Machine Learning Image Reconstruction for Identifying Geophysical Anomalies

RE: Final Report Draft

Presented to:

Instructional Team: Dr. Vikki Remenda and Mr. Evan Dressel

Faculty and Client Advisory Team: Dr. Georgia Fotopoulos, Alex Harvey and Jacqueline Williams

Winter 2025

GEOE 446/7

Smith School of Engineering

Queen's University

Prepared by: Airborne Insights (AI)

[Kathryn Lees, 20216152]

[Max Follett, 20200766]

[Elliot Carusone, 20218549]

[April 18, 2025]

Executive Summary

This capstone project explores the application of unsupervised machine learning (ML) to geophysical anomaly detection, with the objective of developing a semi-autonomous system for processing airborne magnetic, gravity, and satellite imagery data in mineral exploration. Conducted by Airborne Insights, the project centered around the design and implementation of an autoencoder-based image reconstruction pipeline capable of highlighting regions of anomalous geophysical response without requiring labeled training data.

The project was motivated by the increasing complexity and data volume in modern geophysical surveys, which often create bottlenecks in traditional manual interpretation workflows. By leveraging autoencoders (neural networks trained to compress and reconstruct input data) the team aimed to isolate statistically anomalous patterns through reconstruction error, serving as a proxy for potential mineralization. The scope of work included data acquisition and standardization, interpolation and image preparation, model design and training, anomaly scoring via reconstruction error heatmaps, and the development of a user-friendly web interface for data processing and visualization.

Technically, the model architecture was tailored to geoscientific constraints through dynamic latent space calculation, weighted loss functions to rebalance underrepresented gravity and magnetic data, and post-processing strategies such as patch overlap blending and spatial heatmap consolidation. The model's performance was evaluated against known porphyry copper-gold deposits in Utah using publicly available United States Geological Survey (USGS) survey datasets and Landsat 8 spectral imagery.

While the autoencoder showed some capability in isolating patterns in geophysical data, validation against known mineral occurrences revealed that the final outputs failed to consistently align with real anomalies. Spectral data dominated the reconstruction process and obscured geophysically meaningful signals. Even with targeted reweighting, the model was unable to reliably identify key deposits without manual inspection and interpretation.

This outcome led to a critical project insight: unsupervised machine learning alone is not currently feasible for reliable geophysical anomaly detection in real-world exploration contexts. Statistical reconstruction does not inherently prioritize geological relevance, and therefore the outputs require expert validation. The project concludes that AI-based tools should be viewed as assistive technologies, not replacements for geophysicists. Human validation, domain-specific constraints, and supervised refinement remain essential components for integrating ML into exploration workflows.

Despite these limitations, the project achieved several key outcomes: the construction of a functioning geophysical autoencoder pipeline, a web-based interface for users to upload their own survey data to highlight anomalies, and a reproducible workflow for preprocessing and integrating multiple survey models. The environmental impact of the project was minimal, and ethical considerations were addressed using open-access datasets and transparency in model limitations.

Recommendations for future work include integrating supervised or hybrid learning approaches, expanding geophysical data inputs (IP, EM, GPR), developing tools to contextualize and explain reconstruction errors, and automating validation routines to provide confidence metrics alongside outputs. The results of this project lay the groundwork for more intelligent, scalable, and sustainable mineral exploration tools that are designed not to replace, but to enhance expert geophysical interpretation.

Table of Contents

Executive Summary i
Table of Figures iv
List of Tables v
1.0 Introduction 1
1.1 AI in Geophysics 1
1.2 Project Significance and Impact 2
2.0 Project Management 2
2.1 Project Budget 3
3.0 Options Analysis 4
3.1 Model Applications 4
3.2 Data/Survey Types 5
3.3 Databases 6
3.4 Geological Deposits 6
3.4.1 Porphyry-Cu-Au Deposits 7
3.4.2 VMS Deposits 7
3.4.3 Skarn Deposits 8
3.5 Type of Machine Learning 8
4.0 Design Solutions 9
4.1 Data Preprocessing 9
4.1.1 Data Standardization 9
4.1.2 High Density Region 10
4.1.2 Interpolation and Landsat Imagery Gathering 10
4.1.3 Retrieving Landsat Imagery 12
4.1.4 Final Data Gathering and Preparation for Machine Learning Model 12
4.2 Machine Learning and Processing 13
4.2.1 Input data 14
4.2.2 Autoencoder Preprocessing 15
4.2.3 Autoencoder Design and Application 18
4.2.4 Error Processing 22
4.2.5 Output 24
4.2.6 Output Validation 25
4.3 User Interface 26
4.3.1 Development 27
5.0 Environmental, Ethical, and Societal Considerations 29
5.1 Environmental Considerations 29

5.2 Ethical and Societal Considerations 30
5.3 UNSDG 30
6.0 Cost Benefit Analysis 31
7.0 Recommendations 32
8.0 Conclusion 33
9.0 Team Signatures 35
References 1
Appendix 3

Table of Figures

Figure 1 – Autoencoders use neural networks to compress input data via an encoding stage, passing it through a hidden bottleneck layer before reconstructing it in the decoding stage [3]. 1
Figure 2 – Project phases for workflow of project from September to April. 2
Figure 3 – Plot of high data density ROI from magnetic survey in Utah, plot shows the traverse (black), hexbin plots, and the ROI (red). 10
Figure 4 – Linearly interpolated Density data for a survey in Marysvale Utah, Bouguer correction (left) and Isostatic correction (right). 11
Figure 5 – Linearly interpolated magnetic values of a survey in Cedar City Utah, the same survey from Figure 4 above. 11
Figure 6 – Landsat spectral band imagery for a specific ROI, data accessed through GEE 12
Figure 7 – Workflow diagram of machine learning script: Start and End Point (green), Autoencoder Preprocessing (blue), Autoencoder Construction and Training (red), and Error Analysis (orange). 14
Figure 8 – Reconstruction error plots demonstrating the stamp-effect reduction caused by the addition of noise to the input data. 16
Figure 9 – Visualization of the smoothing effect caused by an increase in the amount of patch overlap. 17
Figure 10 – Spectral band 7 and Bouguer density anomaly model inputs, and the Bouguer density anomaly reconstruction error showing extremely high error caused by the encoding of a lake into the density data. 21
Figure 11 – Plots visualizing the process which the target region survey data undergoes throughout the script. 24
Figure 12 – Heatmap plots showing the predicted anomaly locations from each survey and the consolidated heatmap showing the anomaly location which the model predicts. 25
Figure 13 – USGS map of known mineral deposits in the southern Utah region that correlate with the outputs from the model. Known deposits circled in red correlate directly to Bouger density data. 26
Figure 14 – Workflow diagram for user interface. 27
Figure 15 – User interface as seen on local website, showing the interactive layout for processing geophysical survey data with real-time feedback. 28
Figure 16 – User interface as seen on local website, illustrating system behavior: clicking the file upload button opens the user's local file directory, the interpolation method is selected from a dropdown menu, and an error message is displayed if processing is attempted without an uploaded file. 29

List of Tables

Table 1 – Hours worked and budget for project from original projected hours and budget to final cost of project. 3
Table 2 – Decision matrix evaluating potential model applications based on economic potential, geophysical relevance, and machine learning suitability. 4
Table 3 – Decision matrix evaluating geophysical and remote sensing data types based on availability, sustainability, and scalability for machine learning applications. 5
Table 4 – Decision matrix comparing geophysical databases based on data availability, file format compatibility, and high-resolution coverage for effective model training and integration. 6
Table 5 – Decision matrix for geological deposits based on signature, exploration interest, profitability, detection complexity and influence of alteration on signals. 7
Table 6 – Decision matrix for unsupervised vs supervised of machine learning comparing the advantages and disadvantages. 8
Table 7 – Description of layers present within the designed autoencoder. 19
Table 8 – Comparison of Panel and Dash for the user interface. 27
Table 9 – Cost benefit analysis between manual interpretation and a ML model for identifying geophysical anomalies. 32
Table 10 – Metadata Legend from the USGS magnetic database. 3
Table 11 – Gantt chart for 2024/25 project timeline. 4
Table 12 – Budget compliance in terms of tasks, individual progress, hours spent, status and completeness. 5

1.0 Introduction

Mineral exploration relies on geophysical surveys to map subsurface anomalies that can indicate economically viable mineral deposits. These surveys produce vast amounts of data which is traditionally analyzed by expert geophysicists. Manual interpretation can be time-consuming, subjective, and creates bottlenecks in exploration workflows [1].

This project addresses these challenges through the development of a semi-autonomous anomaly detection tool which processes unlabeled survey data and generates georeferenced files highlighting regions of interest for further investigation. By leveraging machine learning techniques, the team aims to accelerate data processing, reduce human bias, and support mineral exploration, an industry of critical economic significance.

1.1 AI in Geophysics

Artificial Intelligence (AI) refers to computational systems capable of mimicking human reasoning and learning [2]. Machine learning (ML) is a subset of AI that enables computers to learn from data and make predictions without being explicitly programed to do so [2]. This project focuses on unsupervised learning, where the model identifies patterns without predefined labels [3]. Specifically, an autoencoder which is a neural network trained to compress and reconstruct data, see Figure 1 below, is used to detect anomalies in geophysical surveys [3]. Image reconstruction plays a crucial role, transforming raw geophysical data into visual representations that highlight deviations from expected patterns.

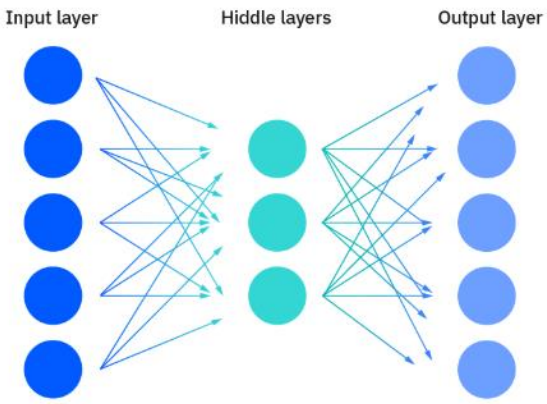


Figure 1 – Autoencoders use neural networks to compress input data via an encoding stage, passing it through a hidden bottleneck layer before reconstructing it in the decoding stage [3].

1.2 Project Significance and Impact

The automation of geophysical anomaly detection aligns with broader industry goals of increasing efficiency and sustainability in mineral exploration [4]. As countries, including Canada, strive to achieve net-zero emissions by 2050, the demand for critical minerals essential for clean energy technologies continues to grow [5]. Deeper and more complex mineral deposits are making exploration increasingly more challenging. By expediting geophysical data processing, this machine learning tool enhances the accuracy and efficiency of anomaly detection, reducing the reliance on extensive manual interpretation. This not only optimizes exploration strategies but also minimizes unnecessary drilling, lowering environmental impact and operational costs [5].

2.0 Project Management

This project started in September 2024 and ended in April 2025. It was broken up into five phases:

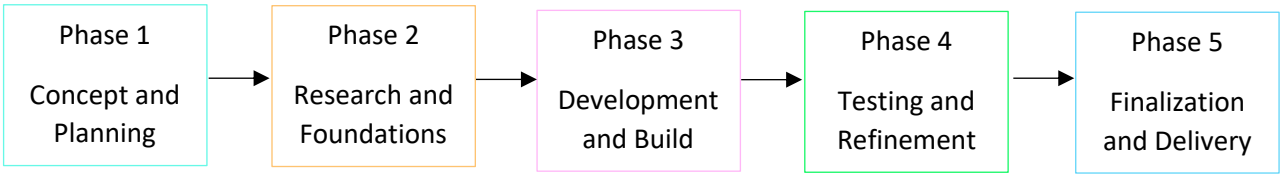


Figure 2 – Project phases for workflow of project from September to April.

Each phase consisted of several tasks and deadlines, which when all completed resulted in the final product that satisfied the project's objectives. The five phases include Concept and Planning, Research and Foundations, Development and Build, Testing and Refinement, and Finalization and Delivery. These project phases provided the structure and time constraints necessary for the project plan. The first phase, Concept and Planning, involved defining project objectives, identifying stakeholders, and outlining the project scope. A high-level timeline was developed using a Gantt chart to break down tasks and set deadlines for major deliverables and milestones, see Table 11 in the Appendix. This phase ensured that the project had a clear direction and that all team members understood their roles and responsibilities.

Phase two, Research and Foundations, focused on building a strong knowledge base to guide model development. This included conducting literature reviews, gathering initial datasets, and researching various data types used in identifying geophysical anomalies. The team also examined machine learning models and their training methodologies. The insights gained during this phase were critical for establishing the framework for the machine learning approach used in the project.

During the Development and Build phase, the team focused on constructing the code and implementing the machine learning model. Task allocation was managed carefully to ensure steady progress and adherence to the project timeline. The first version of the model was developed during this phase, forming the foundation for future refinements.

The fourth phase, Testing and Refinement, was dedicated to ensuring the model functioned as intended. Extensive testing was conducted to identify any issues or areas needing improvement. Feedback from clients and advisors was incorporated to refine the model's accuracy and performance. Adjustments were made to the code to optimize the model's ability to detect geophysical anomalies effectively.

The final phase, Finalization and Delivery, involved completing the project and preparing the final poster, and report. Lessons learned throughout the process were documented, and potential areas for future work were outlined. The completion of this phase marked the successful execution of the project and the delivery of a refined, functional model for identifying geophysical anomalies.

2.1 Project Budget

The only cost associated with this project was the team's time. The project remained on schedule throughout its timeline. The initial allocation of 630 hours was based on an estimate of the team spending 10 hours per week on the project for its duration. When the project commenced some weeks required significantly more time particularly during detailed research and iterative development phases in preprocessing, leading to the project exceeding the budget sooner than anticipated. As a result, workload calculations were revised going into second semester, increasing the projected hours to 785, with an estimated cost of $93 840 at a rate of $120 per hour. The final recorded hours totaled 787, bringing the actual cost to $94 440, only a 0.64% overage compared to the estimated projected cost. A breakdown of this can be found below in Table 1. For the complete updated breakdown of budget compliance in terms of tasks, individual progress, hours spent, status and completeness, see Table 12 in the Appendix.

Table 1 – Hours worked and budget for project from original projected hours and budget to final cost of project.

Allocated Hours			
Kathryn	Max	Elliot	Total
260	263	264	**787**
Final Cost of Project ($120/hour)			**$ 94,440**
Projected Hours (10 hours/week)			630

Projected Budget ($120/hour)	$ 75,600
Updated Projected Hours	785
Updated Budget	$ 94,200

3.0 Options Analysis

Key design decisions for the project were made using decision matrices developed during the research and foundations phase. Each matrix evaluates alternatives across relevant criteria and ranks them using a color-coded scale green (high), yellow (medium), and orange (low). In cases where all options were considered equally viable, blue boxes are used to indicate unranked selections. Final selections are clearly marked with a red box. This approach allowed the team to objectively compare trade-offs and choose the most suitable options for model applications, data types, geological deposits, and ML methods.

3.1 Model Applications

Mineral exploration ranked highest across all categories in **Error! Reference source not found.** due to its strong economic value and high geophysical relevance. With countries like Canada committing to net-zero emissions by 2050, demand for critical minerals such as copper, tin, nickel, and tungsten has increased [6]. These minerals, linked to deposits like VMS, porphyry, and skarn, are harder to find as exploration moves deeper and further from developed areas. Mining also contributes up to 11% of global CO_2 emissions, making efficient and low-impact exploration increasingly important [7].

This project supports that goal by using ML to streamline early exploration, reduce unnecessary drilling, and improve anomaly detection. Prior research confirms that techniques like anomaly detection and image reconstruction are effective for mineral targeting using geophysical data [8]. The model also considers both geological scarcity and market relevance. Although some critical minerals are geologically scarce (e.g., antimony at 10.8 on the scarcity scale), market prices have remained relatively stable, so both availability and demand are prioritized [9].

Geological mapping scored high in geophysical relevance but lower in economic and ML suitability. Environmental analysis received medium scores overall; although geophysics can aid in tasks like contamination detection, it is less directly aligned with this project's data and goals.

Table 2 – Decision matrix evaluating potential model applications based on economic potential, geophysical relevance, and machine learning suitability.

	Mineral Exploration	Geological Mapping	Environmental Analysis
Economic Potential	High	Medium	Medium
Geophysical Relevance	High	High	Medium
ML Suitability	Medium	Low	Medium

3.2 Data/Survey Types

The geophysical data used for this project includes magnetic, gravity, and Landsat spectral band data. Magnetic and gravity surveys are critical in detecting subsurface mineralization, as they respond to contrasts in physical properties, magnetic surveys highlight variations in magnetic susceptibility (e.g., magnetite-rich zones), while gravity surveys detect density differences between ore bodies and surrounding rock. These methods are commonly used to infer geological structures, locate buried intrusions, and identify alteration zones associated with mineralization which is why they were chosen to be input surveys for this project.

Landsat 8 spectral bands 4, 5, 6, and 7 were also selected for spectral analysis due to their proven utility in mapping hydrothermal alteration and lithological variations associated with mineral deposits. Band 4 (near-infrared) is sensitive to vegetation and surface moisture, while bands 5 and 7 (shortwave infrared) are effective in identifying clay minerals and hydroxyl-bearing alteration products common in porphyry and VMS systems [10]. Band 6 also contributes to distinguishing silicate and carbonate compositions. For better spectral discrimination, band ratios such as 4/5 and 5/7 were employed. The 4/5 ratio is particularly effective for highlighting vegetation anomalies and iron oxide content, whereas the 5/7 ratio emphasizes variations in clay and alteration mineral assemblages [10].

Table 3 – Decision matrix evaluating geophysical and remote sensing data types based on availability, sustainability, and scalability for machine learning applications.

	Magnetic	Gravity	EM	IP	Radiometric
Availability	High	High	Medium	Medium	Medium
Suitability for Detection	High	High	High	High	Low
Scalability	High	High	Medium	Low	Low

3.3 Databases

With magnetic and gravity data selected as the primary geophysical inputs for model training, it was essential to identify a consistent, reliable, and easily accessible data source. Three major geophysical databases were considered: the United States Geological Survey (USGS), the Geological Survey of Canada (GSC), and the Geophysical Database of Australia (GDA). A decision matrix was developed to evaluate these sources based on data availability, file formats, and high-resolution geophysical coverage.

