Welcome to the Real World: Efficient, Incremental and Scalable Key Point Analysis

Lilach Eden, Yoav Kantor*, Matan Orbach, Yoav Katz, Noam Slonim, Roy Bar-Haim IBM Research

{lilache, yoavka, matano, katz, noams, roybar}@il.ibm.com

Abstract

Key Point Analysis (KPA) is an emerging summarization framework, which extracts the main points from a collection of opinions, and quantifies their prevalence. It has been successfully applied to diverse types of data, including arguments, user reviews and survey responses. Despite the growing academic interest in KPA, little attention has been given to the practical challenges of implementing a KPA system in production. This work presents a deployed KPA system, which regularly serves multiple teams in our organization. We discuss the main challenges we faced while building a real-world KPA system, as well as the architecture and algorithmic improvements we developed to address these challenges. Specifically, we focus on efficient matching of sentences to key points, incremental processing, scalability and resiliency. The value of our contributions is demonstrated in an extensive set of experiments, over five existing and novel datasets. Finally, we describe several use cases of the deployed system, which illustrate its practical value.

1 Introduction

Getting the gist of a large collection of opinions, such as user reviews and open-ended survey responses, typically requires significant manual work. While word clouds (Heimerl et al., 2014) and key phrases (Hasan and Ng, 2014; Merrouni et al., 2019) are somewhat helpful in providing a highlevel view of the data, they are often too crude to fully replace manual analysis. Plain-text summaries, on the other hand (Chu and Liu, 2019; Bražinskas et al., 2020a,b; Angelidis et al., 2021; Louis and Maynez, 2022), lack a quantitative dimension, as they do not measure the prevalence of each point in the summary. Applying generative AI to summarize large datasets may raise additional



Figure 1: KPA results for the Austin Community Survey. For each KP, the percentage of responses that are matched to it is indicated. Examples for matched sentences are shown for the KP *Austin needs better public transportation*.

issues, such as faithfulness (Maynez et al., 2020) and scalability.

Key Point Analysis (KPA) has been recently proposed as a compelling alternative to the above approaches (Bar-Haim et al., 2020a,b). KPA maps the input texts to a set of automatically-extracted short sentences and phrases, termed *Key Points (KPs)*, which provide a concise plain-text summary of the data. The prevalence of each KP may be quantified as the number of its matching sentences. Figure 1 shows an example of KPA results summarizing a few thousands of responses to a community survey conducted in the City of Austin.

KPA has gained significant academic interest, with 17 teams participating in the 2021 KPA shared task (Friedman et al., 2021). However, little attention has been given in previous work to practical aspects of building a real-world KPA system. In this work, we make a step towards closing this gap, by describing a deployed KPA system that is being regularly used by multiple teams in our organization (IBM). Specifically, we focus on the following issues, which we found to be the most critical when deploying KPA in production:

Efficient Matching. Most of the run time in KPA is spent on matching sentences to KPs or KP can-

^{*}First two authors equally contributed to this work.

didates. We propose a method that combines slow, accurate matching with fast, less accurate matching. This allows run time reduction by a factor of five, while achieving comparable accuracy (§4).

Incremental KPA. In many practical scenarios, KPA needs to be performed periodically, over data that is being accumulated over time. We introduce the notion of *Incremental Key Point Analysis*, and propose a modification to the KPA algorithm that enables efficient incremental processing (§5).

Scalability and resiliency. While academic KPA datasets (Bar-Haim et al., 2020a; Friedman et al., 2021) include a few hundreds of input texts per topic, real-world KPA systems should accommodate much larger datasets – up to hundreds of thousands of comments. Scaling up the system should be straightforward by adding more resources. In addition, the system should serve multiple users in a responsive fashion, and should be able to overcome failures, especially when processing large jobs. Finally, as GPUs are expensive, it is important to utilize them efficiently. We develop an architecture that addresses all of these requirements (§6).

The above contributions are assessed in an extensive set of experiments that measure run time, accuracy, and the trade-off between the two. We perform the most comprehensive evaluation of a KPA system to date, based on a diverse set of five benchmarks, including both internal and publiclyavailable datasets.

Finally, we discuss several use cases of KPA at IBM, which illustrate its practical value for a variety of tasks and datasets (§7).

