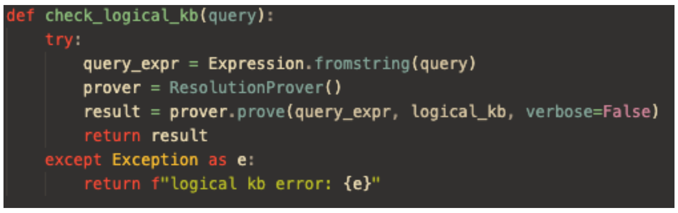
In contrast to the regular CNN, the main employeed difference is the use of the **keras tuner,** in which allows for automated optimization of values such as kernel size & dropout rate, with the expense of increase resource usage & time.

Overall, the tuned CNN generated an accuracy of 91.3%, while the regular CNN’s accuracy was 89.74%. The marginal increase could have possibly been due to overfitting in the tuned CNN caused by the hyperparameter search. This could have been negated with more trials , in which was tried but with no improvement.

Logical / Fuzzy reasoning

The chatbot utilises two logic files **fuzzy-kb.csv** & **logical-kb.csv** respectively for logical & fuzzy logic. This allows the chatbot two handle the user's questions in which require verification. The use of fuzzy logic expands upon the regular logical reasoning by providing degrees of truth / values to categorise characteristics upon value thresholds.

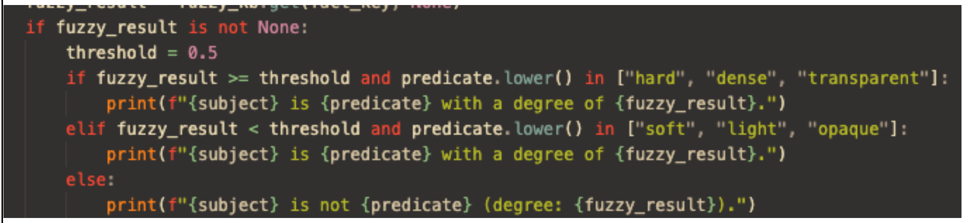
The basic logical reasoning is implemented using the aforementioned NLP techniques combined with a logical knowledge base. To evaluate provided logical statements & verify against the KB, **NLTK Resolution Prover** is used.



```
def check_logical_kb(query):
    try:
        query_expr = Expression.fromstring(query)
        prover = ResolutionProver()
        result = prover.prove(query_expr, logical_kb, verbose=False)
        return result
    except Exception as e:
        return f"logical kb error: {e}"
```

Function **check_logical_kb()** takes the user's input & transforms it into a readable logical expression using NLTK's **Expression** class. To check the statement the statement is true, NTLK's **ResolutionProver** is used in which returns a Boolean which is returned to the main loop.

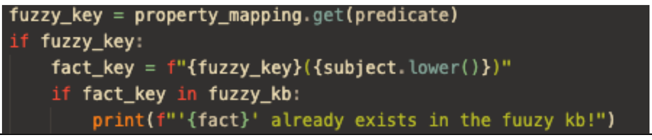
Fuzzy logic provides the ability to handle characteristics that present uncertainty or degrees of truth.



```
if fuzzy_result is not None:
    threshold = 0.5
    if fuzzy_result >= threshold and predicate.lower() in ["hard", "dense", "transparent"]:
        print(f"{subject} is {predicate} with a degree of {fuzzy_result}.")
    elif fuzzy_result < threshold and predicate.lower() in ["soft", "light", "opaque"]:
        print(f"{subject} is {predicate} with a degree of {fuzzy_result}.")
    else:
        print(f"{subject} is not {predicate} (degree: {fuzzy_result}).")
```

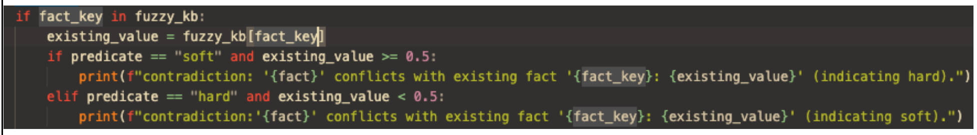
Assuming the user's query exists within the fuzzy KB, the **fuzzy_result** will contain the truth value of the fact (In this case the Mohs hardness). If the **fuzzy_result** is greater than or equal to 0.5, it is determined the mineral is hard.

