**DARPA ASKE DCC – Milestone 4, 2019**

Authors: Akrotirianakis Ioannis, Amar Viswanathan Kannan, Fradkin Dmitriy, Roy Aditi, Canedo Arquimedes, Mohammad Al Faruque

**Contents**

[1. Introduction 2](#_Toc7615682)

[2. Text2Graph 2](#_Toc7615683)

[3. Code2Graph 3](#_Toc7615684)

[4. Knowledge graph alignment 4](#_Toc7615685)

[5. Demo 5](#_Toc7615686)

# Introduction

The purpose of this report is to provide a summary of the work performed under the Defense Advanced Research Projects Agency (DARPA) project titled “Deep Code Curator (DCC)”, under the agreement contract #HR00111990010 for the period from **April 1st to May 1th, 2019**.

During the above period we focused on finetuning the various learning models used in the text2graph module and code2graph modules. In addition, we started the development of the Knowledge Graph alignment module which will merge the information stored in text2graph, image2graph and code2graph. We have also developed a demo, which will be used to highlight the capabilities of the methods we are proposing and the software we have developed during Phase 1. The overall architecture remains the same as in the Phase I. During this month we have streamlined our architecture in a more compact diagram that is shown in Figure 1. This architecture fits better within the overall framework described by the ASKE program. It should be noted, however, that the detailed architecture, defined in all previous milestone reports, remains in effect and will also guide our development through Phase 2.

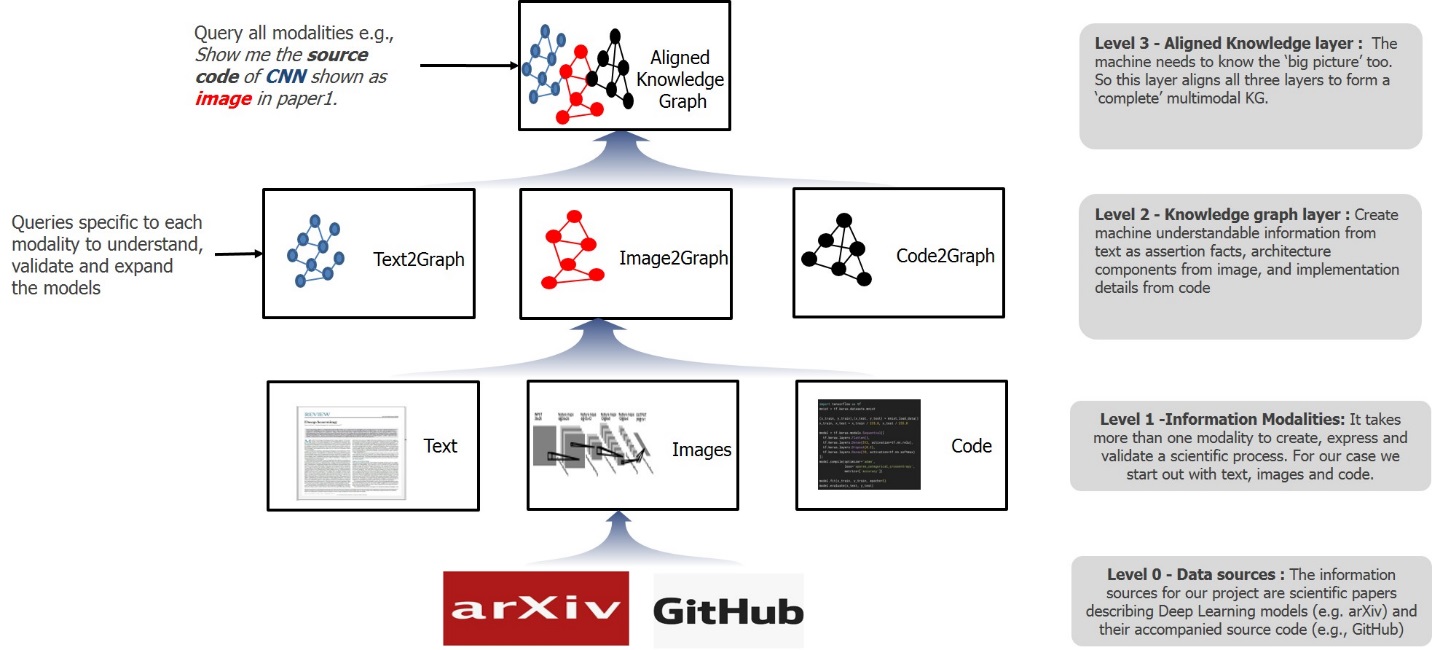


Figure 1: DCC architecture within the ASKE framework

# Text2Graph

We have focused on optimizing the parameters in our NER and RE learning models. In particular, we have found that the batch size has an effect in the training time and accuracy of the model. The strategy we follow is to initialize the batch size to a small number (in our case we set it to 1) and then compound to a larger number that is determined by the size of our training data. Currently, the upper bound we found to have the best performance is equal to 32. This strategy starts by setting the batch size to 1 and increase each batch until it reaches the maximum defined size.

We have also found out that the dropout rate has an effect on overfitting. In our dataset, it is beneficial to set a high dropout rate at initial stages, and subsequently reduce it towards a smaller value. Currently we start with an initial dropout rate equal to 0.5 and reduce it by a factor of 0.01 at every iteration until it reaches the lower value equal to 0.2. The above optimizations have resulted in improved performance of the learning algorithms we have developed so far.

# Code2Graph

For the code2graph module our effort has focused on the following directions:

**Dataset Extraction:** In order to start training the inference algorithms for RDF graph completion, we have focused on both manual and automated dataset extraction of the code RDFs. For the manual method, we have created an ontology for the deep learning architecture and utilized it to manually parse the RDF of the codes to form a baseline and filtered RDFs to be used for testing the inference algorithms. Beside this we have continued automating the RDF extraction from the computational and light-weight methods. For the computational graph we have continued to the task to map the nodes with the TensorFlow APIs. We are exploring both manual and semi-automated annotation options for the RDF extraction and mapping it to the deep learning architecture ontology. The mapping to the ontology will be crucial for creating the knowledge base and effectively aligning it with other RDF graphs.

**Filtering RDF graph:** Both the light-weight and the computational graph method utilized to extract the RDF knowledge graph from the codes consisted of large amount of metadata (either related to the underlying code implementation or the coding structure utilized by the authors). Hence, we have been focusing our work on creating rule-based methodologies filter the RDF graphs. For light-weight method (which extracts the RDF from the static call graph analysis), we have identified and increased nodes which can be filtered out from the call trees. These nodes are based on the structure of the python APIs (both for TensorFlow and Keras). We have extracted these APIs from the webpage of TensorFlow while creating an ontology. For computational graph, we have increased the list of node filters that can detect the C/C++ runtime metadata from the deep learning architecture information.

**Sequence Generation in light-weight method:** Since the light-weight method utilizes static call-graph analysis, we worked on improving the recursive calls made to track the sequence of deep learning architectures. We removed some of the bugs associated with duplication of the nodes in the sequence and addition of edges between the sequence in the RDF graph. We aim to combine this sequence information with the computational graph-based code extraction to further improve the deep learning architecture information extraction from the codes.

**Knowledge graph embedding:** We have started working on triple-translation based knowledge graph embedding algorithms for encoding the entities and relations of the code RDFs. The translation-based algorithms assume the head, relation and tail of the RDF triples to be vectors on a plane. We are exploring other approaches for effective embedding of the knowledge graph of the code. We have utilized simple metrics such a Rank and Hit rate for measuring the accuracy of the embedding algorithms. We aim to further increase the evaluation metrics for effective comparison of the RDF graphs.

# Knowledge graph alignment

We have also started the implementation of the knowledge graph alignment which plays an important role in our project and in Phase 2. The aim of this module is to create a KG that merges the knowledge stored in *text2graoh*, *image2graph* and *code2graph*. We use KGs as shallow representations of ontologies. All the scientific facts extracted by text2graph, image2graph and code2graph will populate the ontology as instances or entities of the domain. This gives us the scope and flexibility of using lightweight semantics and still rapidly construct a graph to represent the information coming from the three considered modalities derived from a scientific paper. Our KG alignment technique involves both alignment at the conceptual level and alignment at the instance level.

Based on this, we looked at some of the different approaches used for knowledge graph alignment and then propose an initial direction for our work. Graph alignment approaches from the logic-based world can be classified under two broad schemes. The first kind utilizes the structure and semantics of the underlying graph to suggest nodes that can be aligned. In the second kind, structural properties of the instances are used to align the graph. These approaches are briefly summarized below:

**Ontology based approaches:** In ontology-based approaches, the semantics of OWL properties and class definitions are utilized as similarity metrics to determine the ability to map concepts. Traditional approaches to ontology matching have focused on either aligning the classes or aligning the instances. The approaches that align classes include techniques such as sense clustering, lexical and structural characteristics and composite approaches. These approaches do not consider sub classes or sub properties in an ontology and thus do not scale well to modern knowledge graphs, which also encode taxonomies or schemas.

**Instance based approaches:** In instance-based approaches, techniques generally use the terminological structure, logical deduction or cluster by relations. As is the case with ontology-based approaches, instance-based approaches utilize only part of the knowledge graph.

**The proposed alignment technique:**

In our knowledge graph, we encode both domain concepts (ontologies) and instances of these concepts. Hence, our alignment approach utilizes the structure of both the ontology data and instance data to look for nodes that require alignment and linking. This is shown in the figure below, where we refer to three diverse knowledge graphs, each representing different modalities of a scientific publication. Each of these knowledge graphs encodes hierarchies and taxonomic concepts as schema definitions and scientific facts as instances of these concepts. To align this, we propose a technique that utilizes *concept alignment* and *instance alignment.* The former uses structural features to align concepts that have similar definitions across schemas, whereas the latter uses an instance data aware methodology, where the features of the nodes are considered to define a similarity metric. This metric is then used to compute similarities between candidate nodes. We also define a threshold metric to merge node features that are common above a threshold value. Using this we link the text2graph, image2graph and code2graph. to form a unified domain knowledge graph. We also investigate the use of a similarity metric to augment existing state-of-the-art entity alignment techniques for knowledge graphs and merge entity nodes. The overall concept is shown in Figure 2.

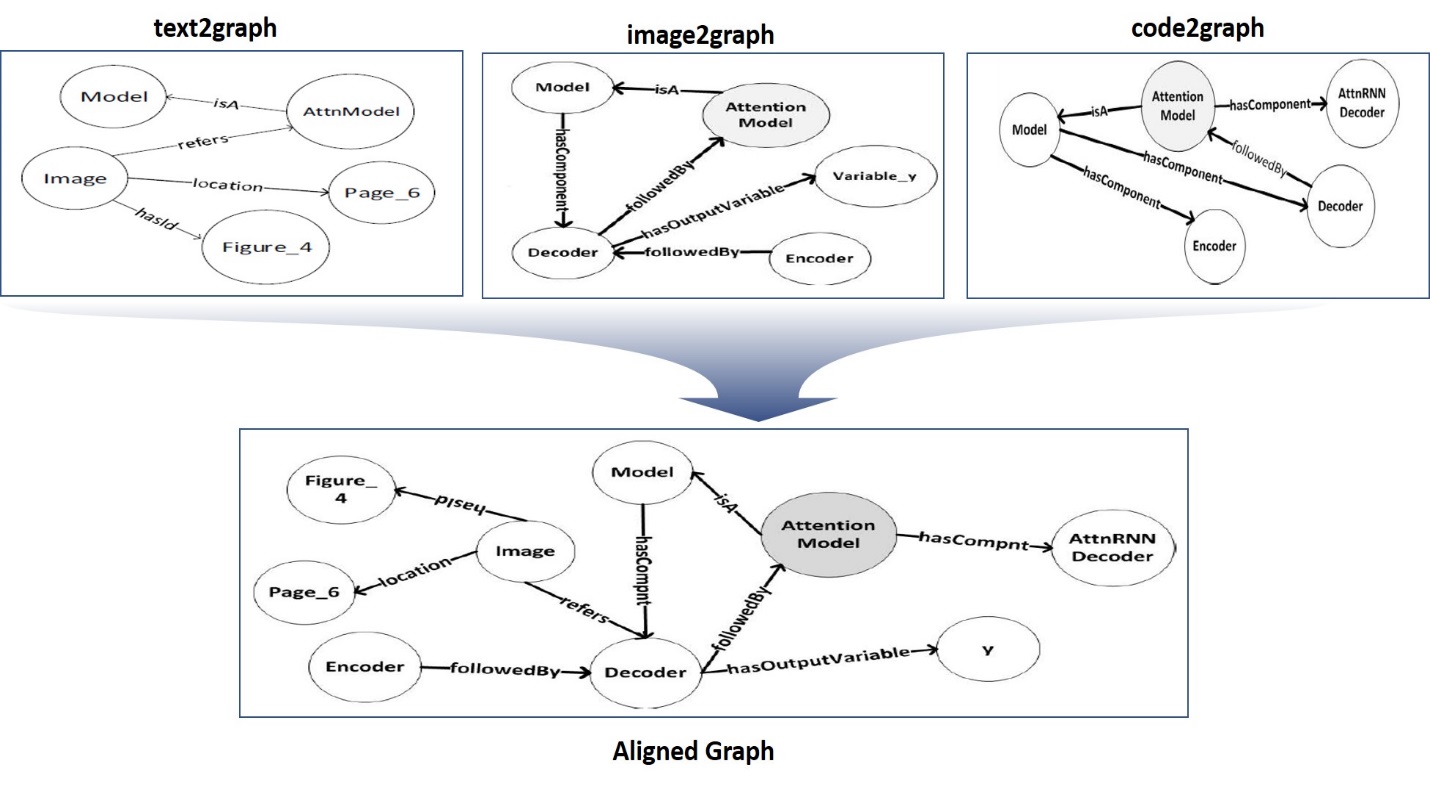
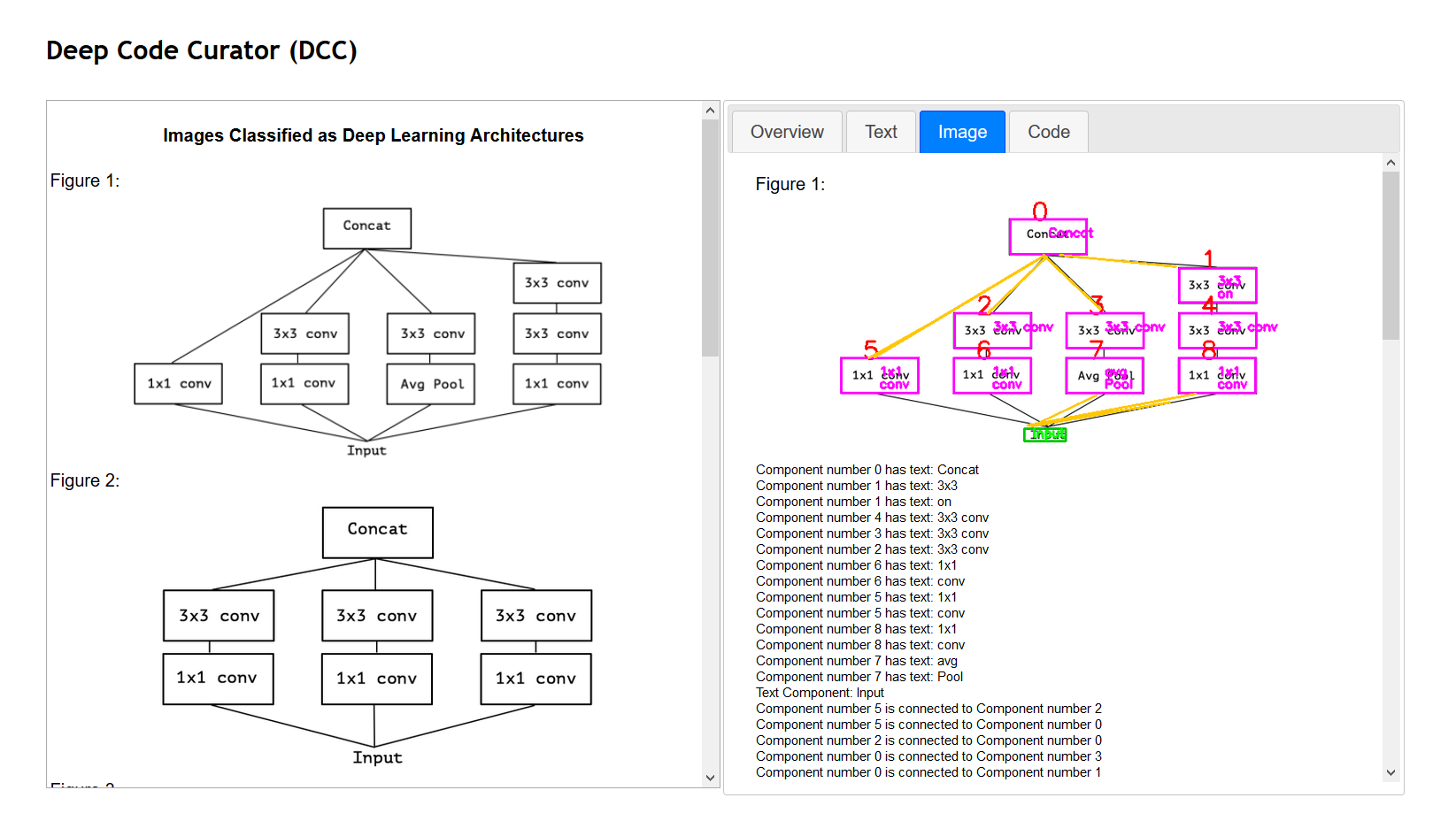


Figure 2: KG alignment

# Demo

We have started the implementation of a demo that will showcase the capabilities of the methods we are proposing and the software we have developed during Phase 1. The demo has four different modes: An *Overview* mode, and a separate mode for each of the three modalities (*Text*, *Image*, and C*ode*). Figure 3 shows a sample view from the *Image* mode. The user interface has two main frames, as marked in Figure 1 with “Frame 1” (left frame) and “Frame 2” (right frame).



Frame 2

Frame 1

**Figure 3. A screenshot of the Deep Code Curator Demo in the *Image* mode**

Different modes are triggered via the tabs on the top of the Frame 3. Table 1 provides a summary of the contents of the two frames for each of the four modes. Currently, the demo provides results from one paper, and the results are obtained in an offline fashion, i.e. the results are stored and displayed, instead of running the corresponding codes from the user interface.

**Table 1**

|  |  |  |
| --- | --- | --- |
| Mode | Frame 1 | Frame 2 |
| Overview | The input paper that is used by the demo is displayed in the PDF format. | In progress: We plan to display the integrated Knowledge Graph of the paper. |
| Text | In progress: Display the text in the deep learning paper that is being used to generate the knowledge graph. | In progress: Include the knowledge graph generated by the triples computed from the NER and relation extraction modules. |
| Image | A list of images that are extracted from the paper and classified as showing deep learning architectures. | For each image from Frame 1, results of our image2graph module are displayed. In particular we display   * an annotated version of the image that shows the detected components and their assigned id’s * a set of triples that describe the extracted components and their relationships. |
| Code | The source code that implements the deep learning architecture described in the scientific paper. | The visualization of the resultant knowledge graph. This is a smaller version of the TensorFlow computational graph, focusing on the main components that are relevant to the deep learning architecture described in the scientific paper. |

**Implementation Details**

The demo is designed as a single web page (HTML). The contents of the frames for different modes are standalone web pages and they are integrated into the demo using Iframe’s. For its implementation, we use JavaScript, and the following JavaScript libraries:

* JQuery[[1]](#footnote-2) and JQuery UI[[2]](#footnote-3): To implement the backbone structures, and interactions (tab selection).
* CodeMirror[[3]](#footnote-4): To display the source code (*Code* mode, Frame 1)

**Next Steps**

We plan to complete the in-progress parts (*Overview* and *Text* modes), and add the following functionalities (as time permits):

* Ability to switch between different papers (Could be a predefined set of papers, or alternatively users could provide the link for the paper)
* Running the DCC code in an online fashion, e.g. accessing them through web services and displaying the results that are retrieved online.

1. <https://jquery.com> [↑](#footnote-ref-2)
2. <https://jqueryui.com> [↑](#footnote-ref-3)
3. <https://codemirror.net> [↑](#footnote-ref-4)