2 KPA Algorithm Overview

Our system is based on the KPA algorithm of Bar-Haim et al. (2020b), which was the best end-to-end performer on the 2021 shared task (Friedman et al., 2021). The input for the algorithm is a collection of *comments*, split into *sentences*. The algorithm comprises the following steps:

- 1. Select *KP candidates*, which are short, highquality input sentences.
- 2. Match the rest of the sentences to the KP candidates, while merging semantically similar candidates.
- 3. Rank the candidates by the number of their matches, and select the top *k* candidates as the final KPs.

4. Return the selected KPs along with the matching sentences for each KP.

The algorithm employs two supervised Transformer models: an *argument quality model* (Gretz et al., 2020) for the first step, and a *matching model* (Bar-Haim et al., 2020a,b) for the second step (for both matching the sentences and identifying semantically-similar candidates¹).

Following Bar-Haim et al. (2021), we incorporated two additional models into the KPA system: First, a *stance (sentiment)* classification model, which labels the input sentences as *positive, negative* or *neutral*.² When this optional step is performed, we subsequently exclude the neutral sentences and run KPA separately on the positive and negative sentences. This improves both the run time and the matching accuracy. Second, we developed an additional *KP quality* supervised model, specifically designed to select candidates with desirable properties (have a clear stance, discuss a single topic, not too general/specific). This classifier is used in conjunction with the argument quality classifier.

The bottleneck of the KPA algorithm is the matching step, in which a Transformer-based matching model is applied to compute the match score for each (sentence, KP candidate) pair, resulting in quadratic complexity. Other models (stance, quality) are only applied once per sentence, and therefore have marginal effect on the overall runtime. Bar-Haim et al. (2021) proposed the following modifications to reduce the matching run time for large datasets: (a) limit the number of KP candidates, and (b) select the KPs based on a subset of sentences; then match the rest of the sentences only to the selected KPs. Even with these improvements, matching run time remains a major issue when deploying a KPA system that should serve many users, and process large datasets.

Thus, a core challenge in developing a realworld KPA system is improving matching efficiency, without degrading its quality. In the following sections we describe the setup of our experiments, and specifically the data (§3) and the various matching approaches we experimented with (§4.1), followed by experimental results (§4.2). Then, we describe additional solutions we implement for real-

¹The latter is performed by applying the match score in both directions and taking the average.

²Our classifier treats suggestions, which commonly occur in surveys, as negative feedback.

world KPA: addressing incremental updates (§5), and a scalable system architecture (§6).

3 Data

In this section we briefly describe the datasets that were used to train the matching model, and to assess its quality.

Out of our three training sets and five test sets, two (ArgKP and ArgKP-21) are existing public datasets. The remaining six datasets were created as part of this work, utilizing diverse public and internal data sources. Each of the datasets includes correct (positive) and incorrect (negative) examples of mapping sentences to KPs. The label of each instance was obtained by consolidating multiple human annotations. Some statistics on the train and test sets are given in Table 1.³

Train and Development sets. The matching model was trained on the following datasets, combined:

- ARGKP: the ARGKP dataset (Bar-Haim et al., 2020a,b) consists of pro and con arguments for 28 controversial topics⁴ that were mapped to KPs composed by a professional debater. The annotators selected the matching KPs for each argument. The train and dev sets comprise 24 and 4 topics, respectively. An initial system was trained on the train set, and its output was utilized for constructing the rest of the datasets, as described below. The dev set was used for tuning the matching thresholds, as described in Section 4.
- EMPLOYEE: an internal employee feedback dataset. The dataset consists of sampled sentences and a manually-revised version of the top KPs extracted by the system. Similar to ARGKP, the annotators selected the matching KPs for each sampled sentence.
- MUNICIPAL: This dataset was generated by running the system over the open-ended responses to the 2018 Austin Community Survey⁵. To ensure the inclusion of difficult examples, we annotated (sentence, KP candidate) pairs on which an ensemble of models disagreed.

| | Dataset | #Pairs | |
|-------|-------------|--------|----------|
| | | Total | Positive |
| | ARGKP | 20,635 | 4,260 |
| Train | Employee | 1,454 | 291 |
| | MUNICIPAL | 2,861 | 1,001 |
| Dev | ARGKP | 3,458 | 738 |
| | ARGKP-21 | 3,923 | 552 |
| | ARGKP-LARGE | 9,281 | 928 |
| Test | Employee | 4,990 | 154 |
| | MUNICIPAL | 15,189 | 356 |
| | PRODUCT | 3,738 | 379 |

Table 1: Statistics on the train, development and test datasets.

