CFO: A Framework for Building Production NLP Systems

Rishav Chakravarti[1] **Cezar Pendus**[2] **Andrzej Sakrajda**[2] **Anthony Ferritto**[1]
Lin Pan[1] **Michael Glass**[2] **Vittorio Castelli**[2] **J. William Murdock**[1]
Radu Florian[2] **Salim Roukos**[2] **Avirup Sil**[2*]

[1]IBM Watson, [2]IBM Research AI

{rchakravarti, cpendus, ansa, panl, mrglass, vittorio, murdockj, raduf, roukos, avi}@us.ibm.com
aferritto@ibm.com

Abstract

This paper introduces a novel orchestration framework, called CFO (COMPUTATION FLOW ORCHESTRATOR), for building, experimenting with, and deploying interactive NLP (Natural Language Processing) and IR (Information Retrieval) systems to production environments. We then demonstrate a question answering system built using this framework which incorporates state-of-the-art BERT based MRC (Machine Reading Comprehension) with IR components to enable end-to-end answer retrieval. Results from the demo system are shown to be high quality in both academic and industry domain specific settings. Finally, we discuss best practices when (pre-)training BERT based MRC models for production systems.

1 Introduction

Production NLP (Natural Language Processing) and IR (information retrieval) applications often rely on a system flow consisting of multiple components that need to be woven together to build an end-to-end system (A. Ferrucci et al., 2010; Yang et al., 2019). This paper presents a novel approach for defining flow graphs and a toolkit for compiling those definitions into deploy-able production grade systems[1].

Though the framework, which we refer to as CFO (COMPUTATION FLOW ORCHESTRATOR), is well suited to a variety of use cases, we demonstrate it by creating an interactive QA (Question Answering) system that can be used both for academic benchmarking as well as industry specific use cases. The interactive system integrates SOTA (state-of-the-art) BERT-based MRC (Machine Reading Comprehension), an Elasticsearch based document retrieval component, and a deduplication & sorting component to provide end-to-end answer retrieval. We will refer to this demonstration system as GAAMA (Go Ahead, Ask Me Anything). The key contributions of this work, therefore, are to (1) introduce a novel framework for stitching together deployable NLP components, (2) demonstrate the framework with an end-to-end QA system, and (3) discuss the training steps necessary for adapting a SOTA MRC model to a data set before plugging it into the QA system.

Section 2 provides the motivations and details of the CFO framework, section 3 discusses the specific model components integrated into GAAMA, section 4 discusses experimentation to adapt the BERT-based MRC component to this system, section 5 discusses related work, and, finally, section 6 provides a conclusion and discussion of future work.

In addition, a (private) screencast video demonstration of GAAMA has been uploaded at `http://ibm.biz/gaama_demo` (along with a supplementary presentation of the CFO framework at `http://ibm.biz/gaama_cfo_demo`).

2 CFO Architecture

The CFO framework relies on two sets of system specifications to define the computation flow graph. First, each node within the graph defines its service name, input message data fields, and output message data fields using Google's Protocol Buffer Interface Definition Language[2]. Second, the orchestrator is described using a custom specification format allowing the user to declare:

1. The set of nodes (provided as containerized gRPC[3] microservices). Each node will have

[*] Corresponding author.

[1]We are actively seeking to open source the toolkit and flow definition language. If successful, we will be releasing the code to `http://ibm.biz/cfo_framework`

[2]`https://developers.google.com/protocol-buffers/`

[3]`https://grpc.io/`

implemented a microservice according to the node's declared interface specification.

2. The set of nodes to treat as entry points to the flow graph.
3. A mapping within the flow for each *data element* comprising the input and output message interfaces for nodes.
4. For ease of deployment, the specification also provides the ability to describe deployment specific configuration settings like service ports, docker registry location etc.

Given these specification files, CFO provides a compiler to auto-generate an orchestrator node which implements the computation flow graph, a (dockerized) launch script, and a simple REST interface / GUI which provide access to the defined entry points and debug information.

The resulting design allows data flows and dependencies to be described both concisely and transparently. The flow specification allows for easy debugging and modification of how data fields enter the computation flow, get transformed, and finally outputted. This is a significant departure from traditional orchestration systems which would require parsing through source code to determine and change the route taken by data fields through the computation flow. Furthermore, the use of Protocol Buffers to describe these data fields ensures a language and platform agnostic representation that does not compromise the system's speed or ability to ensure data type correctness at compile time (as opposed to traditional JSON/XML representations which need to encode/decode from strings at run time).

