Katecheo: A Portable and Modular System for Multi-Topic Question Answering

Shirish Hirekodi, Seban Sunny, Leonard Topno, Alwin Daniel, Reuben Skewes, Stuart Cranney
CV Digital
Sunshine Coast, Queensland, Australia
ailab@cvglobal.co

Daniel Whitenack
Data Dan LLC
West Lafayette, IN, USA
dan@datadan.io

Abstract

We introduce a modular system that can be deployed on any Kubernetes cluster for question answering via REST API. This system, called Katecheo, includes four configurable modules that collectively enable identification of questions, classification of those questions into topics, a search of knowledge base articles, and reading comprehension. We demonstrate the system using publicly available, pre-trained models and knowledge base articles extracted from Stack Exchange sites. However, users can extend the system to any number of topics, or domains, without the need to modify any of the model serving code. All components of the system are open source and available under a permissive Apache 2 License.

1 Introduction

When people interact with chatbots, smart speakers or digital assistants (e.g., Siri[1]), one of their primary modes of interaction is information retrieval (Lovato and Piper, 2015). Thus, those that build dialog systems often have to tackle the problem of question answering.

Developers could support question answering using publicly available chatbot platforms, such as Watson Assistant[2] or DialogFlow[3]. To do this, a user would need to program an intent for each anticipated question with various examples of the question and one or more curated responses. This approach has the advantage of generating high quality answers, but it is limited to those questions anticipated by developers. Moreover, the management burden of such a system might be prohibitive as the number of questions that needs to be supported is likely to increase over time.

To overcome the burden of programming intents, developers might look towards more advanced question answering systems that are built using open domain question and answer data (e.g., from Stack Exchange or Wikipedia), reading comprehension models, and knowledge base searches. In particular, Chen et al. previously demonstrated a two step system, called DrQA, that matches an input question to a relevant article from a knowledge base and then uses a recurrent neural network (RNN) based comprehension model to detect an answer within the matched article. This more flexible method was shown to produce promising results for questions related to Wikipedia articles and it performed competitively on the SQuAD benchmark (Rajpurkar et al., 2016).

However, if developers wanted to integrate this sort of reading comprehension based methodology into their applications, how would they currently go about this? They would need to wrap pre-trained models in their own custom code and compile similar knowledge base articles at the very least. At the most, they may need to re-train reading comprehension models on open domain question and answer data (e.g., SQuAD) and/or implement their own knowledge base search algorithms.

In this paper we present Katecheo, a portable and modular system for reading comprehension based question answering that attempts to ease this development burden. The system provides a quickly deployable and easily extendable way for developers to integrate question answering functionality into their applications. Katecheo includes four configurable modules that collectively enable identification of questions, classification of those questions into topics, a search of knowledge base articles, and reading comprehension. The modules are tied together in a single inference graph that can be invoked via a REST API call. We demonstrate the system using publicly available,

[1] https://www.apple.com/siri/
[2] https://www.ibm.com/cloud/watson-assistant/
[3] https://dialogflow.com/

pre-trained models and knowledge base articles extracted from Stack Exchange sites[4]. However, users can extend the system to any number of topics, or domains, without the need to modify the model serving code. All components of the system are open source and publicly available under a permissive Apache 2 License[5].

The rest of the paper is organized as follows. In the next section, we provide an overview of the system logic and its modules. In Section 3, we outline the architecture and configuration of Katecheo, including extending the system to an arbitrary number of topics. In Section 4, we report some results using example pre-trained models and public knowledge base articles. Then in conclusion, we summarize the system, its applicability, and future development work.

2 System Overview

Katecheo is partially inspired by the work of Chen et al. on DrQA. That previously developed method has two primary phases of question answering: document retrieval and reading comprehension. Together these functionalities enable open domain question answering. However, many dialog systems are not completely open domain. For example, developers might want to create a chatbot that has targeted conversations about restaurant reservations and movie times. It would be advantageous for such a chatbot to answer questions about food and entertainment, but the developers might not want to allow the conversation to stray into other topics.