Test sets. The matching model was evaluated on the following benchmarks:

- ARGKP-21: the test set from the 2021 KPA shared task (Friedman et al., 2021). This dataset was constructed following the same methodology as ARGKP, and includes three topics.
- ARGKP-LARGE: this benchmark maps arguments for ten topics from the Gretz et al. dataset that are not in ARGKP to KPs automatically extracted by the system.
- EMPLOYEE: constructed similarly to the train EMPLOYEE dataset, with data from a different year.
- PRODUCT: an internal product feedback benchmark, generated from responses to Net Promoter Score (NPS) surveys. It is composed of sampled sentences and manually-revised versions of the top KPs extracted by the system.
- MUNICIPAL: generated from the Austin Municipal Survey of 2016-2017, similarly to the PROD-UCT benchmark.

We release two novel KPA benchamrks, ARGKP-LARGE and MUNICIPAL, along with this paper.⁶

4 Efficient Matching

4.1 Models

The design of a matching model is critical for both the quality and the run time of a KPA system. In this section we propose several alternatives for implementing such a model, and assess their tradeoffs empirically. Each of the models described below was fine-tuned on our training set. The matching thresholds for the DeBERTa and SBERT models were tuned over the development set.

³The negative examples in the EMPLOYEE and MUNICI-PAL train sets were downsampled, to obtain a more balanced training set.

⁴A subset of the *IBMArgQ-Rank-30kArgs* dataset (Gretz et al., 2020)

⁵https://data.austintexas.gov/dataset/ Community-Survey/s2py-ceb7

⁶https://research.ibm.com/haifa/dept/vst/ debating_data.shtml

Cross-encoder (DeBERTa). This is our baseline model. The original KPA algorithm (Bar-Haim et al., 2020b) implemented the matching model as a RoBERTa-large cross-encoder, which receives as an input a concatenation of the sentence and the KP/KP candidate. In our experiment we used *DeBERTa-v3-large*⁷ instead, as we found that it provides better results, with a comparable run time. Cross-encoders are accurate, as they can model complex interaction between the texts, but slow, since inference is required for each pair.

Bi-encoder (SBERT). a much faster alternative to cross-encoders is a bi-encoder such as SBERT (Reimers and Gurevych, 2019), a pretrained Transformer model that was fine tuned using a Siamese network architecture to derive semantically meaningful sentence embeddings. The semantic similarity between two texts can be computed as the cosine-similarity between their embeddings. This reduces model inference complexity from quadratic to linear, as each sentence is only encoded once. However, this comes at the expense of a more simplistic modelling of the interaction between the texts. Furthermore, this model is symmetric, while the relation between a sentence and a key point is directional (the KP should summarize the sentence). Therefore, we might expect bi-encoders to be less accurate than cross-encoders. Here, we use the all-mpnet-base-v2 model, which is a pretrained MPNet model (Song et al., 2020) that was fine-tuned over 1.2B sentence pairs⁸.

Combined. To get the best of both worlds, we propose to combine the two methods and use the fast bi-encoder to filter the inputs fed into the slow cross-encoder. First, the matching scores of the SBERT model are computed. Then, only the top matching KPs for each sentence are scored by the slow DeBERTa matcher, while the rest are assigned a zero score. We selected the top 10% key points for each sentence⁹, but no less than two. A similar "retrieve and rerank" approach¹⁰ that applies a pipeline of bi- and cross-encoders has been proposed for zero-shot entity linking (Wu et al., 2020).

⁷https://huggingface.co/microsoft/ deberta-v3-large **Generative LLM (Flan-T5-XL).** Following the recent success of generative models, we also experimented with Flan-T5-XL (Chung et al., 2022; Longpre et al., 2023) for matching sentences to KPs. This model was tuned for the matching task with QLora (Dettmers et al., 2023), for one epoch, since performance on the development set has not improved beyond that point. The prompt used in this experiment is described in the Appendix.

4.2 Experiments

We first compare the quality of the different models (measured as a micro-F1 score over the pairs in each test set), as well as the mean number of pairwise inferences performed per sentence (Table 2).

While the fast SBERT model alone performs poorly, the Combined model results are comparable to or better than the baseline DeBERTa results for 4 of the 5 benchmarks (except for PRODUCT where it's slightly lower), with far fewer pairwise inferences per sentence.