Another benefit of the CFO toolkit is to auto-generate the serialization and connectivity code for each node as well as exposing the entry points via REST interfaces. Relinquishing this responsibility to the CFO toolkit frees the developer up from writing a significant amount of boilerplate code and worrying about distributed system best practices such as enforcing time outs, error propagation, latency logging, and parallelization through asynchronous calls. As an illustrative example, the auto-generated code for the GAAMA demo consists of 2,800 lines of C++ orchestration code.

Finally, the CFO toolkit generates a set of shell scripts, docker images, and config files for deploying and running the flow graph using either

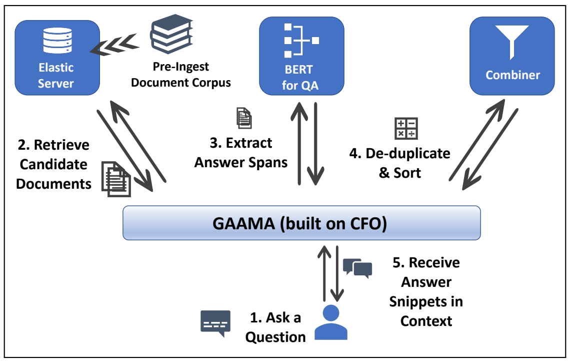

Figure 1: GAAMA System Architecture

docker-compose[4] or kubernetes[5]. This allows the generated project to be deployed both locally for debugging as well as on modern cloud infrastructure.

3 GAAMA Architecture

As a simple case study for CFO, we create a demonstration QA system consisting of four nodes: (1) an Elasticsearch[6] based IR node (2) a BERT based MRC node (3) an answer "deduplication" node and (4) a final answer combiner node. See fig. 1 for an overview of the QA system.

As described in section 2, each component starts by declaring an interface specification allowing the underlying implementation to be swapped out without the need to modify the rest of the QA system. Next, a gRPC server is implemented with the core business logic. Auto-generated gRPC code stubs provide the core communication/serialization logic for the service layer of the server. We describe the IR and MRC nodes in further detail in the following two sections.

3.1 IR with Elasticsearch

The core business logic of the IR node uses Elasticsearch APIs to retrieve the top k documents from the appropriate corpus based on BM25, a popular variant of term frequency overlap between the query and document text (Robertson et al., 1976). Two document corpora are ingested using the standard English analyzer. The first corpus consists of Wikipedia paragraphs used for academic benchmarking and the second corpus consists of an industry dataset made up of IBM Technical Support Documentation. The user interface allows us to choose either corpus when asking

[4]https://docs.docker.com/compose/
[5]https://kubernetes.io/
[6]https://www.elastic.co/products/elasticsearch

questions to evaluate the system. We use "paragraphs" as the base unit of ingestion in line with (Yang et al., 2019) which shows this as an optimal pre-processing step for consumption by a MRC component.

The node's input interface, therefore, accepts query text, a hyperparameter k, and a target corpus identifier. Its output interface produces a list of document texts accompanied by retrieval scores. As discussed earlier, these interfaces are defined using protocol buffer definitions, so we follow standard steps[7] for auto-generating a gRPC server with placeholders for the custom business logic of retrieving documents. Similarly, with the server logic in place, we follow standard steps[8] to create a docker image that CFO will need to launch the gRPC server. Though we are wrapping Elasticsearch's index based retrieval implementation here, we can swap our implementation for more recent Neural IR based techniques (Craswell et al., 2017) without changing the exposed interface or the orchestration code.

3.2 MRC with BERT

The MRC node similarly wraps a BERT-for-QA model in a dockerized gRPC server which accepts a single query-document pair as its input and produces a span from the document along with a prediction score as its output. Note that CFO's orchestrator automatically realizes that the IR node produces a list of documents, while the MRC node accepts a single document at a time. So CFO will take care of calling the MRC node for each of the k retrieved documents (using asynchronous calls to parallelize requests if a configuration flag is set).

