University of Pisa
Human Language Technologies

Key Point Analisys: Key Point Matching and Key Point Generation with Pre-trained Language Models

Irene Pisani[1] and **Alice Bergonzini**[2]

[1] *M.Sc. Computer Science* - 560104 - i.pisani1@studenti.unipi.it
[2] *M.Sc. Digital Humanities* - 560680 - a.bergonzini1@studenti.unipi.it

September, 2022

Abstract

This work aims to describe simple approaches for solving *Key Point Matching* (KPM) and *Key Point Generation* (KPG) tracks proposed at Argument Mining 2021 in the context of the shared task on *Quantitative Summarization and Key Point Analysis* (KPA).
The presented methods rely on the fine-tuning of some state-of-the-art pre-trained language models (PLMs) both for KPM and KPG subtasks. Regarding the KPM task all the models explored were validated using the Hold-Out validation technique and their results were compared to analyze their effectiveness within the task. Leveraging DeBERTa pre-trained transformer, our best model yields to competitive performance since it achieved on TS set a mAP Strict and mAP Relaxed score of, respectively, 0,7035 and 0,8857.
For the KPG task, a simple baseline based on an abstractive summarization approach was provided; our system takes advantage of the pre-trained Google mT5 transformer to generate several key points that are finally properly selected.

1 Introduction

Key point analysis (KPA) is an emerging research field in Natural Language Processing (NLP) that aims to produce a non-redundant, succinct and informative list of key points from a collection of short and opinionated arguments; the generated key points must have the correct level of granularity such that they can be appropriately mapped to the corresponding arguments.
This challenge is closely related to the problems of learning sentence representation and semantic similarity and it was inspired by previous works on other practical applications such as Opinion and Abstractive Summarization, Computational Augmentation and Opinion Analysis [Bar-Haim et al., 2020a]

[Bar-Haim et al., 2020b].

During the 8th Workshop on Arguments Mining (ArgMining 2021) at EMNLP 2021, in an attempt to suggest relevant solutions to this recent problem, the Key Point Analysis (KPA-2021) Shared Task was proposed. As reported in *Overview of the 2021 key point analysis shared task* by Friendman et al. [2021], the overall task consisted of two complementary but independent sub-tasks: a first one on Key Point Generation and a second one on Key Point Matching (KPM).
The first track required generating salient key points from a given set of arguments belonging to a specific topic, while the second track required matching arguments to key points, given a finite set of predefined key points, by assigning a matching score to each <argument, key point> pair within the same topic. [Friedman et al., 2021]

This work mainly focuses on the KPM sub-task by presenting a simple but effective and accurate approach to the problem; the proposed method was inspired and motivated by the main findings and results achieved by the participants in the KPA- 2021 Shared Task.
In order to effectively compare our work with that of the participants, it was decided to use the same dataset provided during the competition and strictly follow the methodological directions provided by the organizers.
All of the following sections will refer to the KPM task, except for the Section 8 where we have reported a brief overview of the experiments run to set up the KPG baseline.
To test the reproducibility of the results obtained in this project the developed code was made available on Github[1].

2 Related Works

Until now, several of the models submitted to the KPA-2021 Shared Task represent the state-of-the-art for the KPM task: all of these are transformed-based architectures obtained through the exploitation of PLMs belonging to the BERT family [Friedman et al., 2021]. Since this structural approach, used by most of the participants, proved to be successful, we decided to take it into account as well. We developed our system for KPM by exploiting transfer learning through fine tuning of pre-trained models rather than relying on deep learning architecture to be implemented and trained from scratch (i.e., convolutional neural network, recurrent neural network or LSTM).

The results of the Shared Task showed that a clear predominance of the highest performing models were based on RoBERTa [Liu et al., 2019], which thus appears to be the most suitable PLM for solving KPM task. In fact, the highest performance was obtained by exploiting a Siamese Sentence BERT that takes as input the arguments and the key points embedded with RoBERTa Large; their overall network is trained with contrastive learning to learn a

[1]The code is available at https://github.com/irenepisani/Key_Point_Analysis

space of embeddings in which matching <arguments, key points> pairs are closer than not matching pairs [Alshomary et al., 2021].
Despite the excellent results obtained, such an architecture requires high computing power and long time to be trained. Since our goal was to develop a proficient system for KPM while being under certain computational limitations, it was decided to take as reference point those models that have a simpler topology.

Those systems used as our benchmarks have looked at the KPM task as a simple classification task by discriminating matching pairs from non-matching ones. In view of these considerations, our work focused on developing a binary classifier based on the fine-tuning of a PLM concatenated to some fully connected feed-forward layer.
More specifically, this work is strictly inspired by the models submitted to KPA-2021 such as Modern Talking [Reimer et al., 2021], Enigma [Kapadnis et al., 2021] and Poledro [Cosenza, 2021]. These mentioned works suggested us some fundamental methodological choices: respectively they motivated us to use the concatenation of argument and key points pairs as input to the model, to integrate the transformer output with additional information before passing it to the feed-forward layers, and finally provided some interesting insights about which PLMs were worth using.

3 Software design overview

The software was implemented in Python 3 using some auxiliary libraries such as Hugging Face[2] to import and exploit the PLMs and PyTorch[3] to implement our model's architecture and training settings. To ensure rapid code execution the whole system was run with Colab GPU.
Several classes were specifically built to perform separate functions that need to be combined together during the general execution of the software: this modular structure ensures conceptual clarity and greater accessibility to the different software components. The description of the used classes follow here.