The fine-tuned Flan-T5-XL model performance is comparable to the DeBERTa model on four out of the five test sets (except PRODUCT) while being nearly 15 times slower¹¹, so overall we did not find it beneficial for our system.

Having established the quality of the Combined model, we next test its impact on the run time of the full KPA system¹². Table 3 presents the end-toend run time of KPA over subsets of different sizes from an internal large-scale employee survey, for both the DeBERTa and the Combined models. The Combined model becomes more beneficial as the input size increases, approaching a five-fold run time reduction for 100,000 comments.

Based on the above experimental results, we selected the Combined matching model for our deployed KPA system, and it is used in the rest of the experiments, to be described in the next sections.

5 Incremental KPA

Previous work applied KPA only to static datasets. In real-world scenarios, however, data is often accumulated over time, and it is required to rerun KPA periodically, over all the data collected so far. Suppose that we have already run KPA over customer feedback collected in January, and now we obtain

⁸https://huggingface.co/sentence-transformers/ all-mpnet-base-v2

⁹out of the key points with the same topic and stance.

¹⁰https://www.sbert.net/examples/applications/ retrieve_rerank/README.html

¹¹Measured on a single A100 GPU.

¹²Run time experiments were conducted over ten K80 GPUs, in a dedicated environment.

| Benchmark | Model | F1 | #Pairwise inferences/ sentence |
|------------|------------|------|--------------------------------------|
| | SBERT | 0.55 | 0 |
| ARGKP- | DeBERTa | 0.65 | 6.62 |
| LARGE | Combined | 0.72 | 2 |
| | Flan-T5-XL | 0.67 | 6.62 |
| | SBERT | 0.54 | 0 |
| ARGKP-21 | DeBERTa | 0.70 | 5.43 |
| AKGKF-21 | Combined | 0.75 | 2 |
| | Flan-T5-XL | 0.73 | 5.43 |
| | SBERT | 0.32 | 0 |
| EMPLOYEE | DeBERTa | 0.48 | 26.68 |
| EMPLOYEE | Combined | 0.49 | 3.49 |
| | Flan-T5-XL | 0.47 | 26.68 |
| | SBERT | 0.33 | 0 |
| PRODUCT | DeBERTa | 0.62 | 9.92 |
| FRODUCI | Combined | 0.59 | 2 |
| | Flan-T5-XL | 0.72 | 9.92 |
| | SBERT | 0.47 | 0 |
| MUNICIPAL | DeBERTa | 0.61 | 38.36 |
| WIUNICIPAL | Combined | 0.60 | 3.8 |
| | Flan-T5-XL | 0.57 | 38.36 |

Table 2: Micro F1 score and mean number of pairwise inferences performed per sentence for each matching model and test set.

| #Comments | Run time DeBERTa | Run time Combined | Ratio |
|-----------|---------------------|----------------------|-------|
| 1,000 | 7.8 | 3.7 | 0.47 |
| 5,000 | 44.2 | 11.2 | 0.25 |
| 10,000 | 91.4 | 21.5 | 0.24 |
| 50,000 | 515.7 | 112.8 | 0.22 |
| 100,000 | 1308.5 | 270.1 | 0.21 |

Table 3: Run time (mins.) of the full KPA system over subsets of different sizes from an internal large-scale employee survey.

additional data from February. We may want to run KPA over the whole period (January+February), or compare the differences between the January and February KPA results. This scenario raises several issues. First, rerunning KPA from scratch on the entire data is inefficient, since it does not leverage computations from previous runs. This can be partially addressed by *caching* of match scores for previously-inferred pairs. However, running KPA over the unified dataset may surface KP candidates (and consequently, KPs) that are paraphrases or near-paraphrases of the KPs found for January. Pairs that include these KPs/candidates may not be in the cache, increasing the computation time. Moreover, it would be difficult to align the January+February KPA results with the January results, as they may contain semantically similar but different key points (the same problem would occur when running only on the February data).

To allow consistent and efficient incremental analysis, we modified the KPA algorithm to reuse KPs from previous runs and incrementally add new ones. In our example, we run KPA over the January+February data while using the final key points from January as key point candidates. The system would only add new candidates that are sufficiently different from the previous ones (according to the matching threshold). New KPs in the summary would represent emerging points that are over-represented in the new data. Since KPA output includes the mapping of each sentence to its corresponding KPs, KPA results can be derived for any subset of the sentences. Thus, given the results for January+February, we can easily extract the results for either January or February. Moreover, since both subsets are mapped to the same set of key points, we can also compare the distribution of the key points in the two sets, and even apply statistical significance tests to the differences in their relative frequencies.

