CRUISE: Cold-Start New Skill Development via Iterative Utterance Generation

Yilin Shen, Avik Ray, Abhishek Patel, Hongxia Jin
Samsung Research America, Mountain View, CA, USA
{yilin.shen,avik.r,abhisehek.p,hongxia.jin}@samsung.com

Abstract

We present a system, CRUISE, that guides ordinary software developers to build a high quality natural language understanding (NLU) engine from scratch. This is the fundamental step of building a new skill for personal assistants. Unlike existing solutions that require either developers or crowdsourcing to manually generate and annotate a large number of utterances, we design a hybrid rule-based and data-driven approach with the capability to iteratively generate more and more utterances. Our system only requires light human workload to iteratively prune incorrect utterances. CRUISE outputs a well trained NLU engine and a large scale annotated utterance corpus that third parties can use to develop their custom skills. Using both benchmark dataset and custom datasets we collected in real-world settings, we validate the high quality of CRUISE generated utterances via both competitive NLU performance and human evaluation. We also show the largely reduced human workload in terms of both cognitive load and human pruning time consumption.

1 Introduction

Artificially intelligent voice-enabled personal assistants have been emerging in our daily life, such as Alexa, Google Assistant, Siri, Bixby, etc. Existing off-the-shelf personal assistants provide a large number of capabilities, referred to as *skills*, and the number of skills keeps growing rapidly. Thus, it is critically desirable to design an easy to use system that facilitates developers to quickly build high quality new skills.

The key of developing a new skill is to understand all varieties of user utterances and carry out the intent of users, referred to as *natural language understanding* (NLU) engine. Existing industrial personal assistant products or open source tools (e.g., API.ai, WIT.ai) require software developers themselves or via crowdsourcing to manually input various natural utterances and annotate the slots for each utterance. Recently, researches have been made to bootstrap the utterance generations. These approaches first generate canonical utterances based on either lexicon/grammar (Wang et al., 2015) or language/SQL templates (Iyer et al., 2017); then utilize crowdsourcing to create paraphrases and correct labels. Unfortunately, they require software developers to have natural language expertise and still heavily rely on costly crowdsourcing. Thus, it is significantly and crucially desirable to develop a system for helping *ordinary developers* quickly build a high quality skill for personal assistants.

In this paper, we present a system, called *Cold-start iteRative Utterance generatIon for Skill dEvelopment* (CRUISE). As the name suggests, CRUISE aims to guide software developers to build a new skill from scratch, a.k.a., *cold-start*. It is defined from two aspects: *cold-start software developers* which refer to the ordinary developers who do not have either linguistic expertise or complete functionalities of the new skill in mind; and *cold-start dataset* which means that there is zero or very few training samples available. Specifically, CRUISE initially takes the list of intents in a new skill as inputs from software developers and runs a hybrid rule-based and data-driven algorithm to automatically generate more and more new utterances for each intent. During the whole process, software developers only need to iteratively *prune* the incorrect samples. As

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics-System Demonstrations, pages 105–110
Melbourne, Australia, July 15 - 20, 2018.

such, CRUISE does not depend on crowdsourcing to conduct the heavy task of manually generating utterances and annotating slots.

2 Background and Related Work

Natural language understanding is a key component in skill development. In personal assistants, since users intend to use spoken language to interact with personal assistant agents, most industrial products are focused on spoken language understanding (SLU) in which it is sufficient to understand user query by classifying the intent and identifying a set of slots (Liu and Lane, 2016). One class of approaches is to paraphrase user utterances to increase the number of training set (Barzilay and Lee, 2003; Quirk et al., 2004; Kauchak and Barzilay, 2006; Zhao et al., 2009; Prakash et al., 2016). However, these approaches depend on the existence of large amount of dataset for training paraphrasing model. As discussed above, the most relevant works (Wang et al., 2015; Iyer et al., 2017) bootstrapped the utterances based on grammar and SQL templates respectively and then relied on crowdsourcing to increase the utterance varieties and correct the labels. Unfortunately, they require both linguistic expertise from software developers and heavy human workload. In this paper, we use NLU and SLU engines equivalently.

3 CRUISE System Overview

3.1 Our Settings

Settings for Software Developers: To build a new skill S, a software developer starts with providing a list of predefined intents in this skill S. For each intent, as shown in Figure 1, the software developer first reviews and prunes an automatically constructed knowledge base. Next, the only human labor of a developer is to iteratively prune incorrect generated utterances. In the end, CRUISE will automatically outputs both a well-trained NLU engine for skill S and a large number of annotated correct utterances that can be used directly by third parties to train their own NLU engines.