- *Parser* class consists of some useful functions for importing, reading and pre-processing the required datasets.
- *Architecture* class is the core of the simulator. It initializes the model with tensor object by generating layers of neurons and configuring the weights of the connections between them. This constructor performs the feed-forward between the layers.
- *Trainer* class considers a specific configuration of hyper-parameter values and it has the fundamental role of launching the fine-tuning of a model to return a trained and evaluated model.
- *Validator* class handles tasks such as grid-search, model selection and model assessment.
- *Predictor* class is useful to load a pre-trained model from a check-point and use it to obtain predictions on new data.

[2]Hugging Face library: https://huggingface.co
[3]Pytorch library: https://pytorch.org

4 Dataset

The employed dataset is the official KPA-2021 Shared Task dataset called ArgKP-2021[4]; it is composed by a set of arguments, a set of key points and a set of labels. The organizer provided the full dataset already splitted in training set (TR), validation set (VL) and test set (TS).

All data covers a set of debated topics; for each topic and stance towards a topic (pro or cons) a set of arguments and a set of key points are given.
While for TR and VL set, the set of arguments are drawn from the IBM-ArgQ-Rank-30kArgs dataset [Gretz et al., 2020], for TS set the arguments set was carefully designed by an expert. All arguments are described by an ID, a corresponding topic in textual form, a stance toward a topic (i.e. pro or cons the topic), and the argument's text.
For both TR, VL and TS set the key points set is manually written by an expert: each key point is described by an ID, its corresponding topic in textual form, the stance towards its topic, and the key points text.
Regarding the labels set, it consists of a set of triplets in the form <arguments ID, key points ID, labels>. All labels are defined by manual annotation with the following procedure: each pair of <argument, key point> under the same topics and stance towards the topic is considered; to each one of these a positive label is assigned if a key point represents an argument (i.e. there is a match between key point and argument) and a negative label otherwise. In case of disagreement between annotators the pair is marked with an undecided labels (neither positive or negative). [Friedman et al., 2021]

4.1 Dataset statistics

	TR set	**VL set**	**TS set**
Num. topics	24	4	3
Num. arguments	5583	932	723
Num. key points	207	32	33
Num. < arg., kp.> pairs	24454	4211	3923
Num. positive pairs	17%	18%	14 %
Num. negative pairs	67%	64%	73%
Num. undecided pairs	14%	18%	13%

Table 1: Total number of topic, arguments and key points for each dataset. Given the total number of <argument, key point> pairs the percentage of pairs marked with positive, negative and undecided labels are reported.

Dataset statistic reported in Table 1 highlight the imbalanced nature of the dataset, which is composed by a clear predominance of negative examples. To prevent this from affecting the final performance, we try to focus on developing a model that is robust to imbalances. This fact has already been observed and reported by some participants to the Shared Task who adopted their own strategy to manage the imbalances.

[4]The dataset is available at: https://github.com/ibm/KPA_2021_shared_task

5 Method

In order to create an effective model that is able to successfully fulfill the KPM task we decided to base it on a pre-trained transformer. More precisely, we explored various options made available by the Hugging Face library. The transformers that we used are: BERT [Devlin et al., 2018], RoBERTa [Liu et al., 2019], ALBERT [Lan et al., 2019] and DeBERTa [He et al., 2020]. All of these were fine-tuned making the model applicable to the KPM task. Then, we proceeded to perform a hyper-parameter selection through a grid search with hold-out validation.

5.1 Pre-processing

Knowing that the transformer tokenizer performs some pre-processing itself we decided to keep other pre-processing procedures of the textual information to the absolute minimum. The few techniques that were applied to the numerous sentences are the following: transformation to lower case and removal of punctuation. These were applied to the textual content of the arguments, the topic and the key points. Additionally, one hot encoding has been performed to the stance datum with the intention of transforming all -1 values into 0s.

The pre-processed data has then been fed to the transformer's WordPiece tokenizer. The argument and topic have been previously concatenated to form the first sentence and the topic represents the second sentence. The tokenizers are of two types depending on the pre-trained model that was used, resulting in two different types of formatting for the models input. For BERT the formatting is `[CLS] sentence [SEP] sentence [SEP]`, while RoBERTa, ALBERT and DeBERTa presuppose the encoding `<s> sentence </s> </s> sentence </s>`. In addition, the tokenizer also carried out truncation to a maximum length of 80 tokens and added padding to the shorter sentences.
From the tokenizer we obtain for each sample the following: the input ids (which are the indexed tokens of the sentences with the special tokens), the attention mask and, only for BERT, the token type ids. These will be fed to the transformer as input.

5.2 Model architecture

The architecture employed incorporates inside itself a PLM; in this way we leverage this model for learning the contextual representations of <argument,key point> pairs. The pre-trained models are all transformers belonging to the BERT family: the representative capabilities of BERT, ALBERT, RoBERTa and DeBERTa were tested out in this project to solve the KPM problem. For each model we exploited the base version specified in Table 2.

The overall design of our developed architecture is shown in Figure 1 and can be divided in 3 main components: an encoding component, a context integrating component, and a matching component.
The encoding component consists of the pre-trained transformer and it takes the concatenation of the tokenized <argument + topic, key point> pair as

Model name	Model type
BERT	`bert-base-uncased`
ALBERT	`albert-base-v2`
RoBERTa	`roberta-base`
DeBERTA	`microsoft/deberta-base`

Table 2: Used pre-trained model and corresponding employed version.

input; here the input is passed through one of the transformers listed above. For each used transformer, the tokenization step was carried out with their corresponding tokenizer.
Then, in the context integrating component, the transformer's output is concatenated with the encoded stance and given as input to a fully-connected feed-forward layer with dropout and identity activation function. The output of this hidden layer is finally passed to the matching component.
This last component is the output layer, a last dense layer in which Sigmoid activation function is used to get the final matching score in the range of [0,1].