To allow the user to append to the logical / fuzzy KBs, the NLP also handles ' I know that statements' The query is split into its subject & predicate, with the predicate initially being used to check if the predicate is a fuzzy property or logical.



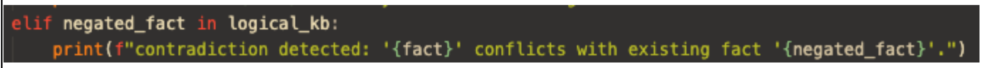
```
fuzzy_key = property_mapping.get(predicate)
if fuzzy_key:
    fact_key = f"{fuzzy_key}({subject.lower()})"
    if fact_key in fuzzy_kb:
        print(f"'{fact}' already exists in the fuuzy kb!")
```

To prevent copies, the **fact_key** is checked to see if the fact already exists within the **fuzzy_kb** or the **logical_kb** if not a fuzzy statement.



```python
if fact_key in fuzzy_kb:
    existing_value = fuzzy_kb[fact_key]
    if predicate == "soft" and existing_value >= 0.5:
        print(f"contradiction: '{fact}' conflicts with existing fact '{fact_key}: {existing_value}' (indicating hard).")
    elif predicate == "hard" and existing_value < 0.5:
        print(f"contradiction:'{fact}' conflicts with existing fact '{fact_key}: {existing_value}' (indicating soft).")
```

In order to prevent contradictions within the **fuzzy_kb,** the key value of the fact_key is checks to see if its value exceeds 0.5. An example of this is saying I know that X mineral is **soft**, but if the **existing_value** is greater than 0.5, it is **hard**.

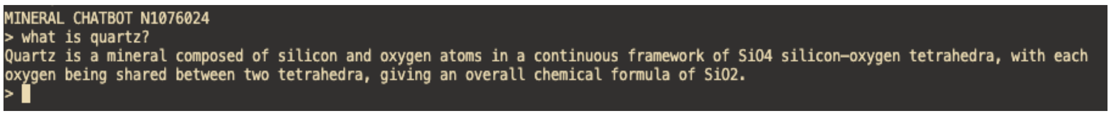


```python
elif negated_fact in logical_kb:
    print(f"contradiction detected: '{fact}' conflicts with existing fact '{negated_fact}'.")
```

To prevent contradictions within the **logical_kb,** the **negated_fact** is used to check if the **opposing fact** of what the user has said is in the KB, if so a contradictory statement is thrown.

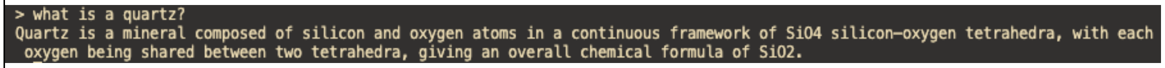
3- Conversation log

Task A -



```
MINERAL CHATBOT N1076024
> what is quartz?
Quartz is a mineral composed of silicon and oxygen atoms in a continuous framework of SiO4 silicon–oxygen tetrahedra, with each
oxygen being shared between two tetrahedra, giving an overall chemical formula of SiO2.
>
```

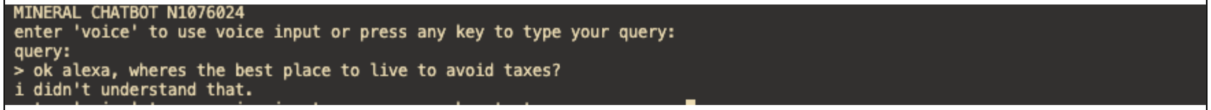
By default AIML functionality is used to determine the user's input, in the case of querying the question answer bank query instances that contain phrases such as 'what is' or 'define' trigger the Q/A functionality. Assuming the query is a direct match to an entry in the Q/A bank, the



```
> what is a quartz?
Quartz is a mineral composed of silicon and oxygen atoms in a continuous framework of SiO4 silicon–oxygen tetrahedra, with each
 oxygen being shared between two tetrahedra, giving an overall chemical formula of SiO2.
```

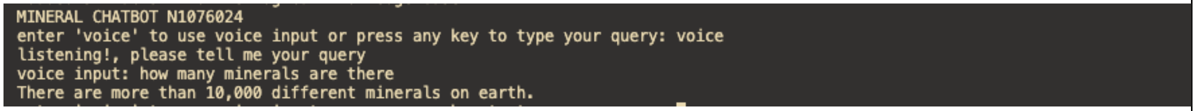
corresponding answer is returned.

