22AIE452: Data Driven Material Modeling

Attention Mechanism Based Combinatorial Design of Perovskites

Project Report Submitted by

Group 17

Group Members:

Akhil Vignesh A - CB.SC.U4AIE23302

Shiva Charan Goud K - CB.SC.U4AIE23305

Tarun Reddy N - CB.SC.U4AIE23320

Sai Akhilesh Y - CB.SC.U4AIE23348

Guided by

Dr. Kritesh

Assistant Professor,

Amrita School of Artificial Intelligence

Amrita Vishwa Vidyapeetham

Coimbatore, Tamil Nadu, India – 641112

Contents

List of Figures

Abstract

The identification and optimization of perovskite materials for electronic and energy devices are a large challenge given the enormous chemical space and intricate structure–property relationships. It is computationally expensive and experimentally prohibitive to accurately predict properties like band gaps, formation energy, and material stability, which involve combining compositional and structural data.

In this project, we formulated an attention mechanism-based combinatorial design method for perovskites through an Enhanced Element Descriptor Matrix (EDM) Transformer model. The methodology combines Word2Vec embeddings of elemental symbols, other elemental features (atomic number, electronegativity, radius, etc.), and enriched fractional embeddings to produce an all-encompassing representation of each compound. The model applies multi-head attention pooling and cross-element attention to decode inter-element correlations and detect significant compositional contributions. Multi-task learning was used to predict regression targets (e.g., band gap) and classification targets (stability).

Our results show that the introduced framework makes accurate predictions on previously unseen perovskite compositions with high correlation between predicted and actual property values for regression tasks and strong classification metrics for stability. Interpretability is provided by attention visualizations, emphasizing the most impactful elements in defining material properties.

In summary, this paper contributes a data-driven, scalable, and interpretable solution to speed up functional perovskite design. The combination of advanced embeddings with attention mechanisms allows for the discovery of high-potential compositions in an efficient manner, potentially minimizing experimental trials and informing future material search endeavors.

1. Introduction

Identification and tuning of perovskite compounds for electronic and energy devices are a persistent challenge owing to the huge chemical space and intricate relationships between composition, structure, and properties. Classical experimental methods are often time, labor, and resource intensive and narrow in scope, whereas computational approaches struggle to accurately capture the complex dependencies that underlie material performance. This work seeks to overcome these challenges using state-of-the-art machine learning methods, specifically transformer-based architectures with attention, to model the rich interactions between constituent species in perovskites. Through a synergy of compositional embeddings, elemental descriptors, and augmented fractional representations, the suggested approach offers a complete perspective on the materials' properties.The objective of this research is to create an interpretable, scalable, and computationally efficient framework that can predict some of the most important material properties like band gaps, formation energy, density, volume, and stability. Multi-task learning is utilized to tackle both regression and classification tasks at the same time, whereas attention-based mechanisms emphasize the most significant elements and interactions in a compound. This methodology not only speeds up the computational design of perovskites but also delivers insight into the underlying drivers of material properties, thus informing experimental verification and future material discovery.

1.1 Motivation of the Study

Perovskite compounds have garnered unprecedented attention in applications ranging from photovoltaics, electronics, and catalysis thanks to their outstanding optoelectronic properties, adjustable band gaps, and structural versatility. The challenge of discovering novel perovskite compositions is hampered by the vast chemical space as well as the complex interplay between multiple elements and thus poses experimental synthesis and examination as extremely resource-demanding. Standard computational techniques, although useful, are constrained by their potential to realistically account for intricate element–property correlations in thousands of potential compositions. There is an immediate need for data-driven strategies that could predict material properties with accuracy and speed, as well as offer interpretability to inform experimental studies.

1.2 Problem Statement

The main challenge in creating new perovskite materials is in the enormous combinatorial potential of composition space and the rich structure–property correlations that control their behavior. Realistic prediction of important properties, including band gap, formation energy, and material stability, needs to combine both compositional and structural knowledge. The older experimental methods are time-consuming and expensive, and the usual computational methods are often not scalable and lack generalizability to novel compositions.

Besides, most current predictive models are not interpretable, such that one cannot easily tell which combinations of elements or structural characteristics have the greatest impact on material properties. Such non-interpretability hinders focused material optimization and retards high-performance perovskite discovery.

Thus, a critical need is realized for a computational protocol that can effectively screen the perovskite chemical space, make accurate predictions of properties, and provide interpretability to inform experimental synthesis. This project fills this void by utilizing an attention mechanism-based combinatorial design based on improved elemental descriptors to embody inter-element interactions and model multiple material properties in one step.

1.3 Plan of Action

To meet the challenges mentioned in the problem statement, the following multi-step process was followed in this project:

1. Data Collection and Preprocessing

- Compiled a dataset of perovskite compositions and related properties like band gap, formation energy, density, and stability.
- Cleaned and standardized the data for model training consistency and reliability.

2. Feature Engineering with Improved Element Descriptors

- Built an Enhanced Element Descriptor Matrix (EDM) for every compound, blending multiple feature types:
  - Word2Vec embeddings of elemental symbols to capture semantic relationships.

  - Elemental attributes like atomic number, electronegativity, atomic radius, and metal type.
  - Improved fractional embeddings to efficiently represent stoichiometric ratios.
- Ensured uniform representation for compounds for compatibility with the transformer model.

3. Model Design – Attention-Based Transformer

- Created an EDM Transformer that utilizes multi-head attention pooling and cross-element attention to decode element interactions.
- Used a CLS token and various pooling techniques to extract rich compound-level features.

4. Multi-Task Learning Framework

- Trained the model to simultaneously predict regression targets (e.g., formation energy, band gap) and classification targets (stability).
- Used task-specific heads and learnable loss weighting to weigh contributions from various tasks.

5. Training and Validation

- Divided the dataset into training, validation, and test sets for strong evaluation.
- Employed advanced optimization methods such as differential learning rates, learning rate scheduling with warmup, and early stopping to facilitate stable and efficient convergence.

6. Interpretability and Visualization

- Extracted attention weights to visualize element contributions and inter-element relations.
- Determined important compositional variables that control material properties to inform experimental verification.

7. Evaluation and Deployment

- Measured model performance against typical regression and classification criteria.
- Verified predictions on novel compositions to exhibit generalizability.
- Prepared visualizations and model outputs to aid reproducibility and additional analysis.

The structured approach enables efficient exploration of the perovskite chemical space and accurate, interpretable, and scalable predictions to facilitate material discovery.

2. Literature Review

Table 1: Literature review addressing summary and the research gap of the project

S.No	Title of the Paper	Author and year	Paper summary	Research Gap
1	Strategic Integration of Machine Learning in the Design of Excellent Hybrid Perovskite Solar Cells	Zhaosheng Zhang, Sijia Liu, et.al.	This work criticizes models such as CNNs and GNNs that require full 3D crystal structures—a narrow approach,as this information typically is not available. The project closes this loop by proposing a flexible, ”composition-only”Transformer model predicting propertiesfrom thechemical formula directly.This is addressing the data dependency problem along withotherchallenges such as multi-tasklearning andclass imbalance.	This paper's models require complete 3D crystal structures, which are often unavailable. Our composition-only Transformer model solves this by predicting properties directly from the more accessible chemical formula.
2	Multimodal deep learning-driven exploration of lanthanide-based perovskite oxide semiconductors for ultra-sensitive detection of 2-butanone	Shaofeng Shao, Liangwei Yan, et.al	This work describes a deep-learning-predicted perovskite, TmRhO, that was prepared for diagnostic applications in medicine. material can be used as an extremely stable sensor with tunable electronic behavior for detecting 2-butanone (50 ppb detection),supports the combined deep learning and experimental method for designing material systems.	This study validates a deep learning prediction for a single material. Our project addresses the gap by providing a scalable, general-purpose model for high-throughput screening of many potential materials.
3	Quotient Complex Transformer (QCformer) for Perovskite Data Analysis	Xinyu You, Xiang Liu, et.al., 2025	This work's QCformer perovskite model breaks GNN restrictions by observing periodicity and multiple-body interactions. In a new Quotient Complex (QC)characterization, state-of-the-art bandgap predictions are attained on benchmark databases such as JARVIS, speeding up material discovery.	Incorporating higher-order topological data features.Caps its predictive power by overlooking topology.Extending model to non-crystalline materials.Limits applicability to crystalline materials only.Investigating higher-dimensional atomic interactions.Incomplete representation can decrease accuracy
4	Optimizing the performance of vapor-deposited perovskite solar cells through advanced predictive modeling	Seyed Hamed Godasiaei, 2024	This project employs machine learning.(Random Forest and LASSO,Deep Learning) to optimize vapor-Embedded perovskite solar cells.The deep learning model unveiled the most precise (R^2=0.91)identifying deposition techniques and short-circuit current with key factors being projected a maximum efficiency of 25.4%.	The model is trained from a small,literature-mined dataset (125 samples), which may restrict its generalizability. Furthermore, it neglects additional factors—High-end, physics-based architecture;Features that can yield increased understanding and insights more than simple statistical correlations.
5	Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties	Tian Xie, Jeffrey C.Grossman, 2018	This work talks about the Crystal Graph Convolutional Neural Network (CGCNN). It represents crystals in graphs where the atoms are nodes and bonds are edges,using convolutions for predicting material properties and establishing an important benchmark for later structure-based Models.	CGCNN's simple graph representation does not find a material periodicity and complex many-body interactions, limitations that more advanced architectures like The QCformer and Matformer are distinguished. to specifically address.

