Analysing the Impact of Ensemble Stacking Methods, Transfer Learning and Dimensionality Reduction on LLM Embeddings

Damien Bose	Roman Karim	Anirudh Neti	Suhail Merali	Bruno Bianchi
20018370	20019613	20006413	20078483	20117182

Abstract

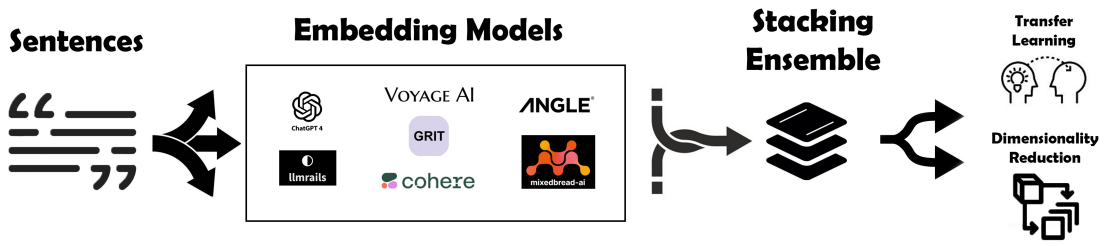
When evaluated on the Massive Text Embedding Benchmark (MTEB), Large Language Model (LLM) embedding models display different levels of performance. To incorporate their individual strengths and overcome their weaknesses, we propose an approach to ensemble various models through the concatenation of their sentence embeddings. We show that stacking these embeddings results in a consistent increase in performance across embedding tasks. We also evaluate the effectiveness of applying PCA to reduce noise while attempting to maintain performance. Lastly, we further transform this stacked embedding space using transfer learning. The experiment code can be found here.

1 Introduction

Natural Language Processing (NLP) models use different methods to calculate numerical vectors, called embeddings, which encode text syntax and semantics for easier processing. Embedding models are vital in various NLP scenarios, such as search engines, Retrieval-Augmented Generation (RAG) and sentiment analysis, due to their efficient representation of text. Whilst ensuring the preservation of the semantics of the sentences they encode, embeddings allow for mathematical operations to be performed on these vectors, enabling comparisons of text to be done algorithmically.

The performance of these embedding models can be evaluated using the wide array of tasks defined in the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2023). MTEB uses a total of 58 datasets to test across several categories, including Classification, Retrieval and Semantic Textual Similarity (STS), and provides a continuously updated leaderboard containing the models that perform best across these tasks. This leaderboard, however, makes apparent the current dilemma with embedding models: whilst there are multiple models that are effective for specific tasks, there is a lack of options that are generalisable across embedding use cases.

In an attempt to combat this, our project presents an approach to combine multiple performant models together, gathering the strengths of each individual model to create a singular, more powerful model capable of achieving high scores across the entire MTEB. The proposed methodology involves stacking the output embeddings of various models together, incorporating the semantic information from each model. Comparing the performance of the stacked model against the individual models, we aim to demonstrate the benefits of concatenating these together. Additionally, we explore the use of dimensionality reduction to get rid of unnecessary overlapping of information from the models whilst attempting to avoid a drop in performance. Furthermore, the concatenations are adjusted, and transfer learning is used to learn the most effective combinations of these models. All these methods are expected to produce a fully adaptable model that can be used across all scenarios involving NLP.



2 Related Works

Our work primarily draws upon research in ensemble techniques, dimensionality reduction and transfer learning within the domain of natural language processing (NLP).

Whilst many ensemble methods exist, Dietterich (2000) was the first to combine multiple learning algorithms to obtain a better predictive performance than a singular instance. A necessary and sufficient condition for ensembling embeddings is the diversity in how the models function and the similarity in accuracy. This is further explored by Al-Azani and El-Alfy (2017) combining different base classifiers using several ensemble learning techniques: voting, bagging, boosting, stacking and random forests. The work concluded with achieving a consistently improved score with the highest overall average attained when using the stacking ensemble learning method.