With Katecheo, one of our goals was to create a question answering system that is more flexible than those relying on curated responses while remaining more targeted than a completely open domain question answering system. The system includes document retrieval (or what we refer to as "knowledge base search") and reading comprehension, but only within sets of curated knowledge base articles each corresponding to a particular topic (e.g., food or entertainment).

When a question text is input into the Katecheo system, it is processed through four modules: (1) question identification, (2) topic classification, (3) knowledge base search, and (4) reading comprehension. This overall logic is depicted in Figure 1.

[4]https://stackexchange.com/sites#

[5]https://github.com/cvdigitalai/katecheo

2.1 Question Identification

The first module in Katecheo, question identification, determines if the input text (labeled Q in Figure 1) is actually a question. In our experience, users of dialog systems provide a huge number of unexpected inputs. Some of these unexpected inputs are questions and some are just statements. Before going to the trouble of matching a knowledge base article and generating an answer, Katecheo completes this initial step to ensure that the input is a question. If the input is a question, the question identification module (henceforth the "question identifier") passes a positive indication/flag to the next module indicating that it should continue processing the question. Otherwise, it passes a negative flag to end the processing.

The question identifier uses a rule-based approach to question identification. As suggested in Li et al., we utilize the presence of question marks and 5W1H words to determine if the input is a question. Based on our testing, this provides quite high performance (90%+ accuracy) and is not a blocker to overall performance.

2.2 Topic Classification

To reach our goal of a question answering system that would be more targeted than previous open domain question answering, we decided to allow the user of the system to define one or more topics. The topic classification module of the system (henceforth the "topic classifier") will attempt to classify the input question into one of the topics and then select a knowledge base article from a set of knowledge base articles corresponding to that topic.

One way we could enable this topic classification is by training a text classifier that would classify the input text into one of the user supplied topics. However, this approach would require (i) the user to provide both the topic and many example questions within that topic, and (ii) the system to retrain its classification model any time a new topic was added. We wanted to prioritize the ease of deployment, modularity and extensibility of the system, and, thus, we decided to take a slightly more naive approach.

Along with each topic, the user supplies the system with a pre-trained Named Entity Recognition (NER) model that identifies entities within that topic. The topic classifier then utilizes these pre-

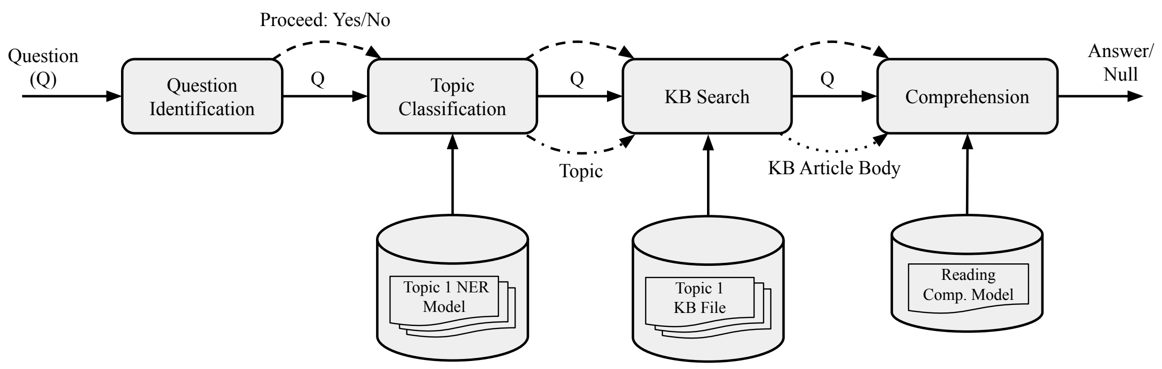

Figure 1: The overall processing flow in Katecheo. *Q* represents the input question text, the dashed lines represent a flag passed between modules indicating whether the next module should proceed with processing, and the cylinders represent various data inputs to the modules.

trained models to determine if the input question includes entities from one of the user supplied topics. If so, the topic classifier classifies the question into that topic. When two of the topics conflict, the system currently suspends processing and returns a null answer.