The underlying BERT-for-QA model is based on (Alberti et al., 2019). BERT (Devlin et al., 2018) is one of a series of pre-trained neural models that can be fine tuned to provide state-of-the-art results in NLP (Peters et al., 2018; Howard and Ruder, 2018; Radford et al., 2019) including on the SQuAD (Rajpurkar et al., 2018) and NQ (Kwiatkowski et al., 2019) tasks that align with our MRC based QA.

We use the Huggingface PyTorch implementation of BERT [9] which supports starting from a Base (a 12 layer, 768 hidden dimension, 12 attention head, 110M parameter transformer network) or a Large (a 24 layer, 1024 hidden dimension, 16 attention head, 340M parameter transformer network) model. An output feed forward layer is added on top of this to produce 3 sets of scores: (1) scores at each token offset marking the likelihood of an answer chunk starting at this offset (2) scores at each token offset marking the likelihood of an answer chunk ending at this offset (3) a score for the entire sequence marking the likelihood of the question being answerable given the current context.

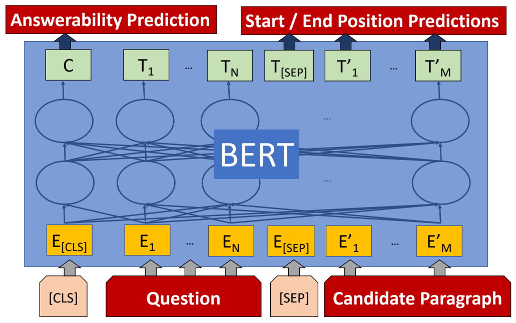

Figure 2: BERT for QA (Devlin et al., 2018)

The parameters of the entire network are fine tuned using a set of question and document pairs where annotators provide the correct start and end offsets or have marked the question-document pair as having no correct answer. Refer to fig. 2 for a visual depiction of the model and to (Alberti et al., 2019) for additional details about model architecture and implementation.

Section 4 includes practical pre-training, fine tuning, and hyperparameter optimization steps for building the final model deployed as part of GAAMA. At the time of writing, our BERT-based MRC model[10] was the best performing submission (dated 7/31/2019), outperforming the next best system by 1% on F1 on the Natural Questions public leaderboard[11].

4 Experiments

Prior to integration of the MRC node into GAAMA, we first experiment with data preparation and training of the BERT MRC model as a standalone component using dev sets provided with the NQ and SQuAD data sets (see 5 for more on these data sets). NQ is preferred for evaluat-

[7] https://grpc.io/docs/tutorials/
[8] https://github.com/grpc/grpc-docker-library/
[9] https://github.com/huggingface/pytorch-transformers

[10] The best performing submission to the leaderboard uses an ensemble of models using different hyperparameters rather than a single model
[11] https://ai.google.com/research/NaturalQuestions/leaderboard

	F1
Prior Work	
DecAtt + Doc Reader (Parikh et al., 2016)	31.4
BERT (Devlin et al., 2018)	50.2
BERT w/ SQuAD 1.1 (Alberti et al., 2019)	52.7
This Work	
BERT w/ U-MRC	53.6
BERT w/ U-MRC & SQuAD 1.1	54.2
BERT w/ U-MRC & SQuAD 1.1 + SQuAD 2.0 Data Aug	**54.5**

Table 1: Dev Set Performance on NQ with different pre-training & data augmentation techniques. We also report some baselines from (Alberti et al., 2019) for context

Pre-Training	EM	F1
BERT (Devlin et al., 2018)	78.7	81.9
BERT w/ U-MRC	82.2	85
BERT w/ U-MRC & NQ	**82.6**	**85.4**

Table 2: Dev Set Performance on SQuAD 2 with different pre-training strategies.

ing production systems since the questions were "naturally" generated and does not suffer from the observational bias inherent in SQuAD's data collection approach (Kwiatkowski et al., 2019). When reporting results with the SQuAD dataset, we use the methodology (and evaluation script) made available with (Rajpurkar et al., 2018). Similarly, when reporting results with the NQ dataset, we use the methodology (and evaluation script) made available with (Kwiatkowski et al., 2019). Once we are satisfied with the performance of this model, we integrate into GAAMA and evaluate manually using an internal corpus.

We use the F-score at an "optimal" threshold for the dev set[12] as the headline metric for assessing the system. Latency measurements are carried out using a random sample of examples on a system with an Intel® Xeon® E5-2690 16-core CPU, 2 Nvidia® Tesla® P100 GPUs, and 128GB of RAM.[13] We then examine the feasibility of deploying base and large models in a production environment on GPUs and CPUs.