The concept of stacking is not new, with previous work ranging from stacking neural networks (Mohammadi and Das, 2016) to combining different classifiers (Taspinar et al., 2022). This can be extended to stacking sentence embeddings as explored by Subba and Kumari (2022). The authors introduce a heterogeneous stacking ensemble framework using Word2Vec, GloVe and BERT embeddings for sentiment analysis. They utilize the aforementioned embeddings to train a classifier model composed of LTSM, GRU and Bi-GRU base-level classifiers. The proposed framework was tested on four different datasets, significantly outperforming other base levels.

However, stacking these vectors increases the feature space's dimensionality leading to computational inefficiency commonly referred to in the literature as the *curse of dimensionality*. One way to mitigate the effects is to construct latent spaces of lower dimensionality and project the origin feature vectors onto that space using Principal Component Analysis (Abdi and Williams, 2010).

Akritidis and Bozanis (2022) expands on this idea to study the trade-off between accuracy and efficiency by varying the number of dimensions from 10^1 to 10^5. The authors found computation time to be faster but at the cost of decreased performance.

In an attempt to increase performance, we can refer to Moreo et al. (2022)'s use of transfer learning techniques such as **generalised funnelling (gFun)**. They build on the concept of *funnelling* to integrate outputs from various classifiers into a unified prediction model. They employ a meta-classifier trained on the aggregated outputs of multiple view-generating functions (VGFs) in an attempt to enrich the overall feature set and allow for effective knowledge transfer across different linguistic domains. The results indicate that this approach not only enhances the adaptability of the model across multiple languages but also significantly improves its overall performance in multilingual text classification tasks.

3 Methodology

3.1 Models

For simplicity, we chose to work with LLMs that generate embeddings of the same dimensions of 1024. We make use of the following LLM embedding models:

voyage-02 (VoyageAI, 2024): *Rank 114*. This model is instruction-tuned for classification, clustering and STS tasks.

UAE-Large-V1 (Li and Li, 2023): *Rank 10.* This uses AnglE optimisation, aiming to avoid the gradient saturation zones that come with the use of the cosine function.

cohere-embed-english-v3.0 (Reimers et al., 2024): *Rank 13.* Compared to its previous versions, v3.0 boosts performance by requiring the user to specify the purpose of the embeddings that will be generated.

GIST-large-Embedding-v0 (Solatorio, 2024): *Rank 16.* This model is a fine-tuned version of FlagEmbedding's 'bge-large-en-v1.5'. It does not require any instruction to generate embeddings.

bge-large-en-v1.5 (Xiao et al., 2023): *Rank 17.* Developed by the FlagEmbedding group from the Beijing Academy of Artificial Intelligence.

ember-v1 (LLMRAILS, 2024): *Rank 22.* By LLMRAILS.

mxbai-embed-large-v1 (Lee et al., 2024): *Rank 24.* Mixedbread AI's sentence embedding model. Similarly to UAE-Large-V1, it was trained using AnglE rather than cosine optimisation.

gte-large (Li et al., 2023): *Rank 25.* Developed by Alibaba Group.

3.2 MTEB Tasks

We evaluated our embedding models on a subset of task types from MTEB (Muennighoff et al., 2023):

Classification: This task involves classifying model embeddings into predefined categories. Using *AmazonCounterfactualClassification, Banking77Classification* and *EmotionClassification*.

Clustering: This aims to group a set of sentences or paragraphs into meaningful clusters. Using *ArxivClusteringS2S* and *RedditClustering*.

Reranking: Involves ranking a list of documents based on their relevance to a given query, evaluated using MRR@k and MAP metrics. Using *AskUbuntuDupQuestions*.

Pair Classification: This task determines the relationship between pairs of sentences or paragraphs, such as identifying duplicates or paraphrases. Using *TwitterSemEval2015*.