After testing data access and integration into the user interface, the USGS was decided to be the most suitable option. It offers extensive publicly available magnetic and gravity data, along with versatile file formats (.csv, .shp, and .xyz) that align well with the project's preprocessing workflow. Additionally, the USGS provides high-resolution coverage across the continental United States, Alaska, and Hawaii, which allows the team to explore a diverse range of geological settings and topographies for model development and validation.

Table 4 – Decision matrix comparing geophysical databases based on data availability, file format compatibility, and high-resolution coverage for effective model training and integration.

	USGS	GSC	GDA
Data Availability	High	Medium	Medium
Available Files	CSV, shape, Tiff, JPEG, XYZ, NetCDF	CSV, SEG-Y, DBF/DXF	SEG-Y, CSV, NETCDF, ASEG-GDF2
High Resolution Coverage	USA	CA	AU

3.4 Geological Deposits

To train the ML model to identify economically significant geophysical anomalies, three major deposit types were considered, Volcanogenic Massive Sulphide (VMS), Porphyry Copper-Gold (Cu-Au), and Skarn deposits. These deposit types were selected because of their geophysical signatures and economic importance in mineral exploration. Each exhibits unique combinations of physical properties like conductivity, density, and magnetic susceptibility, that can be detected through various geophysical survey methods. A decision matrix (Table 5) was developed to evaluate the three deposit types, and Porphyry-Cu was ultimately selected due to its strong geophysical signature, high exploration interest, economic potential, and the relative ease of detection through established survey methods.

Table 5 – Decision matrix for geological deposits based on signature, exploration interest, profitability, detection complexity and influence of alteration on signals.

	Porphyry-Cu	VMS	Skarn
Mappable Ore Bodies	High	Medium	Low
Exploration Interest	High	Medium	Medium
Profitability	High	Low	Medium
Detection Complexity	Medium	High	High
Influence of Alteration on Signals	Low	Medium	High

3.4.1 Porphyry-Cu-Au Deposits

Porphyry Cu-Au deposits are typically large, low-grade ore bodies containing disseminated sulphides, making them ideal candidates for induced polarization (IP) surveys. As highlighted by Sinclair, IP is particularly effective in porphyry exploration due to its sensitivity to chargeable minerals spread over broad areas [11]. The team's original idea was that the model would focus on IP data as the primary input for porphyry systems, specifically targeting high chargeability values ranging between 15-30 mV/V. These deposits also respond to magnetic surveys, especially in regions with hydrothermal magnetite, which results in magnetic highs. Conversely phyllic alteration zones, where magnetite is destroyed, may present as magnetic lows, providing contrasting features useful for classification. The average magnetic susceptibility for porphyry systems is 60 g/cm^3, which helps the model differentiate magnetite-rich zones from alteration zones. Alteration patterns such as potassic, phyllic, and argillic zones also influence geophysical responses and in the future should be incorporated into the model's classification logic to improve anomaly detection accuracy [11].

3.4.2 VMS Deposits

VMS deposits are characterized by high electrical conductivity, significant magnetic responses, and elevated density values due to their abundant sulphide content. Minerals such as chalcopyrite, pyrite, and sphalerite are commonly present, and their conductive nature produces strong anomalies in electromagnetic (EM) surveys. Gravity surveys are effective in detecting VMS deposits, as they are typically denser than their surrounding host rocks. Magnetic responses vary across the deposit due to changes in mineralogy, particularly in areas rich in magnetite or iron-rich sphalerite, which create noticeable magnetic anomalies. However, these signals can be complex, especially near the massive sulphide lens and associated stockwork zones, where alteration affects the magnetic signature which is why they were not selected [12].

3.4.3 Skarn Deposits

Skarn deposits are formed through metasomatic processes and exhibit distinct zonation in metal content, known as metal dispersion halos. These halos can extend up to 1000 meters from the deposit's core and are characterized by elevated concentrations of elements (Au, Te, Bi, Ag, Pb, and Zn). It ended up being too complex to train the model to recognize both proximal and distal geochemical signatures which is essential to avoid overlooking skarn-associated anomalies. Skarns often contain abundant magnetite and pyrrhotite, resulting in strong magnetic signals. However, some skarns exhibit magnetic lows due to magnetite alteration under reduced conditions. Skarns were also considered because gravity surveys are useful as skarns are denser than their host rocks. Strong IP and EM responses can come from disseminated sulphides, but care must be taken when interpreting EM results due to the potential presence of graphite, which can mimic sulphide conductivity. This made finding survey data on skarns challenging and is why they were also not chosen [13].

3.5 Type of Machine Learning

A decision matrix was also created to evaluate potential ML approaches for geophysical image reconstruction. The two options considered were supervised and unsupervised learning models. The team selected an autoencoder, a type of unsupervised ML model, based on its suitability for anomaly detection and image reconstruction tasks.

Unsupervised learning offers several advantages for detecting geophysical applications. An unsupervised model does not require labeled data, which is a key benefit given the large volume of unlabeled magnetic and gravity datasets being used. Autoencoders are well-established for learning patterns within complex datasets and highlighting deviations or anomalies, which allows the team to work with an established autoencoder.

In contrast, supervised learning models rely on extensive labeled datasets, which are not only difficult to obtain in this context but also require significant preprocessing. Additionally, supervised models tend to be more computationally intensive, adding further constraints to their practical implementation in this project. Refer to Table 6 below which breaks down the advantages and disadvantages to an unsupervised and a supervised ML model.

Table 6 – Decision matrix for unsupervised vs supervised of machine learning comparing the advantages and disadvantages.

	Unsupervised	Supervised
Advantages	- Algorithms exist for anomaly detection/image reconstruction - Model can learn from unlabeled data	- Once trained, more accurate - Error analysis is easier (labelled data)
Disadvantages	- Requires human validation - Less accurate	- Requires labelled data - Computationally extensive

4.0 Design Solutions

The design solution goes through the full development process of the ML anomaly detection model starting with geophysical and satellite data preparation, continuing through ML design and training, and ending with output validation and interface integration.

4.1 Data Preprocessing

This project integrates USGS geophysical data with Landsat imagery accessed through Google Earth Engine to train a ML model for mineral exploration. All available magnetic and gravity survey datasets were first standardized, cleaned, and interpolated to ensure consistency across regions. For areas containing known porphyry copper deposits with high-resolution magnetic and density coverage, corresponding multispectral Landsat imagery was retrieved using Google Earth Engine.

4.1.1 Data Standardization

Raw magnetic and gravity data were extracted from the USGS geophysical database, with survey coverage focused primarily on regions within Utah, Wyoming, Montana, and Nevada. These datasets were provided as .zip folders containing a variety of file formats, including .txt metadata files, .jpg survey images, and .xyz files containing radial and magnetic measurement data. Each archive was uploaded to a shared GitHub repository for centralized access.

To process the magnetic data, a custom Python script was developed to extract the relevant .xyz files directly from the repository. The script extracted the necessary columns and visualized the survey data for initial inspection. Based on the USGS metadata conventions, a data legend was created to guide preprocessing. Specifically, columns 6, 7, and 10, representing latitude, longitude, and residual magnetic intensity, respectively, were selected as input features for model training. Refer to Appendix A, Table 10, for a detailed breakdown of the USGS metadata structure for the magnetic survey files.

4.1.2 High Density Region

The primary objective of preprocessing is to convert geophysical point data into structured images that can be segmented into patches and fed into a ML algorithm. This requires interpolating the data onto a unified master grid, a process implemented using standard Python libraries such as NumPy and SciPy. One of the main challenges is that ML models typically require rectangular, vertically oriented inputs, while geophysical surveys vary widely in shape and orientation.

Hexbin plots were utilized to identify high-density, rectangular regions within each geophysical survey area. As two-dimensional spatial histograms, hexbin plots group data points into hexagonal bins based on their spatial coordinates, with each hexagon representing the local data density. To isolate the core survey area, hexagons falling below the 25th percentile in data density were filtered out, a threshold determined through iterative testing. This approach effectively highlights the primary shape and extent of the survey by removing the sparse data. Refer below to a plot of an airborne magnetic survey traverse, with hexbin plots and the defined high-density region of interest (ROI).

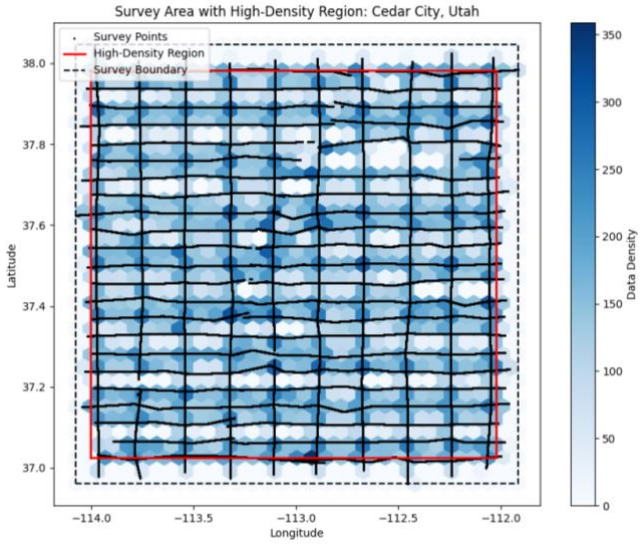


Figure 3 – Plot of high data density ROI from magnetic survey in Utah, plot shows the traverse (black), hexbin plots, and the ROI (red).

4.1.2 Interpolation and Landsat Imagery Gathering

Once a ROI is defined, the geophysical data must be interpolated. Linear interpolation was chosen as the most suitable method due to its simplicity, efficiency, and effectiveness in preserving the spatial trends of the gravity and magnetic data without introducing unnecessary complexity or smoothing. It provided a balance between computational speed and accuracy, making it ideal for large regional datasets.

The gravity data used in this process was sourced from the USGS continental U.S. gravity reference dataset, including both Bouguer and isostatic corrections. These datasets were uploaded to the teams

shared GitHub repository. A Python script was then developed to automatically retrieve the gravity data from GitHub, apply linear interpolation, and output the results.

One issue resolved during the linear interpolation of the geophysical data was the unintended extrapolation occurring at the corners of the ROI, which resulted in distorted arrays and inaccurate interpretations. To address this, the user-defined ROI was temporarily expanded by 0.3 degrees in both latitude and longitude in all directions. Interpolation was then performed over this larger area, but only the data within the original ROI was retained and displayed. This approach eliminated the need for extrapolation by ensuring that all interpolated values were based on surrounding data rather than assumptions beyond the available dataset. Figure 4 and Figure 5 below provide a visual representation of the linear interpolation for the gravity data and magnetic data, respectively.

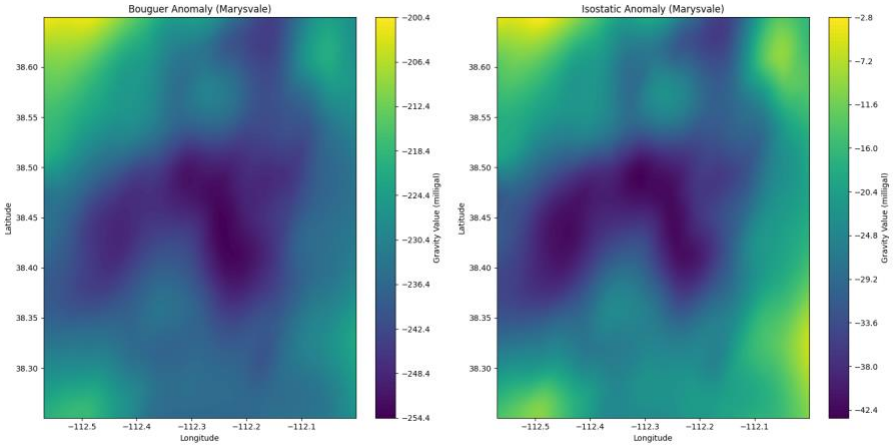


Figure 4 – Linearly interpolated Density data for a survey in Marysvale Utah, Bouguer correction (left) and Isostatic correction (right).

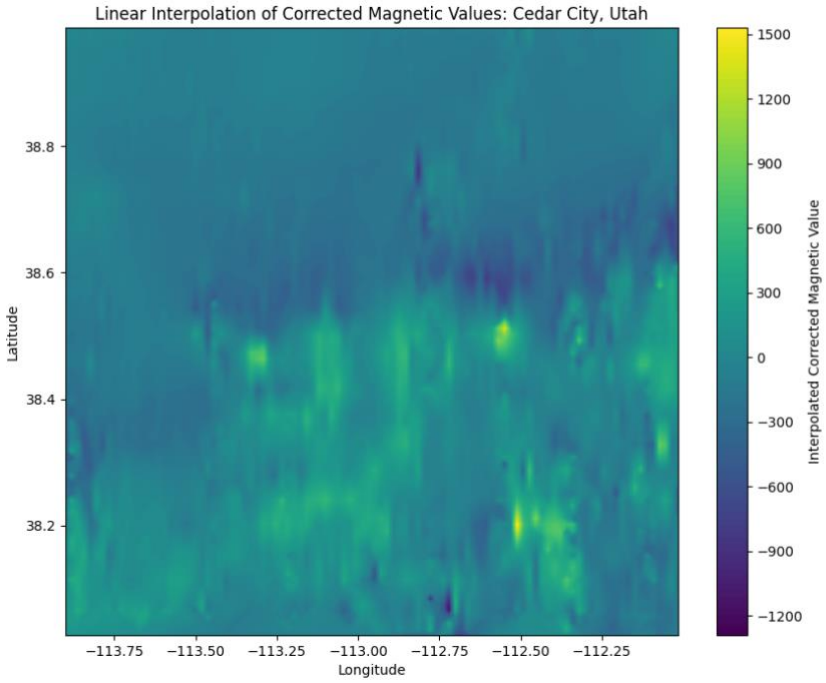


Figure 5 – Linearly interpolated magnetic values of a survey in Cedar City Utah, the same survey from Figure 4 above.

4.1.3 Retrieving Landsat Imagery

To support the ML reconstruction of geophysical imagery, we retrieved Landsat satellite data using Google Earth Engine (GEE), a cloud-based geospatial processing platform. The process began with the setup of a dedicated project in the Google Cloud Console, which was then linked to GEE to enable access through the Python script for retrieving the imagery. The team created a service account within this project that generated a .json key file to authenticate our scripts. This credentials file allowed the team to interact with the Earth Engine Python API, a tool that lets the code communicate directly with Google cloud infrastructure.

To facilitate collaboration, the .json key was securely distributed among the project team, enabling each member to initialize the Earth Engine API locally and independently access the Landsat data. Once authenticated, team members could define regions of interest and timeframes to retrieve relevant imagery. This setup ensured that all team members had synchronized access to consistent datasets, while leveraging GEE's infrastructure for scalable image retrieval and analysis. Figure 6 below is the output of the Landsat preprocessing script showcasing all desired Landsat bands and band ratios.

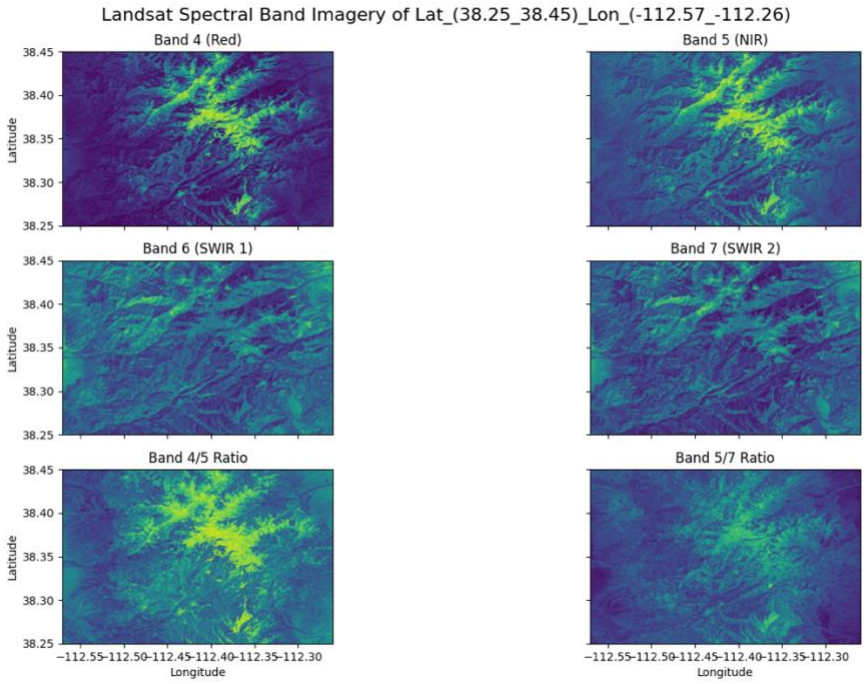


Figure 6 – Landsat spectral band imagery for a specific ROI, data accessed through GEE

4.1.4 Final Data Gathering and Preparation for Machine Learning Model

The final step in preparing the input for the ML model involves running a master script that integrates and standardizes the outputs from the gravity, magnetic, and Landsat preprocessing scripts. This master

script streamlines the workflow by suppressing any intermediate visualizations or print statements to reduce processing time. The autoencoder employed in this project is optimized to take .txt files as inputs, with each file representing a different geophysical survey. To maintain compatibility, all nine .txt files must have identical array dimensions.

These dimensions were standardized based on the survey area's spatial coverage and resolution requirements. Given the high resolution of the Landsat imagery and the lower resolution of the gravity and magnetic datasets, a target pixel resolution of approximately 20 meters, corresponding to around 0.0026 degrees in latitude or longitude for the Utah region, was selected. This resolution provided a suitable balance, enabling uniform array sizes across all data types while preserving spatial detail relevant to the training process. Once the data has been resampled and aligned to this common grid, the master script exports the complete set of nine .txt files into a folder named after the survey area. Each file follows a naming convention that includes the survey type, latitude and longitude bounds, and the date of preprocessing. This folder serves as the final, standardized input package ready for the autoencoder.