S.No	Title of the Paper	Author and year	Paper summary	Research Gap
6	Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals	Chi Chen, Weike Ye, et.al., 2019	This work presented MatErial. Graph Network (MEGNet), a GNN That extends CGCNN by including world state information (e.g., temperature) and bonding information information into the graph representation. It achieved remarkable performance on big test sets from the Project Materials.	Similar to earlier GNNs, MEGNet is pri rather constrained by its pairwise interaction focus. It does not particularly model high-order atomic relationships, or the symmetries in geometry that equivariant networks capture.
7	Periodic Graph Transformers for Crystal Material Property Prediction	Keqiang Yan, Yi Liu, et.al., 2022	Matformer modifies the powerful Transformer structure for periodic crystal structures. It displays offers a periodic table that shows representation to better capture long-range Atomic interaction and potential lattice. periodicity, achieving state-of-the-art Results serve as a direct communication,alternative to models such as QCformer.	While demonstrating strength in periodicity, Matformer does not necessarily use topological constructs like simplicial complexes to model many-body interactions are the hallmark distinction from the QCformer's more geometric approach.
8	E(3)-Equivariant Graph Neural Networks for Data-Efficient and Accurate Interatomic Potentials	Simon Batzner, Albert Musaelian, et.al., 2022	This article defines E(3)- equivariant GNNs (such as That obey the simple three-dimensional symmetry. matrices (rotation, translation). By setting such laws of nature into the architecture, the models become extremely data-efficient and accurate, especially for predictions forces in molecular dynamics.	Equivariant models are computationally intensive and highly complicated to apply. Their chief use has been interatomic potentials, With fewer searches for direct preterms of electronic behavior such as bandgap.
9	Machine learning for perovskite solar cell design	Zhenkun Hui, Mingyao Wang, et.al., 2023	This article discusses ML's role in accelerating perovskite solar cell Design, including high-throughput. Sifting for fresh stuff, pro-cess optimization and stability prediction. It highlights how ML guides experiments by efficiently Exploring deep chemical and process spaces.	The efficiency of screening depends heavily on training data quality and size. It commonly depends on simplified descriptors often lack the nuance. Advanced structural information cap- tured by geometric deep learning.

3. Dataset Description

3.1 Source

The data for this research are taken from the Materials Project database — a popular open-source collection that offers calculated materials properties based on DFT simulations.

This dataset targets perovskite compositions, which are materials of large interest because of their relevance in solar cells, LEDs, and energy storage.

The dataset holds tabular numerical features in correspondence to the physical and electronic properties of each compound. One row is a unique material sample, and one column is a measured or calculated feature.

3.2 Input Feature

The data is concerned with perovskite-type compounds with the general chemical formula ABX_3, where:

A and B are cations (positive ions), and

X is an anion (usually oxygen, halogen, or similar element).

Each record in the dataset refers to a single unique ABX_3 compound, acting as the input feature for the model. Rather than supplying explicit numerical features like density or volume, the model is able to learn representations in the first place from the chemical composition of the compound. These representations encompass periodic, atomic, and structural information underlying the material properties, which are necessary to predict.

3.3 Target Variables

There are two target variables in the dataset for two different prediction tasks.

For the task of regression, the target parameter is the **Band Gap**, which is the difference in energy between the valence and conduction bands of a material. The model predicts this continuous value, and it is measured in electronvolts (eV). The band gap is one of the important electronic properties that decide if a material is a conductor, a semiconductor, or an insulator.

For the classification problem, the target attribute is **Stability**, which is an indicator of whether a compound is thermodynamically stable or unstable. This is a binary attribute, in which 0 represents an unstable material and 1 represents a stable material. The task of classification is useful for identifying energetically favorable materials that can be used in real-world applications.

Both of these targets enable the model to carry out multi-task learning, predicting both a continuous attribute (band gap) and a categorical attribute (stability) simultaneously, and thus provide a more in-depth insight into material properties.

4. Preprocessing Overview

Preprocessing is an important process that transforms the dataset into a state ready for training the neural network model. Preprocessing entails loading, converting, and enriching raw chemical information into a machine learning algorithm-friendly form. The processes involved include dataset loading, creation of elemental embeddings, and integration of other atomic-level features.

4.1 Dataset Loading

The data used in this project are perovskite compounds saved into a file called `Perovskite_data.csv`. Each compound in the data is described by a chemical formula like $BaTiO_3$ with a general format of ABX_3. Because neural networks are not capable of processing text-based chemical formulas directly, the formulas are transformed into numerical vector representations using systematic preprocessing. This process enables the model to compute the structural and compositional characteristics of every compound numerically.

4.2 Elemental Embeddings (Word2Vec)

As a format to represent chemical elements that can be understood by a machine, the project employs Word2Vec-based embeddings from the `mat2vec` model — a pretraining model trained on materials science publications. Each chemical element (e.g., Ba, Ti, O) is encoded as a 200-dimensional vector that reflects its chemical and contextual underlying properties.

This methodology is borrowed from natural language processing (NLP), where text is encoded in the form of vectors that express semantic relationships. In the same vein, `mat2vec` embeddings express the chemical context and similarity between elements, and how elements co-occur and interact in real compounds.

4.3 Other Element Features

To further augment the model's knowledge of each compound, further numerical and categorical features are added for each element using the `pymatgen` library. Ten important features are extracted for each element, rich atomic-level descriptors that are augmentative to Word2Vec embeddings. These are:

- Atomic number

- Atomic mass
- Electronegativity
- Atomic radius
- Average ionic radius
- Group (position in periodic table column)
- Row (period in periodic table)
- Is metal (boolean)
- Is transition metal (boolean)
- Is metalloid (boolean)

Such capabilities enable the model to capture the quantitative and qualitative attributes of the elements, which ultimately enhances prediction accuracy for both regression (band gap) and classification (stability) tasks.

4.4 Fractional Embeddings

Every component in a compound possesses a certain fractional composition—that is, the ratio of the component to the overall composition. In order to well encode this information, fractional embeddings are produced with improved positional embedding methods utilizing 256-dimensional vectors. Three scales are employed to encode different facets of the elemental fractions:

- **Linear Scale:** Expresses the absolute fraction of an element in the compound. This enables the model to capture direct compositional ratios.
- **Logarithmic (Log) Scale:** Represents extreme variations or orders of magnitude differences between elements. This is especially helpful when certain elements occur in trace quantities.
- **Square Root Scale:** Places greater emphasis on small fractions by minimizing their numerical difference, so small constituents also have an effect on the model's perception.

These fractional embeddings assist the network in identifying both major and minor composition influences in advanced materials.

4.5 Aggregated Element Representation (EDM – Element Descriptor Matrix)

Following the derivation of the fractional embeddings, all features pertinent to every element are aggregated to create a single representation. Every element within a compound is described by aggregating the following elements:

- **Word2Vec embedding (200 dimensions):** Identifies semantic and contextual correlations among elements from material science texts.
- **Elemental attributes (10 dimensions):** Contain elements like atomic number, mass, electronegativity, and metallic character (as derived through pymatgen).
- **Fractional embedding (256 dimensions):** Embodies the fraction of the element present within the compound.

These three elements combined constitute a 768-dimensional vector for each element. To make sure every compound will have input sizes that are the same, compounds with fewer elements are padded to a previously agreed maximum number of elements (*max_elements*). This means there is a standard input format that is favorable for batch processing in neural networks.

4.6 Multi-Task Target Encoding

The project does multi-task learning, making predictions of both continuous and categorical attributes of perovskite materials:

- **Regression Target (Band Gap):** The *band_gap* values are normalized to have a standard deviation of 0 and 1. This prevents the model from learning inefficiently due to large numerical scales.
- **Classification Target (Stability):** The stability tag is represented as an integer:

$$0 \rightarrow \text{Unstable compound}$$

$$1 \rightarrow \text{Stable compound}$$

To counteract class imbalance (asymmetric numbers of stable and unstable compounds), class weights are calculated, assigning greater significance to minority classes in the course of training.

Once preprocessed, the resultant Element Descriptor Matrix (EDM) tensor is of size $(4557, 5, 768)$, meaning:

- 4557 compounds (samples),
- 5 elements per compound (padded if less), and
- 768 features per element (combined embeddings and descriptors).

4.7 Class Imbalance Problem

In the context of machine learning applied to materials discovery, particularly when forecasting perovskite stability, class imbalance is a prevalent and serious issue. Class imbalance occurs where one class (e.g., stable compounds) contains far more samples than another (e.g., unstable compounds). If left uncorrected, it can lead to the model's poor performance on the minority class with potentially misleading performance measures.

What is Class Imbalance?

In this dataset, the number of stable compounds is much higher than the number of unstable compounds. This uneven distribution creates a bias in the model toward predicting the majority class (stable). As a result, even though the overall accuracy might appear high, the model struggles to correctly identify unstable compounds, which are crucial for material discovery and innovation.

For example, a model that always predicts "stable" given most input samples could still have very high accuracy, just due to the fact that most samples are in that class. It would nevertheless miss the detection of rare unstable or metastable materials, which are frequently of greater scientific interest.

Why It Matters

Class imbalance has important consequences in materials research:

- **Scientific Significance:** New perovskite materials are discovered mostly based on the presence of rare unstable or metastable compounds, and finding them can be the key to revolutionary progress in energy and semiconductor technology. Disregarding these minority samples restricts scientific investigation.

- **Model Performance:** If the model trains mostly from stable samples, it will not be able to generalize and predict poorly when tested on new data.
- **Fair Learning:** Class imbalance handling ensures that the model has equal focus on stable and unstable compounds, resulting in a stronger and balanced performance in making predictions.

Method Used in This Project

Focal Loss, a dedicated loss function aimed at enhancing learning in the minority class, is used in this project to handle the imbalance.