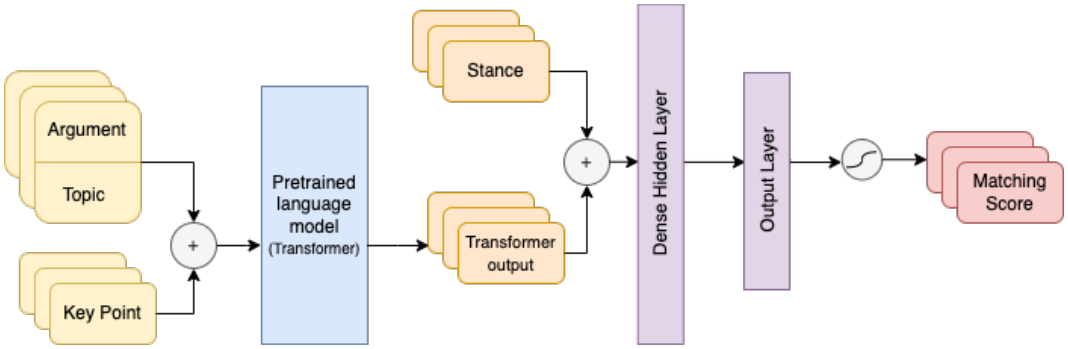

Figure 1: Overall design of model's architecture.

5.3 FineTuning

The whole architecture described in Section 5.2 required to be fine-tuned end-to-end in order to fulfill the KPM task; this implied not freezing the pre-trained transformers but fine-tuning them as well.
The fine-tuning procedure was performed using the Binary Cross Entropy (BCE) loss function and Adam with weight decay as optimizer. Some additional training techniques such as linear learning rate scheduler and gradient clipping were adopted to enhance the learning process: the former linearly decays the learning rate of each parameter group while the latter clips the gradient norm to prevent exploding gradient.

All hyper-parameters values used for fine-tuning belong to a range of values close or even equal to the configurations suggested in the original paper of each model for its fine-tuning. In addition, we have used a preliminary screening phase of the model behavior to fix the value of some hyper-parameters before starting the model selection phase in order to decrease the execution time. Some Adam's parameters, for example, were fixed: we used β equals to (0.9,

0.999), ϵ equals to 1e-08 and weight decay factor equals 0.01. The gradient norm was clipped to a maximum of 0.1 and we use 0.3 and 1.0 as start factor and end factor for multiplying learning rate in the first and last epoch respectively; this linear changing process is carried out for a total number of 5 iterations. The joint impact of other hyper-parameters such as learning rate, batch size, number of neurons and dropout rate of the hidden dense layer were studied within a grid search during the model selection phase.

Each model was fine-tuned over the all the <argument, keypoint> pairs in TR set annotated with a positive or negative label. All the training pairs labelled as undecided were judged useless during fine-tuning procedure.

5.4 Evaluation Metrics

In the evaluation phase, we follow the evaluation method proposed during the KPA-2021 shared task. This method involves evaluating the model by taking into account, for each argument, only the argument-key point pairs with the highest obtained matching score; then the evaluation was executed only on the top 50 <argument, key point> pairs selected based on the confidence score. The adoption of this evaluation technique was made easier by the opportunity to leverage the proper code already developed and made available online by the organizers [5].
Moreover, the quality of the computed outputs was analyzed by means of some alternative evaluation metrics that could take into account the need to evaluate the generalization capabilities also on undecided labeled data as well as the goodness of matching score obtained without the application of a threshold.

As a first step, since the model outputs a matching score between 0 and 1 we set a threshold at 0.5. For all matching scores predicted by the model having the corresponding ground truth label annotated (i.e., with label other than undecided) the model recognizes pairs as matching if their output is greater than 0.5 assigning them a positive predicted label, otherwise it assigns them a negative label. In this step we discarded all those potentially ambiguous pairs marked by an undecided label from the evaluation and we evaluated the model by computing typical classification metrics such as accuracy, precision, recall and F1 measure.

In a second step we have also included the potentially ambiguous pairs marked by annotators with undecided labels in the evaluation and we evaluated the model predictions based on the official metrics in KPA-2021 shared task: Strict Mean Average Precision (mAP-Strict) and Relaxed Mean Average Precision (mAP-Relaxed). More in details, in the strict evaluation undecided labelled pairs are treated as negative matching while in the relaxed evaluation they are treated as positive ones.

[5]The code for properly evaluate models is available at:
https://github.com/IBM/KPA_2021_shared_task/blob/main/code

5.5 Validation Schema

In the validation phase, our purpose was to explore the fitting capability of a PLM fine-tuned with a particular hyper-parameter configuration: all these resulting models were subjected to a rigorous validation method.
Since the ArgKP-2021 dataset is provided already divided into TR, VL and TS set, it was not even necessary to perform the dataset partitioning on our own, but it was possible to take advantage of the given validation scheme (i.e., data partitioning).
In the validation procedure, both the model selection and the model assessment phase were performed through the hold out validation technique. This choice was made to shorten the execution times but it resulted in a less accurate measurement of the models performances reported in the Subsection 6.

5.5.1 Model Selection and Model Assestment

A model selection process was executed for each used PLM: a grid search on the hyper-parameter space has been performed to evaluate them with different configurations of hyper-parameters, therefore, for each combination a validation process is launched. Once the grid search is over the best hyper-parameter configuration was found by looking for the model that has achieved the lowest validation loss. Then, this winning model has been retrained with the best hyper-parameter configuration on the full dataset composed by the concatenation of TR and VL sets, thus we obtain the best model. During the final retrain we don't need to still use a VL set since no early stopping conditions were used. We didn't use them since we have observed most of the models tend to go in overfitting very soon. This motivated us to fix a small number of training epochs (i.e. 2-3 training epochs). At the end of this stage we came up with 4 resulting best models: one for each employed PLM. We ranked these 4 models based on their average mAP score obtained on VL set. Then, the phase of model assessment is carried out on the first-placed model and it was evaluated on the basis of the predictions produced over the given TS set.