KP reuse also leads to substantial run time saving: since the matching scores for the old data are cached, old sentences need to be matched only with the new candidates, whose number is significantly reduced. Moreover, key point selection is faster, since the list of candidates from the old data is already reduced to the final list of key points.

Experimental Results. To test the impact of caching and KPA reuse on run time, we ran KPA incrementally on 50,000 comments from the internal employee engagement survey, adding 10,000 comments at each stage.

The results are summarized in Table 4, demonstrating the contribution of both caching and KP reuse. While in the baseline system, processing 50K comments is about 5 times slower than the first 10K batch, this ratio is reduced to 1.4 in our improved incremental implementation.

To ensure to quality of the incremental results, we compared the top 50 key points generated with and without KP reuse over the entire dataset. While only 5 of the key points overlapped, we found that 80% of the key points output by the full run were covered by the key points of the incremental run, and 92% were at least partially matched, that is, almost all of the insights uncovered by the full run, were also found by the incremental run.

| Comments | No Caching | Caching | Caching +KP Reuse |
|----------|---------------|---------|----------------------|
| 10,000 | 21.5 | 21.7 | 22.8 |
| 20,000 | 44.7 | 33.8 | 17.4 |
| 30,000 | 66.7 | 43.8 | 20.8 |
| 40,000 | 90.4 | 54.9 | 24.3 |
| 50,000 | 112.8 | 58.6 | 29.3 |

Table 4: Run time (mins.) for incremental KPA



Figure 2: KPA System Architecture

6 A Scalable Architecture for KPA

The previous two sections focused on improving the matching component in terms of efficiency and incremental processing. We next turn to describe our overall system architecture, and how it addresses the requirements from a real-world KPA system: scalability, resiliency, efficient GPU utilization, and fair, responsive multi-user service.

The KPA system architecture is depicted in Figure 2. The system consists of the following components: a Python *client* allows the users to submit KPA tasks to the system, and retrieve the results, either synchronously or asynchronously.

The *Backend Service* is the primary service that implements the KPA algorithm. Uploaded comments undergo *preprocessing*, which includes splitting the comments into sentences, and computing sentence-level scores (stance, argument quality and key point quality). Then, an *analysis* task is launched over the preprocessed sentences. The analysis stage executes the core KPA algorithm, including the computation of pairwise matching scores.

To facilitate incremental processing, users can define multiple *domains*, to which comments are uploaded. A domain may contain multiple sets of comments, accumulated over time, and each analysis task is applied to all the comments in the specified domain.

The backend service initiates inference by each of the system's models: stance, argument quality and KP quality for preprocessing; SBERT embedding and DeBERTa pairwise matching for the analysis tasks. The backend service breaks the data to be inferred into batches, and submits batch inference requests via a message queue. To optimize GPU usage and equitably distribute resources between tasks, each task is capped at 20 pending inference requests.

The backend selects the next task for execution based its *size* and *user priority*. Tasks are classified into *small, medium* or *large*, and the system limits the number of active tasks of each type¹³. Users' priority is determined by their current and recent activity, as well as their predefined priority level. In other words, users with a superior predefined level, lower recent activity, and no active tasks are prioritized.

Inference requests are handled in parallel by multiple instances of the *Multi Models Service* - one per each available GPU. Each instance can serve requests for any of the models, swapping models in the GPU memory as required. The number of consecutive inferences of the active model is limited, to ensure that none of the other models is starved.

The processed data, including intermediate and final results, is stored persistently in a *database*. Specifically, we use MongoDB as a cloud service. This adaptable, secure, and robust service allows swift resource expansion - memory, CPU, and network - as needed. In addition, a watchdog mechanism monitors comment-batch processing and analysis task execution. If a task stalls beyond a predefined duration, it is restarted.

The above architecture has several important benefits. Breaking tasks into batches and executing them parallelly enables fast processing of large tasks, as well as serving multiple users. The backend task scheduling mechanism facilitates responsiveness and fairness among users. The multimodels services ensure that the GPUs are fully utilized, by dynamically adapting the mixture of models being served to the incoming inference requests. The system's throughput can be easily scaled up linearly by adding more GPUs. The database provides both a caching mechanism, saving redundant computations, and failure recovery by recording the system's state. Using a message queue further enhances the system's scalability and resilience. It decouples the producers and consumers of inference requests, ensures asynchronous communication, buffers during high loads or failures, and

¹³Up to 1 large, 2 medium and 10 small tasks.