Retrieval: Comprises matching a set of queries with a collection of documents and ranking the results based on relevance, quantified by various metrics. Using *ArguAna* and *SciFact*.

Semantic Textual Similarity (STS): Assesses similarity between sentence pairs using Pearson and Spearman correlations, based on cosine similarity. Using *STS12* to *STS22*, *SICK-R* and *STSBenchmark*

3.3 Stacking Ensemble

The stacking ensemble method is performed by collating the output embeddings of multiple different models and concatenating them together. This creates a new, larger embedding that contains each individual model's semantic information. Each embedding is a vector of size $x_i \in \mathbb{R}^{1024}$, and, when concatenated together, they form a new vector $s = [x_1; ..; x_n] \in \mathbb{R}^{1024n}$, where n represents the **stack size**: the number of models concatenated together.

The embeddings were also normalised before ensembling them. We used L_2 normalisation, using the following formula:

$$x_i = \frac{x_i}{\sqrt{\sum_{j=1}^{1024} x_{ij}^2}}$$

This was a necessary step as the range of values in the embeddings varied between models, and normalising them ensured that all models contributed equally to the stacked embedding.

We experimented with all possible model types and stack sizes, from combining two models to combining all eight, giving 255 total different combinations. The performance of the stacked embeddinga was evaluated across the aforementioned subset datasets the MTEB, and the improvement of the stacked embeddings over the best model on its own was calculated.

3.4 Principal Component Analysis

Principal component analysis is a statistical dimensionality-reduction method usually used to reduce the number of features in a dataset. It con-

verts a set of possibly correlated features into a set of linearly uncorrelated features, called principal components, through an orthogonal transformation. The general steps of this transformation include computing a covariance matrix of the data, selecting the eigenvectors to determine the direction of the new feature space, and then projecting the original vectors onto the selected principal components.

The intuition behind PCA was simple: since multiple similarly trained models were concatenated together, there might be a significant overlap in encoded information by the features. Hence, the required dimensions for effective text encodings are likely lower. Large embedding dimensions may be undesirable for storage reasons and may slow downstream computation.

PCA Architecture: To test this hypothesis, we applied PCA to our generated stacked embedding and experimented with reducing the dimensionality to various sizes: 256, 512, 768, 896, 1024. By selecting datasets from our subset of MTEB that consisted of either a training or validation set, we applied PCA to the concatenated embeddings, fitting the PCA model for each combination and target size to transform the embeddings during evaluation.

3.5 Transfer Learning

This section tries to apply transfer learning to the stacked embedding models to optimise the embedding futher. The goal remains the same: we are trying to create a multi-purpose sentence/paragraph embedding model that works well on various downstream tasks. From the results section, we show that stacking LLM embedding models leads to mostly consistent improvement in all the downstream tasks, including STS, Classification, Clustering, Retrieval, PairClassification and Ranking.

However, it seems apparent that the different LLMs used have different skill sets: they are trained on different data, with different model architecture and design paradigms. Hence, a particular model A might better encode the semantics of Maths, and model B is better at doing so for Biology. Stacking is a simple, effective ad-hoc approach to combining this information into a unified model. However, stacking ensemble alone may be too simplistic. For example, if we stack models A and B and then try to use that ensemble for STS, it may perform worse than model A alone.

Hence, this section tries to intelligently reweigh the features of the stacked model, aiming to make this stacking ensemble generalise better across various domains. We do this by applying a linear transformation to the stacked ensemble. This model is trained with labelled STS data in the form $[text_1, text_2, similarity\_score]$.

Below, we go through the embedding space transformation architecture alongside how this model was trained.

Transformation Architecture: This model applies a transformation on the stacking ensemble. The input sentence *text* is passed through the 8 embedding models individually. This creates embedding of size $\mathbf{x_i} \in \mathbb{R}^{1024}$. Afterwards, these are stacked to create $\mathbf{s} := [\mathbf{x_0}; \ldots; \mathbf{x_7}] \in \mathbb{R}^{8192}$. We then try to learn the function $f_\theta : \mathbb{R}^{8192} \to \mathbb{R}^{8192}$ such that $f_\theta(s)$ leads to better MTEB benchmark performance across all tasks.