4.2 Machine Learning and Processing

An autoencoder was chosen for its efficiency in learning how to create compressed representations of data. The iterative process of training an autoencoder involves minimization of data loss in the encoding and decoding process. During training, the autoencoder should learn to capture the dominant and large-scale patterns in the data as this will minimize the loss. With the model being trained to learn and reconstruct the most frequent and consistent patterns, it will tend to ignore or poorly reproduce anomalous data. This property makes autoencoders effective for anomaly detection. By comparing the input and the decoded or reconstructed data the location of anomalous can be determined. Further, an autoencoder can serve as a tool for uncovering patterns and irregularities that may not be easily identified by a human observer, especially when dealing with large or high-dimensional datasets.

The process designed in the autoencoder script is outlined in the workflow diagram shown below in Figure 7. This workflow diagram is divided into five key parts which connect the data preprocessing to a final output that gets manually validated. The purpose and reasoning for each element of the workflow diagram is discussed throughout this section.

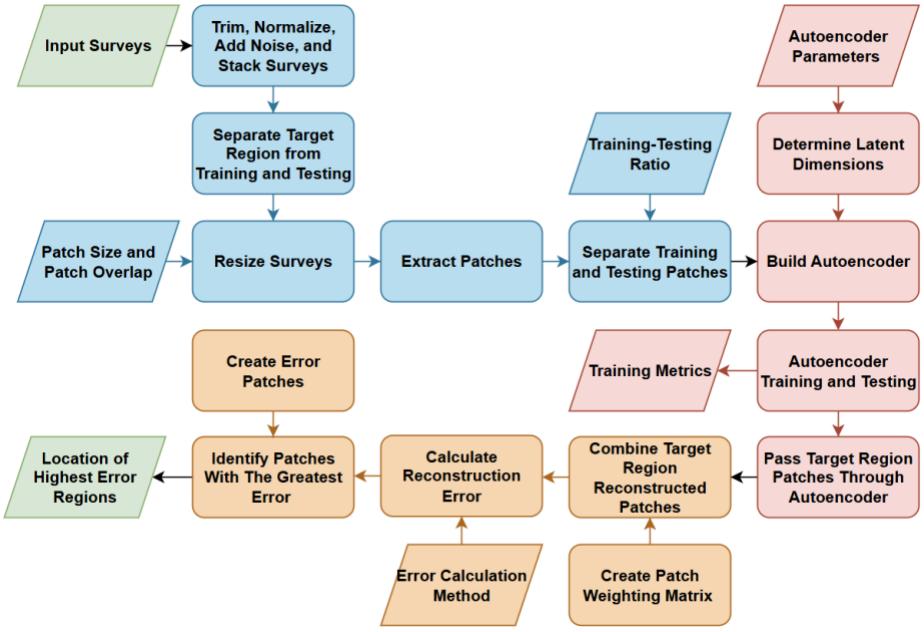


Figure 7 – Workflow diagram of machine learning script: Start and End Point (green), Autoencoder Preprocessing (blue), Autoencoder Construction and Training (red), and Error Analysis (orange).

Using the JupyterLab Python environment each element of the workflow diagram could be coded and tested separately. The notebook interface allows for clear separation of each element of the workflow within the script, making the code easier to navigate and update. The ability to interweave code and visualizations or other outputs provides an effective way to debug, test, and iterate upon code. Further, the persistent in-memory environment significantly speeds up development and testing as it eliminates the need to reload or recompute earlier steps when modifying later parts of the script. These features make the JupyterLab environment well-suited for data-driven tasks such as ML and data processing.

4.2.1 Input data

The input data used in this project consists of three sources; Landsat 8 imagery, gravity survey data, and magnetic survey data (as described in Section 4.1). The dataset includes six spectral bands from the Landsat satellite, two gravimetric surveys, and one magnetic survey, covering the same spatial region. However, this information will not be provided to the autoencoder as it is designed as an unsupervised model so it should learn to corelate the similar survey types.

The .txt file outputs from the data preprocessing are read into the script and are currently the only external input for the script. This approach was chosen for convenience and ease of testing. While the data preprocessing functions capable of dynamically extracting data based on a specified latitude and longitude range, these were excluded from the main script to allow for faster iteration and flexibility. A

more finalized version of the project would process an input that consists of both a coordinate range and preprocessed survey data, making this still relevant for future iteration.

4.2.2 Autoencoder Preprocessing

This stage of the workflow prepares the input data specifically for use with the autoencoder model. Unlike the general data preprocessing described in Section 4.1, this step focuses on formatting and structuring the data in a way that aligns with the autoencoder's input requirements. It is deliberately separated from the main data preprocessing process, as the input to the data preprocessing script and this script may vary depending on how data is received.

The preprocessing within this script is designed to accept any number of survey-inputs and dynamically adjusts the relevant variables. This is required for scalability and testing as it allows the script to work with various types and amounts of data which makes testing and updating more efficient.

To simplify both preprocessing and debugging, each survey is normalized independently to a range between 0 and 1. This approach avoids issues where the value range of one data source might disproportionately influence the training process and allows the model to treat each survey with equal weight. The script scans for rows consisting entirely of zeros or ones, which are likely corrupted or improperly preprocessed data. The survey is flagged and gets manually traced back through the data preprocessing. This ensures that the autoencoder is not trained on meaningless or misleading data.

Before further processing, a check is performed to ensure that all input arrays which are holding the survey data are the same size. Different size arrays could suggest that the surveys represent different physical areas. Despite this, arrays are all trimmed to the smallest dimension for convenience which could lead to a source of error. All survey channels are combined into a single array for processing. This reflects the fact that geophysical data is spatially correlated. By stacking the surveys, the spatial relationships between can be learned by the autoencoder during training.

To improve the general applicability and robustness of the autoencoder random noise is added to the input data. This serves two key purposes: it reduces the influence of outliers and helps prevent the overfitting of characteristic patterns within the data during autoencoder training. The amount of noise added was experimentally determined to remove a stamping effect that occurred in the training process. This stamping effect and the impact of applying 1% noise to all the data is illustrated in Figure 8.

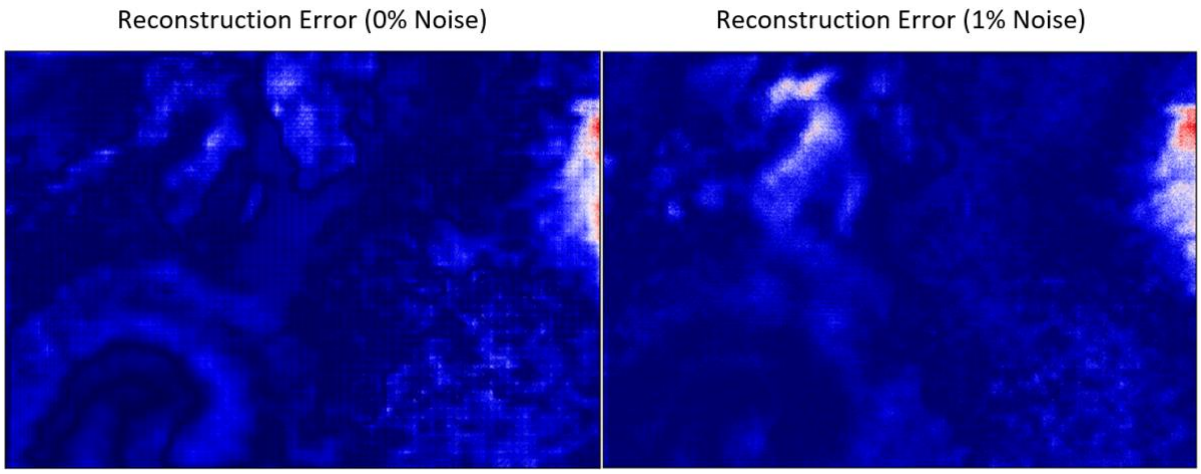


Figure 8 – Reconstruction error plots demonstrating the stamp-effect reduction caused by the addition of noise to the input data.

The stamping effect is quite significant on the left plot with each individual patch being visible. This is not a good reconstruction and causes errors to appear in the wrong locations. With the addition of the 1% noise the stamping effect is significantly reduced.

To prepare for model training and evaluation, the survey region is divided into quadrants and a target region is selected for anomaly detection. As geological features are often local in nature, it is important that the training data comes from nearby regions. This helps the autoencoder learn the characteristic geological patterns present in the area surrounding the target region. For model validation, there is a known deposit within this region which should appear as an anomaly that will be detected by with this script. The remaining quadrants are reserved for the training and testing of the autoencoder. This separation ensures that the data used to verify the design feasibility has not been seen during training. This is also critical for avoiding overfitting, a common theme throughout the design.

The four quadrants need to be divided into patches before being given to the autoencoder. The patch size is kept static to simplify the design process, although the code supports adjusting it if needed. While square patches are used for convenience, the design does not enforce this shape. Patch size impacts both the structure of the autoencoder and the total number of training samples. Larger patches result in fewer inputs, while smaller patches generate more training data, which is needed for improving model performance. A larger dataset helps the autoencoder better capture the underlying patterns in the data which will allow for more accurate anomaly detection.

By overlapping the patches, even more input can be prepared, providing the autoencoder with more opportunities to decipher any underlying spatial patterns. By extracting patches, it should also enable

the autoencoder to capture features at different spatial scales, as local variations within each patch can highlight both local and regional trends. Figure 9 demonstrates the difference between zero and some amount of patch overlap, with the former having a clear checker pattern and the latter be more visually smooth.

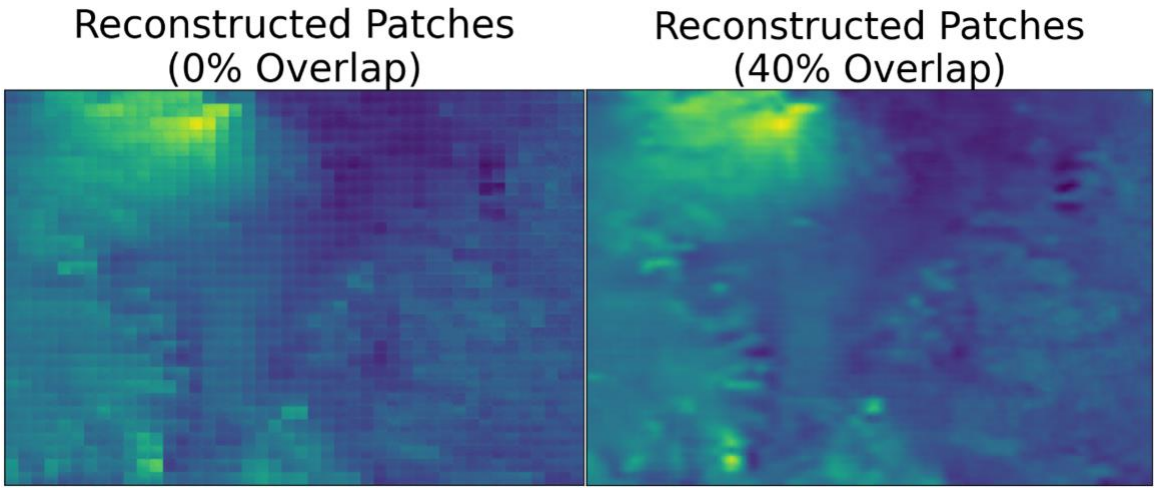


Figure 9 – Visualization of the smoothing effect caused by an increase in the amount of patch overlap.

The overlapping of patches helps to mitigate edge artifacts during reconstruction. When the model's outputs are later reassembled into reconstructed surveys, overlapping regions can be averaged to smooth out abrupt changes at patch boundaries. This results in more consistent and reliable reconstructions, particularly around areas where anomalies might otherwise be masked or exaggerated due to patch discontinuities.

As the patch dimensions and overlap may not divide evenly into the dimensions of the quadrants, it is necessary to calculate new array dimensions to ensure full patch coverage. The maximum size of the quadrants that allows patches to be extracted without exceeding array boundaries is determined with the following equation:

$$Max\ Dim_x = (P_w \cdot P_h - P_o) + \left(Dim_x - \frac{P_w \cdot P_h - P_o}{P_x - P_o}\right) \cdot (P_x - P_o) \quad [1]$$

In equation 1, Dim_x represents either the horizontal or vertical dimension of the quadrants. P_w and P_h are the patch width and height, respectively; P_o is the amount of overlap between patches; and P_x corresponds to either P_w or P_h depending on whether Dim_x refers to the width or height. This formula is needed to avoid partial or clipped patches at the edges. The maximum dimensions are used to trim all

quadrant arrays accordingly. Data from the bottom and right edges of each array are removed to ensure that the dimensions align for patch extraction.

An alternative approach was also tested during development, where the patch size and overlap were forced to fit the existing quadrant dimensions. However, this method introduced significant variation in patch size and overlap, which proved to be problematic. As previously discussed, patch size and overlap are key controls for the autoencoder training process, affecting the number of training samples, the spatial scale of features captured, and the structure of the model itself. Maintaining a consistent patch size was prioritized and trimming the data to fit the selected dimensions was chosen as the more controlled approach for testing.

Patches can now be extracted from both the target region and the training-testing quadrants. Patches from the three non-target quadrants are concatenated into a single dataset for training and testing the autoencoder. These patches are randomly shuffled to ensure that the training and testing sets are representative of the input survey area and not biased by spatial grouping, thus reducing the chance of overfitting during training. The training-testing patches are split with an 80/20 training-testing ratio.

At the end of this preprocessing stage, there are two patch arrays: one containing patches from the target region, and another containing the shuffled and split patches from the remaining three quadrants, used for training and testing the autoencoder. All the data are now normalized, had noise added, consistently sized, and structured for input into the model. With all necessary preprocessing complete, the data is ready to be fed into the autoencoder for training, evaluation, and processing.

4.2.3 Autoencoder Design and Application

This section outlines how the autoencoder is constructed, trained, and applied to the processed the survey data. The autoencoder is implemented using the TensorFlow library, which provides a variety of ML tools and pre-built components that simplify model design and training. TensorFlow's flexibility and compatibility with GPU acceleration also make it well-suited for handling large geophysical datasets efficiently.

An autoencoder is a type of neural network designed to compress and then reconstruct data, learning to capture the most important underlying patterns in the process. It consists of multiple layers stacked together, like an onion. The input is gradually reduced in size through the encoder, passed through a low-dimensional bottleneck, and then expanded back to its original shape through the decoder. Each layer transforms the data while the general structure of the data should be preserved.

The architecture of the autoencoder is designed using a sequence of layers that serve distinct purposes in the encoding and decoding processes. The encoder is composed of a reshape layer, followed by a flatten layer, a fully connected dense layer, batch normalization, Rectified Linear Unit (ReLU) activation functions, and a dropout layer. The decoder reverses the encoding process, using a dense layer, batch normalization, and sigmoid activation functions to reconstruct the data, followed by a reshape layer to restore the original input dimensions. This structure allows the autoencoder to learn a reversable and meaningful compression of the data. Each type of layer performs a specific transformation or function within the network. Table 7 explains the purpose and effect of each layer in the context of the autoencoder's operation.

Table 7 – Description of layers present within the designed autoencoder.

Autoencoder Layer	Layer Function
Reshape	Reshape the input data in preparation for later layers.
Flatten	Convert multi-dimensional input into a 1D vector, making it compatible with dense layers.
Dense	Fully connected layer which enables complex pattern learning.
Batch Normalization	Normalizes the inputs to a layer across a match.
Activation	Applies a function to allow the network to learn complex patterns.
Dropout	Randomly disables a fraction of neurons during training to prevent overfitting and improve generalization.

The purpose of the autoencoder is to learn and encode the general patterns present in the input survey data, while anomalous data gets lost. This design choice relies on the assumption that anomalous data are not well-represented in this compressed space after training. To ensure this effect is achieved preventing overfitting is crucial.

The prevention of overfitting is mostly achieved within the encoder. The flatten layer converts each set of patches into a one-dimensional vector to allow processing by dense layers. The dense layer is a fully connected layer meaning it links every neuron to all neurons in the previous layer. This is vital as it enables the model to learn complex feature relationships across the entire input. ReLU activation functions introduce non-linearity, allowing the network to learn more expressive patterns. Batch normalization improves training stability by standardizing the output of intermediate layers. Lastly,

dropout randomly disables a proportion of neurons during training to discourage the network from becoming too reliant on any one path through the model, further helping to prevent overfitting.

The decoder reverses this compression process. A dense layer expands the compressed data back toward its original shape. The sigmoid activation functions constrain output values between 0 and 1 to match the normalized input range. Batch normalization is applied to promote stable training, and the final reshape layer returns the data to its original format. This allows for direct comparison between the reconstructed and original patches.

The final step in the autoencoder construction is the definition of a custom mean squared error loss function. This modification was necessary to address an overfitting issue observed during early testing, where the autoencoder focused too heavily on the more abundant and higher resolution Landsat spectral bands, while neglecting the gravity and magnetic surveys. As a result, features characteristic of the Landsat data, such as the outline of a lake, began appearing in the reconstructed gravity and magnetic outputs, which isn't possible given just the density data.

The weighting function was introduced into the loss calculation, assigning higher penalties to reconstruction errors in the gravity and magnetic layers. By increasing their contribution to the overall loss, the model is required to give redistribute the importance of different survey types. This adjustment ensures the autoencoder learns to accurately reconstruct all survey types, rather than overfitting to the dominant Landsat input. Figure 10 illustrates this issue, showing artifacts from the Landsat data appearing in the gravity survey reconstructions prior to the application of the weighted loss function.

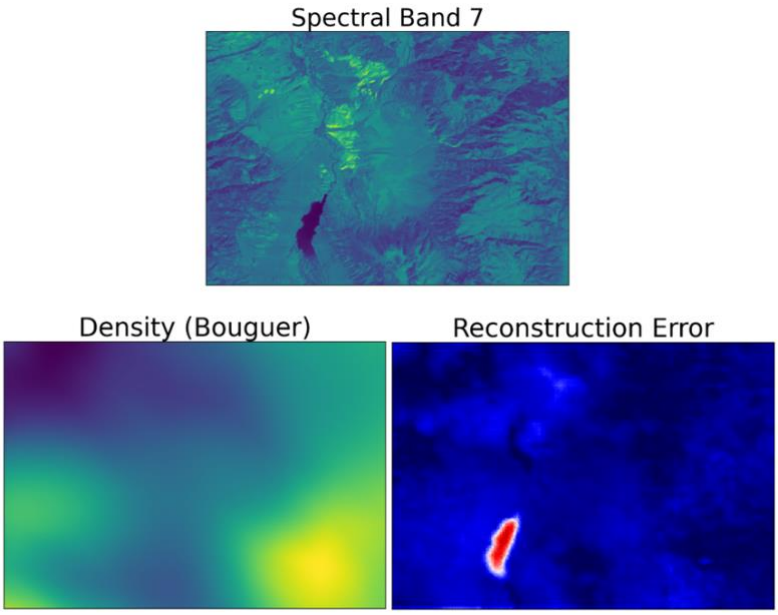


Figure 10 – Spectral band 7 and Bouguer density anomaly model inputs, and the Bouguer density anomaly reconstruction error showing extremely high error caused by the encoding of a lake into the density data.