Figure 3: KPA run time improvement when adding GPUs (10,000 comments).

mitigates the impact of slower multi-models services. Finally, the watchdog mechanism ensures that tasks are progressing and not getting stuck.

Scalability Evaluation. We tested empirically the impact of the number of GPUs on the run time for analyzing 10,000 comments (split into 16,220 sentences). The results, shown in Figure 3, demonstrate that the expected inverse proportional relation between the number of GPUs and the run time does hold in practice in our implemented system.

7 Use Cases

The deployed KPA system serves multiple teams across IBM. Below we describe several use cases, in which KPA has been applied to extract finegrained insights from large textual datasets, saving the time and cost of manual analysis.

Feedback on internal applications. IBM uses a common tool to collect internal user feedback and verbatim comments. Each product owner or application team has to dedicate time and efforts to manually read, categorize user comments and feedback, summarize and identify actions to be taken to address them. The My Cognitive Adviser (MCA) tool was created as a common solution to be used by owners of internal applications. Using KPA services, it analyzes and displays applicationspecific feedback summarized by key points, sentiment, associated sentence count and trends over time. Automated analysis is provided monthly or quarterly, depending on the volume of feedback. It allows users to understand the top pain points for their applications, without having to manually review, categorize and label each user feedback.

Productivity improvement ideas. IBM Finance and Operations (F&O) launched an organization-wide initiative to improve productivity within the

company, by organizing and executing 900+ workshops resulting in 7500+ ideas from employees (posted in Slack and Mural boards). KPA was used to analyze and summarize these ideas to identify the top 15 key areas for productivity improvements within the company. This analysis gave the F&O executives a very quick synopsis of the workshops, enabling them to hear the voice of their employees, without having to spend weeks manually reviewing and categorizing the ideas.

Employee engagement survey. KPA has also been applied to analyze the annual IBM employee engagement survey. Over 300K employees wrote more than 550K sentences in total. These sentences were automatically classified into positive and negative, and KPA was applied to each set separately to identify positive and negative key points. We also compared year-to-year differences in key points distribution, as well as differences between suborganizations, and between detractors and advocators. KPA results enabled the HR survey research team to extract actionable and valuable insights from this very large dataset, with significantly less effort.

8 Discussion and Conclusion

KPA is a promising approach for large-scale quantitative summarization of opinions, with many practical applications. Yet, moving from academic experimentation to a real-world implementation poses significant research and engineering challenges.

We described a deployed KPA system, for which we presented a comprehensive architecture, as well as algorithmic improvements that enable efficient matching and incremental processing. Our contributions have been evaluated in terms of both quality and run time over the most comprehensive set of KPA benchmarks to date, including both internal and public datasets.

It is important to note the compound effect of combining the above individual improvements. For example, as shown in Table 3, the baseline run time for 50K comments is 515.7min. Using our Combined model, this is reduced to 112.8min, and in an incremental setting, it is further reduced to 29.3min, using caching and KP reuse (Table 4). Additional speed-up can be obtained by adding more GPUs, as shown in Figure 3.

In future work, we plan to further explore the use of generative LLMs for KPA, aiming to achieve both hiqh-quality results and feasible run time.

Ethical Considerations

The datasets used in this work include anonymous feedback and do not contain personal details. All internal datasets were provided by the respective data owners in our organization, and were processed on a strict need-to-know basis. Data has been encrypted both in transit and at rest, for increased security and privacy.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. We also thank our collaborators at IBM, Laura Ashley Bridges, Anne Kline, David Koch and Neha Shahani, for their feedback on the various use cases of the deployed KPA system.