The system accepts NER models that are compatible with spaCy (Honnibal and Montani, 2017). As discussed further below, the user can supply a link to a zip file that contains each topic NER model.

Note, it might be possible to remove the dependence on NER models in the future. We are currently exploring the use of other topic modeling techniques including non-negative matrix factorization and/or Latent Dirichlet Allocation (LDA). These techniques could enable the system to automatically match the input question to most appropriate topical knowledge base, and thus only rely on the user to supply knowledge base articles.

2.3 Knowledge Base Search

Once the topic has been identified, a search is made to match the question with an appropriate knowledge base article from a set of user supplied knowledge base articles corresponding to the user supplied topic. This matched article will be utilized in the next stage of processing to generate an answer.

The user supplied sets of knowledge base articles for each topic are in a JSON format and include a title and body text for each article. The system assumes that the knowledge base articles are in the form of a question and answer knowledge base (e.g., like a Stack Exchange site), rather than any arbitrarily structured articles. In this way, we are able to utilize the titles of the articles (i.e., the questions) in matching to user input questions.

In the knowledge base search module of Katecheo (henceforth the "KB Search" module), we use the Python package FuzzyWuzzy[6] to perform string matching between the input question and the knowledge base article titles. FuzzyWuzzy uses Levenshtein Distance (Levenshtein, 1966) match the input string to one or more input candidate strings.

We eventually plan to update this knowledge base search to an approach similar to that of Chen et al. using bigram hashing and TF-IDF. However, the fuzzy string matching approach works reasonably well as long as the supplied knowledge bases are of a type where many of the article titles are in the form of topical questions.

2.4 Reading Comprehension

The final module of the Katecheo system is the reading comprehension (or just "comprehension") module. This module takes as input the original input question plus the matched knowledge base article body text and uses a reading comprehension model to select an appropriate answer from within the article.

The current release of Katecheo uses a Bi-Directional Attention Flow, or BiDAF, model for reading comprehension (Seo et al., 2017). This BiDAF model includes a Convolutional Neural Network (CNN) based character level embedding layer, a word embedding layer that uses pre-trained GloVE embeddings, a Long Short-Term

[6] https://github.com/seatgeek/fuzzywuzzy

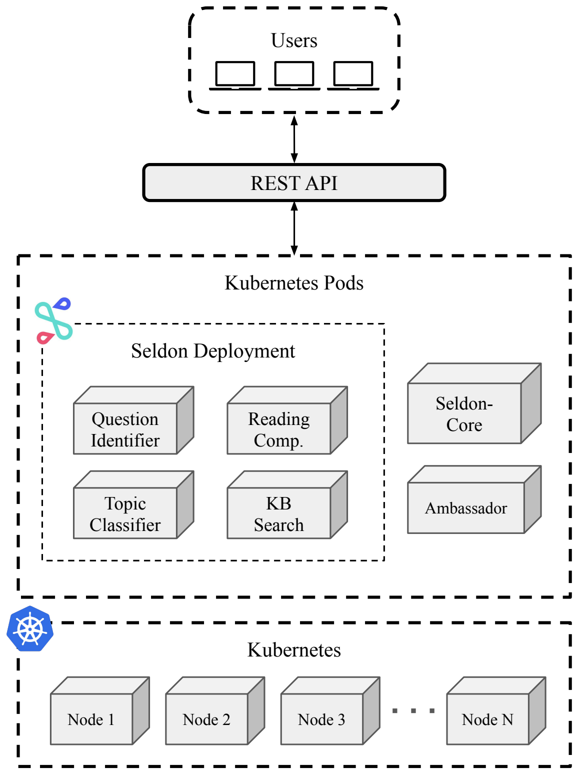

Figure 2: The overall Katecheo architecture. Each *node* in Kubernetes may be a cloud instance or on-premise machine.