Offline Preprocessed Components: CRUISE also consists of the following components which have been preprocessed offline *without the involvement of software developers*: *(1) Publicly available InfoBox template* Ω *(InfoBox, 2017):* contains a subset of information/attributes about an object, i.e., a set of object-attribute pairs. For example, the object `food` has attributes such as `course`, `region`, etc. (2) *Language model*: pre-trained on a public corpus (e.g., Wikipedia). (3) *Pre-built concept hash table*: for each word in the language model vocabulary, we use MS term conceptualizer (Wang and Wang, 2016) to find its most likely concept/category. For example, the word *pizza* is considered as an instance of the concept `food`. Then a hash table is constructed to map each concept to its instances.

3.2 CRUISE Design

We discuss the goals of CRUISE system design and the key ideas to achieve them.

Cold start Support: As discussed in introduction, the first trade-off in CRUISE design is between cold start (lack of training data and expertise from developers) and a high quality NLU engine. In order to accommodate the developers who do not have any linguistic expertise and reduce their workload to manually generate various utterances, we design an *Iterative Utterance Generation* approach. It starts from an intent as a simplest natural language utterance and decomposes the complex utterance generation into small tasks to tackle them one by one iteratively.

Reduced Human Workload: Another trade-off in CRUISE design is between human workload minimization and high quality of generated utterances. To address this, we design CRUISE from two aspects: (1) *Human-in-the-loop Pruning* allows developers iteratively *prune* the incorrect generated utterances. In next iteration, more utterances are generated *only* based on the previously selected *correct* utterances. (2) *Automated Utterance Annotation* generates an utterance corpus that can be directly used to train NLU engine without extra human efforts. The idea is to generate *tagged utterances* in which each tag can be simply coupled with its corresponding instances to automatically annotate the slots. For example, a tagged utterance contains `@food` and `@size` tags rather than their instances such as *pizza* and *large* (in Figure 1).

3.3 CRUISE Components & Workflow

As the running example shows in Figure 1, CRUISE has two main steps, *knowledge base construction* and *iterative utterance generation*.

Step 1. Knowledge Base Construction: for each

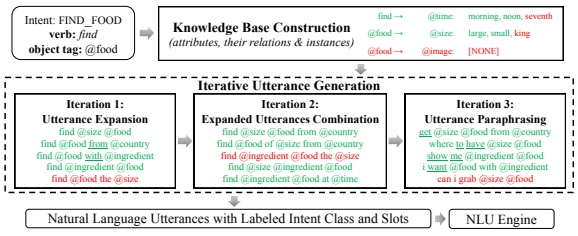


Figure 1: A running example of CRUISE system: it takes a new intent "FIND_FOOD" as input and outputs both annotated natural language utterances (each utterance with an intent label and slots labels) and a trained NLU engine. Each black box corresponds to a component in CRUISE, with both correct outputs (in green) and incorrect outputs (in red) to be pruned by developers. The underlined words are generated by the data-driven tagged sentence filler (Section 4.2) and the other words are generated by rules (Section 4.1).

intent (e.g., "FIND_FOOD" includes a verb *find* and an object tag @FOOD), CRUISE constructs *a knowledge base* with the following information: (1) identified list of attribute tags depending on object tag using Infobox template Ω (e.g. attribute tag @SIZE depends on object tag @FOOD); (2) the sample instances belong to each tag using pre-built concept hash table (e.g. instances "large", "small" for tag @SIZE). In addition, developers can also add or remove the tags and instances of each tag.

Step 2. Iterative Utterance Generation: CRUISE iteratively generates more and more utterances with human-in-the-loop pruning. CRUISE outputs the generated natural language utterances with both intent and slots annotations as well as a ready-to-use NLU engine trained on these utterances.

4 Iterative Utterance Generation

In this section, we describes the details of utterance generation in each iteration. *The key idea to generate utterances in each iteration is the hybrid of rule-based and data-driven approaches.* In brief, we utilize a small set of rules to derive a list of incomplete tagged utterances with blanks (word placeholders); then use data-driven tagged utterance filler algorithm to fill in the blanks. The rest of this section includes rule-based iteration design and data-driven tagged utterance filler algorithm respectively.