If the user were to propose a query in which does not match the contents the question-answer bank, utilising TF/IDF & cosine similarity the chatbot will find the most appropriate answer (assuming the cosine similarity score is appropriate)



```
MINERAL CHATBOT N1076024
enter 'voice' to use voice input or press any key to type your query:
query:
> ok alexa, wheres the best place to live to avoid taxes?
i didn't understand that.
```

If the user's query results in a cosine similarity of zero or does not satisfy any AIML patterns, the default response is returned.

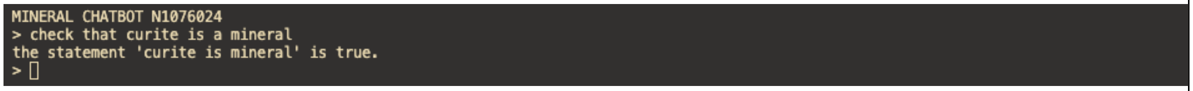


```
MINERAL CHATBOT N1076024
enter 'voice' to use voice input or press any key to type your query: voice
listening!, please tell me your query
voice input: how many minerals are there
There are more than 10,000 different minerals on earth.
```

The user can also utilise a microphone to ask questions, via typing 'voice' to initiate the voice based input.

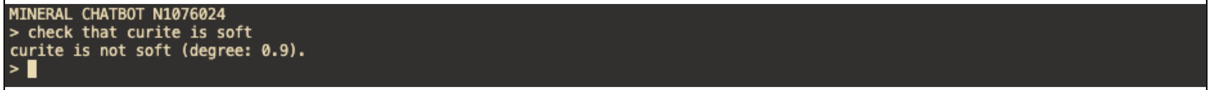
Task B –

The chatbot refers to two knowledges for task b, logical and fuzzy knowledge bases (extension task)



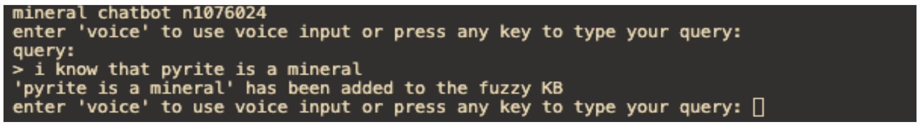
```
MINERAL CHATBOT N1076024
> check that curite is a mineral
the statement 'curite is mineral' is true.
>
```

The user can check to see if a statement is true or false based on the contents of the two KBs. The above example demonstrates the chatbot referring to the logical knowledge base, in which statements with a boolean characteristic are stored.



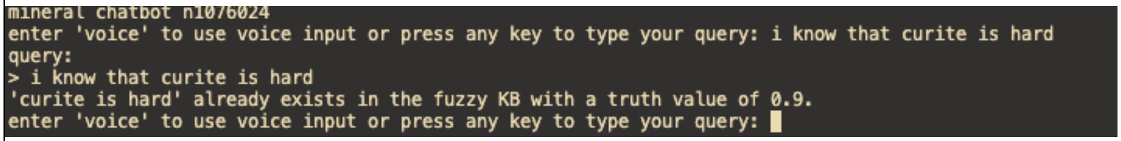
```
MINERAL CHATBOT N1076024
> check that curite is soft
curite is not soft (degree: 0.9).
>
```

If the user's statement coincides with a property in which has a numerical value, the fuzzy knowledge base is consulted, and its corresponding value is checked.



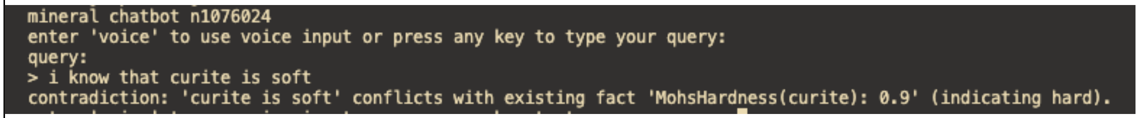
```
mineral chatbot n1076024
enter 'voice' to use voice input or press any key to type your query:
query:
> i know that pyrite is a mineral
'pyrite is a mineral' has been added to the fuzzy KB
enter 'voice' to use voice input or press any key to type your query:
```