4.1 Pre-Training & Data Augmentation

We explore two types of pre-training. The first follows (Alberti et al., 2019) by leveraging a similar task for which supervised labels are available and pre-training the model on it before moving onto fine tuning on the target dataset. Specifically, we use SQuAD 1.1 (Rajpurkar et al., 2016). Table 1 shows that this strategy can provide an absolute improvement of 2.5% over a model that starts with just the default BERT language model.

We also employ (Glass et al., 2019)'s approach to using an unsupervised auxilary task that is better aligned to our final task (i.e. MRC) than the default Masked Language Model and Next Sentence Prediction used in (Devlin et al., 2018) to pre-train the BERT models. Using the Wikipedia corpus, we create cloze style queries by masking out terms (named entities or noun phrases) in a sentence. Then we identify an answer bearing passage from the Wikipedia corpus that is relevant to the query (as identified by BM25 IR). This allows us to pre-train all layers of the BERT model including the answer extraction weights by training the model to extract the answer term from the selected passage. Like the Masked Language Model, this task relies on predicting a masked component of an input sequence, but the prediction is generated by extraction rather than generation. Table 1 labels these results as "BERT w/ U-MRC" and shows that this additional training on a MRC specific unsupervised task improves the model's final fine-tuned performance on the NQ task by 1.5%. Table 2 similarly shows the benefits of these pre-training strategies on the SQuAD 2.0 dataset.

In addition, as noted by the authors of the original BERT-for-QA submission to SQuAD (Devlin et al., 2018), there can be a benefit to fine tuning the entire network with labelled examples from multiple datasets. The last row of table 1 shows an incremental gain of 0.3% by introducing SQuAD 2.0 during the fine-tuning phase. For now, the additional data is simply shuffled into the first 80% of mini batches during the fine-tuning phase.

4.2 BERT Models & Latency

Most model settings are taken from (Alberti et al., 2019) with the exception of batch size and learning rate which are tuned using the approach from (Smith, 2018). In addition, we experiment with models trained on BERT base and BERT large to understand trade-offs between latency and accuracy. Using the hardware described in section 4, we evaluate F1 and latency on a subset of the NQ

[12] See `http://www.ibm.biz/confidence_thresholding` for more on choosing business specific thresholds

[13] We only use 1 P100 GPU or 8 CPU threads in latency experiments

Model	$F1$	T^G_{50}	T^G_{95}	T^C_{50}	T^C_{95}
Base	42.5	0.05	0.49	0.53	2.32
Large	50.8	0.10	0.66	1.51	6.00

Table 3: F1 and latencies for BERT base and large models running on GPU and CPU for a subset of the NQ dev set. T^D_K is the K-th percentile query latency in seconds when running on device D (GPU or CPU).

dev set[14]. In order to decrease latency, we simulate passage retrieval to send GAAMA the most relevant passage by selecting the first correct top level candidate if there is one and the first (incorrect) top level candidate if there is not. We find in table 3 that switching from base to large yields an 8.3% absolute increase in F1 in exchange for 1.3x to 2.8x increases in latency. When running GAAMA on a GPU these result in manageable 95th percentile query latencies of less than a second; whereas on the CPU the 95th percentile times are in excess of two and five seconds for base and large respectively. For large, even the median latency is greater than one and a half seconds, effectively cementing GPUs as a requirement for deploying to production environments. In future work we intend to explore network pruning or knowledge distillation techniques for potential speedups with the large model.

5 Related Work

Recently (Yang et al., 2019) proposed BERTSerini, an end-to-end QA pipeline demo that leverages the Anserini IR toolkit (Yang et al., 2017) to look for relevant documents for a question, then uses BERT-based techniques (Devlin et al., 2018) to extract the correct answer. However, their reliance on a Lucene based IR toolkit means that constructing a NLP pipeline would either require pipeline components to be written as Lucene based plugins (which comes with a variety of constraints on programming language and structure) or writing custom orchestration code to connect components outside of the toolkit. Similar constraints are imposed by other popular NLP pipeline toolkits such as StanfordNLP (Qi et al., 2018) and Spacy[15] (both of which require development in Python with limited flexibility in training neural models with other frameworks such as Tensorflow[16]).