4.1 Rule-based Iteration Design

The key idea is to decompose the task of generating complex utterances into three subtasks and tackle each task in one iteration. Specifically, we divide the utterance generation task into the following subtasks in three iterations (idea

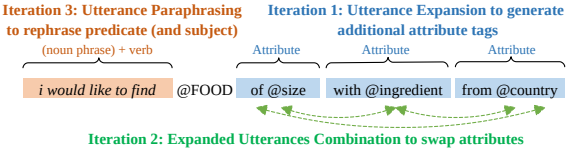


Figure 2: A running example to illustrate *subtasks* in each iteration towards generating a tagged utterance

illustration in Figure 2 and examples in Figure 1): *(1) Utterance Expansion:* generate attributives and adjuncts to expand an utterance into utterances with an additional new tag. *(2) Expanded Utterances Combination:* swap attributives and adjuncts to concatenate previously expanded utterances. *(3) Utterance Paraphrasing:* rephrase predicate (and subject) to paraphrase each utterance. At the end of each iteration i, we provide all generated utterances to the software developer for pruning such that only the selected *correct* utterances will be the input of next iteration. At last, we output the natural language utterances by substituting instances into tags with slot annotations.

Iteration 1. Utterance Expansion: Given an intent as a simplest verb phrase consisting of only a verb and its direct object (tag), the goal is to expand it into utterances such that each generated utterance has an additional attribute tag associated with the object tag. Thanks to the simple verb phrase, the expansion has no semantic ambiguity. Thus, for each new tag t, we expand the utterance by inserting t before and after the object tag. While it is straightforward to insert t directly before the object, the insertion of t after the object need joiner words where we introduce blanks. In the end, we fill out the blanks (usually 1-3) in tagged utterances as described in Section 4.2.

Iteration 2. Expanded Utterances Combination: The goal is to combine the previously generated correct utterances (two tags in each utterance) into long utterances with all combination of different tags in each utterance. We generate the permutations of attribute tags themselves (and with joiner words) before (after) the object. This iteration then outputs utterances these attribute tag permutations before and after the object with non-overlapping attribute tags. Thanks to the correct input utterances, most of combined utterances are surely correct, which saves a lot of pruning efforts.

Iteration 3. Utterance Paraphrasing: Since the previous iterations have covered the varieties for attributives phrases and clauses, the goal

of iteration 3 is to increase the varieties of predicates. We generate different predicates for a tagged utterance as follows: (1) verb replacement, (2) wh-word question rephrasing, and (3) "I" started utterance rephrasing. Both (2) and (3) are motivated by the application of personal assistants that help the user who initiates the utterance. Likewise, we invoke tagged utterance filler to fill out the blanks (usually 3) for each predicate. In order to further reduce the human pruning workload, we group the same predicates for developers to prune them at once instead of pruning every single utterance again and again.

4.2 Data-driven Tagged Sentence Filler

As a key data-driven subroutine, this module takes a tagged utterance with blanks (i.e., word placeholders) u as the input and outputs the complete tagged utterance with all blanks filled by natural language words, called *filler words*. We first instantiate the attribute and object tags in u using their mapped instances in the pre-built concept hash table. Since all instances belong to the vocabulary of the pre-trained language model, we avoid tackling out of vocabulary problem. Based on the intuition that good filler words are usually generated repeatedly in many instantiated utterances, we fill out the blanks in all instantiated utterances and return the K_t (up to 20) filler words ranked by the frequency of their appearances.

To fill out the blanks in an incomplete natural language utterance, we use an efficient beam search algorithm via RNN based pre-trained language models. This returns a list of K_n (up to 30) best filler words, ranked according to their likelihood scores.

5 Experimental Evaluation

We implement the CRUISE system with an easy-to-use user interface (Figure 3) with the thumb up/down mechanism for efficient human pruning. We have internal developers to use and evaluate this real system in terms of both utterance quality and human workload.