The user can also append to the KBs via ‘I know that’ statements.



```
mineral chatbot n1076024
enter 'voice' to use voice input or press any key to type your query: i know that curite is hard
query:
> i know that curite is hard
'curite is hard' already exists in the fuzzy KB with a truth value of 0.9.
enter 'voice' to use voice input or press any key to type your query: █
```

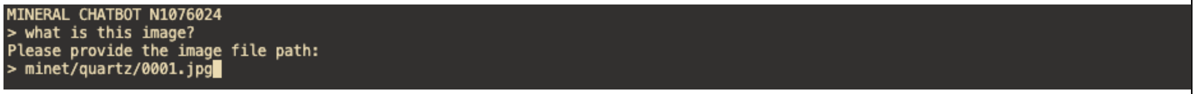
If the fact already exists, its prompted back to the user.



```
 mineral chatbot n1076024
 enter 'voice' to use voice input or press any key to type your query:
 query:
 > i know that curite is soft
 contradiction: 'curite is soft' conflicts with existing fact 'MohsHardness(curite): 0.9' (indicating hard).
```

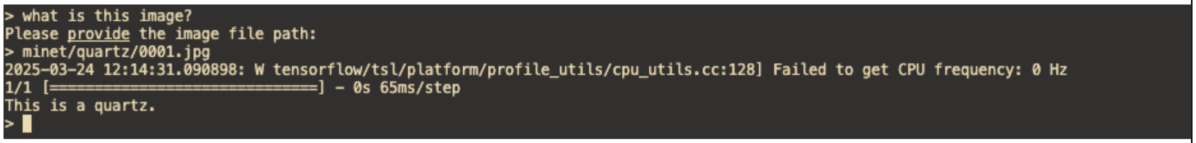
If the fact contradicts the relevant KB, the user is also prompted.

Task C –



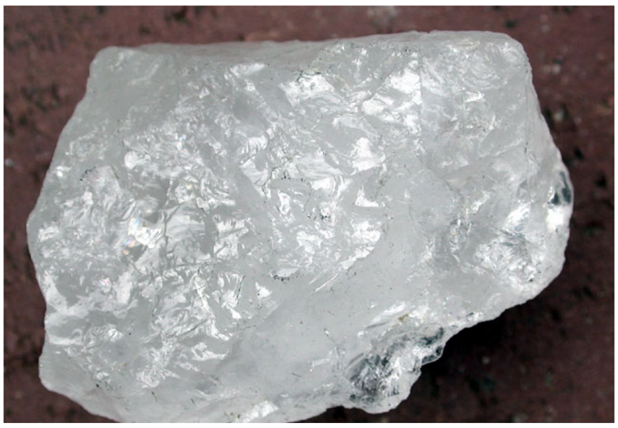
```
MINERAL CHATBOT N1076024
> what is this image?
Please provide the image file path:
> minet/quartz/0001.jpg█
```

If the user inputs ‘what is this image?’, the chatbot will prompt the user to provide the directory path to the desired image.

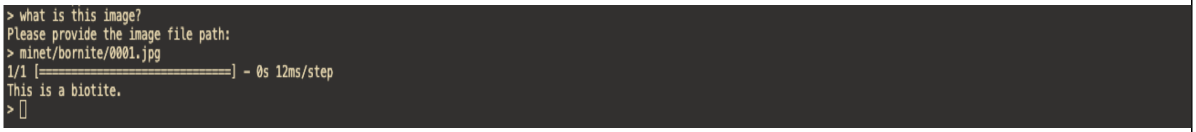


```
> what is this image?
Please provide the image file path:
> minet/quartz/0001.jpg
2025-03-24 12:14:31.090898: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
1/1 [==============================] - 0s 65ms/step
This is a quartz.
> █
```

Assuming the directory path provided refers to a suitable file format, the image is processed using a hyperparameter tuned CNN.



Quartz, https://www.kaggle.com/datasets/asiedubrempong/minerals-identification-dataset



```
> what is this image?
Please provide the image file path:
> minet/bornite/0001.jpg
1/1 [==============================] - 0s 12ms/step
This is a biotite.
> ▯
```

While the chatbot can identify a numerous minerals, unfortunately the model does mix up similar appearing minerals. In this case biotite & bornite.

Biotite,https://www.kaggle.com/datasets/asiedubrempong/minerals-identification-dataset

Bornite, https://www.kaggle.com/datasets/asiedubrempong/minerals-identification-dataset