In contrast to the baseline Cross-Entropy Loss, which assigns equal treatment to all samples, Focal Loss adaptively weights each sample during training. It reduces the weight of easy-to-classify samples (dominant class) and emphasizes hard-to-classify samples (rare class).

It ensures that the model provides more learning focus to unstable compounds that are not very common in the dataset.

The formula of Focal Loss is provided as:

$$FL(p_t) = -\alpha(1 - p_t)^\gamma \log(p_t)$$

Where:

- p_t = predicted probability for the true class
- α = balancing parameter (regulates significance of every class)
- γ = focusing parameter (regulates intensity of focus on hard samples)

Here in this project, appropriate values of α and γ are determined from experimental tuning to ascertain that the model is able to learn from stable and unstable compounds effectively.

With Focal Loss, the model becomes more responsive to minority class samples and is prevented from being overwhelmed by the majority class, eventually enhancing recall and F1-score for unstable compounds.

4.8 Focal Loss Parameters and How They Influence Model Training

Within this project, the Focal Loss function was used to combat the class imbalance issue in perovskite stability prediction. Two of the most important hyperparameters — α (alpha) and

γ (gamma) — are responsible for controlling how the model balances and pays attention to various classes during training.

4.9 Parameters Utilized

α (Alpha) = 0.25 $\rightarrow$ Class Balance Factor

The alpha parameter specifies the relative weight assigned to each of the classes in the dataset. As the dataset has a much larger number of stable compounds compared to unstable ones, a small value of $\alpha = 0.25$ assists in assigning a higher weight to the minority class (unstable compounds) while training. This makes sure that the model does not neglect unstable samples and that they have a greater influence on the gradient updates, thereby creating better recognition of minority class instances.

γ (Gamma) = 2.0 $\rightarrow$ Focusing Parameter

The gamma parameter determines the extent to which the model should concentrate on hard-to-classify samples. A γ value of 2.0 decreases the contribution of loss from easy samples (that the model already predicts accurately) and adds more weight on misclassified or hard examples. Such a mechanism enables the model to learn more discriminative features that enhance the model's performance in detecting less frequent unstable compounds accurately.

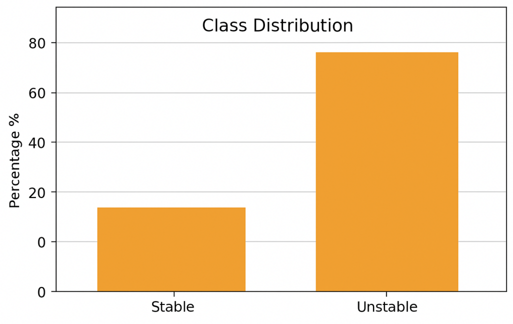
4.10 Impact on Model Training

The values of α and γ employed had a very positive influence on the learning behavior of the model:

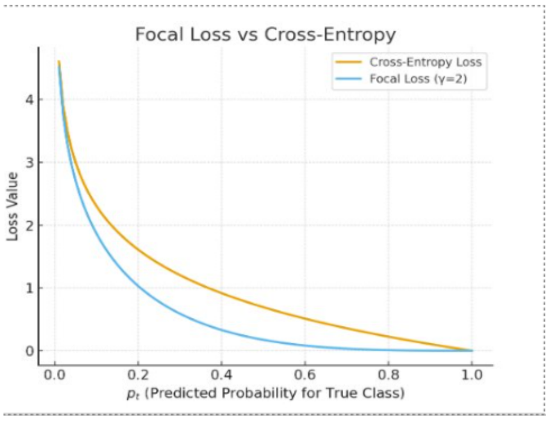
- **Prevents the model from neglecting the minority class:** By giving more effective weight to the unstable (minority) class, the model keeps these samples in focus while training and avoids being biased towards the stable (majority) class.

- **Enhances classification performance (higher F1-score):** The Focal Loss design results in more balanced learning that directly increases the F1-score — a measure that accounts for precision and recall. This enhancement means that the model is not only accurately identifying stable samples but also accurately predicting for unstable ones.

- **Guarantees balanced learning for both classes:** The combination of parameters ensures equitable performance by making sure the model provides equal weightage to both stable and unstable compounds, ensuring improved generalization and stability in multi-task learning problems.



(a) Class Distribution



(b) Focal Loss vs Cross Entropy

Figure 1: Analysis of class imbalance and comparison of focal loss with cross-entropy loss.

Interpretation of Class Imbalance and Focal Loss Graphs: The left plot shows the distribution of classes, highlighting the imbalance present in the dataset. The right plot compares the performance of focal loss against standard cross-entropy loss, demonstrating how focal loss better handles imbalanced classes by giving higher weight to minority classes.

The graph gives a general idea about how class imbalance impacts model training and how Focal Loss prevents this. The left graph easily illustrates the unbalanced distribution of samples in the dataset — with stable compounds constituting close to 20% of the whole data and unstable compounds occupying only around 80%. This skew will make the model biased towards predicting the dominant class (stable), leading to good overall accuracy but bad performance in detecting unstable compounds, which tend to be more scientifically important.

The correct plot shows the performance of Focal Loss compared to regular Cross-Entropy Loss during training. While Cross-Entropy gives equal weights to all samples, Focal Loss (represented by the blue curve) dynamically down-weights well-classified or "easy" examples and puts more weight on hard-to-classify or minority examples. This allows the model to learn better from infrequent, unstable compounds and enhance its generalization ability as well as detect underrepresented cases.

In general, the graph illustrates how Focal Loss contributes to learning balance between majority and minority classes, resolving the imbalance reflected in the distribution of datasets.

EDM of Al_2O_3

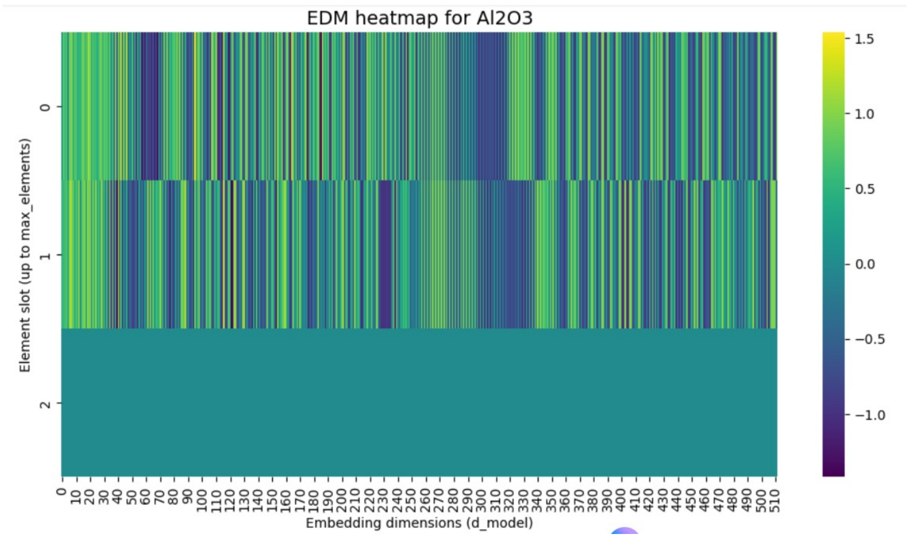


Figure 2: EDM Heatmap for Al_2O_3

The heatmap above illustrates the **Elemental Descriptor Matrix (EDM)** representation of the compound **Al_2O_3 (Aluminum Oxide)**. Each row refers to an *element position* in the compound, and each column refers to an *embedding dimension* that represents the combined chemical and structural feature information of the elements. The intensity of color reflects the size of embedding values, with yellower shades corresponding to greater feature activations and darker shades (purple/blue) corresponding to lesser ones. This visual aids in comprehending how the model represents various elements and their contributions in different dimensions. The empty (teal) rows correspond to *padding positions*, which are introduced to maintain a constant input size across compounds with different numbers of elements.

5. Methodology

5.1 Self-Attention Mechanism

Self-Attention is a mechanism that enables a model to calculate, for every item in a sequence or set, a **weighted representation** of all the other items in the same set.

That is, each element “pays attention” to other elements and discovers how much weight to assign to each of them when creating its representation.

This allows the model to learn *contextual dependencies* — the interactions between features, words, or tokens — dynamically for each sample.

Mathematically, from a collection of input vectors, self-attention calculates three projections with the following formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- Q = Query matrix (what we are paying attention to)
- K = Key matrix (what we compare against)
- V = Value matrix (information to be aggregated)
- d_k = Key vectors’ dimension (utilized for scaling)

The output is a **weighted sum of the value vectors**, in which the weights are calculated based on the similarity between queries and keys.

5.1.1 Why Self-Attention Instead of Classical Models?

Classical machine learning models such as decision trees, logistic regression, or feed-forward neural networks tend to treat each input feature independently, or they learn feature interactions only *implicitly*. This limits their ability to capture context-dependent relationships among features.

5.1.2 Limitations of Classical Models

Decision Trees: These models represent feature interactions through pairwise or hierarchical splits on individual features. However, once a split is created, the model no longer considers how other features may collectively affect that decision.

Linear Models (e.g., Logistic Regression): Such models assume additive relationships among features and cannot directly encode non-linear or contextual dependencies between them.

Feed-forward Neural Networks: While capable of modeling non-linear relationships, the interactions between features are learned through *static shared weights* across all samples. Thus, they do not adapt their behavior dynamically based on the specific input context.

Consequently, these classical models find it challenging to compute contextualized, sample-specific interactions among features. In real-world tasks, the importance of one feature often depends on the presence, value, or context of other features.

For instance, in natural language processing, the meaning of the word "bank" depends strongly on its context ("river bank" vs. "money bank"). These contextual dependencies are difficult to represent using fixed-weight models.