5.5.2 Grid Search

The hyper-parameters space was explored following a two-step approach.
The experimental screening phase involved a manual analysis of the effect of hyper-parameters values and of small combinations of them: these observations motivated us to discard those ranges of values that influenced the model's behaviour in a negative way (i.e. causing strong instability, slowing down or affecting the convergence, stuck the learning process in poor local minimum). Thanks to this manual procedure it was possible to reduce the number of combinations of values on which to perform the grid search and to minimize its computational cost. We also agreed that in most cases the values worth analyzing through grid-search were those reported in the presentation articles of each PLM.
The final grid search was performed on the remaining possible ranges of values in order to identify the most promising configurations among the tested ones. More accurate details concerning the values tested in each grid search for each

language model can be find in Appendix A.2, Table 10.

6 Experimental results

This section reports the outcomes of the experiments carried out by fine-tuning different PLMs. Since a model selection procedure was launched for each studied transformer, we ended up with 4 final models chosen. The performances of these 4 models are listed in the Table 3.

Training Performance (TR)					
Model	**Loss**	**Precision**	**Recall**	**F1**	**Accuracy**
BERT	0,3049	0,74	0,68	0,62	0,68
ALBERT	**0,2551**	**0,78**	**0,75**	**0,72**	**0,75**
RoBERTa	0,3799	0,7	0,62	0,56	0,62
DeBERTa	0,2825	0,75	0,73	0,7	0,73

(a) Hyper-parameters values for fine-tuning $\text{ALBERT}_{\text{BASE}}$.

Validation Performance (VL)							
Model	**Loss**	**Precision**	**Recall**	**F1**	**Accuracy**	**mAP(S)**	**mAP(R)**
BERT	0,5176	0,7433	0,74	0,7333	0,74	0,5252	0,8366
ALBERT	0,4182	0,7866	0,7866	0,7866	0,7866	0,5923	0,8428
RoBERTa	0,3171	0,8133	0,8066	0,8066	0,8066	0,5973	0,8711
DeBERTa	**0,3163**	**0,8266**	**0,83**	**0,82**	**0,83**	**0,6156**	**0,8832**

(b) Hyper-parameters values for fine-tuning $\text{ALBERT}_{\text{BASE}}$.

Test Performance (TS)						
Model	**Precision**	**Recall**	**F1**	**Accuracy**	**mAP(S)**	**mAP(R)**
BERT	0,7066	0,6766	0,6266	0,6766	0,6332	0,7948
ALBERT	0,7700	0,7666	0,7500	0,7666	0,6884	0,8294
RoBERTa	0,7966	0,7933	0,7900	0,7933	0,6829	0,8721
DeBERTa	**0,8266**	**0,8133**	**0,7933**	**0,8000**	**0,7035**	**0,8857**

(c) Hyper-parameters values for fine-tuning $\text{ALBERT}_{\text{BASE}}$.

Table 3: Models performance achieved in training (TR), validation (VL) and testing (TS) phase. Precision, Recall, and F-1 score are computed with weighted modality (i.e. not macro); mAP(S) and mAP(R) stands respectively for Strict and Relaxed mean Average Precision. All reported values refer to the average of 4 different runs for TR, VL and TS.

The training results have suggested that the ALBERT-based model was able to fit all the data better than other models. Nevertheless, looking at validation performance it became clear that the model that outperformed the other ones in the predictive phase on never-before-seen data turned out to be DeBERTa, followed in order by RoBERTa and ALBERT. The BERT-based model ranked last in both training and validation phase.

Results obtained on the VL set led us to confirm the fine-tuned DeBERTa-based model with hyper-parameter values reported in Table 10 as the best final model found for this KPM task; this choice was made by looking at the Average mAP value obtained on the VL set. Afterwards, we proceeded to evaluate this final best model in model assestment phase on the TS set.
However, since we had 3 other fully fine-tuned models we decided to evaluate them on TS set as well; all performances are shown in the Table 4. In this step no rules were broken since we did not use TS set to choose our best model, but only to get some more information about the other models already discarded. It is good to mention that this obtained information did not affect the final ranking. which was done by looking only at the performance reached on VL set.

We can observe that for each transformer-based models, the values of the evaluation metrics obtained on TS set correspond, more or less, to these obtained on VL set, except for the strict map which shows a significant increase of 8.5% on TS set with respect to the VL set. At the same time, the classification metrics show a small decrease from the values obtained on the VL set.

Rank	**Model name**	**Average mAP (VL)**	**Average mAP (TS)**
1° model	DeBERTa	0,7617	0,7946
2° model	RoBERTa	0,7342	0,7775
3° model	ALBERT	0,7176	0,7589
4° model	BERT	0,6809	0,7140

Table 4: Final ranking for fined-tuned models based on the avarage mAP (mean between mAP Strict and mAP Relaxed). All reported values refer to the average of 4 different runs for TR, VL and TS.

Our final Deberta-based model outperforms on TS set the other fine-tuned models by 0.02%, as shown in the table 4, and is also able to compete with many of the participants in the Shared Task. In fact, the table 12 in the Appendix B.2 shows that despite our results not approximating the first-ranked ones and the ones with a more sophisticated architecture, it still manages to compete with simpler architectures such as those proposed by the teams XLNet and RoBERTa-LC [Friedman et al., 2021] which are based on the fine tuning of RoBERTa and XLNet, respectively.