Memory Network (LSTM) based contextual embedding layer, an "attention flow layer", and a modeling layer include bi-directional LSTMs. We are using a pre-trained version of BiDAF available in the AllenNLP (Gardner et al., 2017) library.

Future releases of Katecheo will include the ability to swap out the reading comprehension model for newer architectures based on, e.g., BERT (Devlin et al., 2018) or XLNet (Yang et al., 2019) or custom trained models.

3 Architecture and Configuration

All four of the Katecheo modules are containerized with Docker (Merkel, 2014) and are deployed as pods on top of Kubernetes (Hightower et al., 2017) (see Figure 2). In this way, Katecheo is completely portable to any standard Kubernetes cluster including hosted versions in AWS, GCP, Digital Ocean, Azure, etc. and on-premises version that use vanilla Kubernetes, OpenShift, CaaS, etc.

To provide developers with a familiar interface to the question answering system, we provide a REST API interface. Developers can call Katecheo via a single endpoint with ingress to the system provided by Ambassador[7], a Kubernetes-native API Gateway.

Seldon-core[8] is used to simplify the routing between the four modules, create the REST API, and manage deployments. To create the Seldon deployment of the four modules, as depicted in Figure 2, we: (1) create a Python class for each module that contains standardized Seldon-specified methods and that loads the various models for making predictions; (2) wrap that Python class in a standard, containerized Seldon model server using a public Seldon Docker image and s2i [9]; (3) push the wrapped Python code to DockerHub [10]; (4) create a Seldon inference graph that links the modules in a Directed Acyclic Graph (DAG); and (5) deploy the inference graph to Kubernetes. After all of these steps are complete, a single REST API endpoint is exposed. When a user calls this single API endpoint the Seldon inference graph is invoked and the modules are executed using the specified routing logic.

To specify the topic names, topic NER models, and topic knowledge base JSON files (as mentioned in reference to Figure 1), the user need only fill out a JSON configuration file template in the following format:

```
[
  {
    "name": "topic 1 name",
    "ner_model": "<link>",
    "kb_file": "<link>"
  },
  {
    "name": "topic 2 name",
    "ner_model": "<link>",
    "kb_file": "<link>"
  },
  etc...
]
```

where each $\langle link \rangle$ would be replaced with a respective URL containing the NER model or knowledge base JSON file. The linked NER models need to be spaCy compatible and compressed into a single zip file, and the linked knowledge

[7]https://github.com/datawire/ambassador

[8]https://github.com/SeldonIO/seldon-core

[9]https://github.com/openshift/source-to-image

[10]https://hub.docker.com/

base JSON files need to include both titles and bodies as specified in the Katecheo GitHub repository README file. Once this configuration file is created, a deploy script can be executed to automatically deploy all of the Katecheo modules.

4 Example Usage

We demonstrated the utility of Katecheo by deploying the system for question answering in two topics, *Medical Sciences* and *Christianity*. These topics are diverse enough that they would warrant different curated sets of knowledge base articles, and we can easily retrieve knowledge base articles for each of these subjects from the Medical Sciences[11] and Christianity[12] Stack Exchange sites, respectively.

We also have access to NER models for both of these topics. For the *Medical Sciences* NER model, we utilized the *en_ner_bc5cdr_md* model from scispaCy (Neumann et al., 2019), which is trained on the BC5CDR corpus (Wei et al., 2015). For the *Christianity* topic, we utilize a custom spaCy NER model trained on annotated data from the GotQuestions website[13].

Example inputs and outputs of the system are included in Table 1. As can be seen, the system is able to match many questions with an appropriate topic and subsequently generate an answer using the BiDAF comprehension model. Not all of the answers would fit into conversational question answering in terms of naturalness, but others show promise.