5.1.3 Advantages of Self-Attention

The **Self-Attention mechanism** overcomes these shortcomings by dynamically computing relationships among all elements (features, words, or tokens) in an input. It allows each element to interact with and attend to every other element, enabling flexible, adaptive, and context-aware feature representations.

- **Contextual Awareness:** Each feature (or token) can pay attention to other informative features, identifying which parts of the input are most relevant for understanding the current element.
- **Dynamic Representation:** The relative importance of each relationship is computed separately for each input sample, making the model *instance-specific* rather than relying on fixed weights.
- **Interpretability:** The attention weights produced by the model provide clear insights into which features contribute most to a particular decision, improving interpretability compared to traditional black-box models.

- **Long-Range Dependency Modeling:** In sequential data (such as text), Self-Attention can directly relate distant elements without requiring multiple intermediate layers, unlike recurrent or convolutional architectures.

Summary Overall, Self-Attention is a more expressive, flexible, and interpretable mechanism for modeling complex relationships in data. It not only identifies what is important but also determines when and under what context each feature is significant. This adaptability forms the foundation of modern deep learning architectures such as the **Transformer**.

5.2 Input Projection

5.2.1 Purpose

Before feeding data to the self-attention mechanism, each input element (e.g., token or feature vector) must be mapped into a suitable vector space. The **Input Projection** layer applies a linear transformation that:

- Converts raw input embeddings into a feature space appropriate for attention operations.
- Normalizes and enhances features for proper contextual learning.
- Ensures all elements share the same feature dimension for multi-head attention compatibility.

5.2.2 Operation

A linear layer is applied to each input vector:

$$X_{\text{proj}} = XW + b$$

where:

- X is the input matrix containing N elements, each of dimension d_{in}.
- W is the trainable weight matrix.
- b is the trainable bias vector.
- X_{proj} is the projected output that will be fed into the self-attention mechanism.

Key Notes

- Each input element embedding is linearly transformed without any non-linear activation.
- The output dimension is d_{model}, ensuring compatibility with multi-head attention layers.
- The parameters (W, b) are learned to maximize the representational power of the features.

This process allows the model to convert raw embeddings into a contextualized vector space suitable for effective attention computation.

5.3 What are Q, K, and V?

5.3.1 Purpose

During self-attention, every input element interacts with all others to determine its relevance or *attention weight*. To enable this interaction, the model projects input representations into three distinct spaces: the **Query (Q)**, **Key (K)**, and **Value (V)** matrices. These projections capture different types of relational information between input elements.

Mathematical Formulation Given the projected input embeddings X_{proj}:

$$Q = X_{\text{proj}} W_Q$$
$$K = X_{\text{proj}} W_K$$
$$V = X_{\text{proj}} W_V$$

where:

- X_{proj} : Projected input embeddings.
- W_Q, W_K, W_V : Trainable weight matrices.
- Q, K, V : Output Query, Key, and Value matrices.

Here, N denotes the number of elements (tokens or features), d_{model} is the embedding dimension (e.g., 768), and d_k represents the internal dimensionality used in the attention computation (e.g., 64).

5.3.2 Intuition

- **Q (Query):** Represents what an element is *asking for* — it determines which information the current feature seeks from others (e.g., “Does density depend on volume?”).
- **K (Key):** Represents what information a feature can *provide* — it serves as a potential answer (e.g., “Volume can explain density.”).
- **V (Value):** Represents the actual information carried by a feature, which is propagated forward after being weighted by its attention score.

Each element in the input learns to query other elements (through Q), respond to others (through K), and share its own content (through V). This dynamic interaction allows the model to learn deep contextual relationships among features.

5.3.3 Output

After projection, we obtain the three matrices Q, K, and V, which are subsequently used in the attention computation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This operation produces a contextualized representation for each input element, computed as a weighted sum of all other elements based on their relevance.

Summary The projection of inputs into Q, K, and V spaces is essential for enabling self-attention to model contextual, sample-specific dependencies dynamically. It ensures that each feature can selectively focus on the most relevant parts of the input sequence.

5.4 Attention Scores

5.4.1 Purpose

In the attention mechanism, each feature is represented by two vectors: a **Query (Q)** and a **Key (K)**. These vectors enable the model to determine how much one feature should pay attention to another. The resulting attention scores represent the *similarity* between various features in the input sequence.

Goal The primary goal of attention scoring is to measure the similarity between features, allowing the model to determine which features should focus more on others. For example, in natural language processing, certain words in a sentence depend more strongly on specific other words, and the attention mechanism quantifies these dependencies.

Formula

The attention scores are calculated using the dot product of the Query and Key matrices:

$$\text{Scores} = QK^T$$

Each element $\text{Scores}[i, j]$ represents the similarity between the Query vector of the i^{th} feature and the Key vector of the j^{th} feature. If there are 5 features, the resulting **Scores** matrix will be of size 5×5.

5.4.2 Scaling

When the dimensionality of the Key vectors (d_k) is large, the dot product values can become very large, which may cause unstable gradients during training. To prevent this, a scaling factor is applied by dividing the score by the square root of d_k:

$$\text{Scaled Scores} = \frac{QK^T}{\sqrt{d_k}}$$

This scaling operation stabilizes the gradients during training and ensures smoother optimization.

*Summary Scaled attention scores form a crucial component of the Transformer architecture. They allow the model to efficiently determine the most important features by computing normalized similarity scores between every pair of input features, enabling effective contextual learning.

5.5 SoftMax and Attention Weights

Objective After calculating the raw attention scores using the dot product of Queries (Q) and Keys (K), the resulting values can be both positive and negative, without any fixed range or clear interpretation. To make these scores meaningful, they must be normalized so that they can be interpreted as probabilities, indicating how much attention each feature gives to every other feature.

5.5.1 Why SoftMax?

The **SoftMax** function transforms these raw scores into normalized attention weights that sum to 1 across each row. This normalization ensures that:

- The attention of each feature is distributed among all other features probabilistically.
- Higher similarity (larger raw score) leads to higher attention weight.
- The model selectively focuses on the most relevant sections of the input while ignoring less important ones.

The formula for computing the attention weights is:

$$A = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

where:

- Q: Query matrix,
- K: Key matrix,
- d_k: Dimension of the Keys (e.g., 64).

5.5.2 Explanation of the Process

1. Compute pairwise similarity scores between features using QK^T.
2. Scale these values by dividing with $\sqrt{d_k}$ to prevent large magnitudes and unstable gradients.
3. Apply the SoftMax function to obtain a probability distribution:

   $$\text{SoftMax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   ensuring that all attention weights are positive and each row sums to 1.

5.5.3 Interpretation

Each element $A[i, j]$ in the attention matrix A represents the extent to which feature i attends to feature j. A higher value of $A[i, j]$ indicates that feature i pays greater attention to feature j, whereas a smaller value indicates less focus.

5.5.4 Significance

- **Context-Dependent Learning:** Each feature adjusts its attention based on the current input context.
- **Improved Interpretability:** Attention weights indicate which input components have the most influence.
- **Stable Training:** SoftMax normalization converts high raw values into bounded probabilities between 0 and 1.

Thus, the SoftMax operation converts raw similarity scores into meaningful and interpretable attention weights, forming the foundation of the self-attention mechanism in Transformer architectures.

5.6 Weighted Sum with Values

The weighted sum operation is a central component of the self-attention mechanism, where feature representations are dynamically updated through learned attention scores.

*Formula

$$Z = AV \tag{1}$$

$\mathbf{A} \in \mathbb{R}^{5\times5}$ — attention weight matrix, where each element specifies how much attention one feature (token) gives to another. These weights are calculated via a softmax operation on query-key similarity scores.
$\mathbf{V} \in \mathbb{R}^{5\times64}$ — denotes the value matrix, which includes the original feature vectors carrying semantic or contextual information for each token.
$\mathbf{Z} \in \mathbb{R}^{5\times64}$ — denotes the new feature representation, which is derived by multiplying the attention weights with the value matrix. This output indicates how every feature now combines information from all the others based on their importance in attention.

5.6.1 Intuitive Explanation

Every feature vector in the model refreshes itself by averaging all other features' value vectors with learned weights. The weights are derived from how similar or relevant the features are to each other, allowing the model to place higher emphasis on meaningful portions of the input.

5.6.2 Why This Matters

Legacy models: Generally process features as separate or use static weights, which restricts how well they can represent contextual relationships.

Self-Attention: Enables features to dynamically engage and adjust according to the input A

5.7 Multi-Head Attention

Goal

The multi-head attention mechanism allows the model to more effectively capture intricate relationships in the data by utilizing multiple parallel attention mechanisms, referred to as *heads*.

Whereas single-head attention can only observe one type of relationship at a given point in time, multi-head attention enables the model to simultaneously focus on multiple aspects of element interactions.

Each attention head operates independently, learning unique dependencies or patterns, which are then merged to produce a richer and more expressive feature representation. Because of this capability, multi-head attention has become a key component in Transformer architectures and deep learning models designed for sequential or structured information.

Concept Explanation

Each multi-head attention layer consists of several attention heads, and each head possesses its own **query (Q)**, **key (K)**, and **value (V)** projections. These projections allow each head to learn a distinct input feature representation. The process can be explained as follows:

Independent Projections: The input features are linearly projected into separate Q, K, and V matrices for each attention head. These projections allow each head to focus on different parts of the input sequence or feature space.

5.7.1 Head-wise Attention Calculation

Each head calculates its own output attention using the scaled dot-product attention mechanism:

$$Z_h = \text{Attention}(Q_h, K_h, V_h) \tag{2}$$

Here, Z_h represents the result of the h^{th} attention head. This operation determines how much attention should be allocated to other tokens or features while updating the representation of a single token. In essence, it allows the model to selectively focus on relevant portions of the input sequence, enhancing contextual understanding.

5.7.2 Projection and Concatenation

The outputs from each of the attention heads are then combined into a single vector and passed through a linear transformation (using the output weight matrix W_o) to project it back into the original model dimension:

$$Z_{\text{multi}} = \text{Concat}(Z_1, Z_2, ..., Z_H)W_o \tag{3}$$

This ensures that while multiple heads capture different types of relational information, the resulting representation remains consistent with the model's dimensional requirements.

Number of Heads (H)

The total number of attention heads, denoted as H, is typically treated as a hyperparameter (for example, $H = 12$ in most Transformer-based models). Increasing the number of heads enables the model to capture a wider variety of relationships and dependencies among elements. However, a larger number of heads also increases computational cost and resource requirements, so an appropriate balance must be maintained depending on the application and available resources.

5.8 How Multi-Head Attention Works

5.8.1 One-Head Limit Breakthrough

Single-head attention utilizes only one set of query (Q), key (K), and value (V) matrices. As a result, the model can capture only one type of interaction or dependency at a time. The multi-head attention mechanism overcomes this limitation by employing multiple heads, each attending to the input data from a different perspective. This allows the model to process multiple types of relationships simultaneously, breaking the bottleneck of single-head attention.

5.8.2 Further Context Understanding

Each attention head becomes specialized in identifying different kinds of relationships — for instance, distinguishing between short-range and long-range dependencies or focusing on local versus global regularities. This parallel focus across multiple heads significantly enhances the model's ability to understand contextual relationships in complex input data.

5.8.3 Increased Representational Ability

By integrating the information obtained from several attention heads, the final output representation can capture subtle dependencies and inter-component relationships that a single attention mechanism might miss.

Applications: In domains such as *natural language processing*, this enables the model to simultaneously comprehend both syntactic and semantic aspects of text. In *scientific applications* such as *material informatics*, multi-head attention facilitates the extraction of intricate structural and property-based dependencies — for example, identifying how multiple atomic arrangements collectively influence perovskite stability.

5.9 Residual, Normalization, and Feedforward Layers

5.9.1 Problem Following Attention

While the multi-head attention mechanism (Z_{multi}) enables the model to capture intricate dependencies among tokens or features, it also introduces certain challenges. The strong transformation capability of attention can sometimes distort the original feature representation, resulting in the loss of valuable input information. Furthermore, during training, this may cause instability due to oscillations in gradient magnitudes or over-specialization in certain attention patterns. Another limitation is that attention itself is largely linear and therefore lacks sufficient non-linearity to model highly complex data distributions. These limitations necessitate the introduction of mechanisms that stabilize learning while preserving the representational capability of the network.

5.9.2 Residual Connection

To address these concerns, Transformer architectures employ **residual connections**. The idea is to add the original input X back to the attention output Z_{multi}, as expressed by:

$$X' = X + Z_{multi} \tag{4}$$

This skip connection ensures that the original information from the input is not lost even after multiple layers of transformation. It also facilitates gradient propagation directly through the network during backpropagation, thereby preventing issues such as vanishing or exploding gradients. As a result, the training process becomes more stable and efficient, even in very deep networks.

Furthermore, residual connections provide a shortcut path for gradients, which accelerates convergence and improves the model's generalization performance. From a theoretical perspective, the network effectively learns only the *residual mapping*—that is, the difference between the desired output and the input—thereby simplifying the overall learning problem.

5.9.3 Layer Normalization

After applying the residual connection, **layer normalization** is employed to ensure numerical stability and balanced activations across layers. The process of normalization is mathematically represented as:

$$Norm(X') = \frac{X' - \mu}{\sigma} \tag{5}$$

where μ and σ denote the mean and standard deviation of the features, respectively. Unlike *batch normalization*, which performs normalization across the batch dimension, layer normalization operates feature-wise. This makes it particularly well-suited for sequential models such as Transformers that process variable-length sequences.

This normalization step plays several crucial roles:

- It maintains uniform feature scaling, ensuring that no single neuron dominates the activations.
- It stabilizes training, especially for long sequences or deep network architectures.

- It mitigates exploding and vanishing activations, keeping gradient magnitudes balanced throughout the network.
- It enhances convergence rates and reduces sensitivity to variations in the learning rate.

In summary, the combination of **residual connections** and **layer normalization** provides a robust architectural foundation for the Transformer model. This pairing maintains information preservation while ensuring stable gradient flow through multiple layers, enabling efficient and reliable deep learning performance.

5.9.4 Feedforward Network (FFN)

The **Feedforward Network (FFN)** is a very important element in every Transformer layer. It is a two-layer neural network that acts independently on each feature vector. While the attention mechanism discovers the relationships between tokens, the FFN works to generalize the representation of each token into a higher abstraction level.

Mathematical Representation

Mathematically, the FFN can be represented as:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where:

- x is the input feature vector,
- W_1 and W_2 are the weight matrices for the two linear transformations,
- b_1 and b_2 are bias terms, and
- $\max(0, \cdot)$ denotes the ReLU activation function, which introduces non-linearity.

Important Functions of FFN

1. **Non-Linearity Addition:** The inclusion of the ReLU activation function adds non-linearity, enabling the model to learn complex, non-linear transformations that go beyond the capabilities of a simple linear function.
2. **Feature Transformation:** The FFN maps input embeddings to a more expressive and richer feature space. This enhances the model's representational capacity and allows it to learn more complex patterns from the data.

3. **Independent Operation:** The FFN treats each token (or position) in the sequence independently and identically. This operation makes positional information invariant and strengthens the individual token representation.

4. **Capturing Complex Dependencies:** While the attention layer focuses on establishing relationships between tokens, the FFN improves the internal representation of each token, allowing the model to capture deeper dependencies and contextual relationships between features.

Block Structure

A **Transformer Block** is the building block of the Transformer model. Each block combines attention mechanisms and feedforward networks along with normalization and residual connections to facilitate stable and efficient learning.

Structure of a Transformer Block

A Transformer block can be expressed as:

$$\text{Multi-Head Attention} \rightarrow \text{Residual Connection + Normalization}$$

The **multi-head attention** mechanism allows the model to attend to various segments of the sequence simultaneously, learning multiple types of relationships.

A **residual connection** is introduced to improve gradient flow and prevent the vanishing gradient problem.

Layer Normalization stabilizes training and accelerates convergence.

$$\text{Feedforward Network (FFN)} \rightarrow \text{Residual Connection + Normalization}$$

The **Feedforward Network (FFN)** applies non-linear transformations on the output of the attention layer.

Another residual connection and normalization step are used to ensure stable feature scaling and prevent overfitting.

Stacking Multiple Blocks — Deep Reasoning

Multiple Transformer blocks are stacked on top of each other to enable deep reasoning.

- Lower layers extract simple or local relationships (such as word-level patterns).

- Higher layers extract abstract and complex reasoning (such as contextual meaning or semantics).

Strengths of the Transformer Block Design

1. **Parallel Processing:** Unlike Recurrent Neural Networks (RNNs), Transformers process tokens in parallel, which makes training significantly more efficient.
2. **Stable Learning:** Residual connections and normalization ensure that information is preserved across layers, preventing degradation in deep training.
3. **Expressive Representations:** The alternating structure of attention and feedforward layers enhances the model's ability to capture long-range dependencies and intricate relationships.
4. **Scalability:** Stacked Transformer layers allow the model to scale effectively, supporting deeper reasoning and a more comprehensive understanding of the data.

5.10 Final Prediction Layer

The **Final Prediction Layer** is the last stage in the Transformer architecture, where the information processed from the previous layers is aggregated and transformed into an output suitable for downstream tasks such as classification, regression, or other prediction objectives.

After passing through several Transformer blocks (each containing Multi-Head Attention and Feedforward layers), the model produces a sequence of feature embeddings — one for each input token or sequence position.

For most prediction tasks (such as sentiment classification, sentence-level regression, or document classification), a single vector representation that summarizes the entire sequence is required.

5.10.1 Aggregation of Features

After the input sequence passes through all Transformer blocks, each token position obtains a contextualized embedding that captures its relationship with all other tokens in the sequence.

Thus, the model outputs a tensor with the shape:

$$(\text{Batch}, \text{Sequence}, \text{Embedding})$$

In Our Project:

$$(4577, 5, 768)$$

where:

- $4577 \rightarrow$ number of samples in the batch,
- $5 \rightarrow$ sequence length (number of tokens per sample),
- $768 \rightarrow$ embedding dimension per token.

To generate the final prediction, this 3D tensor cannot be directly fed into a classifier. Instead, it must be **compressed into a single feature vector per sample** that effectively represents the meaning of the entire sequence.

5.10.2 Purpose of Pooling

Pooling refers to the operation employed to pool information from all token embeddings into one single representation. It serves as a bridge between the deep contextual representations produced by the Transformer and the output layer.

Types of Pooling Techniques:

- **[CLS] Token Pooling:**
  Prevalent in the case of models such as BERT, where the sequence starts with a special [CLS] token. Its representation (after Transformer layer traversal) is used as the final representation of the whole sequence.
- **Mean Pooling:**
  The mean of all the token embeddings is computed, summarizing the sequence by retaining overall semantic information.
- **Max Pooling:**
  The maximum value across tokens for every embedding dimension is taken, highlighting the most salient features.
- **Attention Pooling:**
  A learnable attention mechanism places weights on tokens, allowing the model to determine which tokens are most important in the final representation.

Why Pooling is Critical:

Transforms a sequence of embeddings into a vector of fixed length, independent of input sequence length.

Allows classification or regression heads (typically straightforward linear or MLP layers) to be applied to the data.

Pools contextual knowledge learned from all Transformer layers into a dense and informative vector.