In any case, the comparison of our models with those of shared task revealed both strengths and weaknesses.
In this analysis it remains crucial to note that most of the participants mentioned above achieved the results reported in the table by exploiting the large versions of the PLMs; in this project, instead, we were able to achieve competitive results by exploiting only the base versions. This point is relevant as it emphasizes how it is not guaranteed that the base version of PLMs performs worse than the larger one; under some settings good behavior can be attained as well.

The obtained results also led us to investigate the quality of the implemented architecture and the fine-tuning performed. On one hand, we were able to obtain final models that surpassed similar proposals (e.g., our version of RoBERTa surpassed the one proposed by RoBERTa-LC); on the other hand, the comparison with Poledro's work [Cosenza, 2021] revealed a possible underfitting condition of our model. This reflection stems from the fact that although the performances on TS set are extremely close to each other (with a slight advantage of 0.014 % for Poledro) it needs to be taken into account that its performances achieved on the TR set and VL set are significantly higher than ours. This forced us to make the assumption that by boosting DeBERTa's fine tuning more we would be able to achieve an even better performance. This last assumption, however, has not been substantiated by practical experiments, which could either verify or disprove it.

All of this led us to raise a final consideration about our fine-tuned DeBERTa: the final model was chosen based on the validation loss as this is a conventional rule in model selection procedures. However, some posthumous experiments showed that by choosing the final model on the basis of the highest average mAP achieved on VL set, the model on TS set could reach a maximum peak of about 0.72 of strict mAP and 94.5 of relaxed mAP.

7 Error Analysis

To understand what elements to focus on for possible future refinements of our model we carried out a manual analysis of the errors made during the testing phase by our final best model (fine-tuned DeBERTa) looking for possible conditions that may affect predictions.

A first look at the misclassified predictions shows how semantic similarity between words in the argument and in the key-point might be a reason for the model to fail. When elaborating arguments and key-points, the model is working with information coming from the same semantic area and also with overlapping words, making it hard to recognize small facets of meaning between terms. As an example consider the following key-point and argument couple: "Social media regulation can lead to political abuses by the government", "it would be a bit weird to have the government behind you watching and controlling everything you do". Terms like *government*,*political abuses*, *controlling* are likely to be associated with a similar meaning.

The unbalanced nature of our TR set, where the vast majority of the samples are labeled as 0, leads us to assume that the accuracy level must be in a way influenced by the number of samples available for each category, making samples with label 0 easier to predict for our model. A more sophisticated analysis has however demonstrated that there is no big difference between the two classes. To prove this, a confusion matrix has been built based on the predictions made by the model on TS set (excluding non-labeled samples) and the outcome has been related to the totals of each category. As we can see in Table 5, the percentage of true positives detected differs by only 1% from

the percentage of true negatives making the predictions of the model highly balanced.

	Correct prediction	**Wrong Prediction**
Actual pair with negative label	88%	12%
Actual pair with positive label	87%	13%

Table 5: Percentage of correct and wrong predictions computed by the model for each class.

Taking a closer look at the predictions we can state some additional observations. When a value very close to 0.5 is assigned to a couple (key-point,argument) we can notice that most of the time those specific matchings are more difficult to determine also as humans. In these cases the arguments seem to be more ambiguous and vague (some examples are given in Table 6). So it is important to remember that key-point matching is a task that is very much influenced by subjectivity (disagreement between annotators is key evidence of this).

Key point	**Argument**	**Label**	**Score**
Mandatory vaccination contradicts basic rights	Every parent should be able to decide whether to vaccinate their children or not.	0	0.585
Mandatory vaccination contradicts basic rights	No, the obligation to vaccinate children should not happen because God made us free for us to make our decisions.	1	0.496
Child vaccination saves lives	Routine childhood vaccinations should be mandatory for virus prevention.	0	0.545

Table 6: Matching score examples with value close to 0.5

A more specific analysis has been carried out to check any particular conditions that may influence the outcome of the model. We checked for the presence of correlations, for example the number of stopwords has been count, but it doesn't seem like the matching score has any correlation with the number of stopwords found in the numerous arguments. Next, we checked if the difference between the length of the argument and the length of the key-point could in a way impact the efficiency of the model. We found that the average difference in those examples misclassified with 0 is significantly higher than in the other cases. This leads us to believe that the model might struggle with longer arguments, and thus bigger amounts of information to elaborate.

8 Baseline for Key Point Generation

After concentrating most of our efforts in the Key Point Matching task, to complete our contribution to the more general problem of Key Point Analysis we decided to equip our project with a baseline system for Key Point Generation (i.e., the first of the two tracks proposed at the KPA-2021 Shared Task).

To set up our KPG baseline we have tried to follow an approach that could be consistent with the one adopted for the KPM track; therefore, we made the methodological decision to rely, again, on a PLM.

This choice led us to take as our guideline the systems developed by Enigma [Kapadnis et al., 2021] during its participation in the Shared Task. Their approach, as well as the one we have adopted, looks at the KPG task as an abstractive summarization task. They fine-tuned Pegasus [Zhang et al., 2019] PLM giving it as input the concatenation of arguments and topics and as target the ground truth key points provided in the competition. The model returned in output a set of summaries that have been sorted according to the rouge-1 score calculated with respect to the ground-truth key points. Then the top 5 generated key point are selected as final generated key points.

8.1 Methodological survey

Our plan for carrying out the KPG task was to reproduce the Enigma's KPG system by introducing some changes and improvements.