There were cases in which the system was not able to classify an input question into an appropriate topic, even when there would have been a closely matching knowledge base article. In particular when testing the system on the *Medical Sciences* topic, we noticed a higher number of these cases (see the fourth and fifth rows of Table 1). This is due to the fact that the pre-trained *Medical Sciences* NER model from scispaCy is primarily intended to recognize chemical and disease entities within text, not general medical sciences terminology. On the other hand, the NER model utilized for the *Christianity* topic is more generally applicable within that topic.

[11]https://medicalsciences.stackexchange.com/

[12]https://christianity.stackexchange.com/

[13]https://www.gotquestions.org/

5 Conclusions

In conclusion, Katecheo is a portable and modular system for reading comprehension based question answering. It is portable because it is built on cloud native technologies (i.e., Docker and Kubernetes) and can be deployed to any cloud or on-premise environment. It is modular because it is composed of four configurable modules that collectively enable identification of questions, classification of those questions into topics, a search of knowledge base articles, and reading comprehension.

Initial usage of the system indicates that it provides a flexible and developer friendly way to enable question answering functionality for multiple topics or domains via REST API. That being said, the current configurations of Katecheo are limited to answering from knowledge bases constructed in a question and answer format, and the current topic classification relies on topical NER models that are compatible with spaCy. In the future, we plan to overcome these limitations by extending our knowledge base search methodology, enabling usage of a wider variety of pre-trained models, and exploring other topic matching/modeling techniques to remove our NER model dependency.

The complete source code, configuration information, deployment scripts, and examples for Katecheo are available at https://github.com/cvdigitalai/katecheo. A screencast demonstration of Katecheo is available at https://youtu.be/g51t6eRX2Y8.

References

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Kelsey Hightower, Brendan Burns, and Joe Beda. 2017. *Kubernetes: Up and Running Dive into the*

Question	Matched Topic	Generated Answer
What happens when I take high doses of iodine?	Medical Sciences	the blood loss can be much more severe. This could lead to anaemia and potentially death
What effect does alcohol have on children?	Medical Sciences	memory loss, loss of coordination and alcohol poisoning
How can you treat colon cancer?	Medical Sciences	removed before bowel cancer even begins to develop
Why do we cold sores on the lips?	None	None
What is the ideal salt intake?	None	None
What is the basis for Limited Atonement?	Christianity	Unconditional Election
What does the Bible say about vegetarianism?	Christianity	I think you would be hard pressed to say that the Bible commands a vegetarian diet
Should a Protestant read the apocryphal books?	Christianity	Knowing why it was rejected (was it heretical or known to be falsified or simply not surely inspired) might help in such an evaluation. Personally I have found the extra-biblical works quite interesting in their own right

Table 1: Example inputs, outputs, and matched topics from a Katecheo system deployed to provide question answering on two topics, Medical Sciences and Christianity.

Future of Infrastructure, 1st edition. O'Reilly Media, Inc.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

V. I. Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

Baichuan Li, Xiance Si, Michael R. Lyu, Irwin King, and Edward Y. Chang. 2011. Question identification on twitter. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 2477–2480, New York, NY, USA. ACM.

Silvia Lovato and Anne Marie Piper. 2015. "siri, is this you?": Understanding young children's interactions with voice input systems. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 335–338, New York, NY, USA. ACM.

Dirk Merkel. 2014. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239).

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. 2019. Scispacy: Fast and robust models for biomedical natural language processing.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. *ArXiv*, abs/1611.01603.

Chih-Hsuan Wei, Yifan Peng, R Leaman, Allan Peter Davis, C.J. Mattingly, J Li, T.C. Wiegers, and Zhiyong lu. 2015. Overview of the biocreative v chemical disease relation (cdr) task. pages 154–166.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *ArXiv*, abs/1906.08237.