Final Prediction Stage

Once pooled, the one resultant vector (typically of size 768 in a model such as BERT) is fed into one or more dense (fully connected) layers to perform the final prediction task.

Depending on the use case:

For classification, a softmax or sigmoid activation is used to yield class probabilities.

For regression, the dense layer outputs a real value.

This last output vector captures the model's comprehension of the whole input sequence and is the basis for making decisions about the prediction task.

5.11 Pooling Methods Employed in Our Model

Pooling is a critical process in Transformer-based models to get a fixed-size representation from variable-length sequences. In our model, we employ two primary pooling methods: **CLS token pooling** and **multi-head attention pooling**.

5.11.1 CLS Token Pooling

Concept:

The [CLS] (classification) token is a special learnable token inserted at the start of each input sequence.
In training, this token learns to aggregate information from all other tokens (or components) in the sequence.

How it works:

The input sequence of embeddings is modified as:

$$[\text{CLS}], x_1, x_2, \ldots, x_n$$

The whole sequence (including the CLS token) is fed through the Transformer encoder.

After encoding, the output for the CLS token — usually represented as:

$$z[:,0,:]$$

is used as the pooled representation of the whole sequence.

Interpretation:

The CLS token is a summary vector that accumulates global context over all the elements of the sequence.

This approach is commonly employed in models such as BERT, where the CLS embedding is passed to a classifier for a downstream classification, regression, or prediction task.

Mathematical Intuition:

Suppose H is the matrix of encoded representations:

$$H = [h_{\text{cls}}, h_1, h_2, \dots, h_n]$$

Then the pooled output is simply:

$$h_{\text{pooled}} = h_{\text{cls}}$$

Here, h_{cls} contains global information collected from all h_i vectors through attention mechanisms within the Transformer.

5.11.2 Multi-Head Attention Pooling

Concept:

Rather than taking the simple average (mean pooling) or max value (max pooling), attention pooling enables the model to pick up on which elements are more significant in the sequence.

This is achieved with a learnable query vector that attends over all elements with the Transformer's attention mechanism.

How it works:

Each element embedding h_i interacts with a learnable query vector q through attention.

Attention scores specify how much each component contributes to the overall pooled representation.

Multi-head attention has multiple query heads — each head can attend to distinct aspects or relationships within the sequence.

Mathematical Form:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Here:

Q is the learnable query (or set of multiple queries for multi-head attention),

K and V are obtained from the sequence embeddings,

The output is a weighted sum of all components.

Benefit:

Captures context-dependent significance of each component.

The model can emphasize more on informative or critical factors and less-weight less relevant ones.

This results in a more expressive pooled vector than with simple averaging.

What is Multi-Head Attention Pooling?

Objective:

To reduce a sequence of embeddings (such as tokens in a sentence) to one vector with learnable attention weights — and not mere average or max.

It's based on the multi-head attention mechanism of the Transformer architecture.

Shape Transformations

Input:

$$(B, L, D)$$

- B: Batch size
- L: Length of the sequence (number of tokens or time steps)
- D: Embedding dimension (features per token)

Every sequence element is represented by a D-dimensional vector.

Multi-head projections:

The input is mapped to several “heads,” and thus every head can pay attention to different parts of the sequence.

$$(B, L, D) \to (B, H, L, d_k)$$

where:

- H: Number of attention heads
- $d_k = \frac{D}{H}$: Dimension per head

Every head works on the input separately.

Attention scores:

There is a learnable query vector (not input token tied) used to calculate attention scores over all positions in the sequence.

These scores guide how much weight to assign to each token when constructing the pooled representation.

Output:

After calculating attention for each head:

All heads’ outputs are concatenated.

Shape:

$$(B, H \times d_k) = (B, D)$$

Then projected into a single vector of dimension D.

So final shape:

$$\text{Output: } (B, D)$$

5.11.3 Max Pooling

Picks the maximum value over the sequence length (L) along each feature dimension.

Ignores padded tokens (so padding tokens don’t influence results).

Example:

$$\text{max\_pooled}[i, d] = \max_{t=1}^{L} x[i, t, d]$$

5.11.4 Mean Pooling

Calculates the mean value along the sequence dimension for every feature.

Also ignores padded tokens.

Example:

$$\text{mean\_pooled}[i,d] = \frac{1}{L}\sum_{t=1}^{L} x[i,t,d]$$

5.11.5 Combined Pooling

Once getting various summary vectors:

- **CLS pooling:** the [CLS] token's embedding (from BERT-like models).
- **Attention pooling:** the learnable attention-based pooled output.

All these are joined together to create one single representation.

This means the final representation includes:

- Global context from attention
- Specific feature extremes from max pooling
- Overall context from mean pooling
- Task-level representation from CLS token

5.12 Shared Feature Extraction

Following the pooling operation (which combines data using CLS, attention, max, and mean pooling), the model contains a number of different feature representations. These are concatenated (joined end-to-end) into one large feature vector.

Example:

Suppose each pooled vector is of size 256, and there are 4 types (CLS, Attention, Max, Mean), then the combined vector has size

$$4 \times 256 = 1024.$$

This combined vector is the summary of the whole sequence — it captures:

- Task-level semantics (from CLS token)
- Contextual dependencies (from attention pooling)

- Strong features (from max pooling)
- Global averages (from mean pooling)

Processing the Combined Vector

After this combined vector is achieved, it is fed into a stack of fully-connected (dense) layers. These layers are:

- **Normalization:** (e.g., LayerNorm or BatchNorm) — stabilizes learning and facilitates convergence.
- **GELU Activation:** a smooth non-linear activation function (Gaussian Error Linear Unit), employed in transformers rather than ReLU since it is better suited for deep models.
- **Dropout:** randomly discards some neurons during training in order to avoid overfitting.

Purpose:

To combine all the representations that are pooled into a single shared feature space — that is to say, to produce an overall representation that can be used as a base for several downstream tasks.

5.12.1 Task-Specific Heads

After producing the common features, the network splits into separate heads — one for every type of task.

Regression Heads:

Utilized whenever the model has to predict continuous values (for instance, sentiment score, age, or numeric output).

Every regression target (say, predicting numerous continuous attributes) gets its own head.

Every head comprises fully-connected layers that transform the common features into scalar predictions.

Classification Heads:

Used when the model is required to make predictions over categories or classes (e.g., positive vs. negative sentiment, spam vs. not spam).

Each head corresponds to a classification task.

These heads produce logits (unnormalized scores) which are then turned into probabilities (through softmax or sigmoid).

- For binary, a single logit is output (then sigmoid is used).
- For multi-class, many logits are output (then softmax is used).

Combined Multi-Task Loss

$$\mathcal{L}_{\text{total}} = \sum_{i \in \text{regression targets}} \mathcal{L}^{i}_{\text{regression}} + \sum_{j \in \text{classification targets}} \mathcal{L}^{j}_{\text{classification}}$$

This means: **Sum of All Regression and Classification Losses**

In multi-task learning, one neural network is trained to do many tasks at the same time — for instance:

- Regression tasks (estimating continuous values such as scores or prices)
- Classification tasks (estimating discrete labels such as sentiment or category)

Each task has its own loss value that estimates how far the model's prediction is from the actual value.

5.12.2 Regression Loss – Smooth L1 Loss (Huber Loss)

Definition:

Smooth L1 Loss (or Huber Loss) is given as:

$$\text{Smooth L1 Loss} = \begin{cases} 0.5(y - \hat{y})^2/\beta & \text{if } |y - \hat{y}| < \beta \\ |y - \hat{y}| - 0.5\beta & \text{otherwise} \end{cases}$$

Parameters:

- y: Actual target value
- $\hat{y}$: Predicted value
- β: Smoothing parameter (here, $\beta = 0.5$)

Explanation:

When the absolute difference between actual and predicted values ($|y - \hat{y}|$) is small, the loss is quadratic — similar to Mean Squared Error (MSE).

When the absolute difference is large, the loss is linear — similar to Mean Absolute Error (MAE).

This hybrid construction bestows upon it the benefits of L1 and L2 losses:

- Robust and less outlier-sensitive (as compared to MSE, which severely penalizes large errors)
- Smooth around zero error, providing stable gradient updates (contrasting with MAE, which provides constant gradient)

Use case:

The Smooth L1 Loss works especially well for regression tasks like:

- Object detection bounding box regression (employed in models such as Faster R-CNN and SSD)
- Any regression task in which outliers can occur but should not control training

It enables the model to strike a balance between precision and robustness, which results in smoother training convergence and stability against extreme values.

5.12.3 Classification Loss – Focal Loss

Definition:

The Focal Loss is defined as:

$$L_{\text{classification}}(p_t) = -\alpha(1 - p_t)^{\gamma} \log(p_t)$$

Parameters:

- p_t: Predicted probability for the true class
- α: Balancing factor (e.g., $\alpha = 0.25$)
- γ: Focusing parameter (controls how much easy examples are down-weighted)

Explanation:

Standard cross-entropy loss treats all examples as equally important, which is a problem in class-imbalanced datasets (e.g., lots of negative examples, but not many positives).

The focal loss adapts cross-entropy by adding a modulating factor $(1 - p_t)^\gamma$:

- When a sample is classified correctly ($p_t \approx 1$), the term $(1 - p_t)^\gamma$ makes it small, and its contribution becomes smaller.
- When a sample is misclassified ($p_t \approx 0$), the term becomes large, and that example receives more attention during training.

Therefore, the model gives more attention to hard or misclassified samples and less to easy ones, resulting in improved performance on minority or hard classes.

Purpose:

- Works against class imbalance by giving more emphasis to hard-to-classify samples.
- Avoids the model from being biased towards major classes.
- Applied extensively in object detection tasks (e.g., RetinaNet) and imbalanced classification tasks.

5.12.4 Why These Two Together?

In multi-task learning (such as detection or joint regression-classification models):

- Smooth L1 Loss takes care of the regression task (such as predicting coordinates or continuous outputs).
- Focal Loss takes care of the classification task (such as predicting object class or label).

By using both together:

$$L_{\text{total}} = \sum_i L_{\text{regression}_i} + \sum_j L_{\text{classification}_j}$$

the network learns precise regression outputs as well as manages class imbalance well.

6. Evaluation Metrics

In this multitask learning task, the model performs two kinds of predictions in tandem:

- **Regression task** → Prediction of continuous attributes such as band gap (eV).
- **Classification task** → Categorical prediction such as stability (stable / unstable).

As such, distinct sets of metrics were employed to measure the performance of the individual tasks.

6.1 Regression Task Metrics

Regression metrics measure how well the model's continuous predictions match the actual target values. The following metrics were employed:

6.1.1 Mean Absolute Error (MAE)

Definition: MAE calculates the average size of the errors in a collection of predictions, irrespective of their direction. It assigns equal importance to all errors.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{6}$$

where y_i represents the true value, $\hat{y}_i$ is the predicted value, and N is the total number of samples.

Interpretation: Lower MAE indicates better model performance. It is easy to interpret since it is in the same unit as the target variable (eV for band gap).

6.1.2 Mean Squared Error (MSE)

Definition: MSE calculates the mean of squared differences between observed and predicted values. Squaring gives more weight to larger errors.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{7}$$

Interpretation: MSE penalizes larger errors more heavily than MAE, making it suitable when we need to discourage high deviations or outliers in predictions. A smaller MSE indicates better model performance.

6.1.3 Coefficient of Determination (R^2)

Definition: R^2 (R-squared) is the ratio of explained variance in the dependent variable to the total variance. It measures how well the model explains the variability in the target data.

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2} \tag{8}$$

where $\bar{y}$ is the mean of the true target values.

Interpretation:

- $R^2 = 1$: perfect prediction.
- $R^2 = 0$: model performs no better than predicting the mean.
- $R^2 < 0$: model performs worse than predicting the mean.

Reason for Use: MAE provides the average error magnitude, MSE penalizes large deviations, and R^2 offers an overall measure of goodness-of-fit.

6.2 Classification Task Metrics

For classification (e.g., predicting stable vs. unstable perovskite), we evaluate how effectively the model separates the two classes using the following metrics:

6.2.1 Accuracy

Definition: Accuracy measures the proportion of correctly predicted samples out of all predictions made.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

where TP, TN, FP, and FN represent True Positives, True Negatives, False Positives, and False Negatives, respectively.

Interpretation: Accuracy gives an overall view of model correctness but may be misleading for imbalanced datasets.

6.2.2 Precision

Definition: Precision measures how many of the samples predicted as positive are actually positive.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{10}$$

Interpretation: High precision indicates fewer false positives. This is critical when false positives are costly (e.g., labeling an unstable material as stable).

6.2.3 Recall (Sensitivity)

Definition: Recall measures how many of the actual positive samples were correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

Interpretation: High recall means fewer false negatives, which is important when missing a positive case is costly.

6.2.4 F1-Score

Definition: The F1-score is the harmonic mean of Precision and Recall, balancing both measures.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

Interpretation: The F1-score ranges between 0 and 1. A score of 1 indicates perfect precision and recall. It is preferred over accuracy when dealing with imbalanced datasets.

6.2.5 Confusion Matrix

Definition: A 2×2 table summarizing the model's classification results:

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

Interpretation: The confusion matrix provides a comprehensive view of model errors and helps identify class-specific imbalances.

Reason for Use: Accuracy gives a global performance overview, while Precision, Recall, and F1-score help analyze model performance on imbalanced data. The confusion matrix provides deeper insight into the distribution of prediction outcomes.

7. Results and Discussion

The multitask learning model proposed was tested on the perovskite dataset to learn two vital material properties simultaneously — band gap energy (eV) as a regression output and stability (stable/unstable) as a classification output. The performance on both tasks was tested by using the evaluation measures outlined in the above section.

7.1 Regression Task: Band Gap Prediction

For the regression task, the model reported the following performance metrics:

Mean Absolute Error (MAE): 0.4267

Root Mean Squared Error (RMSE): 0.7741

Coefficient of Determination (R^2): 0.7211

These figures mean that the model can precisely predict the band gap values with relatively small error. The value of $R^2 = 0.7211$ indicates that the model explains about 72% of the band gap data variance, indicating good predictive ability.

The small MAE and RMSE values indicate that the model reflects the intrinsic physical and chemical relations underlying perovskite material band gaps. The performance can be explained through the combination of *mat2vec* embeddings, *pymatgen*-based elemental descriptors, and attention-based transformer architecture, which together enable the model to extract meaningful patterns from compositional and structural data.

7.2 Classification Task: Stability Prediction

For the classification task, the model obtained the following results:

Accuracy: 0.7741

Precision: 0.7450

Recall: 0.7741

F1-Score: 0.7405

These measures show how well the model is able to differentiate between stable and unstable perovskite structures, maintaining a good balance between precision and recall. The F1-score value of 0.7405 demonstrates consistent performance across both positive and negative classes.

Upon closer examination of the evaluation report, it was observed that the model tends to misclassify samples belonging to **Class 1 (stable class)**. This shows that while the model correctly identifies most stable materials, it occasionally confuses borderline samples where the material properties lie near the stability threshold. This issue may arise due to a slight imbalance in the dataset or overlapping feature space between the two stability classes.

In future work, this limitation can be mitigated by using *class-weighted loss functions* or *data augmentation techniques* to improve class discrimination and model robustness.

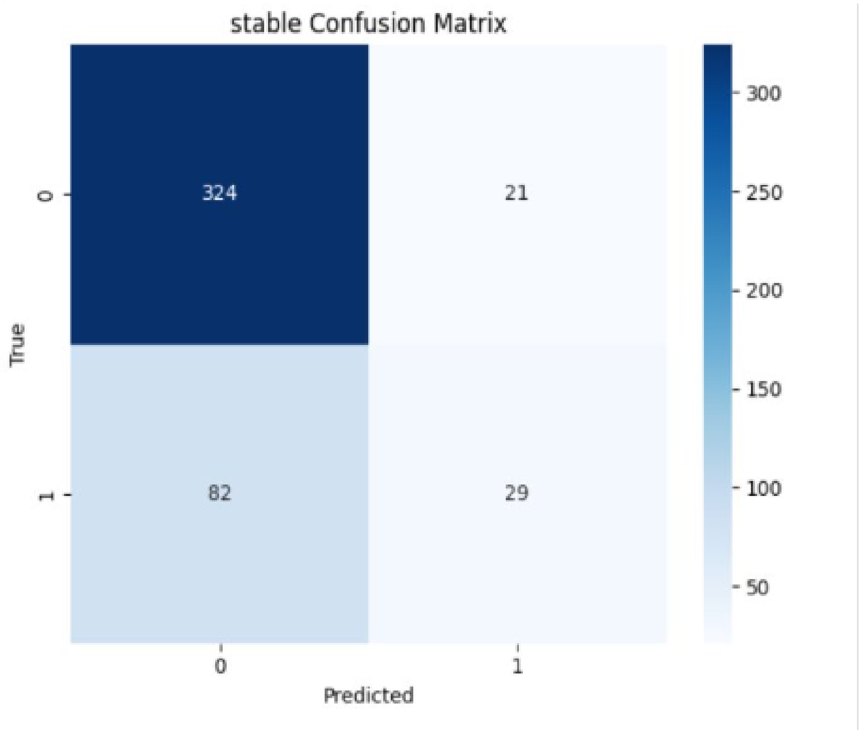


Figure 3: Confusion matrix showing the classification performance of the model.

This table presents the performance of the model on 456 test cases. The model achieved an overall **accuracy of 77.85%**. The model performed exceptionally well on the negative class (0), correctly predicting **324 instances**, which resulted in a high **specificity of 93.91%**.

However, the model struggled with the positive class (1), correctly identifying only **29 true positives** while producing **82 false negatives**. This imbalance led to a low **recall of 26.13%**, indicating that the model failed to capture most of the true positive samples.

This observation suggests that the model is biased toward the negative class and lacks sufficient sensitivity in recognizing stable (positive) instances. To address this issue, future improvements could involve applying *class-weighted training*, *resampling strategies*, or *focal loss* to enhance the detection of minority classes and improve overall model balance.

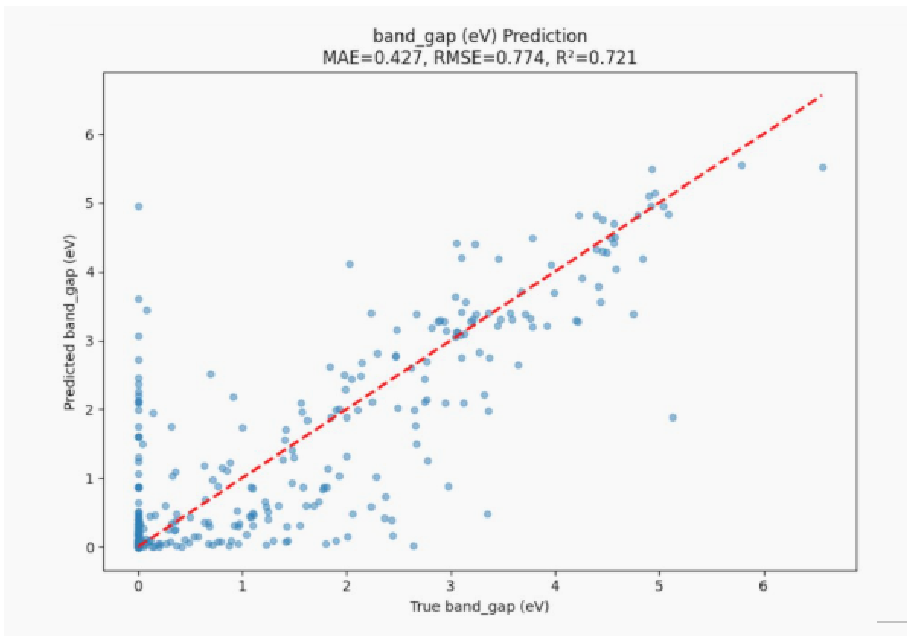


Figure 4: Predicted vs True band_gap (eV).