Our neural approach differs from Enigma's one as it is not based on the Pegasus model but rather revolves around mT5 language model [Xue et al., 2020], a Text-to-Text Transfer Transformer that we use to leverage for the abstractive summarization problem.
The mT5 architecture is based on T5, that was pretrained in a text-to-text framework, and the overall model is extremely versatile, it offers interesting abilities such as understanding long passages and generating coherent text that captures salient features in the input. We therefore decided to use mT5 for our summarization problem since it is a state-of-art model in performing many sequence to sequence tasks.
Nonetheless, in contrast to Enigma, we did not pass the concatenation of <arguments, topic> as input to the model but only the arguments since some initial experiments showed that by passing the <arguments, topic> pair to the model the generated key points turned out to be too short, very general, and uninformative, thus looking much more like a topic than like a key point. By giving only the arguments as input to the model instead, the quality of the key point produced in output has been improved significantly in terms of informativeness. The last difference with the Enigma project is in the scoring and ranking modalities. For each generated key point, the rouge-1 score was computed with respect to the corresponding argument instead of with respect to the ground truth key point.

A detailed description of the employed method is reported below.

8.2 Dataset and validation split

It is worth mentioning that since this task remains contextualized within KPA-2021 Shared Task the used dataset is always ArgKP-2021, composed by the same argument set, key point set, label set. Again, we proceeded to use the dataset partitioning in TR, VL e TS provided by the organizers.
However, while the label set was ignored because it was judged useless for our purposes, the argument and key point sets were manipulated as follows before being fed to the neural model.
For each set (TR, VL and TS), every argument under a given topic and stance has been coupled with every key point under the same topic and stance. For all these <argument, key point> pairs, arguments and key points were used, respectively, as input and target for the model so that it could be fine-tuned in a supervised manner.

8.3 Model fine-tuning

The only pre-processing procedure applied on TR, VL and TS set was the tokenization performed through mT5 tokenizer. We configured the tokenizer by setting the maximum lengths of the input (argument) and target (ground-truth key point) respectively at 512 and 60, using the truncation technique when necessary.

Next, the data were passed to our chosen model: Google mT5, in the `mt5-small` version. Both fine-tuning, evaluation and prediction phase are carried out with the Hugging Face Trainer API [6]. The model fed with the TR set was fine-tuned for 3 training epochs, with learning rate and batch size fixed to 1e-5 and 16; weight decay and learning rate scheduler with warm up have been also used. More details on the exact hyper-parameters values employed are reported in Appendix A.1, Table 8.
While fine-tuning was running, the model has been evaluated on the VL set to monitor its performance during the learning process; to evaluate the generated summaries a post-processing function that splits the generated summaries into separated sentences was implemented so that their quality can be measured by rouge score. The choice of this metric, as well as the fine-tuning procedures to be adopted, were suggested by HuggingFace's guide for summarization task[7].
After this step was completed, the fine-tuned model was used to predict a set of new key points (or summaries) based on the arguments in TS set.

[6] Hugging Face Trainer API:
https://huggingface.co/docs/transformers/main_classes/trainer

[7] HuggingFace's guide for summarization:
https://huggingface.co/course/chapter7/5?fw=ptsummarization

8.4 Key points ranking

As already explained in the above section, our model generates one key point for each possible <argument, key point> pair under the same topic and stance; hence, we ended up with a very large number of key points where many of them are frequently repeated.
Thus, performing a selection on the generated key points was proved to be necessary to obtain a maximum of 5 distinct key points for each topic and stance. To do this, we have devised a system for scoring and ranking key points. First of all, we have considered all key points generated under the same topic and stance and for each one we have computed the rouge-1 score with their corresponding argument; for identical key points we have taken into account the maximum of their scores. Lastly, we have ranked all key points based on the assigned score and the 5 top ones were selected as final key points for those topics and stances. This procedure had to be repeated for every topic and stance.

8.5 Results and considerations

Several experiments led us to fix the hyper-parameters at certain values that enabled the model to achieve the performance shown in Table 7. Overall, since the hyper-parameter space was not exhaustively examined, the model resulting from our analysis may achieve sub-optimal performance.

	Loss	**Rouge-1**	**Rouge-2**	**Rouge-L**	**Rouge-LS**	**Length (avg)**
TR	3.129	-	-	-	-	-
VL	4.270	7.637	1.060	6.943	6.946	12.260
TS	3.756	12.5816	3.031	12.348	12.345	19.863

Table 7: Performance of Google mT5 on KPG task (in abstractive summarization context). The values reported here come out from a single run of the model over TR, VL and TS set (i.e., they do not represent the average value of multiple runs).

The statistics reported in Table 7 evaluate the abstractive summarization process embedded in our KPG system but are not informative about the quality of the final key points.
Moreover, the study of the quality of the results carried out according to different aspects would have required the simultaneous use of different evaluation metrics. This complexity, also mentioned in *Overview of the 2021 Key Point Analysis Shared Task* [Friedman et al., 2021], led us to perform a qualitative analysis of the obtained key points only in a manual manner, without the support of specific metrics and without referring to the evaluation method adopted in the Shared Task.

Table 11 in Appendix B.1 shows a visual comparison between 10 generated key points and 10 ground truth key points (5 per stance) concerning the same topic of *child vaccination*.
We have observed that key points with positive stance match quite well the

original one, despite the fact that most of them came in the unexpected form "Legalizing ...". At the same time, we have noted that the model struggles more in reporting good key points with negative stances: in fact, in the given example, only 2 out of 5 generated key points appear to be against the given topic, 2 are not relate to the topic, and one is poorly constructed and erroneously pro-topic.
This kind of behavior, which shows greater weaknesses in generating and appropriately selecting key points with negative stance, occurred for all topics in TS set. Studying this fact more deeply, we have concluded that while some smears in sentence construction are due to possible deficits in the mT5 model (also due to sub-optimal fine-tuning), a large part of the sub-optimal choice of final key points is caused by inefficient ranking method. In fact, we can state that ranking the key points based on the rouge-1 score with its corresponding argument is not a proficient method to evaluate their quality and informativeness.