The architecture chosen for f_θ was a simple dropout layer followed by a linear layer. The dropout was introduced in training to prevent overfitting. Furthermore, a linear layer was chosen instead of some non-linear deep neural network because of issues with overfitting due to a lack of training data.

Dataset: For training data in the form $[text_1, text_2, similarity\_score]$. We note that there are multiple ways to represent data to optimise embeddings, but Li and Li (2023) showed that labelled data in this form performed best. The data was taken from the STS12 and STSBenchmark datasets, as these were the only datasets with training and validation splits. We did not train on the test data used during the MTEB evaluation to prevent overfitting.

Training: We used a Siamese architecture (Fig. 4) similar to that of Reimers and Gurevych (2019). However, one key difference is the loss function used for training. We refer back to Li and Li (2023)'s attempt to create a near SOTA model for the MTEB benchmark using *Angle Loss*. The loss function used is as follows:

$$L_{angle} = \log\left[1 + \sum_{s(x_i,x_j)>s(x_i,x_j)} e^{\frac{\triangle\theta_{ij}-\triangle\theta_{mn}}{\tau}}\right],$$

where τ is a temperature hyperparameter (which we set to 20 for this research), x_i, x_j are the vector embeddings of two sentences $text_i, text_j$ in the dataset, and $s(x_i,x_j)$ is their similarity score in the training set. Finally, $\Delta\theta_{ij}$ is the angle between embedding vector x_i, x_j. By optimising L_{angle}, we expect the angles between sentences with higher scores to be smaller than those with lower scores.

The authors introduce this loss function to optimise the angle between vector embedding directly rather than optimising the cosine function with its saturation zones. This reduces the impacts of vanishing gradients around 0 and 1 of the cosine function. To do this, they decompose the vectors into complex representations.

3.6 PCA + Transfer Learning

In the previous section, we learn the function $f_\theta : \mathbb{R}^{8192} \to \mathbb{R}^{8192}$ but given that the base models (3.1) produce embeddings of size 1024, we propose to learn the function in $\mathbb{R}^{1024}$. We build on from Section 3.4 to reduce the dimensions of our 8-stacked embeddings to 1024 and follow through with transfer learning as explained in Section 3.5.

4 Results & Analysis

4.1 Stacking Ensemble

We explore performance gains stacking ensemble vs taking the best model in the stack on its own. Figure 1 shows that stacking results in an overall improvement over the base models used in the stack. This difference in performance increases as we increase the stack size. This seems intuitive; stacking more models provides our ensemble with more information. However, the performance improvement is not drastic, and this is potentially because of information overlap from the embeddings generated by the different models. This increase in performance also comes with a downside: operations on the bigger stacked embeddings are much costlier, so there is a trade-off of convenience for a marginal performance increase.

We found stacking to have the biggest impact on STS tasks. We now compare 4096-dimensional embedding models. The STS top-performing model in the MTEB leaderboard is GritLM-7B, with an STS score of 0.8335 across 10 tasks. Our 4-stack models, also 4096 in dimension, averaged an STS score of 0.8436; that is, 1.22% over GritLM-7B. Our top scoring model scored 0.8464, ourperform-

	Length	Overall	STS
Best Base	1024	75.5788	84.5021
Ensemble	8192	**75.9794**	84.5583
PCA	256	74.4484	83.0545
	512	75.2456	83.6659
	768	75.3821	83.6737
	896	75.3945	83.6639
	1024	75.3811	83.6535
Transfer	8192	75.9738	**85.9459**
PCA + TL	1024	75.7017	85.2821

Table 1: Performance of stacking 8 embeddings and applying various transformations on STS and overall (subset of MTEB). The best base model overall refers to GIST whilst the best base model STS is Mixed-Bread.