This scatter plot graphs the Predicted band_gap (eV) against the True band_gap (eV). The model is seen to have a decent fit, represented by the points clustering around the red dashed diagonal line (ideal prediction), and has an R^2 of 0.721. That being said, the Mean Absolute Error (MAE) of 0.427 eV and the Root Mean Square Error (RMSE) of 0.774 eV indicate that there is scope for improvement. The model exhibits significant scatter and confusion when the true band gap approaches 0 eV, primarily because the dataset contains a large concentration of zero values, making accurate differentiation in this region difficult for the model.

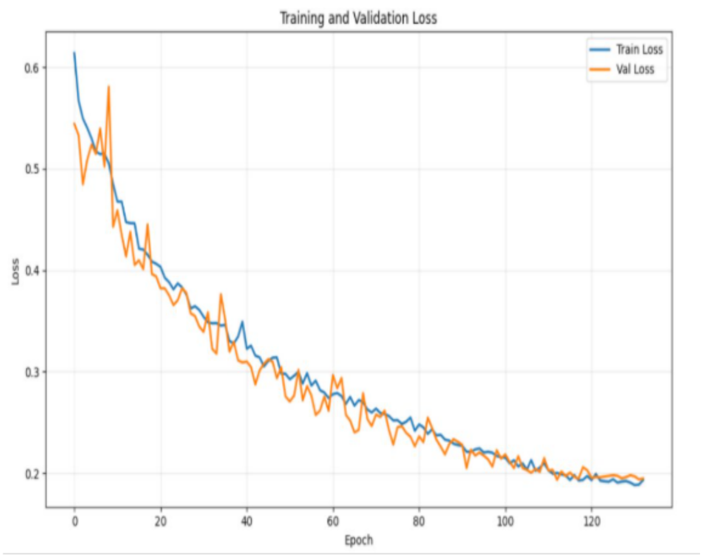


Figure 5: Learning curve of the model.

This graph shows the model's learning curve over 130 epochs. The Training Loss (blue) and Validation Loss (orange) both decrease steadily and come close to one another, showing the model is learning well without any appreciable drop in performance on new data. The two lines follow one another nicely across the training, which is indicative of the model not being underfit or seriously overfit. The slight variations in the Validation Loss are typical, and the ultimate stable low loss values close to 0.2 reflect an adequately generalized and stable model performance.

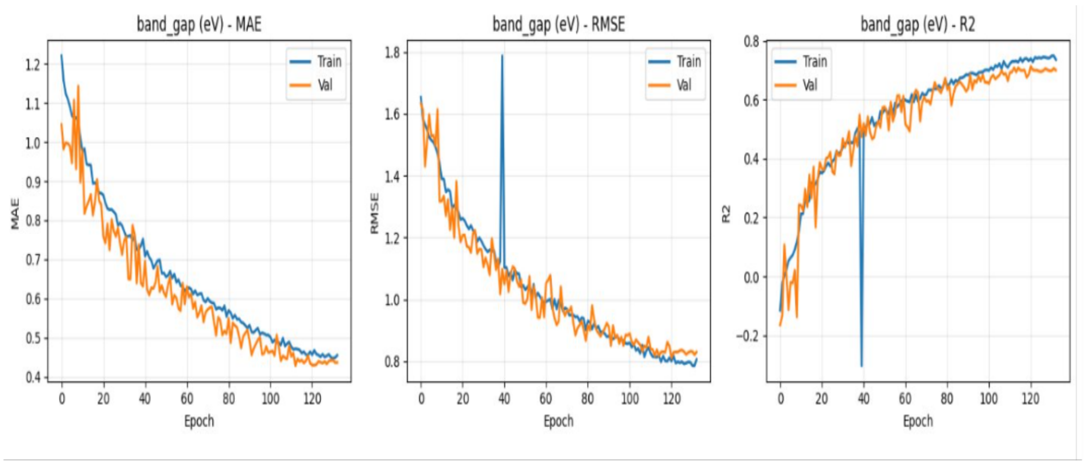
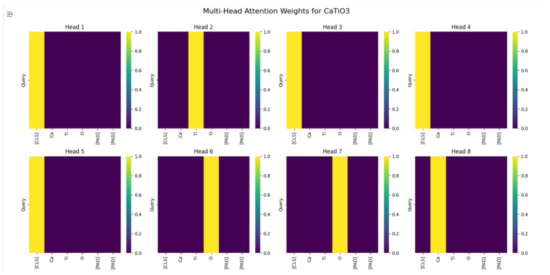


Figure 6: Evolution of regression metrics (MAE, RMSE, R^2).

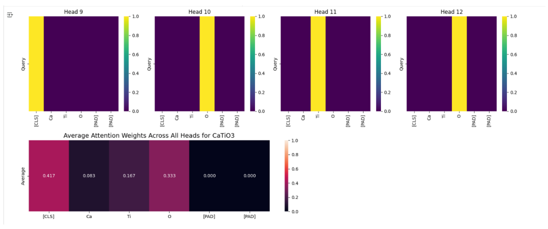
This plot demonstrates the evolution of three prominent regression measures—Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R^2—for training (blue) and validation (orange) sets over 130 iterations.

MAE and RMSE: Both the error measures decrease gradually and are close to each other throughout training, settling at MAE $\approx$ 0.45 eV and RMSE $\approx$ 0.8 eV. This means that the model is learning well and minimizing its error in predictions without much divergence between the training and unseen validation data.

R^2 **(Right Plot):** The R^2 measure, which accounts for the model's proportion of variance explained, continues to rise steadily for both datasets, settling around 0.75. The tight movement of the two lines verifies that the model generalizes well to new data, and the overall trend in all three plots indicates that the model has reached a stable and well-generalized level of performance by the last epoch.



(a) Attention Heads (1-12)



(b) Average Attention Weights Across Heads

Figure 7: Attention scores of the compound $CaTiO_3$ showing multi-head and average attention weights.

Multi-Head Attention Analysis for $CaTiO_3$:

The plots illustrate the attention mechanisms of the Transformer model, i.e., the attention weights of each head (Heads 1-12) and the average thereof, indicating how various components of the input sequence ([CLS], Ca, Ti, O) interact.

Attention Heads (Heads 1–12): The 12 heatmaps represent the self-attention weights of the multi-head attention layer. Query tokens (Y-axis) pay attention to Key tokens (X-axis), with color intensity representing weight (bright yellow $\approx$ 1.0, dark purple $\approx$ 0.0).

Regular Pattern: All query tokens mainly attend to two key tokens: the [CLS] token and the Ca token.

- **High Attention:** Ca and [CLS] columns are bright yellow, indicating near-maximal attention weights. The [CLS] token acts as a global aggregator; Ca is the most important chemical element considered by all heads.
- **Low Attention:** Ti, O, and [PAD] columns are mostly dark purple, showing that these tokens are down-weighted relative to Ca and [CLS].

Average Attention Weights Across All Heads:

Token	Average Attention Weight	Interpretation
[CLS]	0.417	Highest weight; main global information sink
Ca	0.083	Lowest among chemical elements
Ti	0.167	Doubles the attention compared to Ca
O	0.333	Second-highest; four times attention of Ca
[PAD]	0.000	No attention as expected

Conclusion: The head-by-head analysis (Heads 1-12) and the average analysis give an interesting insight:

Heads 1–12 View (Visual): Individual heads show strong attention on Ca and [CLS], making Ca appear highly significant.

Average View (Quantitative): Statistically, the attention on chemical elements follows O > Ti > Ca. The [CLS] token receives the highest overall attention (0.417).

This reveals that although each query token strongly attends to Ca individually, the overall attention pooled to Ca is the lowest among elements. O and Ti, on the other hand, are crucial contextual inputs for the [CLS] token. This demonstrates the model's learned distinction between globally important tokens ([CLS]) and contextually important chemical elements (O and Ti), while Ca serves as a critical source of information for other tokens.

Conclusion

In this work, we created a Transformer-based model for the prediction of major properties of perovskite materials, specifically the band gap, from available material datasets. The model proved to have a good capacity to learn the underlying patterns in the data, with an R^2 of 0.721, Mean Absolute Error (MAE) of 0.427 eV, and a Root Mean Square Error (RMSE) of 0.774 eV. The regression metrics and learning curves show that the model is properly generalized and free from underfitting or overfitting.

The investigation demonstrates that the model runs strongly throughout the majority of the data range but has greater uncertainty around the areas with dense zero values within the band gap, pointing to the difficulties in handling imbalanced datasets. Nonetheless, the Transformer model was efficient in modeling intricate relationships in the perovskite dataset and offered reliable and stable predictions across training and validation sets.

In summary, this project illustrates the capability of attention-based deep learning models in material property prediction. Improvements in low-value regions, the use of larger and more varied datasets, and the application of explainable AI methods to identify the most significant features affecting the model's predictions can be areas of future work. These can expedite the search and design of novel perovskite materials with specific electronic and structural properties.