9 Conclusion and future work

In this work, we have approached the Key Point Analysis problem by providing a two-component system to deal with Key Point Matching and Key Point Generation sub-tasks.
Addressing these subtasks starting from the main findings of KPA-2021 allowed us to leverage solid guidelines for setting up our work, to better understand the model skills required to handle a KPA task, and to more easily identify weaknesses in our model through comparison with other ones.

Regarding the KPM track, where the goal was to match key points against appropriate arguments, our primary outcomes emphasize how DeBERTa has overtaken others PLMs. Even if fully understanding model's capabilities remain an active area of research, we can claim that DeBERTa's capabilities of extracting contextual representation played a crucial role in enhancing the quality of the obtained results.
From this point of view, we can assert that our work has verified and confirmed the work of Enigma, who was the first to notice the effectiveness of DeBERTa within this downstream task. Although most of the Shared Task experiments agree in seeing RoBERTa as the most suitable model for solving the KPM problem, we suppose that future developments may benefit even more from the use of DeBERTa; interesting results could also arise by exploiting the large version of DeBERTa.

While we were looking for our best architecture, we have deduced that some future work should focus on the development of a more sophisticated architecture in which to include PLMs. At the same time we think that a supplementary effort in refining the fine-tuning technique could lead to an even better model.

Furthermore, we concluded that most future developments should focus on the KPG sub-task, the more challenging among the proposed ones. Although

our attempt in this context involved the simple implementation of a baseline, this first approach to this sequence-to-sequence task was sufficient to let us recognize some relevant aspects to be considered for future developments. While transformer-based models are robust enough to capture higher linguistic knowledge sufficient to generate a good quality summary for each arguments, an abstractive summarization approach does not seem robust enough to successfully handle the entire KPG task.
In fact, we found out that models like mT5 need to be supported by a system for selecting top key points; this support system should prefer key points whose semantic content overlaps with that of multiple arguments (i.e., prefer key points that match with a reasonable number of arguments).

This last observation was determinant in helping us understand how the two tasks of KPM and KPG are actually closely related. In fact, a robust and well validated system for KPM could be used with automatically generated key points to identify those ones that, due to their optimal granularity and informativeness levels, can be matched with multiple arguments. Sorting the generated key points with this information would lead to a higher quality selection of the final key points.

In the end, working on Key Point Analysis gave us the opportunity to further explore and understand the potential of multiple state-of-art models by embracing a critical method that could smartly drive our choices and speed up our efforts in training these neural structures.

References

[Alshomary et al., 2021] Alshomary, M., Gurke, T., Syed, S., Heinisch, P., Spliethöver, M., Cimiano, P., Potthast, M., and Wachsmuth, H. (2021). Key point analysis via contrastive learning and extractive argument summarization. *CoRR*, abs/2109.15086.

[Bar-Haim et al., 2020a] Bar-Haim, R., Eden, L., Friedman, R., Kantor, Y., Lahav, D., and Slonim, N. (2020a). From arguments to key points: Towards automatic argument summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4029–4039, Online. Association for Computational Linguistics.

[Bar-Haim et al., 2020b] Bar-Haim, R., Kantor, Y., Eden, L., Friedman, R., Lahav, D., and Slonim, N. (2020b). Quantitative argument summarization and beyond: Cross-domain key point analysis. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 39–49, Online. Association for Computational Linguistics.

[Cosenza, 2021] Cosenza, E. (2021). Key point matching with transformers. In *Proceedings of the 8th Workshop on Argument Mining*, pages 190–199, Punta Cana, Dominican Republic. Association for Computational Linguistics.

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[Friedman et al., 2021] Friedman, R., Dankin, L., Hou, Y., Aharonov, R., Katz, Y., and Slonim, N. (2021). Overview of the 2021 key point analysis shared task. *CoRR*, abs/2110.10577.

[Gretz et al., 2020] Gretz, S., Friedman, R., Cohen-Karlik, E., Toledo, A., Lahav, D., Aharonov, R., and Slonim, N. (2020). A large-scale dataset for argument quality ranking: Construction and analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7805–7813.

[He et al., 2020] He, P., Liu, X., Gao, J., and Chen, W. (2020). Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654.

[Kapadnis et al., 2021] Kapadnis, M. N., Patnaik, S., Panigrahi, S. S., Madhavan, V., and Nandy, A. (2021). Team enigma at argmining-emnlp 2021: Leveraging pre-trained language models for key point matching. *CoRR*, abs/2110.12370.

[Lan et al., 2019] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

[Phan et al., 2020] Phan, H. V., Nguyen, L. T., and Doan, K. (2020). Matching the statements: A simple and accurate model for key point analysis. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online. Association for Computational Linguistics.

[Reimer et al., 2021] Reimer, J. H., Luu, T. K. H., Henze, M., and Ajjour, Y. (2021). Modern talking in key point analysis: Key point matching using pretrained encoders. In *Proceedings of the 8th Workshop on Argument Mining*, pages 175–183, Punta Cana, Dominican Republic. Association for Computational Linguistics.

[Xue et al., 2020] Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2020). mt5: A massively multilingual pre-trained text-to-text transformer. *CoRR*, abs/2010.11934.

[Yang et al., 2019] Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

[Zhang et al., 2019] Zhang, J., Zhao, Y., Saleh, M., and Liu, P. J. (2019). PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. *CoRR*, abs/1912.08777.

Appendix A

A.1 Key Point Generation (KPG): Supplementar information on mT5 fine-tuning procedure.