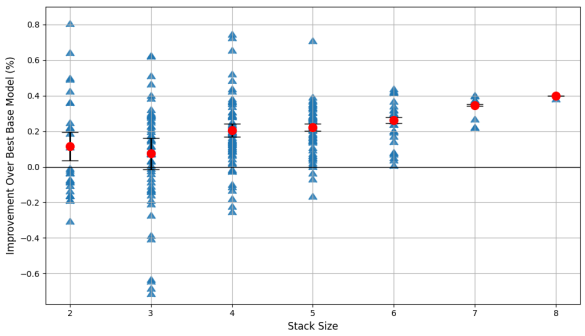


Figure 1: Improvement of stacked embeddings compared to the best base model by stack size, across all tasks. The red dot indicates average improvement, and black bars represent variance.

ing GritLM-7B by a margin of 1.55%. Our worst performing stacked model also performs slightly better than GritLM-7B by 0.4% on STS.

Some stacked ensemble models (notably of stack size 2, 3 and 4) lead to a decrease in overall performance, such as Gist+GTE (Figure 7). This is likely because GIST performs better than GTE, and the mistakes between GTE and GIST correlate during MTEB evaluation. Hence, ensembling GIST with GTE only decreases performance in this case. We theorise that stacking ensemble works best with similarly capable models which make uncorrelated encoding mistakes, for example Cohere + GTE (Figure 6).

4.2 PCA

As shown in Figure 2, looking at the performance of all the tasks in our subset, we see that PCA fails to improve the stacked embeddings.

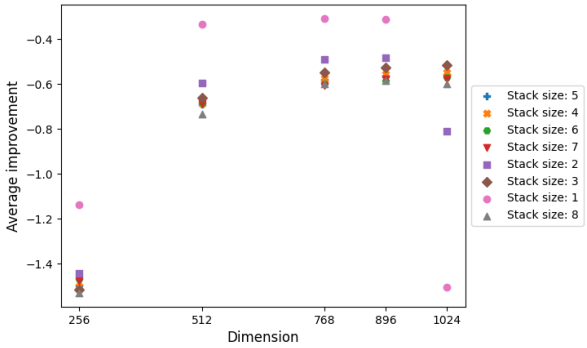


Figure 2: Average performance improvement of embeddings by stack size post-PCA, plotted against the reduced dimensions. Improvement is measured as the average percentage change in performance across all tasks in our MTEB subset, for all embeddings at the corresponding stack-size and dimension, compared to pre-PCA results.

From Figure 3, larger embedding stacks typically show better performance. However, Figure 2 indicates that PCA often reduces the effectiveness of these larger stacks compared to themselves before PCA. The decline in performance may stem from the high uncorrelation among the individual embeddings of the base models, influenced by varied LLM pre-training, the architecture of the neural networks that make up the LLM, and many other factors. Thus, while PCA targets high-variance components, assuming they hold more information, it might remove low-variance elements needed for a good performance in the MTEB.

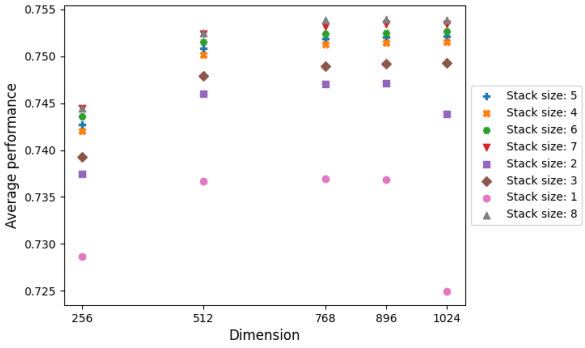


Figure 3: Average performance of PCA-reduced embeddings, plotted by dimension and stack size. Performance is the mean across all tasks in our MTEB subset for each stack size.