References

- Stefanos Angelidis, Reinald Kim Amplayo, Yoshihiko Suhara, Xiaolan Wang, and Mirella Lapata. 2021. Extractive opinion summarization in quantized transformer spaces. *Transactions of the Association for Computational Linguistics*, 9:277–293.
- Roy Bar-Haim, Lilach Eden, Roni Friedman, Yoav Kantor, Dan Lahav, and Noam Slonim. 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.
- Roy Bar-Haim, Lilach Eden, Yoav Kantor, Roni Friedman, and Noam Slonim. 2021. Every bite is an experience: Key Point Analysis of business reviews. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 3376–3386, Online. Association for Computational Linguistics.
- Roy Bar-Haim, Yoav Kantor, Lilach Eden, Roni Friedman, Dan Lahav, and Noam Slonim. 2020b. Quantitative argument summarization and beyond: Crossdomain 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.
- Arthur Bražinskas, Mirella Lapata, and Ivan Titov. 2020a. Few-shot learning for opinion summarization. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 4119–4135, Online. Association for Computational Linguistics.

- Arthur Bražinskas, Mirella Lapata, and Ivan Titov. 2020b. Unsupervised opinion summarization as copycat-review generation. In *Proceedings of the* 58th Annual Meeting of the Association for Computational Linguistics, pages 5151–5169, Online. Association for Computational Linguistics.
- Eric Chu and Peter Liu. 2019. MeanSum: A neural model for unsupervised multi-document abstractive summarization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1223–1232. PMLR.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.
- Roni Friedman, Lena Dankin, Yufang Hou, Ranit Aharonov, Yoav Katz, and Noam Slonim. 2021.
 Overview of the 2021 key point analysis shared task. In *Proceedings of the 8th Workshop on Argument Mining*, pages 154–164, Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Shai Gretz, Roni Friedman, Edo Cohen-Karlik, Assaf Toledo, Dan Lahav, Ranit Aharonov, and Noam Slonim. 2020. A large-scale dataset for argument quality ranking: Construction and analysis. In AAAI.
- Kazi Saidul Hasan and Vincent Ng. 2014. Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1262–1273, Baltimore, Maryland. Association for Computational Linguistics.
- Florian Heimerl, Steffen Lohmann, Simon Lange, and Thomas Ertl. 2014. Word cloud explorer: Text analytics based on word clouds. In 2014 47th Hawaii International Conference on System Sciences, pages 1833–1842.
- Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning.
- Annie Louis and Joshua Maynez. 2022. Opinesum: Entailment-based self-training for abstractive opinion summarization. *ArXiv*, abs/2212.10791.

- Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings* of the 58th Annual Meeting of the Association for Computational Linguistics, pages 1906–1919, Online. Association for Computational Linguistics.
- Zakariae Alami Merrouni, Bouchra Frikh, and Brahim Ouhbi. 2019. Automatic keyphrase extraction: a survey and trends. *Journal of Intelligent Information Systems*, 54:391 – 424.
- Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.
- Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pretraining for language understanding.
- Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable zeroshot entity linking with dense entity retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6397–6407, Online. Association for Computational Linguistics.

9 Appendix

A Training Details

DeBERTa. Fine tuning was performed as described in (Bar-Haim et al., 2020b), except using a *deberta-v3-large* model instead of *roberta-large*, and using 6 trainig epochs.

SBERT. The *all-mpnet-base-v2* model was finetuned for 3 epochs with learning rate 5e-6. We used mean pooling over each sentence embedding, cosine similarity over the vectors representing each pair and contrastive loss. The sequence length was limited to 256 tokens, and the train batch size was 32.

Flan-T5-XL. When tuning with QLora, the parameters were set to r = 8, $\alpha = 32$ and dropout 0.05. LoRA update matrices were only applied to the query and value modules within the transformer. Bias parameters were not trained. The learning rate was 1e - 05, and the optimizer was paged Adamw with 32 bits.

B Prompts

The following prompt was considered for the experiment with the *Flan-T5-XL* model (§4):

{{opening_sentence}} A key point matches a sentence if it either: - Summarizes or repeats a part of the sentence or expresses the same main point.

- Is directly supported by a point made in the sentence. The sentence can support the key point by an example, elaboration, discussing an aspect of the point made in the key point, suggesting a solution to a problem mentioned in the key point, etc.

You will be presented with a key point and a sentence and asked to determine if the key point matches the sentence. The options are: - Yes

- Faulty sentence (not a valid sentence or unclear)

Here is the sentence: "{{sentence_text}}" And here is the key point: "{{key_point}}" Does the sentence match the key point?

The placeholder {{opening_sentence}} was replaced with a sentence describing the dataset, for example, "I am going to show you sentences which are replies to a community survey about the city of Austin." with the MUNICIPAL dataset. The {{sentence_text}} and {{key_point}} placeholders were replaced with the sentence and key point texts, respectively.

⁻ No