Hyper-parameters	**Values**
Num. epochs	3
Batch size	16
Weight Decay	0.01
Learning Rate	1e-5
LR Linear Scheduler - Warm-up steps	500

Table 8: Hyper-parameters whose values have been costumized for fine-tuning Google mT5.

A.2 Key Point Matching (KPM): Supplementar information on models fine-tuning procedure.

Fixed Hyper-parameters	**Default value**
Weight Decay	0.01
Adam ϵ	1e-8
Adam β	(0.9, 0.999)
Gradient Clipping	1.0
LR Linear Scheduler - Start factor	0.333
LR Linear Scheduler - End factor	1.0
LR Linear Scheduler - Tot. iters	5

Table 9: Hyper-parameters values fixed in the preliminary screening phase. These values were shared across all the explored pre-trained language models.

BERT Hyper-parameter	Grid Search values	Best value
Learning rate (Adam η)	1e-05, 2e-05, 3e-05	2e-05
Batch siz	16, 32	32
Num. epochs	3	3
Hidden Layer (num. neurons)	10, 50, 100	100
Hidden Layer (drop-out rate)	0.2, 0.3, 0.4	0.3

(a) Hyper-parameters values for fine-tuning $\text{BERT}_{\text{BASE}}$.

ALBERT Hyper-parameter	Grid Search values	Best value
Learning rate (Adam η)	1e-05, 2e-05, 3e-05	2e-05
Batch size	16, 32	32
Num. epochs	3	3
Hidden Layer (num. neurons)	10, 50, 100	100
Hidden Layer (drop-out rate)	0.1, 0.2, 0.3	0.2

(b) Hyper-parameters values for fine-tuning $\text{ALBERT}_{\text{BASE}}$.

RoBERTa Hyper-parameter	Grid Search values	Best value
Learning rate (Adam η)	1e-05, 2e-05, 3e-05	2e-05
Batch size	16, 32	32
Num. epochs	3	3
Hidden Layer (num. neurons)	10, 50, 100	50
Hidden Layer (drop-out rate)	0.2, 0.3, 0.4	0.3

(c) Hyper-parameters values for fine-tuning $\text{RoBERTa}_{\text{BASE}}$.

DeBERTa hyper-parameters	Grid Search values	Best value
Learning rate (Adam η)	1e-05, 2e-05, 3e-05	1e-05
Batch size	16, 32	32
Num. epochs	2	2
Hidden Layer (num. neurons)	10, 50, 100	10
Hidden Layer (drop-out rate)	0.2, 0.3, 0.4	0.2

(d) Hyper-parameters values for fine-tuning $\text{DeBERTa}_{\text{BASE}}$.

Table 10: Overview of hyper-parameters values explored via grid search for each fine-tuned language model.

Appendix B

B.1 Key Point Generation (KPG): Comparison between ground truth key points and generated key points.

Topic Routine child vaccinations should be mandatory	
Generated key-point (stance +1)	**Original key-points** (stance +1)
• The childhood vaccine is essential for the child • Private child vaccination is essential to prevent diseases • Legalizing vaccination is essential • Legalizing vaccine prevents diseases • Legalizing vaccine prevents diseases • Legalizing vaccination is important	• Routine child vaccinations are effective • Child vaccination saves lives • Routine child vaccinations are necessary to protect others • Routine child vaccinations should be mandatory to prevent virus/disease spreading • Children should not suffer from preventable diseases
Generated key-point (stance -1)	**Original key-points** (stance -1)
• Legalizing child vaccinations are expensive • Women should be able to choose their child • Legalizing vaccines are harmful • Women should be able to be able to adopt their child • Legalizing child vaccination	• Routine child vaccinations, or their side effects, are dangerous • Mandatory vaccination contradicts basic rights • The parents and not the state should decide • Routine child vaccinations are not necessary to keep children healthy

Table 11: Comparison between key points generated during testing phase and ground truth key points for a given topic and stance.

B.2 Key Point Matching (KPM): Comparison between our final DeBERTa model and others participant models to the Shared Task.

Model	Rank	mAP(S)	mAP(R)	Average mAP
[Alshomary et al., 2021] SMatchToPR	1	0.789(1)	0.927(4)	0.858(1)
[Lan et al., 2019] NLP@UIT	2	0.746(3)	0.930(3)	0.838(2)
[Reimer et al., 2021] ModernTalk	3	0.754(2)	0.902(6)	0.828(4)
[Kapadnis et al., 2021] Enigma	4	0.739(5)	0.928(4)	0.833(3)
[Phan et al., 2020] MatchTStm	5	0.745(4)	0.902(6)	0.824(5)
...	...	...	...	...
[Cosenza, 2021] Poledro	8	0.717(6)	0.901(8)	0.809(8)
DeBERTa (*)		0.704	0.886	0.795
...	...	...	...	...
RoBERTa (*)		0.682	0.886	0.777
[Yang et al., 2019] XLNet	10	0.6848(8)	0.869(13)	0.777(10)
[Liu et al., 2019] RoBERTa-LC	11	0.661(13)	0.885(10)	0.773(11)
...	...	...	...	...
[Bar-Haim et al., 2020b] BarH	13	0.674(11)	0.865(14)	0.77(13)
ALBERT (*)		0.688	0.829	0.759
...	...	...	...	...
BERT (*)		0.633	0.795	0.714
[Devlin et al., 2018] SiamBERT	16	0.555(16)	0.729(16)	0.642(16)
...	...	...	...	...
...	18	...	...	...

Table 12: Comparison between our models and KPA-2021 participant model based on the performance achived on TS set. All our models are marked with (*) and coloured in green, the darker row highlights our DeBERTa best model.