Although the dimensionality reduction of the stacked embeddings leads to performances worse than the original stacked model, we still receive a performance better than the best original base model in some cases (Fig. 5). This hints at a potential tradeoff between optimal performance and smaller dimension embeddings that are easier and more efficient to work with.

Despite the graphs indicating that PCA worsens performance, 22% of stacked embeddings slightly improve after the dimensionality reduction. This prompts exploration into different dimensionality reduction techniques as feature selection that does not rely solely on variance may prove more effective.

4.3 Transfer Learning

In this section, we discuss the result of applying transfer learning to the stacked ensemble of our 8 models discussed in the Methodology.

We remind ourselves that we train the transfer learning model on the training splits from the STS12 and STSBenchmark as these are the only STS datasets with training splits. Our results are shown in Table 1.

From this, we see significant improvements in the STS task performance. The average STS MTEB

score is 0.845583 for the "Transfer" model compared to 0.859459 for the stacking ensemble.

We achieved this 1.4% point STS improvement by taking active measures to prevent training set overfitting. During training, the model tries to learn the parameters of a 8192×8192 matrix, amounting to 64 Million parameters - which is likely too high for the amount of training data present. To mitigate overfitting, we implement early stopping by calculating the loss on our validation set (20% of training) to determine when overfitting was starting. Furthermore, a dropout rate of 0.4 was implemented to prevent the model from over-relying on a small subset of features. This was our form of regularisation.

However, overall, Transfer learning seems to lead to a lower overall score. The stacking ensemble had an overall MTEB score of 0.759794, while the score dropped slightly to 0.759738 when transfer learning was applied to that. This decrease of 0.0056% was surprising given that Li and Li (2023) showed that their loss function generalises well to other embedding tasks like Retrieval. However, our shortcoming was because the transfer learning had overfitted to the data distributions of STS12 and STSBenchmark. Combined, both datasets only have approximately 13k pairs of texts with similarity scores. This is likely not enough training data to be representative of the other MTEB datasets. Furthermore, tasks like Retrieval encode large paragraphs of data, and the training set we used is not very representative as it only contains short sentences of approximately 12 words. We strongly believe that major additional generalised gains could be achieved with more training data.

4.4 PCA + Transfer Learning

From reducing the dimensions of our 8-stacked ensemble model to 1024 using PCA and then training the transfer learning model on the training/validations splits from STS12 and STSBenchmark, we see how our model compares to others shown in Table 1. Our model has a 0.1626% performance increase over the best base model (GIST) overall and a 0.9231% increase over the best base model (Mixed-Bread) on STS. Given the reduction of the feature spaces dimensionality, we do not perform better than the $\mathbb{R}^{8096}$ counterpart losing out by 0.7784% on STS and by 0.3594% overall. Our PCA results all performed subpar to the best base model and thus, transfer learning remedies the loss of information to outperform base levels.

Similarly to Section (4.3), our shortcomings were due to the training datasets and the distributions when testing overall as we notice a 0.3655% drop in performance on our ensemble model whilst we have a 0.856% improvement on STS.

5 Conclusion

We have explored a novel application of a widely-used ensembling method for sentence embeddings, showing that they outperform the base embedding models they comprise across all tasks in the Massive Text Embedding Benchmark (MTEB) we tested on. Our approach highlights the potential of ensemble methods as a means to combine the strengths of different LLM embedding models and also clarifies the opportunities to further enhance these embeddings through the use of transfer learning.

Utilising transfer learning to train our eight-stacked embedding in semantic text-similarity tasks, we achieved 1.5 percentage points better performance in STS over the original stacked embedding. We also show that although using PCA worsens the performance when compared to the original stacked embedding, one might prefer to do this instead of using high-dimension embeddings that require more computing to work with. Furthermore, different dimensionality reduction methods could potentially result in better embeddings if features are correctly removed. Finally, we show that the combination of PCA and transfer learning often leads to embeddings that outperform the original embeddings, allowing for the best balance between ideally sized embeddings and performance.