In contrast, CFO's inherent programming language and platform agnostic microservice architecture encourages flexibility and robustness in being able to switch out individual components without re-doing boilerplate code. In addition, CFO's out-of-the-box support for containerization provides flexibility in the compute infrastructure that can be leveraged for rapid deployment to both local and cloud environments.

This flexibility is important in a domain such as Machine Reading Comprehension (MRC) where recent advances in language-modeling based pre-trained embeddings like ELMO (Peters et al., 2018) and BERT (Devlin et al., 2018) along with large scale open data sets like the Stanford Question Answering Dataset (SQuAD) 1.1 (Rajpurkar et al., 2016) and its successor SQuAD 2.0 (Rajpurkar et al., 2018) have spurred a diverse array of model architecture improvements in a short time span. Recent work has even produced systems that surpass human-level exact match accuracy on the SQuAD datasets, causing us to focus on the challenging new Natural Questions (NQ) dataset (Kwiatkowski et al., 2019) where the questions do not have any observational bias as they were not artificially created. To the best of our knowledge, there is no current software framework paper that shows its analysis on the NQ dataset and displays strong empirical performance.

UIMA (Ferrucci and Lally, 2004) is an integration framework that provides defined APIs for analyzing unstructured information and a shared data structure for storing the results of that analysis. When paired with the UIMA Asynchronous Scaleout layer (Apache UIMA Community, 2018) and the Distributed UIMA Cluster Computing tool (DUCC Team, 2013), this technology stack provides many of the same core capabilities that CFO does: pipeline orchestration, microservice deployment and management, data serialization and connectivity, etc. However, CFO is designed for modern cloud environments and includes built-in integration with docker-compose and kubernetes; the UIMA stack *can* be used with these technologies but facilities for doing so are not built-in to the stack so more development effort is needed in those contexts. Also, UIMA and CFO both require that each component expresses its data model (including input and output specifications) declaratively, but UIMA then unifies the data model of all components into a global type hierarchy, which requires some level of compatibil-

[14]Used 500 random examples from dev set for experiments
[15]`https://spacy.io/`
[16]`https://www.tensorflow.org/`

ity across type definitions. In contrast, CFO only requires consistency in the data model for components that directly connect to each other, and the names of corresponding types and fields do not need to match. These differences make CFO easier to use in cases where components were developed by different developers and integrated by a third party.

6 Conclusion

This paper introduces CFO, a novel methodology and toolkit for rapid development of production grade systems for use cases which can be represented as computation flow graphs. We demonstrate the use of this framework to build an end-to-end QA system composed of a SOTA MRC model that is adapted to answering "natural language questions". We also show experimentation using the NQ dataset to illustrate training techniques that can be used to build SOTA systems on top of existing pre-trained language models like BERT.

We are actively seeking to open source the CFO framework and hope that, once available, the community will be able to quickly build and deploy their own SOTA NLP components as interactive multi-component systems. We also intend on expanding GAAMA to incorporate additional QA components to improve its performance through approaches like query expansion for improved recall and network pruning for improved latency.

References

David A. Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Christopher Welty. 2010. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31:59–79.

Chris Alberti, Kenton Lee, and Michael Collins. 2019. A bert baseline for the natural questions.

Apache UIMA Community. 2018. UIMA Asynchronous Scaleout: Version 2.10.3.

Nick Craswell, W Croft, Maarten de Rijke, Jiafeng Guo, and Bhaskar Mitra. 2017. Sigir 2017 workshop on neural information retrieval (neu-ir'17).

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

DUCC Team. 2013. Distributed UIMA cluster computing.

David Ferrucci and Adam Lally. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327348.

Michael Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, Bhargav GP Shrivatsa, Dinesh Garg, and Avirup Sil. 2019. Span selection pre-training for question answering.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *TACL*.

Ankur Parikh, Oscar Tckstrm, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *EMNLP*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *NAACL*.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. CoNLL. ACL.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2019. Improving language understanding by generative pre-training.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *EMNLP*.

C S. Robertson, H. Zaragoza, Stephen Robertson, and Hugo Zaragoza. 1976. The probabilistic relevance framework: Bm25 and beyond.

Leslie N. Smith. 2018. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay.

Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of lucene for information retrieval research. SIGIR. ACM.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini.