Jing LI **Research Statement**

http://www.li-jing.com

School of Computer Science and Engineering, Nanyang Technological University, Singapore

1 PhD Thesis Work

My research aims to build up **semantic Web systems** to support information needs of software developers using **information retrieval**, **natural language processing**, and **machine learning** techniques.

Programming is evolving because of the prevalence of the Web. Nowadays, it is a common activity that developers search the Web to find information in order to solve the problems they encounter while working on software development tasks. Understanding *information needs* of developers can motivate the design of current software development and maintenance tools. We conducted an ***empirical study*** [Li et al., 2016, Bao et al., 2017] to investigate the strategies that how developers seek and use web resources at a micro-level. The empirical study revealed ***three key insights***: ***First***, *developers might have an incomplete or even incorrect understanding of their needs*; ***Second***, *there is a gap between the producers and consumers of software documentation*; ***Third***, *many important pieces of information that developers need are explicitly undocumented in software documentation*. There insights motivate further studies of supporting developers' information needs.

1.1 Discovering Learning Resources for Programmers

Software developers need access to correlated information (*e.g.*, API documentation, Wikipedia pages, Stack Overflow questions and answers) which are often dispersed among different web resources. The first insight suggests that *developers might have an incomplete or even incorrect understanding of their needs*. There are many situations in which developers only know partial topics they are looking for, but at the same time is willing to explore correlated web resources to extend their knowledge or to satisfy their curiosity. Specifically, we present an item-based collaborative filtering technique, named LinkLive, for automatically recommending a list of correlated web resources for a particular web page. The recommendation is by exploiting **hyperlink associations** from the crowdsourced knowledge on Stack Overflow. We motivate our research using an exploratory study of hyperlink dissemination patterns on Stack Overflow. We then present our LinkLive [Li et al., 2018e] technique that uses multiple features, including hyperlink co-occurrences in Q&A discussions, locations (*e.g.*, question, answer, or comment) in which hyperlinks are referenced, and votes for posts/comments in which hyperlinks are referenced. Experiments using 7 years of Stack Overflow data show that, LinkLive recommends correlated web resources with promising accuracy in an open setting. A user study of 6 participants suggests that practitioners find the recommended web resources useful for web discovery.

Figure 1: The LinkLive Recommendation.

We implement a proof-of-concept tool of our LinkLive approach. The backend hyperlink associative network (*i.e.*, hyperlink correlation model) is mined from training data, which is the Stack Overflow data dump (July 2008 to September 2014). When a user visits a web page or mouse hovers over a

hyperlink in the web page, the LinkLive tool searches for hyperlinks from the backend model. If the hyperlink is found in the backend, and the hyperlink has been referenced 5 times or more in the training dataset, the tool recommends top-10 correlated web resources for the given hyperlink. Figure 1 shows the LinkLive recommendation when mouse hovers over hyperlink (`https://en.wikipedia.org/wiki/Singleton_pattern`) in a Stack Overflow question[1]. In addition to the recommendation of correlated web resources, the LinkLive tool also shows a bar chart of citation history for the seed hyperlink and each recommended hyperlink.

1.2 Learning to Answer Programming Questions

The second insight reveals that *this is a gap between the producers and consumers of software documentation*. Programming languages and software packages are often well supported through formal documentation. Official software documentation provides a comprehensive overview of software usages, but not on specific programming tasks or use cases [Li et al., 2018d]. Often there is a mismatch between the documentation and the question stating specific programming tasks because of different wordings.

We observe from Stack Overflow that the best answers to programmers' questions often contain links to formal documentation. In this study, we propose a novel ***deep-learning-to-answer*** framework, named `QDLinker` [Li et al., 2018b], for answering programming questions with software documentation. `QDLinker` learns from the large volume of discussions in community-based question answering site to bridge the semantic gap between programmers' questions and software documentation. Specifically, `QDLinker` learns question-documentation semantic representation from these question answering discussions with a four-layer neural network, and incorporates semantic and content features into a learning-to-rank schema.

Shown in Figure 2, the `QDLinker` framework consists of three core modules: *candidate document generation*, *a four-layer neural network*, and *learning a ranker*. Given a programming question in natural language, *candidate document generation* returns a small set of software documents which are considered relevant to the question. The DNN module learns the semantic representation of query-documentation pairs in a latent space, and generates features for the ranker module. The features by DNN are fully automatic without human intervention. Nevertheless, the network does allow handcrafted features to be inserted in the join layer, illustrated in Figure 2. The learned features are then fed to a learning-to-rank schema to train a ranker, to pick up the more relevant software documents among the candidates.

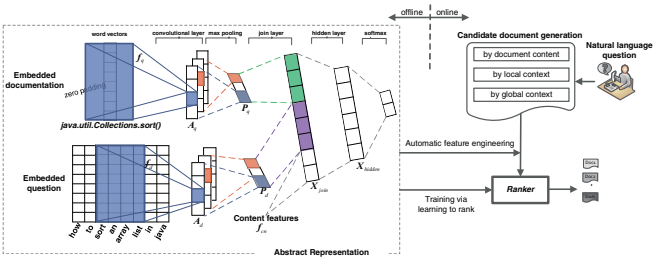

Figure 2: The architecture of `QDLinker`.

Our approach does not require manual feature engineering or external resources to infer the degree of relevance between questions and documentation. Through extensive experiments, results show that `QDLinker` effectively answers programming questions with direct links to software documentation. `QDLinker` significantly outperforms the baselines based on the traditional retrieval models and Web search services dedicated for software documentation retrieval. The user study shows that `QDLinker` effectively bridges the semantic gap between **the intent** of programming questions and **the content** of software documentation.

[1] `http://stackoverflow.com/questions/23360052`

1.3 Distilling Crowdsourced Negative Caveats

The third insight reveals that *many important pieces of information that developers need are explicitly undocumented in software documentation*. Application Programming Interfaces (APIs) are foundations of software development. To program to an API, developers need to know not only "how to use the API", but also "how **not** to use the API". Negative caveats of API are about "how not to use an API", which are often absent from the official API documentation. When these caveats are overlooked, programming errors may emerge from misusing APIs, leading to heavy discussions on Q&A websites like Stack Overflow. If the overlooked caveats could be mined from these discussions, they would be beneficial for programmers to avoid misuse of APIs.

However, it is challenging because of the discussions are informal, redundant, and diverse. In this study, we propose **DISCA** [Li et al., 2018c], a novel approach to automatically **DIS**tilling desirable API negative **CA**veats from large-scale unstructured Q&A discussions, as shown in Figure 3. **DISCA** consists of five core modules: selecting candidate sentences, filtering out context-dependent sentences, identifying prominent domain-specific terms, discovering semantically diverse aspects and selecting API negative caveats.

We formulate our research problem as a text-summarization task. Through sentence selection and prominent term clustering, **DISCA** ensures the distilled caveats are *context-independent*, *prominent*, *semantically diverse and non-redundant*. Quantitative evaluation in our experiments shows that the proposed **DISCA** significantly outperforms four text-summarization techniques. We also show that the distilled API negative caveats could greatly augment API documentation through qualitative analysis.

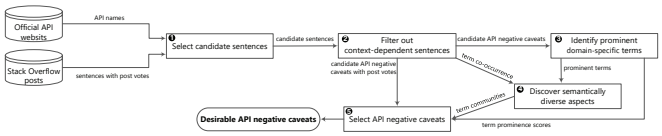

Figure 3: Overview of the **DISCA** framework.

2 Ongoing Work

2.1 Neural Document Understanding

Distributed representation has been explored in various natural language tasks at fine granularities such as words and phrases. However, document-level articles are ubiquitous documents on the Web, such as new article and technical documentation. A fundamental goal of information retrieval systems is to identify, given a query, documents that have relevant text. Understanding the meaning of text in documents is beneficial for many downstream applications such as question answering, document retrieval and reading comprehension. We cast the document understanding as a three-stage problem: **retrieval**, **segmentation** and **comprehension**.

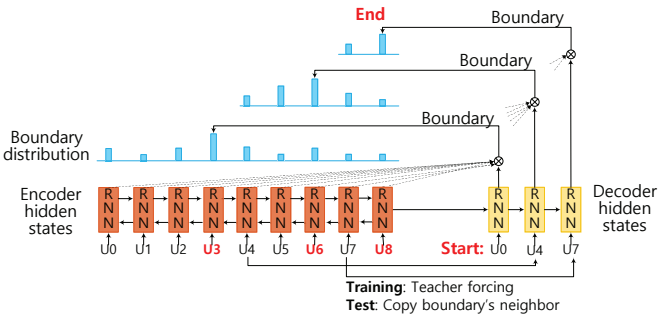

Figure 4: The model architecture of SEGBOT. Input sequence: $U0, U1, ..., U8$. Identified boundaries: $U3, U6, U8$.

We have investigated some approaches to retrieval in Section 1.2. In the future, we will investigate reading comprehension. In this study, we focus on text segmentation for document understanding.

Depending on the levels of granularity, the task can be defined as segmenting a document into topical segments, or segmenting a sentence into elementary discourse units (EDUs). Traditional solutions to the two tasks heavily rely on carefully designed features. The recently proposed neural models do not need manual feature engineering, but they either suffer from sparse boundary tags or they cannot well handle the issue of variable size output vocabulary.

Figure 4 shows our generic end-to-end segmentation model, named SEGBOT [Li et al., 2018a], consisting of three components: encoding phase, decoding phase and pointing phase. SEGBOT uses a bi-directional recurrent neural network to encode input text sequence. The model then uses another recurrent neural network together with a pointer network to select text boundaries in the input sequence. In this way, SEGBOT does not require hand-crafted features. More importantly, our model inherently handles the issue of variable size output vocabulary and the issue of sparse boundary tags. We conduct experiments at two levels of granularity: document-level topic segmentation, and sentence-level EDU segmentation. The results show that SEGBOT achieves new state-of-the-art results on both tasks. We make SEGBOT available online and provide users with Application Programming Interface (API): `http://138.197.118.157:8000/segbot/`

2.2 Neural Named Entity Recognition

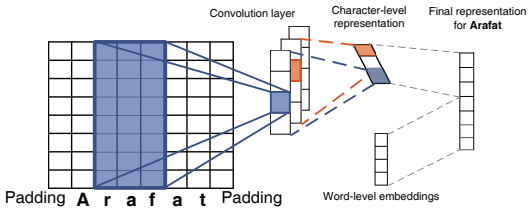

Figure 5: Comprehensive word representation.

Named entity recognition (NER) is a challenging task that has traditionally relied on hand-crafted features (*e.g.*, capitalization patterns) and domain-specific knowledge (*e.g.*, gazetteers) to achieve high performance.

In this study, we propose a neural end-to-end model to recognize named entities. Our model requires no task-specific resources and manual feature engineering beyond pre-trained word embeddings on unlabeled corpora. More specifically, our model consists of two modules: comprehensive word representation (as shown in Figure 5) and neural NER model (as shown in Figure 6). The module of comprehensive word representation uses a convolutional network and pre-trained word embedding to represent English words. The module of NER model uses three recurrent neural networks to recognize the named entities.

Take for example: **Israel**$_{LOC}$ approves **Arafat**$_{PER}$'s flight to **West Bank**$_{LOC}$. Figure 5 illustrates that how to represent "Arafat" using distributed representations. Figure 6 illustrates that how to identify the three named entities (Location: Israel, Person: Arafat, Location: West Bank) in given text. The neural model in Figure 6 consists of encoder network, begin pointer network and end pointer network, which are implemented with different recurrent neural networks. We are conducting experiments on CoNLL 2003 corpus and OntoNotes corpus to evaluate the effectiveness of our proposed model.

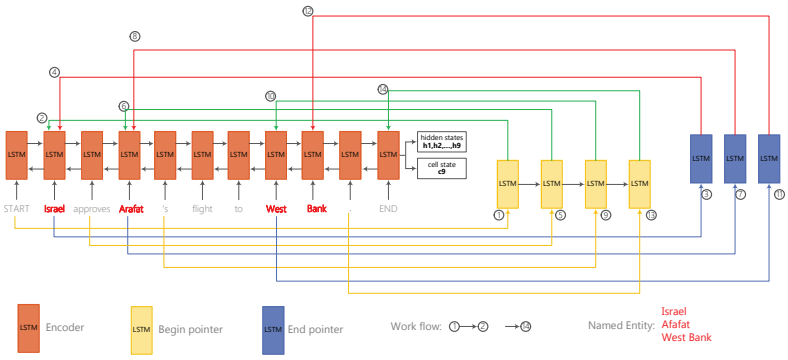

Figure 6: The Proposed Neural NER Model.

References

[Bao et al., 2017] Bao, L., Li, J., Xing, Z., Wang, X., Xia, X., and Zhou, B. (2017). Extracting and analyzing time-series hci data from screen-captured task videos. *Empirical Software Engineering*, 22(1):134–174.

[Li et al., 2016] Li, J., Bao, L., Xing, Z., Wang, X., and Zhou, B. (2016). Bpminer: mining developers' behavior patterns from screen-captured task videos. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1371–1377. ACM.

[Li et al., 2018a] Li, J., Sun, A., and Joty, S. (2018a). Segbot: A generic neural text segmentation model with pointer network. In *IJCAI*. Under Review.

[Li et al., 2018b] Li, J., Sun, A., and Xing, Z. (2018b). Learning to answer programming questions with software documentation through social context embedding. In *Information Science (INS)*. Revision.

[Li et al., 2018c] Li, J., Sun, A., and Xing, Z. (2018c). To do or not to do: Distill crowdsourced negative caveats to augment api documentation. In *Journal of the Association for Information Science and Technology (JASIST)*. Revision.

[Li et al., 2018d] Li, J., Xing, Z., and Kabir, A. (2018d). Leveraging official content and social context to recommend software documentation. In *IEEE Transactions on Services Computing (TSC)*. Revision.

[Li et al., 2018e] Li, J., Xing, Z., and Sun, A. (2018e). Linklive: Discovering web learning resources for developers from q&a discussions. In *WWWJ*. Under Review.