5.1 Future Work

Due to computational limitations, we were only able to test a subset of the tasks that make up the MTEB. Future work on this project can include the computation and evaluation of the stacked embeddings for the rest of the datasets, using the same models or more.

5.1.1 Qualitative and quantitative data

Although MTEB provides datasets for each task, the majority are used for testing the models and are not suitable for training. Focusing specifically on our transfer learning methodology, only a few of the tasks in the STS category provided a training and validation dataset that could be used. Furthermore, a disproportionate number of sentence comparisons in the training dataset had a higher score: there was a higher number of sentences which are closely linked to each other, and consequently, fewer sentence comparisons were dissimilar. Future work could involve obtaining both a larger amount and a larger variety of data that can be used for training.

5.1.2 Correlation analysis

The outcomes of our experiment show that there is significant variance in the benchmark performances of the different stacked embeddings. This result implies potential strategies for making optimal concatenations, necessitating further analysis into why the different concatenations perform as they do and what makes a good combination of embeddings. As the differences in pre-training and LLM architecture lead to completely distinct embedding spaces, a direct comparison of these vectors may not be informative. Instead, one method would be to examine the strengths and weaknesses of each base model to identify the performance correlations between them; this could potentially reveal complementary or redundant stacks. We hypothesise that models whose strengths are close to orthogonal would create the best stacks.

5.1.3 Neural Network

A simple but effective strategy to increase the performance of the stacked embeddings on the benchmark involves using another type of transfer learning, specifically, utilising neural networks that accept the stacked embeddings as input and output a vector of the same dimensions. This neural network would train on the MTEB training data, allowing the model to learn more MTEB task-specific features that would lead to higher performances.

However, this method might inadvertently lead to worse generalisation in real-world applications, as training the neural network on specialised tasks may lead to overfitting. Regularisation, cross-validation and external training data could mitigate this issue and ensure the model's performance across tasks outside the benchmark.

5.1.4 Stacking Embeddings Across Modalities

As stacking sentence/paragraph embeddings shows promising results, future work could target ensembling embeddings derived from different modalities, such as image, video, and audio. This approach can either stack embeddings from single-modal embedding models or multimodal models, both of which have their own advantages and potential challenges. Since embeddings from single-modal models might have a more complex and deeper understanding of the modality they are designed for, they could potentially perform better than general multi-modal models. However, embeddings which are inherently designed to process multi-modal information might combine more seamlessly.

References

Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.

Leonidas Akritidis and Panayiotis Bozanis. 2022. How dimensionality reduction affects sentiment analysis nlp tasks: an experimental study. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 301–312. Springer.

Sadam Al-Azani and El-Sayed M El-Alfy. 2017. Using word embedding and ensemble learning for highly imbalanced data sentiment analysis in short arabic text. *Procedia Computer Science*, 109:359–366.

Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer.

Sean Lee, Aamir Shakir, Darius Koenig, and Julius Lipp. 2024. ember v1. https://www.mixedbread.ai/blog/mxbai-embed-large-v1/. Accessed on: 2024-04-15.

Xianming Li and Jing Li. 2023. Angle-optimized text embeddings. *arXiv preprint arXiv:2309.12871*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning.

LLMRAILS. 2024. ember v1. https://huggingface.co/llmrails/ember-v1/. Accessed on: 2024-04-15.

Milad Mohammadi and Subhasis Das. 2016. Snn: stacked neural networks. *arXiv preprint arXiv:1605.08512*.

Alejandro Moreo, Andrea Pedrotti, and Fabrizio Sebastiani. 2022. Generalized funnelling: Ensemble learning and heterogeneous document embeddings for cross-lingual text classification. *ACM Transactions on Information Systems*, 41(2):1–37.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2023. Mteb: Massive text embedding benchmark.

Nils Reimers, Elliott Choi, AMR Kayid, Alekhya Nandula, Manoj Govindassamy, and Abdullah Elkady. 2024. Introducing embed v3. https://txt.cohere.com/introducing-embed-v3/. Accessed on: 2024-04-15.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Aivin V. Solatorio. 2024. Gistembed: Guided in-sample selection of training negatives for text embedding fine-tuning.

Basant Subba and Simpy Kumari. 2022. A heterogeneous stacking ensemble based sentiment analysis framework using multiple word embeddings. *Computational Intelligence*, 38(2):530–559.

Yavuz Selim Taspinar, Ilkay Cinar, and Murat Koklu. 2022. Classification by a stacking model using cnn features for covid-19 infection diagnosis. *Journal of X-ray science and technology*, 30(1):73–88.

VoyageAI. 2024. Voyage ai embeddings. https://docs.voyageai.com/docs/embeddings. Accessed on: 2024-04-15.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding.

A Appendices

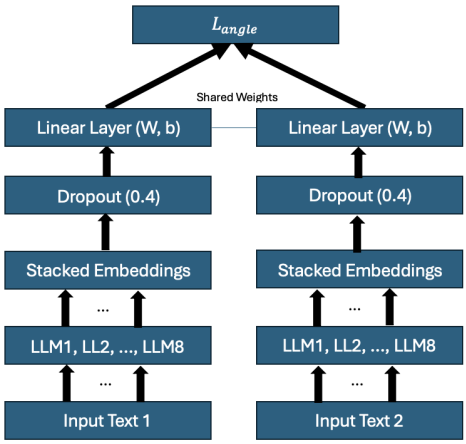


Figure 4: The Siamese Training Architecture Used for Transfer Learning of the Stacked Ensemble on the STS Data

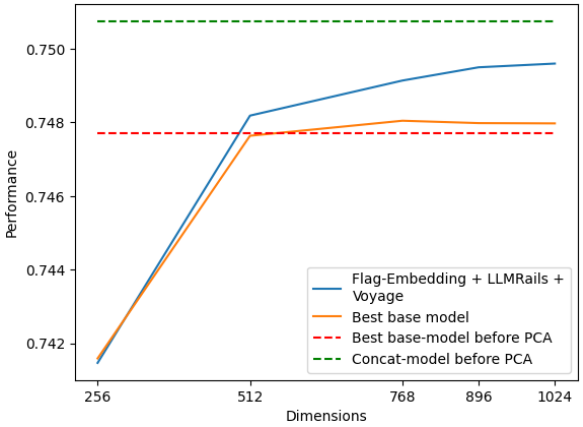


Figure 5: Average performance of stacked embeddings from Flag-Embedding, LLMRails, and Voyage, plotted by reduced dimension. Solid lines indicate post-PCA performance, and dotted lines represent pre-PCA. Performance averages are calculated across all tasks in our MTEB subset.

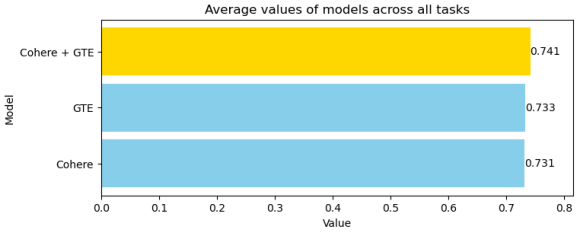


Figure 6: Average performances of the Cohere, GTE and Cohere+GTE embeddings across all MTEB tasks in our subset (Section 3.2).

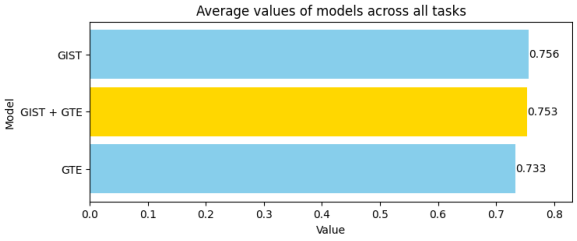


Figure 7: Average performances of the Gist, GTE and Gist+GTE embeddings across all MTEB tasks in our subset (Section 3.2).