Several parameters must be defined to use the autoencoder: the latent dimension, dropout rate, activation functions, and the number of training epochs. The key parameter is the latent dimension, which determines the size of the compressed representation produced by the encoder. The latent dimension is dynamically calculated based on the input patch dimensions and the number of input surveys.

Initially, the latent dimension was set to 25% of the total number of input values (patch height x patch width x number of layers). This was found to lead to overtraining, with the model producing uniform reconstruction error across all survey types. This indicated that the model was encoding the data too well and masking potential anomalies. To address this a new function was designed to determine the latent dimension which decreases the latent dimension as the number of surveys increases. This forces the autoencoder to compress the input more aggressively, encouraging it to prioritize finding patterns or correlations across the inputs. The equation used to define the latent dimension is shown below:

$$Latent\ Dimension = Patch\ Height \cdot Patch\ Width \cdot \frac{\sqrt{Number\ of\ Layers - x_0}}{s} - b \qquad [2]$$

In equation 2, x_0 is a horizontal offset constant, s is a scaling factor, and b is a vertical offset constant. These parameters were experimentally determined to perform well with the specific survey area. By dynamically adjusting the latent dimension the pattern identifying capabilities of autoencoders can be leveraged to reduce overfitting.

Training is performed using the training and testing patches and is carried out over a defined number of epochs. During each epoch, the autoencoder iteratively adjusts its internal weights to minimize the loss calculated by the custom error function. This process refines both the encoding and decoding paths, enabling the model to compress and reconstruct input data more accurately with each pass. The use of a separate testing set ensures that the model's performance is not only improving on the training data, but also generalizing to unseen data, which is necessary for accuracy in the reconstruction process.

Once training is complete, the target region patches are passed through the encoder and decoder, producing a set of reconstructed patches. These reconstructions can then be directly compared to the original input to assess reconstruction error, which forms the basis for detecting potential anomalies within the target region.

4.2.4 Error Processing

In this section of the workflow the reconstruction of the target region is used to identify anomalous data by comparing it to the original data. The areas where a predicted physical anomaly exists will be highlighted.

Two strategies for calculating reconstruction error were considered: one is to compare each patch individually to its corresponding reconstructed patch; the other is to rebuild the full target region from the reconstructed patches and compare it against the original target region. The second approach was chosen as it allows for capturing more regional-scale error patterns that may span across multiple patches. While larger patches could achieve similar results, they reduce the number of available training samples and may obscure smaller, localized features during training.

A function was developed to stitch the reconstructed patches together back into the target region. Due to the use of patch overlap during preprocessing, overlapping areas between adjacent patches must be blended to avoid visible seams or artifacts in the reconstructed image as seen in Figure 9. Simple averaging of overlapping values proved insufficient, as it resulted in unnatural transitions and the outlines of individual patches becoming visible in the final output. A weighted blending function was created to assign higher weights to values near the center of each patch and gradually reduce the influence of values near the edges. Several types of weighting distributions were considered, including inverse, inverse square, linear, and Gaussian, but a sigmoid function was selected. The sigmoid distribution provided the most visually seamless results, creating smooth transitions between patches.

The sigmoid weighting function assigns a weight of 1 at the center of each patch, where the data is most representative, then gradually reduces the weight toward the patch edges within the overlapping region. The function does not reduce the weight to zero, as even the edge values retain meaningful information. The slope of the gradient is dynamically determined based on the size of the patch overlap, with shallower gradients for larger overlaps. This behavior allows the function to adapt to different overlap sizes while preserving the smooth visual transitions between patches. The equation used for the weighting function is shown below:

$$Weight = \frac{1}{1 + e^{k(x-m)}} \quad [3]$$

The variables in equation 3, the sigmoid function is defined as follows: x represents the Euclidean distance from the center of the patch. The parameter m defines the midpoint of the sigmoid curve, the distance at which the function outputs a weight of 0.5. This value scales with the size of the patch

overlap, ensuring that the transition from high to low weights remains centered within the overlapping region. The parameter k controls the steepness of the curve and is also scaled based on the patch overlap, allowing the function to dynamically adapt its gradient to different overlap sizes.

Using the sigmoid function, a weighting patch is generated and applied to each reconstructed patch prior to reconstruction of the full region. This ensures that overlapping areas contribute proportionally, with central values carrying more influence than edge values. Once all patches are weighted, they are summed together to form the final reconstruction of the target region.

There are many ways to calculate reconstruction error, each of which may emphasize different aspects of the data and reveal different types of anomalies. Two simple methods were considered: squared error and absolute error. Both are calculated on a pixel-by-pixel basis by comparing the original target region to its reconstruction. Squared error emphasizes larger differences, making it useful for highlighting localized anomalies. However, since geophysical anomalies are often regional rather than point-based, the absolute error, which treats all differences linearly, was selected as the more appropriate measure. In retrospect, applying some form of normalization to the error prior to analysis might have provided additional insight, such as distinguishing between over and under-reconstruction, which could carry physical meaning.

To detect suspected anomalous regions, the reconstruction error map is divided into a new set of patches specifically for regional error analysis. As the goal is to identify broader patterns of reconstruction error, these error-searching patches are larger than those used during autoencoder training. As with the earlier patching process, patch overlap is included; however, here it is to avoid missing significant errors that occur near patch boundaries.

For each individual layer of the reconstructed target region, the error values within each of these larger patches are summed, creating a heatmap of regional reconstruction error. This allows the script to pinpoint specific areas where the autoencoder poorly reconstructed the input survey data. These regions are candidates for further investigation, as they likely correspond to geophysical anomalies or other features of interest. The progress of the workflow up to this point is illustrated in Figure 11, showing the transition from the original target region to its reconstruction, followed by the generation of an error map, and ultimately the creation of a high error heatmap used to identify the location of potential physical anomalies.

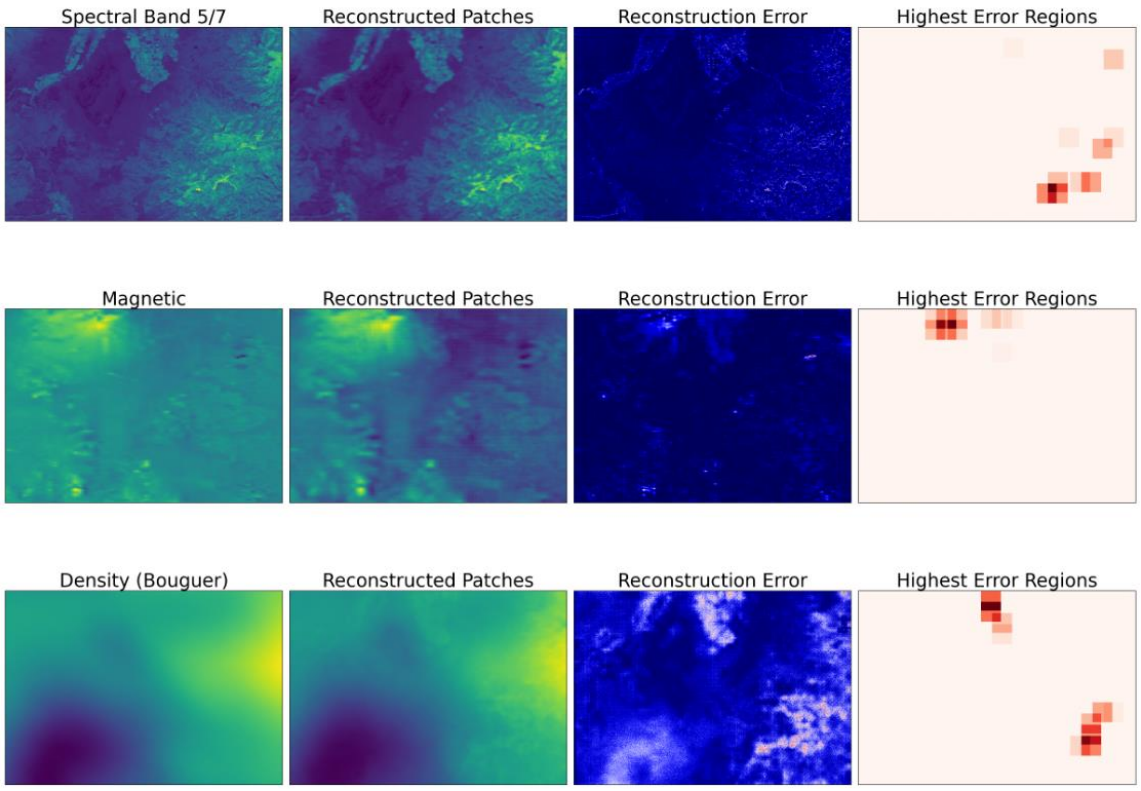


Figure 11 – Plots visualizing the process which the target region survey data undergoes throughout the script.

Figure 11 highlights three distinct input layers with different characteristics, which are reflected in the distribution of reconstruction error. The spectral band ratio and magnetic data exhibit sharp, localized peaks in error, consistent with smaller-scale surface or near-surface anomalies, while the gravity survey shows broader, more regional error patterns, better aligning with the physical anomalies being searched for. These shows that the model is effective in identifying different types of physical anomalies rather than one geophysical anomaly. With error heatmaps created, the last step workflow is to generate a final output that consolidates the heat maps and highlights a specific region of interest for further analysis.

4.2.5 Output

The final output of the model is designed to display the locations of suspected geophysical anomalies based on the high error heatmaps. Another heatmap is generated which allows for a clear assessment of where an anomaly is most likely present.

To ensure that the most valuable data types have the greatest influence on the final output, a weighting function is applied to the error layers before they are combined into a single heatmap. Specifically, greater weight is given to the gravity and magnetic surveys, as these datasets probe deeper into the

subsurface and are more directly related to underlying geological structures than the spectral band data. This weighting approach is controllable, and it offers another way to refine the anomaly detection performance of the model.

Finally, the high error region heatmaps are combined into a single composite heatmap. This final output visualizes the spatial distribution of anomaly likelihood across the target region.

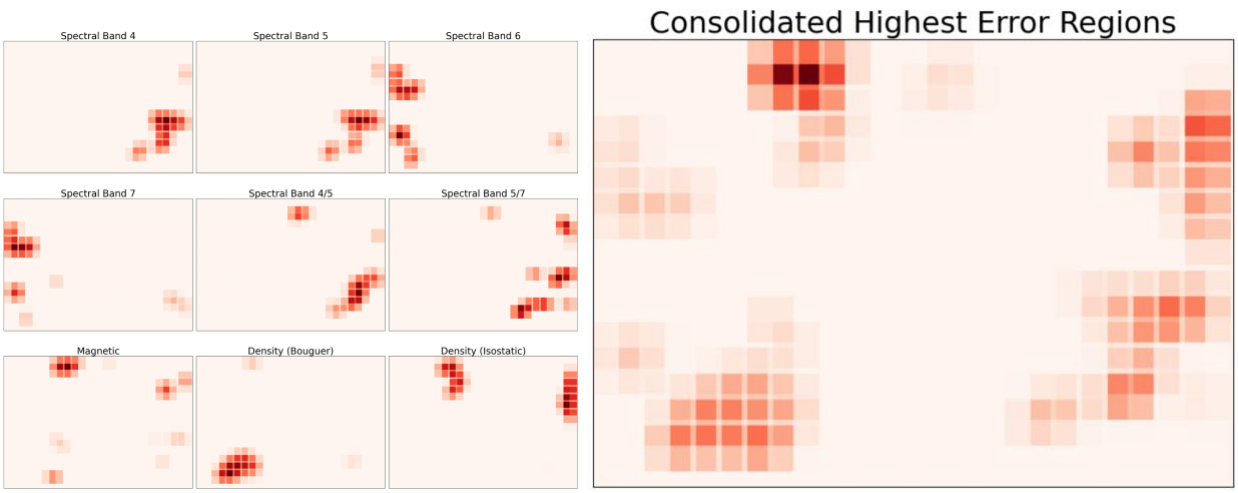


Figure 12 shows all nine before and after being consolidated. Some of the heatmaps show co-located error which suggest a higher likelihood of the detected anomaly being physically present.

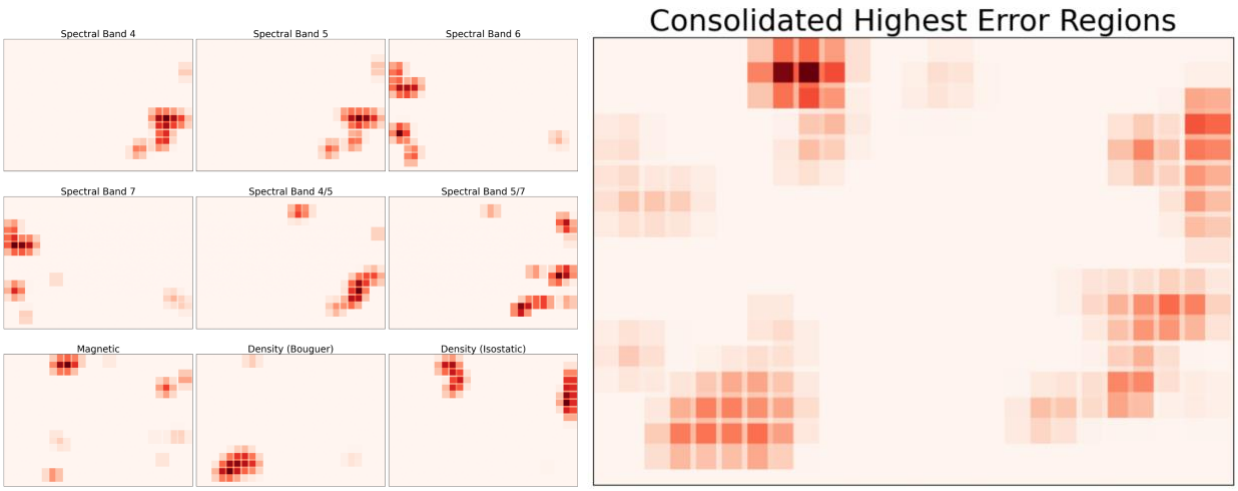


Figure 12 – Heatmap plots showing the predicted anomaly locations from each survey and the consolidated heatmap showing the anomaly location which the model predicts.

By examining the most intense areas of the heatmap, regions of consistently high error across multiple survey types can be identified. These are interpreted as the most probable locations of geophysical anomalies. In this data there are multiple potential anomalies detected, with the strongest signal coming from just left center near the top of the target region.

4.2.6 Output Validation

To validate the accuracy of the model's predictions, the final consolidated heatmap is compared against known mineral deposit locations obtained from the USGS database. This visual comparison is needed to determine whether the identified regions of high reconstruction error align with known deposits that are likely to generate geophysical anomalies, as code for this function was not designed.

In this case, the model's predicted anomaly location showed a poor spatial correlation with the location of known deposits. Figure 13 shows that within the top right quadrant, which was used as the target region during model development, there are deposits in the lower left section. While this aligns with the prediction from the Bouguer density data, the combination of all surveys led to an incorrect prediction. This result suggests that there may be some value workflow design, but significant design decisions must be changed.

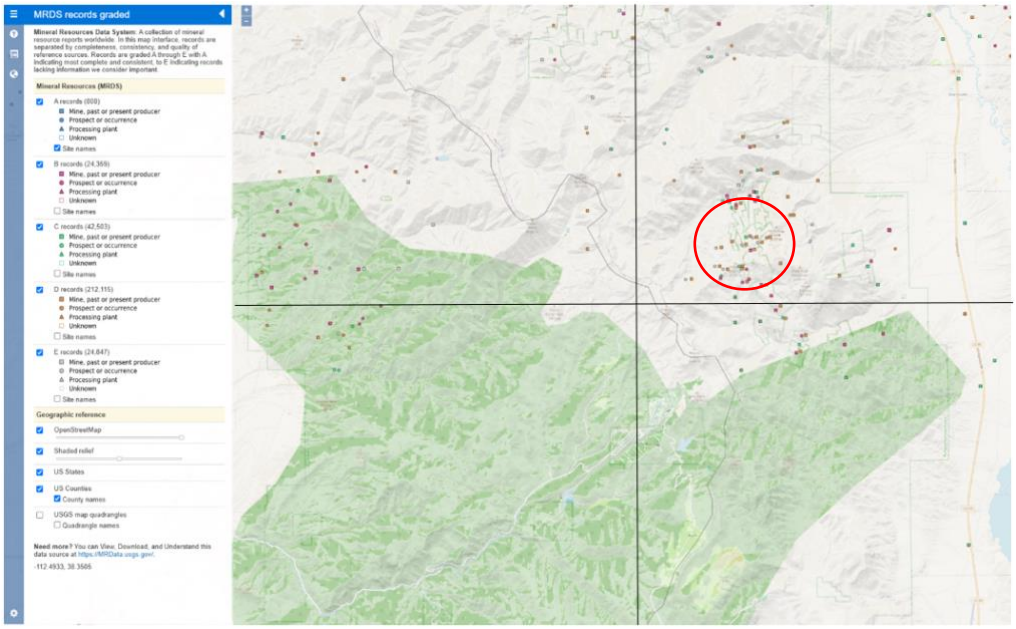


Figure 13 – USGS map of known mineral deposits in the southern Utah region that correlate with the outputs from the model. Known deposits circled in red correlate directly to Bouger density data.

4.3 User Interface

The user interface for this project was designed as a web-based application to facilitate the input, processing, and visualization of geophysical survey data. The interface serves as the primary point of interaction for geologists and geophysicists using the ML model to identify geophysical anomalies. It provides a streamlined workflow that allows users to upload survey data, select interpolation methods, and visualize processed outputs. The website is locally hosted as it runs on the user's local network

rather than being deployed on the internet. This allows for secure testing and development without requiring an external server.

4.3.1 Development

The website was built using Dash, a Python framework for creating web applications with interactive data visualizations. Dash was chosen due to its flexibility in integrating ML models, handling geospatial data, and providing a user-friendly experience without requiring extensive front-end development [14]. The core workflow for the interface, shown in Figure 14, was intentionally designed to be simple and intuitive to ensure ease of use. This simplicity allows geophysicists to quickly upload their data, process it through the model, and visualize potential anomalies freeing up their time to focus on more complex tasks that require expert geological interpretation.

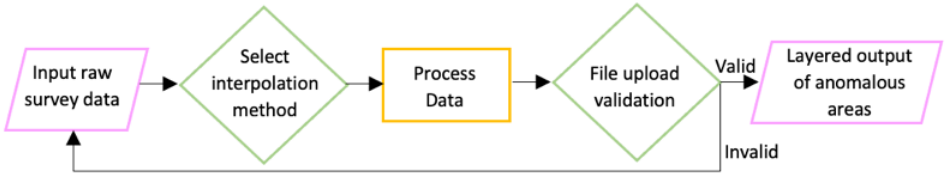


Figure 14 – Workflow diagram for user interface.

Table 8 compares Dash with Panel, another framework considered for the user interface. While Panel is well-suited for JupyterLab-based applications, Dash was ultimately chosen for its flexibility, ability to support multiple file uploads, and capability to function as a standalone web application.

Table 8 – Comparison of Panel and Dash for the user interface.

Options	Best For	Pros	Cons
Panel	JupyterLab apps	Simple, works in notebooks Supports CSV, PDF, and XYZ files Can display uploaded file content	Requires JupyterLab
Dash	Full web apps	More flexible, can be hosted online Allows uploading multiple files Can be hosted online Works outside Jupyter	Needs Flask server

Figure 15 below is the user interface which was designed to provide an interactive platform for processing geophysical survey data. The structure of the website was built using HTML components and interactive elements that allow users to upload files, select interpolation methods, process the data and

get layered outputs to visualize potential anomalies. The application layout is defined using html.Div(), which acts as a container for all elements on the page.

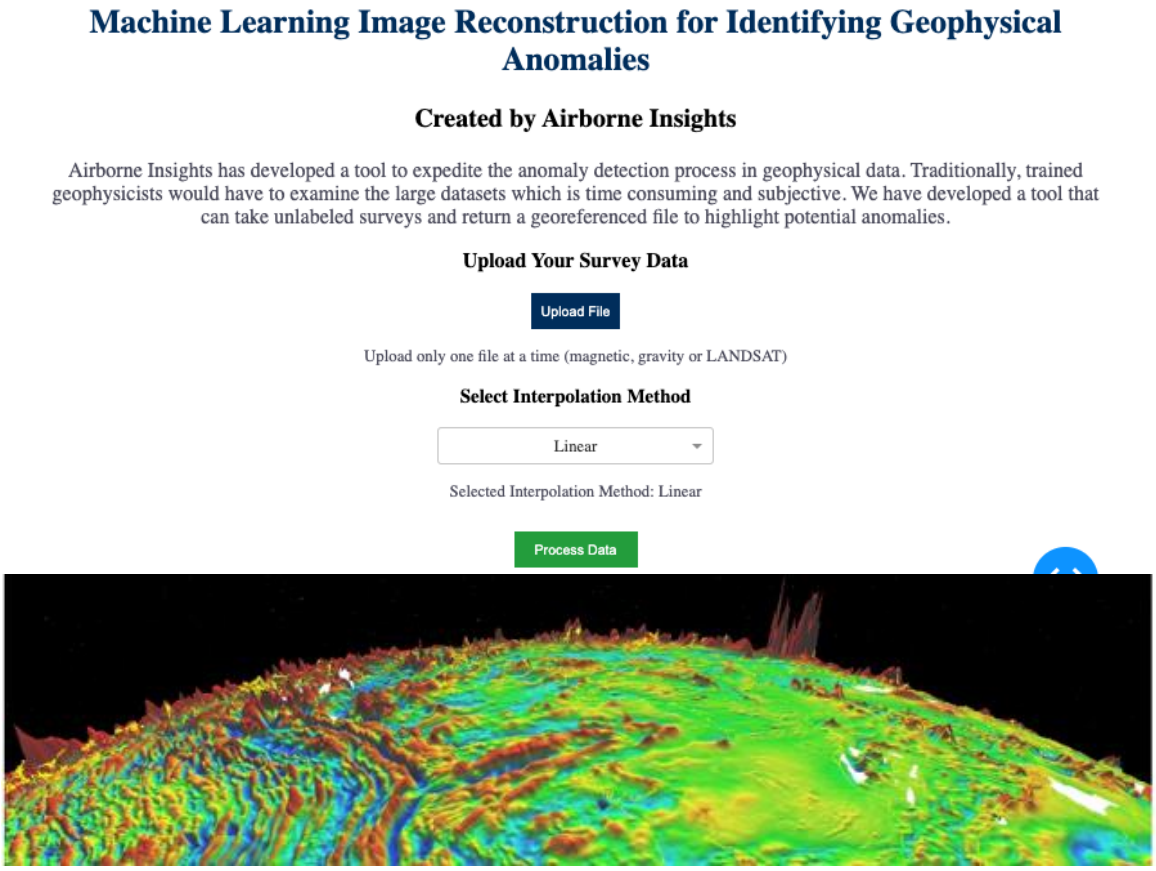


Figure 15 – User interface as seen on local website, showing the interactive layout for processing geophysical survey data with real-time feedback.

The backend of the user interface is designed to handle user input, file validation, and real-time updates. An interpolation method drop down allows users to choose between linear, cubic, or nearest neighbour, which is displayed based on user input. Users can upload geophysical survey data using a file upload component that supports .csv, .xyz, and .pdf formats. When the “Process Data” button is clicked, the system checks whether a valid file has been uploaded. If the file is missing or of an unsupported format, an error message is displayed in red, prompting the user to “Please upload a valid file before processing.”, see Figure 16. If the file is valid it is processed, and the results are displayed in an output section without requiring a page reload. The @app.callback functions ensure that changes in user input (such as file uploads or dropdown selections) trigger updates in the corresponding output sections. This backend logic enables responsive interactions and ensures a seamless experience for geophysicists working with various data formats and interpolation methods.

Figure 16 – User interface as seen on local website, illustrating system behavior: clicking the file upload button opens the user's local file directory, the interpolation method is selected from a dropdown menu, and an error message is displayed if processing is attempted without an uploaded file.

5.0 Environmental, Ethical, and Societal Considerations

As the integration of AI becomes more prominent in geological applications, it is essential to evaluate the technical performance of these systems and their broader environmental, ethical, and societal implications. This section outlines the sustainability of model training and data management practices, addresses key ethical concerns such as transparency, job displacement, and data privacy, and assesses how the project aligns with relevant United Nations Sustainable Development Goals (UNSDG). By proactively considering these factors, the project ensures that its development and deployment reflect responsible engineering practices and contribute positively to long-term environmental stewardship and social accountability.

5.1 Environmental Considerations

AI model training, particularly using deep learning methods such as autoencoders, can consume significant amounts of energy. Research shows that training large models can emit as much CO_2 as several cars over their lifetime [15]. To mitigate this, our code is stored and managed using GitHub, a cloud-based version control platform and model training and computation are performed on Google Cloud's carbon-neutral infrastructure for optimized sustainability. Equation 4 is used to estimate the carbon emitted from developing and training the model. Total carbon emissions were 1.35 kg of CO_2 equivalent which is the same as an average American car driving 5.45 km or 0.68 kgs of coal being burned [16].

$$power\ consumption\ x\ time\ x\ carbon\ produced\ based\ on\ the\ local\ power\ grid \quad [4]$$

$$250W\ x\ 180h = 45\ kWh\ x\ 0.03\ kg\ eq.\frac{CO_2}{kWh} = 1.35\ kg\ eq.CO_2$$

This is a low footprint compared to larger-scale ML projects where emissions can range from 300 000 – 600 000 kg of CO_2 for a natural language processing model which is equivalent to five times the lifetime emissions of an average American car [15].

5.2 Ethical and Societal Considerations

The integration of ML into geophysical anomaly detection introduces several ethical considerations. One potential concern is job displacement, as automation may reduce the demand for manual interpretation [17]. However, in this application a trained geophysicist remains essential to validate and interpret the model's outputs, ensuring that the system functions as a decision-support tool rather than a replacement for expert judgment. Privacy and data protection also present ethical challenges, particularly in the context of AI [17]. This project mitigates these risks by using only publicly accessible datasets for model training. While this approach significantly limits the potential consequences of a data breach, it is acknowledged that any compromise in data security could negatively affect stakeholder trust. The team recognizes the issue of transparency and explainability, as anomaly detection models often function as "black boxes" with limited visibility into decision-making processes [18]. This can easily be resolved by promoting open-source software and accompanying manuals or high-level code commenting to encourage adoption and progress in the field.

Common societal concerns associated with artificial intelligence such as bias and discrimination are acknowledged, but they are not directly applicable to this project in the same way they are in more sensitive domains such as healthcare or criminal justice, where biased outcomes can significantly impact individual lives and societal equity [18]. In this case, the model is applied to geophysical data, where the primary risk lies in misidentifying or overlooking subsurface anomalies rather than affecting human populations. Regardless, the project remains committed to transparency, responsible data handling, and ensuring that any limitations or uncertainties in the model's outputs are clearly communicated to end users.

5.3 UNSDG

In addition to addressing ethical, societal, and environmental concerns, this project aligns with several of the UNSDG. These goals serve as a global blueprint for achieving a better and more sustainable future, and the responsible application of ML in geophysical exploration supports multiple key objectives.

The project supports SDG 9: Industry, Innovation, and Infrastructure by promoting the integration of advanced technologies within the mineral exploration sector [19]. The use of ML for anomaly detection encourages modernization and innovation, driving the development of more efficient and data-driven approaches to resource discovery. This not only enhances operational efficiency but also reduces unnecessary fieldwork and associated costs.

In line with SDG 12: Responsible Consumption and Production, the project contributes to more sustainable exploration practices [19]. By narrowing down high-probability targets through geophysical data analysis, it reduces the need for widespread exploratory drilling and associated environmental disruption. This targeted approach promotes the responsible use of natural resources and helps minimize the environmental footprint of early-stage exploration.

The project also contributes to SDG 13: Climate Action by actively considering the carbon footprint associated with model training [19]. With an estimated energy usage of only 45 kWh and emissions of 1.35 kg CO_2 equivalent, the project maintains a relatively low environmental impact. By exploring options for sustainable computing infrastructure, such as green cloud platforms or energy-efficient training strategies, the project supports global efforts to mitigate climate change.

This approach also supports SDG 15: Life on Land by enabling less invasive mineral exploration [19]. Traditional exploration methods can lead to significant land disturbance, whereas a model-driven approach focuses attention on the most promising areas, therefore preserving ecosystems and reducing unnecessary surface impacts. This reflects a broader commitment to sustainable land use and biodiversity protection.

6.0 Cost Benefit Analysis

ML is not yet a fully feasible replacement for manual geophysical interpretation. Expert geophysicists are still required to review error plots, validate outputs, and interpret the underlying data. ML models also cannot confirm the economic viability of anomalies and therefore exploration and drilling are still necessary to verify the presence of mineralization. ML algorithms can uncover hidden patterns across multiple surveys, but it may not always be clear why these correlations exist or what they represent which is why expert interpretation remains critical. However, ML integration does reduce reliance on extensive manual labor. Automated anomaly detection decreases inspection time and allows geophysicists to focus on higher-level analysis and decision-making, which can contribute to overall cost savings in geophysical surveys.

Studies using deep learning for anomaly detection in subsea oil and gas pipelines have shown potential in identifying features like cracks, but these models face significant limitations due to the large volume of unlabeled data, which still requires domain-specific expertise [20]. This is the same issue in mineral exploration. With the large amount of unlabeled data there is also the issue of costly high-quality geophysical data. Much of the airborne USGS survey data was collected in the 1960’s, and while UAV

acquired data is making updated datasets more accessible and affordable, issues of data quality and consistency remain [21]. Computational constraints are also relevant with survey images often being too large and require substantial processing power. Many neural network architectures must resize or downscale these images to improve processing speed, which can result in the loss of critical geophysical information [22]. Table 9 outlines a comparison of key metrics between manual and ML assisted workflows. The cost per project assumes a one-year duration, with a geophysicist earning the average Canadian salary and the ML model requiring high-latency infrastructure and complex algorithmic support [23] [24].

Table 9 – Cost benefit analysis between manual interpretation and a ML model for identifying geophysical anomalies.

Metric to compare	Geophysicist	Machine Learning
Initial set up	Low	High
Time per project	High	Low
Accuracy	High (expert)	Medium-High (with verification)
Cost per project	$90 000 [23]	$10 000 [24]
Scalability	Low	High
Repeatability	Low	High

7.0 Recommendations

With the current model demonstrating the potential of autoencoders for geophysical anomaly detection, there remains significant room for improvement and optimization. To enhance model performance, increase adaptability, and support broader application, the following next steps are recommended:

- More geophysical data consider IP, EM, seismic, Ground penetrating rader or radiometric data
- Better resolution density and magnetic surveys.

To improve upon the machine learning script, the following next steps are recommended:

- Separate geophysical surveys that target different depths.
- Assess the impacts of varying patch size and overlap on model accuracy
- Consider the impacts of normalizing survey data between the minimum and maximum possible readings on the instruments, as opposed to just the available data.
- Add more dense layers into the construction of the autoencoder.
- Have known anomalies as a target, instead of just known deposits.
- Analyze the impact of calculating error on a patch-by-patch basis.
- Explore more error calculation methods.

- Automate the output validation process to give an accuracy score.

8.0 Conclusion

This project developed and evaluated an unsupervised ML pipeline for geophysical image reconstruction and anomaly detection, using a custom-designed autoencoder architecture trained on magnetics, gravity, and Landsat spectral data. The pipeline integrated extensive preprocessing, including standardized interpolation, spectral normalization, patch-based segmentation, and dynamic latent dimension scaling. Despite the theoretical strengths of this architecture, particularly in compressing complex spatial inputs and isolating anomalous deviations via reconstruction error, the results demonstrate clear limitations in practical applicability and reliability.

Although the model was able to reconstruct survey data with acceptable accuracy and produce structured error maps, its predictive performance did not consistently correlate with known mineral occurrences. Even after weighting gravity and magnetic data more heavily in the loss function and post-processing steps, the final heatmaps failed to reliably identify true anomaly regions when all inputs were combined. This suggests that the model remains highly sensitive to overrepresented features, particularly spectral bands, and lacks the domain awareness necessary to prioritize geologically meaningful patterns over high variance but non-informative signals.

This work highlights a key insight about current ML methods, particularly unsupervised architectures, are not sufficient by themselves for geophysical anomaly detection. Without embedded geological context or expert-informed constraints, the model can identify statistical irregularities but cannot assess their geological plausibility or economic relevance. This confirms that anomaly detection in mineral exploration will likely always require human validation. A trained geophysicist remains essential, not only to verify and interpret model outputs but to guide preprocessing decisions, contextualize results within broader geological frameworks, and recognize the many forms of survey noise, data artifacts, or false positives that ML cannot reliably differentiate on its own.

From a systems design perspective, the development of the user interface and model infrastructure demonstrates that AI-based tools can support and streamline interpretation workflows. However, positioning these models as assistive tools not autonomous decision-makers, is both more realistic and more aligned with current and foreseeable limitations in geophysical AI applications.

In terms of scalability, environmental impact, and accessibility, the project still offers significant benefits. With a minimal carbon footprint (about 1.35 kg CO_2 equivalent), the model training process aligns with

responsible computing practices, and the deployment of a flexible, locally hosted interface shows promise for eventual integration into field workflows. The project also establishes a generalizable pipeline for integrating and preparing geophysical data for ML, which can serve as a foundation for future research in hybrid or semi-supervised approaches.

Moving forward, the team recommends shifting the focus from full anomaly detection autonomy toward robust, interpretable decision-support tools. Improvements such as incorporating expert-defined anomalies, adding supervised elements, increasing geophysical modal diversity (IP, EM or seismic), and developing transparent error explanation mechanisms could enhance the model's utility in professional contexts. Ultimately, the goal should not be to replace geophysicists but to strengthen their capacity with tools that enhance efficiency, repeatability, and exploratory insight while respecting the irreplaceable value of geological expertise.

9.0 Team Signatures

Our signatures below attest that this submission is our original work.

Following professional engineering practice, we bear the burden of proof for original work. We have read the Policy on Academic Integrity posted on the Smith Engineering website (https://smithengineering.queensu.ca/policy/Honesty.html) and confirm that this work is in accordance with the Policy.

Individual 1:

Signature:		Date:	04/18/25
Name:	Kathryn Lees	ID #:	20216152

Individual 4:

Signature:		Date:	04/18/25
Name:	Max Follett	ID #:	20200766

Individual 5:

Signature:		Date:	04/18/25
Name:	Elliot Carusone	ID #:	20218549

References

[1] B. Goss, “What are the Limitations of Geophysical Surveys? | Rangefront,” Rangefront Mining Services. Accessed: Mar. 28, 2025. [Online]. Available: https://rangefront.com/blog/the-limitations-of-geophysical-surveys-in-mineral-exploration/

[2] “What Is Artificial Intelligence (AI)? | IBM.” Accessed: Mar. 28, 2025. [Online]. Available: https://www.ibm.com/think/topics/artificial-intelligence

[3] “What Is Unsupervised Learning? | IBM.” Accessed: Mar. 28, 2025. [Online]. Available: https://www.ibm.com/think/topics/unsupervised-learning

[4] R. S. Davies, M. Trott, J. Georgi, and A. Farrar, “Artificial intelligence and machine learning to enhance critical mineral deposit discovery,” *Geosystems and Geoenvironment*, vol. 4, no. 2, p. 100361, May 2025, doi: 10.1016/j.geogeo.2025.100361.

[5] S. Canada, “The Canadian Critical Minerals Strategy.” Accessed: Mar. 28, 2025. [Online]. Available: https://www.canada.ca/en/campaign/critical-minerals-in-canada/canadian-critical-minerals-strategy.html

[6] E. and C. C. Canada, “Canada’s Climate Plan.” Accessed: Apr. 15, 2025. [Online]. Available: https://www.canada.ca/en/environment-climate-change/campaigns/climate-change-plan.html

[7] A. Kerr, “Critical Minerals in the Context of Canada: Concepts, Challenges and Contradictions,” *Geoscience Canada*, vol. 50, no. 3, Art. no. 3, Oct. 2023, doi: 10.12789/geocanj.2023.50.199.

[8] Z. Luo, Y. Xiong, and R. Zuo, “Recognition of geochemical anomalies using a deep variational autoencoder network,” *Applied Geochemistry*, vol. 122, p. 104710, Nov. 2020, doi: 10.1016/j.apgeochem.2020.104710.

[9] “World Energy Outlook 2020 – Analysis,” IEA. Accessed: Apr. 15, 2025. [Online]. Available: https://www.iea.org/reports/world-energy-outlook-2020

[10] C. A. P, “Enhancement of Landsat Thematic Mapper imagery for residual soil mapping in SW Minas Gerais State Brazil, a prospecting case history in greenstone belt terrain,” *Proceedings of the 7^<th> ERIM Thematic Conference on Remote Sensing for Exploration Geology, 1989*, 1989, Accessed: Apr. 15, 2025. [Online]. Available: https://cir.nii.ac.jp/crid/1571417125462433536

[11] W. D. Sinclair, “Porphyry Deposits in Mineral Deposits of Canada, GAG-MDD Special Publication,” *Geological Survey of Canada*, vol. 5, pp. 233–244, Jan. 2007.

[12] L. A. Morgan, “7. Geophysical Characteristics of Volcanogenic Massive Sulfide Deposits”.

[13] L. D. Meinert, “Skarns and Skarn Deposits,” *Geoscience Canada*, vol. 19, no. 4, Dec. 1992, Accessed: Nov. 14, 2024. [Online]. Available: https://journals.lib.unb.ca/index.php/GC/article/view/3773

[14] “Comparing Panel and Dash — Panel v1.6.1.” Accessed: Mar. 25, 2025. [Online]. Available: https://panel.holoviz.org/explanation/comparisons/compare_dash.html

[15] J. Koch, “Curbing the Carbon Footprint of Machine Learning | Inria.” Accessed: Apr. 13, 2025. [Online]. Available: https://www.inria.fr/en/carbon-footprint-machine-learning-hci

[16] “Machine Learning CO2 Impact Calculator.” Accessed: Apr. 13, 2025. [Online]. Available: https://mlco2.github.io/impact

[17] V. Tadi, “Establishing Ethical Frameworks And Best Practices For Fair And Transparent Machine Learning-driven Anomaly Detection Systems,” *International Journal of Engineering Management*, vol. 6, pp. 33–45, Dec. 2019.

[18] C. Pazzanese, “Ethical concerns mount as AI takes bigger decision-making role,” Harvard Gazette. Accessed: Apr. 13, 2025. [Online]. Available: https://news.harvard.edu/gazette/story/2020/10/ethical-concerns-mount-as-ai-takes-bigger-decision-making-role/

[19] “THE 17 GOALS | Sustainable Development.” Accessed: Apr. 13, 2025. [Online]. Available: https://sdgs.un.org/goals

[20] M. Brebbia, “AI in Geophysical Surveys: Enhancing Data Acquisition, Processing, and Reporting,” Medium. Accessed: Apr. 11, 2025. [Online]. Available: https://massimobrebbia.medium.com/ai-in-geophysical-surveys-enhancing-data-acquisition-processing-and-reporting-f889ef0f522a
[21] D. B. Hoover, D. P. Klein, and D. C. Campbell, “Geophysical Methods In Exploration and Mineral Environmental Investigations,” Aug. 2016.
[22] R. Spahić, K. Poolla, V. Hepsø, and M. A. Lundteigen, “Image-based and risk-informed detection of Subsea Pipeline damage,” *Discov Artif Intell*, vol. 3, no. 1, p. 23, Jun. 2023, doi: 10.1007/s44163-023-00069-1.
[23] “Geophysicist salary in Canada.” Accessed: Apr. 11, 2025. [Online]. Available: https://ca.indeed.com/career/geophysicist/salaries
[24] V. Shashkina, “Machine Learning (ML) Costs: Price Factors and Real-World Estimates,” ITRex. Accessed: Apr. 11, 2025. [Online]. Available: https://itrexgroup.com/blog/machine-learning-costs-price-factors-and-estimates/

Appendix A: Tables and Figures

Table 10 – Metadata Legend from the USGS magnetic database.

1	2	3	4	5	6	7	8	9
Line (alphanumeric Value)	**Fiducial Number**	**Time**	**Day**	**Year**	**Latitude (decimal Degrees)**	**Longitude (decimal degrees)**	**Radaralt (m)**	**totmag (nanoTeslas)**
flight line number, often assigned by airborne survey crew	A fiducial number is a mark which indicates points of simultaneity. It is a user-defined integer used during airborne operations to correlate recording devices (magnetometers) with navigational records (altimeters, camera film, strip charts) that were recorded at the same time.	time of data point collection. Time is given in either local time or Greenwich Meridian Time. Values of 999999 indicate missing data	Julian day, where Jan. 1 = 1 and Dec. 31 = 365 or 366 (leap year)	year of data point collection (Source: self evident)	latitude - geographic coordinate	longitude - geographic coordinate	radar altimetry reading, in meters An aircraft navigational system in which short electromagnetic waves are transmitted, and the energy scattered back by reflection is detected. From this measurement, the distance between the aircraft and the ground is calculated and recorded as radar altimetry. Values of -999.9 indicate missing data.	Raw magnetic value with base magnetometer corrections applied. Because the magnetic field of the earth varies diurnally, a stationary base magnetometer is maintained on the ground during airborne surveying. Frequency of measurement: The magnetometer has a recharging interval of 0.5 seconds. The data were recorded at 1 second intervals. The base magnetometer records changes in the magnetic field (in nanoTeslas) as a function of time. The magnetic changes may have an amplitude of 20 to 50 nanoTeslas. If changes are more severe, as would occur from a magnetic storm, surveying is discontinued or the data recorded are not used. Diurnal variations are then removed from the airborne magnetic data based on
10		**11**			**12**			**13**
resmag (nanoTeslas)		**Diurnal (nanoteslas)**			**Geology**			**ResmagCM4**
residual magnetic value The total magnetic value minus a geomagnetic reference field (GRF), which is a long-wavelength regional magnetic field. The most commonly used reference field is determined from a model developed by the International Association of Geomagnetism and Aeronomy (IAGA). The International Geomagnetic Reference Field (IGRF), is a predictive model adopted at the beginning of a model period (e.g. in 1989 for 1990-1995). After the model period, a revised definitive model is adopted, the DGRF. This is the preferred model to use for removing regional magnetic fields for epochs when CM models are not available. For this survey, the field removed is IGRF 1975. Some contractors add a constant to the residual value in order to avoid negative values. This constant can vary from one data set to another. Values of -9999.9 indicate missing data.		Base magnetometer values used to correct the total magnetic value. Because the magnetic field of the earth varies diurnally, a stationary base magnetometer is maintained on the ground during airborne surveying. The base magnetometer records changes in the magnetic field (in nanoTeslas) as a function of time called the diurnal. The magnetic changes may have an amplitude of 20 to 50 nanoTeslas. The diurnal is monitored during the survey and if changes are severe, as would occur from a magnetic storm, surveying is discontinued or the data recorded are not used. Diurnal variations are then removed from the airborne magnetic data based on the common time. Values of -9999.9 indicate missing data. Frequency of measurement: The base magnetometer has a recharging interval of 0.5 seconds. The data were recorded at 2-4 second intervals.			Geologic identification codes or numeric codes indicating the rock types immediately beneath the flight line. These codes were determined by plotting the flight lines on a surficial geologic map. A description of the codes can be found in the GJBX- or GJO- report for this quadrangle listed in the Cross Reference section above. Values of -99 indicate missing data.			The total magnetic field value minus the core component of geomagnetic field is known as the comprehensive model (CM). CM4 (the fourth generation of CM) is the modern reference field determined from a long-wavelength spatially and temporally continuous model of the geomagnetic field that simultaneously considers the Earth's core, lithospheric, and external magnetic fields (external ionospheric and magnetospheric fields and the components induced by them inside the Earth). Spherical harmonic degree 13 of CM4 is the preferred model to use for removing regional magnetic fields for the time periods it is valid for (1960 to June 2002). Values of -9999.9 indicate missing data. range of values were not determined

Table 11 – Gantt chart for 2024/25 project timeline.

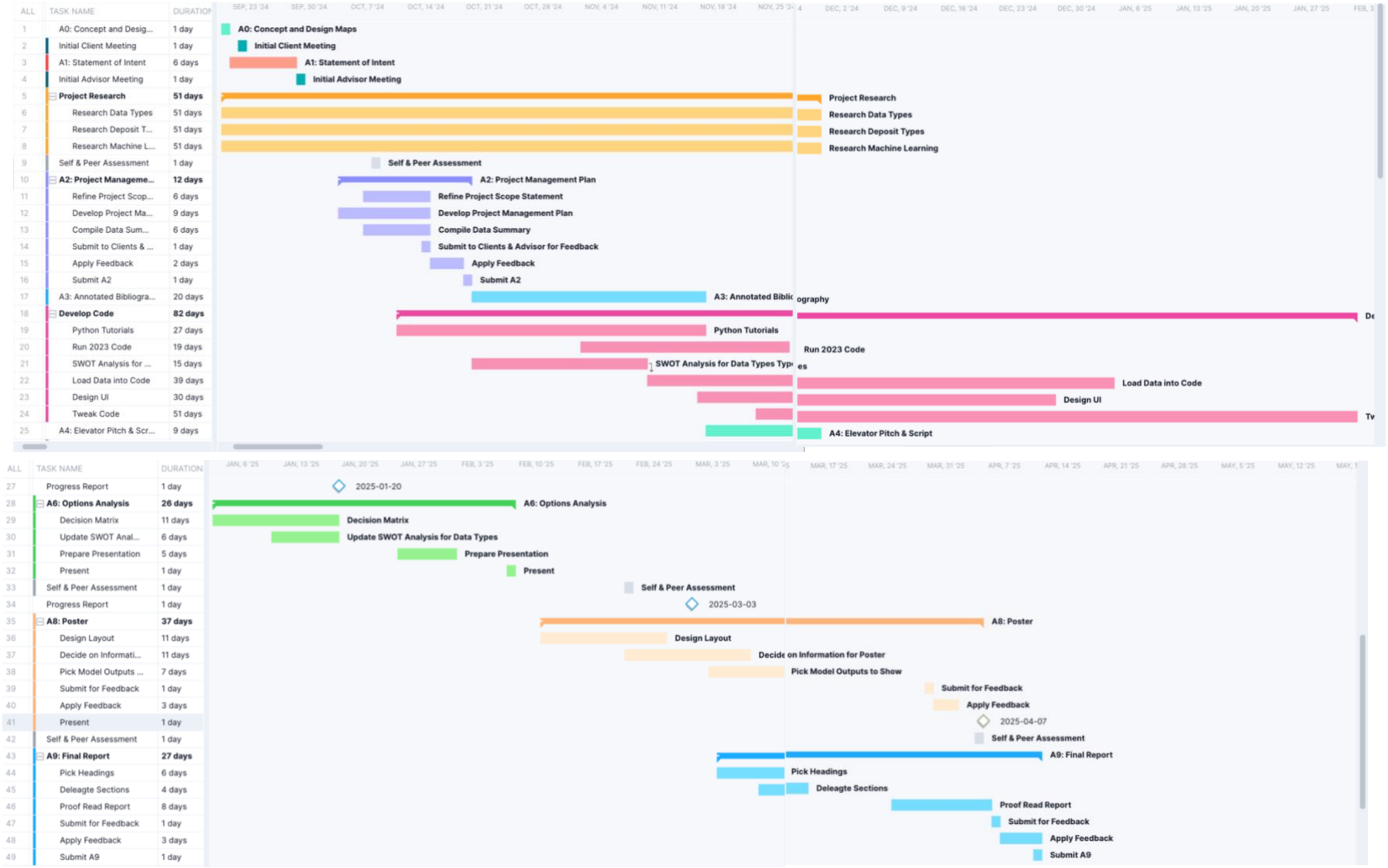


Table 12 – Budget compliance in terms of tasks, individual progress, hours spent, status and completeness.

	A	B	C	D	E	F	G
1	Tasks	Description	Allocated Hours			Status (%)	Complete (Y/N)
2			Kathryn	Max	Elliot		
3	**Concept and Design Map**						
4		Brainstorm ideas	1	1	1		
5	**Statement of Intent**						
6	Design Team Logo	Hand drew Airborne Insights logo	2			100%	Y
7	Project Goals	Decide what goals we want to achieve by April 2025	1	1	1	100%	Y
8	Deliverables and Constraints	Decipher challenges we will face throught the project	3			100%	Y
9	Preliminary Options Analysis	Identify design options for model		3		100%	Y
10	Impact of Project	Research impacts to the climate, Indigenous peoples and ecosystems			3	100%	Y
11	USGS Goals	Research SDGs related to project	1	1	3	100%	Y
12	Qualifications	Undergrad classes that prepared us for the project and previous work experience	0.5	0.5	0.5	100%	Y
13	Project Advisors/Clients	Why and how these people will be useful to the project	0.5	0.5	0.5	100%	Y
14	**Project Management Plan**						
15	Project Management Plan	Schedule, Gantt Chart, Document Management Plan, Research Plan, Bibliogrpahy Standard, Lit Review Plan, Poster Plan, Transmittal Letter & Summary	13			100%	Y
16	Project Scope	Project Objectives, Options Analysis, Scope Statement		3		100%	Y
17	Modelling/Testing Plan	Data Collection, Preprocessing, Evaluating Testing, Iteration		5		100%	Y
18	Data Summary	Data Resolution and Quality, Scale of Data, Types of Data			7	100%	Y
19	**Annotated Bibliography**						
20	Purpose of the Project	Researched the Government of Canada's critical minerals, economic minerals, and the best commodities to train the machine learning model to look for	6	8		100%	Y
21	Geological Deposit Types	Researched VMS, porphyry Cu-Au and skarn deposits.	17			100%	Y
22	Geophysical Interpretation	Researched advanced geophysical techniques for mineral exploration using magnetic and gravity data.		12		100%	Y
23	Machine Learning Models Ideal for Image Reconstruction	Researched machine learning and data processing aspect of project.		2	12	100%	Y
24	**Elevator Pitch**						
25	Script	script	5	5	5	100%	Y
26	Presentation Slide	Created slide and visuals	3			100%	Y
27	**Ongoing Project Tasks**						
28	Client Meetings	Meet with IBM to ask questions and get advice on next steps	10	10	10	100%	Y
29	Advisor Meetings	Meet with Dr. Fotopoulos to go over progress	10	10	10	100%	Y
30	Team Meetings	Meet as a team to work on, review, and edit deliverables, and assist eachother with individual tasks	48	48	48	100%	Y
31	Capstone Class	Workshops and guest speakers	26	26	26	100%	Y
32	**Options Analysis**						
33	Machine Learning Model	Supervised vs Unsupervised, autoencoder	2	2	2	100%	Y
34	Data	Gravity, magnetic, LANDSAT	2	2	2	100%	Y
35	Weigh Options	Decide on a model and data to feed model	1	1	1	100%	Y

36	Presentation	Create presentation	3	3	3	100%	Y
37	**Design Development**						
38	UI Flowchart	How the UI will be designed	15			100%	Y
39	UI Website	Code a webiste for the UI for clients to use (not done)	40			100%	Y
40	Preprocessing Data	Coding		60		100%	Y
41	Machine Learning Model	Coding	10	40	80	100%	Y
42	Iterations of Model	Refining code			32	100%	Y
43	Progress Report	Write progress report to update clients	5	5	5	100%	Y
44	**Final Deliverable**						
45	Writing	Writing	12	5	3	100%	Y
46	Poster	Create poster	6	2	2	100%	Y
47	Poster Presentation	Create poster presentation	6	1	1	100%	Y
48	Final Report	Write final report using all previous work and deliver MVP to clients	11	6	6	100%	Y
49	Total		**260**	**263**	**264**	100%	
50							
51	Projected Work (10hrs/week)	630					
52	Projected Cost for Work ($120/hr)	$75,600					
53	**Cumulative Hours**	**787**					
54	**Final Projected Cost of Work**	**$94,440**					
55							

Appendix B: Code

B.1 Preprocessing

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
import os
import requests
from io import StringIO

# Define constants
file_name = "richfield_mag.xyz"
Survey_name = "Richfield, Utah"

# Get user's desktop path
desktop_path = os.path.join(os.path.expanduser("~"), "Desktop")
output_folder = os.path.join(desktop_path, "magnetic_txt_files")
os.makedirs(output_folder, exist_ok=True)

# Function to load CSV data from GitHub
def load_csv_from_github():
    url = f'https://github.com/maxfollett/AirBorneInsight2/raw/main/CapDatabases/Raw/{file_name}'
    response = requests.get(url)

    if response.status_code == 200:
        csv_data = response.text
        df = pd.read_csv(StringIO(csv_data), header=None, sep='\\s+', usecols=[5, 6, 12],
                         names=["lat", "long", "corrected_magnetic"])
        return df
    else:
        print(f"Failed to load data, status code: {response.status_code}")
        return None

# Function to perform interpolation
def perform_interpolation_with_extrapolation(df, grid_size=1114):
    grid_x, grid_y = np.mgrid[df['long'].min():df['long'].max():grid_size*1j,
                              df['lat'].min():df['lat'].max():grid_size*1j]
    points = np.column_stack((df['long'], df['lat']))
    values = df['corrected_magnetic'].values

    grid_z = interpolate.griddata(points, values, (grid_x, grid_y), method='linear')
    nan_mask = np.isnan(grid_z)
    grid_z[nan_mask] = interpolate.griddata(points, values, (grid_x[nan_mask], grid_y[nan_mask]), method='n

    return grid_x, grid_y, grid_z

# Function to save data to a .txt file
def save_to_txt(grid_z, lat_input, long_input):
    filename = f"MAG_{lat_input}_{long_input}.txt"
    filepath = os.path.join(output_folder, filename)
    np.savetxt(filepath, grid_z, fmt='%.6f')
    print(f"Saved file: {filepath}")

# Main function
def main():
```

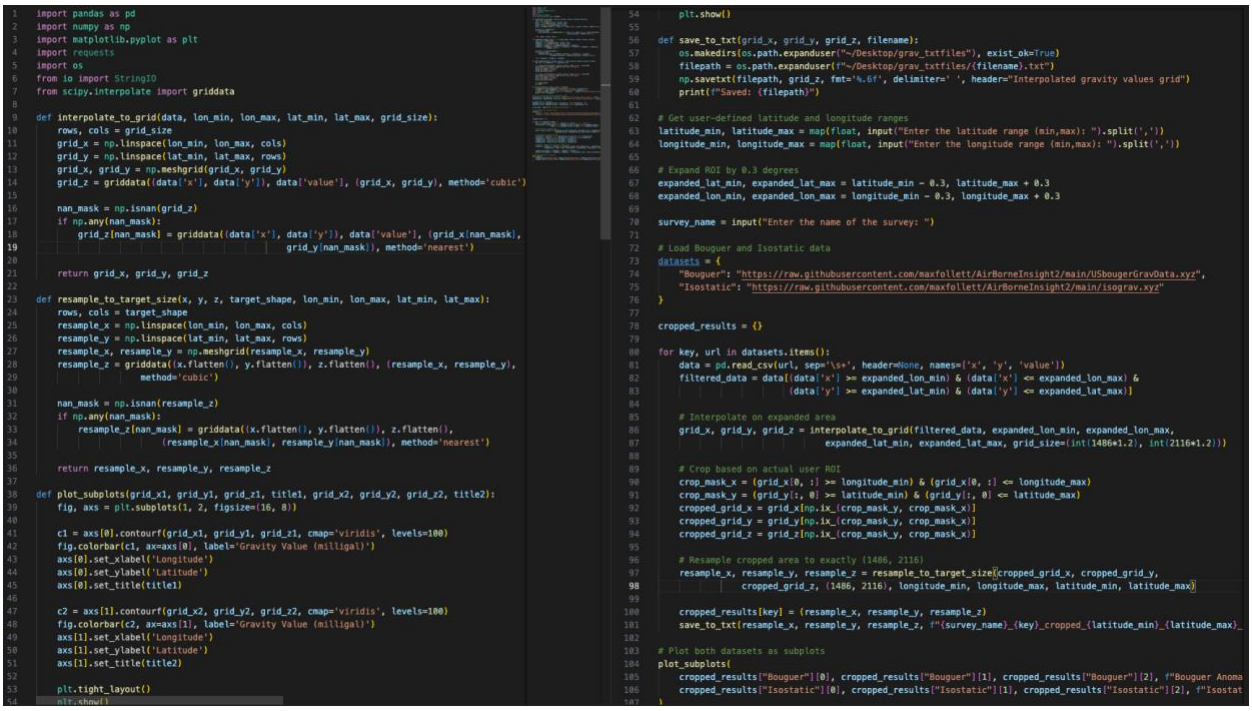
```python
def perform_interpolation_with_extrapolation(df, grid_size=1114):
    return grid_x, grid_y, grid_z

# Function to save data to a .txt file
def save_to_txt(grid_z, lat_input, long_input):
    filename = f"MAG_{lat_input}_{long_input}.txt"
    filepath = os.path.join(output_folder, filename)
    np.savetxt(filepath, grid_z, fmt='%.6f')
    print(f"Saved file: {filepath}")

# Main function
def main():
    df = load_csv_from_github()
    if df is None:
        return

    try:
        lat_input = float(input("Enter latitude: "))
        long_input = float(input("Enter longitude: "))
    except ValueError:
        print("Invalid input. Please enter numeric values.")
        return

    df_filtered = df[(df['lat'].between(lat_input - 0.1, lat_input + 0.1)) &
                     (df['long'].between(long_input - 0.1, long_input + 0.1))]

    if df_filtered.empty:
        print("No data found for the given latitude and longitude range.")
        return

    grid_x, grid_y, grid_z = perform_interpolation_with_extrapolation(df_filtered)
    save_to_txt(grid_z, lat_input, long_input)

if __name__ == "__main__":
    main()
```

Magnetic Data preprocessing

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
import os
from io import StringIO
from scipy.interpolate import griddata

# Function to load XYZ file from GitHub repository
def load_xyz_from_github(url):
    response = requests.get(url)
    if response.status_code == 200:
        xyz_data = response.text
        cleaned_data = []

        # Track malformed lines
        malformed_lines = []

        # Process each line
        for i, line in enumerate(xyz_data.splitlines(), start=1):
            columns = line.split()
            if len(columns) == 3:  # Keep valid lines
                cleaned_data.append(" ".join(columns))
            else:
                malformed_lines.append((i, line))

        # Log malformed lines
        if malformed_lines:
            print(f"Found {len(malformed_lines)} malformed lines. Skipping these:")
            for idx, malformed in malformed_lines[:10]:  # Display first 10 errors
                print(f"Line {idx}: {malformed}")
            print("... (skipping remaining malformed lines)")

        # Join cleaned data
        cleaned_xyz_data = "\n".join(cleaned_data)

        # Load into pandas
        try:
            df = pd.read_csv(StringIO(cleaned_xyz_data), sep='\s+', header=None,
                             names=['x', 'y', 'value'])
            return df
        except Exception as e:
            print(f"Error reading the cleaned data: {e}")
            return None
    else:
        print(f"Failed to load data, status code: {response.status_code}")
        return None

# Function to get user input for the range
def get_range_input(prompt):
    while True:
        try:
            user_input = input(prompt)
            range_values = user_input.split(',')
```

```python
def interpolate_to_grid(data, longitude_min, longitude_max, latitude_min, latitude_max):
            (data['x'], data['y']), data['value'], (grid_x[nan_mask], grid_y[nan_mask]), method='nearest'
        )

    return grid_x, grid_y, grid_z

# Function to plot interpolated data
def plot_data(grid_x, grid_y, grid_z, title):
    plt.figure(figsize=(12, 10))
    plt.contourf(grid_x, grid_y, grid_z, cmap='viridis', levels=100)
    plt.colorbar(label='Gravity Value (milligal)')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.title(title)
    plt.show()

# Function to save data as .txt file
def save_to_txt(grid_x, grid_y, grid_z, filename):
    os.makedirs(os.path.expanduser("~/Desktop/grav_txtfiles"), exist_ok=True)
    filepath = os.path.expanduser(f"~/Desktop/grav_txtfiles/{filename}.txt")

    np.savetxt(filepath, grid_z, fmt='%.6f', delimiter=' ', header="Interpolated gravity values grid")
    print(f"Saved: {filepath}")

# GitHub raw URLs
url1 = "https://raw.githubusercontent.com/maxfollett/AirBorneInsight2/main/USbougerGravData.xyz"
url2 = "https://raw.githubusercontent.com/maxfollett/AirBorneInsight2/main/isograv.xyz"

# Get user-defined latitude and longitude ranges
latitude_min, latitude_max = get_range_input("Enter the latitude range (min,max) (e.g., 40,42) for both plots: ")
longitude_min, longitude_max = get_range_input("Enter the longitude range (min,max) (e.g., -114,-112) for both plots: ")

# Get survey name
survey_name1 = input("Enter the name of the survey: ")

# Load, process, and plot Bouguer gravity data
data1 = load_xyz_from_github(url1)
if data1 is not None:
    filtered_data1 = data1[(data1['x'] >= longitude_min) & (data1['x'] <= longitude_max) &
                           (data1['y'] >= latitude_min) & (data1['y'] <= latitude_max)]
    if not filtered_data1.empty:
        grid_x, grid_y, grid_z = interpolate_to_grid(filtered_data1, longitude_min, longitude_max, latitude_min, latitude_m
        plot_data(grid_x, grid_y, grid_z, f"Bouguer Anomaly Map for {survey_name1}")
        save_to_txt(grid_x, grid_y, grid_z, f"{survey_name1}_Bouguer_{latitude_min}_{latitude_max}_{longitude_min}_{longitu

# Load, process, and plot Isostatic gravity data
data2 = load_xyz_from_github(url2)
if data2 is not None:
    filtered_data2 = data2[(data2['x'] >= longitude_min) & (data2['x'] <= longitude_max) &
                           (data2['y'] >= latitude_min) & (data2['y'] <= latitude_max)]
    if not filtered_data2.empty:
        grid_x, grid_y, grid_z = interpolate_to_grid(filtered_data2, longitude_min, longitude_max, latitude_min, latitude_m
        plot_data(grid_x, grid_y, grid_z, f"Isostatic Anomaly Map for {survey_name1}")
```

Density Data Preprocessing



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
import os
from io import StringIO
from scipy.interpolate import griddata

def interpolate_to_grid(data, lon_min, lon_max, lat_min, lat_max, grid_size):
    rows, cols = grid_size
    grid_x = np.linspace(lon_min, lon_max, cols)
    grid_y = np.linspace(lat_min, lat_max, rows)
    grid_x, grid_y = np.meshgrid(grid_x, grid_y)
    grid_z = griddata((data['x'], data['y']), data['value'], (grid_x, grid_y), method='cubic')

    nan_mask = np.isnan(grid_z)
    if np.any(nan_mask):
        grid_z[nan_mask] = griddata((data['x'], data['y']), data['value'], (grid_x[nan_mask],
                                                grid_y[nan_mask]), method='nearest')

    return grid_x, grid_y, grid_z

def resample_to_target_size(x, y, z, target_shape, lon_min, lon_max, lat_min, lat_max):
    rows, cols = target_shape
    resample_x = np.linspace(lon_min, lon_max, cols)
    resample_y = np.linspace(lat_min, lat_max, rows)
    resample_x, resample_y = np.meshgrid(resample_x, resample_y)
    resample_z = griddata((x.flatten(), y.flatten()), z.flatten(), (resample_x, resample_y),
                          method='cubic')

    nan_mask = np.isnan(resample_z)
    if np.any(nan_mask):
        resample_z[nan_mask] = griddata((x.flatten(), y.flatten()), z.flatten(),
                                        (resample_x[nan_mask], resample_y[nan_mask]), method='nearest')

    return resample_x, resample_y, resample_z

def plot_subplots(grid_x1, grid_y1, grid_z1, title1, grid_x2, grid_y2, grid_z2, title2):
    fig, axs = plt.subplots(1, 2, figsize=(16, 8))

    c1 = axs[0].contourf(grid_x1, grid_y1, grid_z1, cmap='viridis', levels=100)
    fig.colorbar(c1, ax=axs[0], label='Gravity Value (milligal)')
    axs[0].set_xlabel('Longitude')
    axs[0].set_ylabel('Latitude')
    axs[0].set_title(title1)

    c2 = axs[1].contourf(grid_x2, grid_y2, grid_z2, cmap='viridis', levels=100)
    fig.colorbar(c2, ax=axs[1], label='Gravity Value (milligal)')
    axs[1].set_xlabel('Longitude')
    axs[1].set_ylabel('Latitude')
    axs[1].set_title(title2)

    plt.tight_layout()
    plt.show()

def save_to_txt(grid_x, grid_y, grid_z, filename):
    os.makedirs(os.path.expanduser("~/Desktop/grav_txtfiles"), exist_ok=True)
    filepath = os.path.expanduser(f"~/Desktop/grav_txtfiles/{filename}.txt")
    np.savetxt(filepath, grid_z, fmt='%.6f', delimiter=' ', header="Interpolated gravity values grid")
    print(f"Saved: {filepath}")

# Get user-defined latitude and longitude ranges
latitude_min, latitude_max = map(float, input("Enter the latitude range (min,max): ").split(','))
longitude_min, longitude_max = map(float, input("Enter the longitude range (min,max): ").split(','))

# Expand ROI by 0.3 degrees
expanded_lat_min, expanded_lat_max = latitude_min - 0.3, latitude_max + 0.3
expanded_lon_min, expanded_lon_max = longitude_min - 0.3, longitude_max + 0.3

survey_name = input("Enter the name of the survey: ")

# Load Bouguer and Isostatic data
datasets = {
    "Bouguer": "https://raw.githubusercontent.com/maxfollett/AirBorneInsight2/main/USbougerGravData.xyz",
    "Isostatic": "https://raw.githubusercontent.com/maxfollett/AirBorneInsight2/main/isograv.xyz"
}

cropped_results = {}

for key, url in datasets.items():
    data = pd.read_csv(url, sep='\s+', header=None, names=['x', 'y', 'value'])
    filtered_data = data[(data['x'] >= expanded_lon_min) & (data['x'] <= expanded_lon_max) &
                         (data['y'] >= expanded_lat_min) & (data['y'] <= expanded_lat_max)]

    # Interpolate on expanded area
    grid_x, grid_y, grid_z = interpolate_to_grid(filtered_data, expanded_lon_min, expanded_lon_max,
                                expanded_lat_min, expanded_lat_max, grid_size=(int(1486*1.2), int(2116*1.2)))

    # Crop based on actual user ROI
    crop_mask_x = (grid_x[0, :] >= longitude_min) & (grid_x[0, :] <= longitude_max)
    crop_mask_y = (grid_y[:, 0] >= latitude_min) & (grid_y[:, 0] <= latitude_max)
    cropped_grid_x = grid_x[np.ix_(crop_mask_y, crop_mask_x)]
    cropped_grid_y = grid_y[np.ix_(crop_mask_y, crop_mask_x)]
    cropped_grid_z = grid_z[np.ix_(crop_mask_y, crop_mask_x)]

    # Resample cropped area to exactly (1486, 2116)
    resample_x, resample_y, resample_z = resample_to_target_size(cropped_grid_x, cropped_grid_y,
                cropped_grid_z, (1486, 2116), longitude_min, longitude_max, latitude_min, latitude_max)

    cropped_results[key] = (resample_x, resample_y, resample_z)
    save_to_txt(resample_x, resample_y, resample_z, f"{survey_name}_{key}_cropped_{latitude_min}_{latitude_max}_

# Plot both datasets as subplots
plot_subplots(
    cropped_results["Bouguer"][0], cropped_results["Bouguer"][1], cropped_results["Bouguer"][2], f"Bouguer Anoma
    cropped_results["Isostatic"][0], cropped_results["Isostatic"][1], cropped_results["Isostatic"][2], f"Isostat
)
```

Landsat magery data preprocessing

B.2 Machine Learning

```python
import os
os.system('cls')  # clear terminal
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # suppress tensorflow info

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf  # machine learning library
import random

from sklearn.metrics import accuracy_score, precision_score, recall_score  # training metrics
from tensorflow.keras import layers, losses  # ML stuff
from tensorflow.keras.models import Model  # ML stuff
```

```python
# read from file
def read_txt_file(file_path):
    with open(file_path, 'r') as file:
        data = file.readlines()
    return np.array([list(map(float, line.split())) for line in data])

# normalize from 0-1
def normalize_array(arr):
    min_val, max_val = arr.min(), arr.max()
    return (arr - min_val) / (max_val - min_val)

# add random noise to array, ensure it stays betwenn 0-1
def add_noise(arr, noise_level=0.01):
    noise = np.random.uniform(-noise_level, noise_level, arr.shape)
    noisy_arr = arr + noise
    noisy_arr = np.clip(noisy_arr, 0, 1)
    return noisy_arr

# resize array
def resize_array(arr, new_height, new_width):
    return arr[:new_height, :new_width]

# patch extraction
def extract_patches_with_overlap(arr, patch_size, overlap):
    height, width, depth = arr.shape
    patches = []
    step = patch_size - overlap
    for i in range(0, height - patch_size + 1, step):
        for j in range(0, width - patch_size + 1, step):
            patches.append(arr[i:i+patch_size, j:j+patch_size, :])
    return np.array(patches)

# read folder of txt files, arrays need to be the same size, combine them
def process_folder(folder_path):
    arrays = [add_noise(normalize_array(read_txt_file(os.path.join(folder_path, file_name))))
              for file_name in sorted(os.listdir(folder_path)) if file_name.endswith('.txt')]

    min_height = min(arr.shape[0] for arr in arrays)
    min_width = min(arr.shape[1] for arr in arrays)
    arrays = [arr[:min_height, :min_width] for arr in arrays]
```

```python
    stacked_array = np.dstack(arrays)
    return stacked_array, min_height, min_width

# four quadrants, one selectabe 'target' quadrant
def divide_into_quadrants(arr, target_quadrant):
    mid_height, mid_width = arr.shape[0] // 2, arr.shape[1] // 2

    quadrants = {
        1: (arr[:mid_height, :mid_width], "Top-left"),
        2: (arr[:mid_height, mid_width:], "Top-right"),
        3: (arr[mid_height:, :mid_width], "Bottom-left"),
        4: (arr[mid_height:, mid_width:], "Bottom-right"),
    }

    target_region, target_name = quadrants[target_quadrant]
    training_testing = [quad for idx, (quad, _) in quadrants.items() if idx != target_quadrant]

    print(f"Selected Target Region: {target_name}")

    return *training_testing, target_region

#select target region
target_quadrant = 2

# grab survey data and split quadrants
folder_path = 'xxxxxxxxxxx'
stacked_array, min_height, min_width = process_folder(folder_path)
training_testing_1, training_testing_2, training_testing_3, target_region = divide_into_quadrants(stacked_array, target_quadrant)
n_layers = stacked_array.shape[2] #need number of surveys for dynamic calculation

# check original stacked array dimensions
#print(f"Original stacked array dimensions: {stacked_array.shape}")

# Set patch size and overlap
patch_size = 25
patch_overlap_percentage = 40 #this'll actually crash later is theres no overlap... whoopsie
patch_overlap = int(patch_size * (patch_overlap_percentage / 100)) # need overlap in pixels

# quadrants need resizing based on patch sizze and overlap
# grab patches too
def process_quadrant(arr, return_resized=False):
    resized_width = (2 * patch_size - patch_overlap) + ((arr.shape[1] - (2 * patch_size - patch_overlap)) // (patch_size - patch_overlap)) * (patch_size - patch_overlap)
    resized_height = (2 * patch_size - patch_overlap) + ((arr.shape[0] - (2 * patch_size - patch_overlap)) // (patch_size - patch_overlap)) * (patch_size - patch_overlap)
    resized_array = resize_array(arr, resized_height, resized_width)
```

```python
    if return_resized:
        return resized_array, extract_patches_with_overlap(resized_array, patch_size, patch_overlap)
    return extract_patches_with_overlap(resized_array, patch_size, patch_overlap)

# concatenate training/testing quadrant patches
training_testing_patches = []
for quadrant in [training_testing_1, training_testing_2, training_testing_3]:
    patches = process_quadrant(quadrant)
    training_testing_patches.append(patches)
training_testing_patches = np.concatenate(training_testing_patches, axis=0)

# Process target region
resized_target_region, target_region_patches = process_quadrant(target_region, return_resized=True)

# print results for testing
#print(f"Resized Target Region Shape: {resized_target_region.shape}")
#print(f"Target Region Patches Shape: {target_region_patches.shape}")
#print(f"Training/Testing patches shape: {training_testing_patches.shape}")
```

```python
# shuffle patches
shuffled_patches = training_testing_patches[np.random.permutation(training_testing_patches.shape[0])]

# split into training 80% and testing 20%
split_index = int(0.8 * len(shuffled_patches))
training_patches = shuffled_patches[:split_index]
testing_patches = shuffled_patches[split_index:]

#print("training_data shape:", training_patches.shape)
#print("testing_data shape:", testing_patches.shape)
```

```python
# define Autoencoder class, used tensorflow tutorial for initial setup
class Autoencoder(Model):
  def __init__(self, latent_dim, shape):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.shape = shape

    # define encoder, want balance between under and over fitting more focus on over
    self.encoder = tf.keras.Sequential([
      layers.Reshape(shape),
      layers.Flatten(),
      layers.Dense(latent_dim, activation=None),
      layers.BatchNormalization(),
      layers.Activation('relu'),
      layers.Dropout(0.2)
    ])

    # define decoder, reverse encoding process
    self.decoder = tf.keras.Sequential([
      layers.Dense(tf.math.reduce_prod(shape).numpy(), activation=None),
      layers.BatchNormalization(),
      layers.Activation('sigmoid'),
      layers.Reshape(shape)
    ])

  def call(self, x):
    # put input through encoder and decoder
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

# custom weighted MSE loss function
def weighted_mse(y_true, y_pred, weights=(1, 1, 1, 1, 1, 1, 3, 2, 2)):
    # Convert weights to TensorFlow constant and reshape for broadcasting
    weights = tf.constant(weights, dtype=tf.float32)
    weights = tf.reshape(weights, (1, 1, 1, 9))

    # element-wise squared error
    squared_error = tf.square(y_true - y_pred)

    # apply weights and get mean
    weighted_error = tf.reduce_mean(weights * squared_error)
```

```python
    return weighted_error

# shape of the test data excluding batch size
shape = testing_patches.shape[1:]

# Dynamically Define Shape of latent dimension based on patch size and number of layers # want to test this further
# want to slow down the increase of the latent dim as the number of layers increases to force the autoencoder to find correlations
latent_dim = int(patch_size * patch_size * ((n_layers - 0.5) ** 0.5) / 2) - 65

# Instantiate the Autoencoder model with the given latent dimension and input shape
autoencoder = Autoencoder(latent_dim, shape)

# Compile the Autoencoder model with the Adam optimizer and the custom weighted MSE loss function
autoencoder.compile(optimizer='adam', loss=weighted_mse)

# Train the autoencoder for X epochs, using the training data as both input and target
autoencoder.fit(training_patches, training_patches,
                epochs=10,  # Set the number of training epochs # want to test this further
                shuffle=True,  # shuffle the training data before each epoch
                validation_data=(testing_patches, testing_patches))  # test data for validation
```

```python
# After training, encode the target region images into their compressed representations
encoded_target_patches = autoencoder.encoder(target_region_patches).numpy()
# Decode the latent representations back into the reconstructed images
decoded_target_patches = autoencoder.decoder(encoded_target_patches).numpy()

#print("Target Region patches Shape:", target_region_patches.shape)
#print("encode:", encoded_target_patches.shape)
#print("decode:", decoded_target_patches.shape)
```

```python
def reconstruct_array(patches, patch_size, trimmed_array, patch_overlap,n):
    # Extract the height and width from the trimmed array
    trimmed_height, trimmed_width = trimmed_array.shape[:2]

    # Calculate the step size
    step = patch_size - patch_overlap

    # Calculate the number of patches in the vertical and horizontal directions based on the trimmed dimensions
    num_patches_vertical = (trimmed_height - patch_size) // step + 1
    num_patches_horizontal = (trimmed_width - patch_size) // step + 1

    # Initialize an array to hold the reconstructed values and a count array for weights
    reconstructed_array = np.zeros((trimmed_height, trimmed_width,n))
    weight_array = np.zeros((trimmed_height, trimmed_width,n))

    # Create a weight matrix based on distance from the center of the patch
    center = patch_size // 2
    distance_weights = np.zeros((patch_size, patch_size))
    d_max = np.sqrt(2) * center  # Max possible distance in the patch
    k = 6/patch_overlap # experimentally determined steepness
    d_mid = d_max - patch_overlap/2  # Midpoint for sigmoid transition
    for i in range(patch_size):
        for j in range(patch_size):
            # Calculate the distance from the center of the patch
            distance = np.sqrt((i - center)**2 + (j - center)**2)
            # Assign weight (Sigmoid) designed distrobutions here #https://www.desmos.com/calculator/5b3thqmwvu
            ## Other options: Inverse. Inverse Square, Linear, Gaussian
            distance_weights[i, j] = 1 / (1 + np.exp(k * (distance - d_mid)))

    # patches to be placed back based on index, same order as patcehs are grabbed
    patch_index = 0
    for i in range(num_patches_vertical):
        for j in range(num_patches_horizontal):
            # Calculate the position to place the current patch
            row_start = i * step
            col_start = j * step
            patch = patches[patch_index]

            # Apply the weights to the patches across all layers
            weighted_patch = patch * distance_weights[:, :, np.newaxis]
```

```python
            # Accumulate the weighted patch values and increment the weight array
            reconstructed_array[row_start:row_start+patch_size, col_start:col_start+patch_size, :] += weighted_patch
            weight_array[row_start:row_start+patch_size, col_start:col_start+patch_size, :] += distance_weights[:, :, np.newaxis]

            patch_index += 1

    # Normalize based on accumulated weights
    reconstructed_array /= (weight_array)
    return reconstructed_array


# Reconstruct an array using the patches array and target array dimensions
reconstructed_patches = reconstruct_array(decoded_target_patches, patch_size, resized_target_region, patch_overlap,n_layers)
abs_error = abs(resized_target_region - reconstructed_patches)

# checks
#print(f"Quadrant shape: {target_region.shape}")
#print("Reconstructed patches shape:", reconstructed_patches.shape)
```

```python
# new patches for error searching
error_patch_size = 75
error_patch_overlap_percentage = 44
error_patch_overlap = int(error_patch_size * error_patch_overlap_percentage / 100)

# reuse previous patch function
abs_error_patches = extract_patches_with_overlap(abs_error, error_patch_size, error_patch_overlap)
#print("Error Patches Shape:", abs_error_patches.shape)

# find M patches with the highest average error, done separately for each channel layer
def find_heighest_error_patches(abs_error_patches, M):
    num_patches, patch_size, _, num_channels = abs_error_patches.shape
    top_m_indices_per_channel = {}
    top_m_errors_per_channel = {}

    for c in range(num_channels):  # Iterate over channels
        avg_errors = np.mean(abs_error_patches[:, :, :, c], axis=(1, 2))

        # Get indices of the M highest-error patches for channel
        top_m_indices = np.argsort(avg_errors)[-M:][::-1]  # descending order

        # Store results for each channel
        top_m_indices_per_channel[c] = top_m_indices
        top_m_errors_per_channel[c] = avg_errors[top_m_indices]

    return top_m_indices_per_channel, top_m_errors_per_channel

M = 10  # Get top M patches
top_m_indices_per_channel, top_m_errors_per_channel = find_heighest_error_patches(abs_error_patches, M)
```

```python
# Mark the top M patches with scaled values 1 to 1/M, others replaced with zeros
def mark_top_m_patches_and_reorder(abs_error_patches, top_m_indices_per_channel, M):
    marked_patches = np.zeros_like(abs_error_patches)  # Initialize with zeros
    # Create a new list to store patches in original order
    reordered_patches = np.copy(abs_error_patches)  # Copy of the original patches to reorder later
    for c in range(abs_error_patches.shape[3]):  # Iterate over channels
        for rank, patch_idx in enumerate(top_m_indices_per_channel[c]):  # Iterate over top M patches
            value = 1 - (rank / M)  # Scale from 1 down to 1/M
            # Modify the patch with scaled value
            marked_patches[patch_idx, :, :, c] = value
        # Reorder the patches to their original positions
    patch_index = 0
    for i in range(abs_error_patches.shape[0]):
        for j in range(abs_error_patches.shape[1]):
            for c in range(abs_error_patches.shape[3]):
                reordered_patches[i, j, :, c] = marked_patches[i, j, :, c]
                patch_index += 1
    return reordered_patches
```

```python
heighest_error_ordered = mark_top_m_patches_and_reorder(abs_error_patches, top_m_indices_per_channel, M)
#print("Reordered Heightest Error Patches Shape:", heighest_error_ordered.shape)
# new patch stiching function with no weighting function
def reconstruct_from_patches(patches, patch_size, trimmed_array, patch_overlap):
    # Extract the height and width from the trimmed array
    trimmed_height, trimmed_width, n = trimmed_array.shape
    step = patch_size - patch_overlap
    # Calculate the number of patches in the vertical and horizontal directions
    num_patches_vertical = (trimmed_height - patch_size) // step + 1
    num_patches_horizontal = (trimmed_width - patch_size) // step + 1
    reconstructed_array = np.zeros((trimmed_height, trimmed_width, n))

    # Reshape patches to be placed back
    patch_index = 0
    for i in range(num_patches_vertical):
        for j in range(num_patches_horizontal):
            # Calculate the position to place the current patch
            row_start = i * step
            col_start = j * step
            patch = patches[patch_index]
            # add patch
            reconstructed_array[row_start:row_start+patch_size, col_start:col_start+patch_size, :] += patch

            patch_index += 1

    return reconstructed_array

heighest_error = reconstruct_from_patches(heighest_error_ordered,error_patch_size,abs_error,error_patch_overlap)
#print("Heighest Error Shape:", heighest_error.shape)
```

```python
# Define the names for the first plot in each row, yes this is hard coded for making figures
layer_names = [
    "Spectral Band 4", "Spectral Band 5", "Spectral Band 6", "Spectral Band 7",
    "Spectral Band 4/5", "Spectral Band 5/7", "Magnetic",
    "Density (Bouguer)", "Density (Isostatic)"
]

plt.figure(figsize=(24, 6 * n_layers), dpi=200)  # 4 by number of layers figure

for i in range(n_layers):
    # Get the corresponding name for the first plot in each row
    title_name = layer_names[i % len(layer_names)]  # Cycle through names

    ax1 = plt.subplot(n_layers, 4, 4*i + 1)
    plt.imshow(resized_target_region[:, :, i], cmap='viridis')
    plt.title(f"{title_name}", fontsize=28)  # Use layer name
    ax1.get_xaxis().set_visible(False)
    ax1.get_yaxis().set_visible(False)

    ax2 = plt.subplot(n_layers, 4, 4*i + 2)
    plt.imshow(reconstructed_patches[:, :, i], cmap='viridis')
    plt.title(f"Reconstructed Patches", fontsize=28)
    ax2.get_xaxis().set_visible(False)
    ax2.get_yaxis().set_visible(False)

    ax3 = plt.subplot(n_layers, 4, 4*i + 3)
    plt.imshow(abs_error[:, :, i], cmap='seismic', vmin=0)
    plt.title(f"Reconstruction Error", fontsize=28)
    ax3.get_xaxis().set_visible(False)
    ax3.get_yaxis().set_visible(False)

    ax4 = plt.subplot(n_layers, 4, 4*i + 4)
    plt.imshow(heighest_error[:, :, i], cmap='Reds', vmin=0)
    plt.title(f"Highest Error Regions", fontsize=28)
    ax4.get_xaxis().set_visible(False)
    ax4.get_yaxis().set_visible(False)

plt.tight_layout()
plt.show()
```

```python
# Define scaling vector. We want to weight mag and the two gravs higher
scaling_vector = np.array([1, 1, 1, 1, 1, 1, 2, 1.5, 1.5])
# Scale each layer by its corresponding weight
weighted_array = heighest_error * scaling_vector.reshape(1, 1, n_layers)
# sum the weighted values
combined_channel = np.sum(weighted_array, axis=2)
plt.figure(figsize=(10, 10))
im = plt.imshow(combined_channel, cmap='Reds')
plt.title('Summed Highest Error Regions')
plt.show()
```

B.3 User Interface

[1]:
```python
from dash import Dash, dcc, html, Output, Input, State  # Dash components for layout and interactivity
import pandas as pd                                     # For handling CSV file data
import base64                                           # For encoding images and uploaded files
import io                                               # For in-memory file processing

# Initialize the web application instance
app = Dash(__name__)

# Function to read and encode an image to base64 so it can be displayed in HTML
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')

# Encode the Geomagnetism image for later embedding in the page
encoded_image = encode_image("/Users/kathrynnlees/Downloads/Geomag.png")  # Path may need to be updated

# Define the layout and visual structure of the web application
app.layout = html.Div([
    # Title header
    html.H1("Machine Learning Image Reconstruction for Identifying Geophysical Anomalies",
            style={'textAlign': 'center', 'color': '#003366', 'marginTop': '24px'}),

    # Subtitle / team credit
    html.P("Created by Airborne Insights",
           style={'textAlign': 'center', 'fontSize': '25px', 'fontWeight': 'bold', 'color': 'black;'}),

    # Short introduction / abstract
    html.P(
        "Airborne Insights has developed a tool to expedite the anomaly detection "
        "process in geophysical data. Traditionally, trained geophysicists would have to "
        "examine large datasets which is time consuming and subjective. We have "
        "developed a tool that can take unlabeled surveys and return a georeferenced "
        "file to highlight potential anomalies.",
        style={'textAlign': 'center', 'fontSize': '20px', 'color': '#556', 'margin': '20px'}
    ),

    # Upload section header
    html.H3("Upload Your Survey Data",
            style={'textAlign': 'center', 'marginTop': '18px', 'fontSize': '20px', 'fontWeight': 'bold'}),

    # File upload component with a styled button
    dcc.Upload(
        id='upload-data',
        children=html.Button("Upload File", style={'backgroundColor': '#003366', 'color': 'white', 'padding': '10px', 'border': 'none'}),
        multiple=False,  # Restrict to one file at a time
        style={'textAlign': 'center', 'marginTop': '20px'}
    ),

    # Instructional note for supported data types
    html.P("Upload only one file at a time (magnetic, gravity or LANDSAT)",
           style={'textAlign': 'center', 'color': '#556', 'fontSize': '16px'}),

    # Interpolation method dropdown header
    html.H3("Select Interpolation Method", style={'textAlign': 'center', 'marginTop': '20px'}),

    # Dropdown for selecting interpolation method
    dcc.Dropdown(
        id='interpolation-method',
        options=[
            {'label': 'Linear', 'value': 'linear'},
            {'label': 'Cubic', 'value': 'cubic'},
            {'label': 'Nearest', 'value': 'nearest'}
        ],
        value='linear',  # Set default method
        clearable=False,
        style={'width': '50%', 'margin': 'auto', 'textAlign': 'center'}
    ),

    # Output field to display selected method dynamically
    html.Div(id='selected-method', style={'textAlign': 'center', 'marginTop': '16px', 'color': '#556'}),

    # Button to trigger data processing
    html.Div(style={'textAlign': 'center', 'marginTop': '30px'}, children=[
        html.Button("Process Data", id="process-button", n_clicks=0,
                    style={'backgroundColor': '#28a745', 'color': 'white', 'padding': '10px 20px', 'border': 'none'})
    ]),

    # Output field for results or error messages
    html.Div(id='output-data', style={'textAlign': 'center', 'marginTop': '10px'}),

    # Embeds the encoded image below the outputs
    html.Div(style={'textAlign': 'center', 'marginTop': '0px'}, children=[
        html.Img(
            src=f"{encoded_image}",
            style={'width': '100%', 'height': 'auto'}
        )
    ])
])

# Function that parses uploaded file contents and displays results based on type
def parse_contents(contents, filename, method):
    content_type, content_string = contents.split(',')
    decoded = base64.b64decode(content_string)

    # Handle .csv files
    if filename.endswith('.csv'):
        df = pd.read_csv(io.StringIO(decoded.decode('utf-8')))
        return html.Div([
            html.H5(f"Uploaded File: {filename}"),
            html.Pre(df.head().to_string(), style={'whiteSpace': 'pre-wrap', 'textAlign': 'left'}),
            html.H5(f"Processing using: {method.capitalize()} Interpolation")
        ])
    # Handle .pdf files
    elif filename.endswith('.pdf'):
        return html.Div([html.H5(f"Uploaded PDF: {filename}")])
    # Handle .xyz files
    elif filename.endswith('.xyz'):
        return html.Div([html.H5(f"Uploaded XYZ file: {filename}")])
    # Handle unsupported formats
    else:
```

```
    else:
        return html.Div(["Unsupported file format. Please upload a CSV, PDF, or XYZ file."])

# Callback to update interpolation method display when selection changes
@app.callback(
    Output('selected-method', 'children'),
    Input('interpolation-method', 'value')
)
def update_interpolation(method):
    return f"Selected Interpolation Method: {method.capitalize()}"

# Callback triggered when "Process Data" button is clicked
@app.callback(
    Output('output-data', 'children'),
    Input('process-button', 'n_clicks'),
    State('upload-data', 'contents'),
    State('upload-data', 'filename'),
    State('interpolation-method', 'value')
)
def process_data(n_clicks, contents, filename, method):
    # If button is clicked and file is uploaded
    if n_clicks > 0 and contents is not None:
        return parse_contents(contents, filename, method)
    # If button is clicked but no valid file is uploaded
    elif n_clicks > 0:
        return html.Div("Please upload a file before processing.", style={'color': 'red'})
    # Default case (no action)
    return ""

# Launch the server and run the application in debug mode
if __name__ == '__main__':
    app.run_server(debug=True)
```