5.1 Data Quality Evaluation

5.1.1 Objective Evaluation via NLU Engines

We validate our CRUISE system by first evaluating the performance of existing NLU engines trained using our generated utterances compared with using benchmark or manually

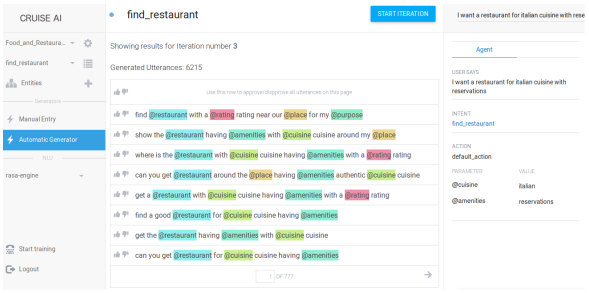


Figure 3: CRUISE User Interface

Table 1: Human NLU engine vs. CRUISE NLU engine results in benchmark and custom datasets

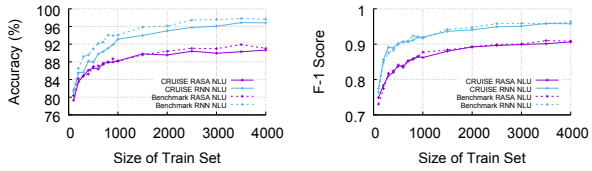
		Human NLU		CRUISE NLU	
Dataset	NLU Engine	Intent Accuracy	Slot Tagging F-1 Score	Intent Accuracy	Slot Tagging F-1 Score
ATIS	RASA	93.29%	90.84	83.33%	80.25
	RNN	97.96%	96.02	82.60%	84.70
Food	RASA	99.4%	91.92	**99.58%**	**93.91**
	RNN	99.31 %	92.28	**99.73%**	**94.70**
Hotel	RASA	-	92.22	-	89.92
	RNN	-	92.09	-	**94.85**

generated utterances. For simplicity, we refer to our CRUISE generated datasets as *CRUISE Dataset* in comparison of *Benchmark/Human Dataset*. Correspondingly, the NLU engines trained on CRUISE and benchmark human generated datasets are referred to as ***CRUISE NLU*** and ***Human NLU*** engines respectively. Both NLU engines are evaluated by testing on benchmark or user generated utterances.

NLU Engines & Performance Metrics: To maximally reduce the bias from NLU engines, we evaluate the performance using different existing NLU engines: open source *RASA* (RASA, 2017) and deep learning based *RNN* NLU engine with joint learning of intent classifier and slot tagger (Liu and Lane, 2016). Both NLU engines target on classifying the intent of each whole utterance and identifying tags/entities (a.k.a. slot tagging). Thus, we use the accuracy and F-1 score as the metrics for intent classifier and slot tagging respectively. We run RASA NLU engine using their default parameters.

Benchmark Dataset Evaluation: Although the benchmark NLU trained on crowdsourced data is expected to perform much better than CRUISE NLU trained on machine generated dataset from cold start, we show that CRUISE NLU still achieves a high accuracy and efficiently trades off NLU performance and human workload.

We evaluate our system on the ATIS (Airline Travel Information Systems) dataset (Hemphill



(a) Intent Classifier **(b)** Slot Tagging

Figure 4: Mixed NLU results in ATIS Dataset

et al., 1990), a widely used dataset in SLU research. It contains 4,978 training utterances and 893 testing utterances with 127 distinct slot labels and 22 different intents. We generate 767,985 unique tagged utterances using CRUISE system. For a fair comparison, we randomly sample 5,000 utterances from CRUISE dataset as training set. Since ATIS is relatively larger, we select both word embedding and LSTM hidden dimension as 128 with 1 hidden layer in RNN NLU.

Table 1 reports the result of NLU performance comparison. As one can see, the performance of CRUISE NLU engine is roughly around 10-15% worse than benchmark NLU engine trained on crowdsourced benchmark data for both intent classification and slot tagging. After a detailed analysis, we find that CRUISE data has smaller vocabulary size (301 words) than the crowdsourced benchmark data (949 words) due to the selection of high likelihood words in beam search. Hence, we attribute a significant cause of errors because of the out-of-vocabulary words in test set. We further test CRUISE NLU on the subset of test set without out-of-vocabulary words and observe 5-6% improvement of NLU performance. Importantly, we observe that CRUISE NLU performs much better on more complex utterances, e.g., *"show me fares for round trip flights with first class of delta from miami into houston"*, where the benchmark NLU fail for both intent classification and slot tagging.

In addition to CRUISE NLU, we further test the performance of NLU engines which are trained by mixed CRUISE and benchmark datasets, named ***Mixed NLU***. The benchmark data is treated as the manual entry data from developers such that we can better study another trade-off between additional human workload and NLU engine performance. Figure 4 reports the result of mixed NLU engine performance with half CRUISE data and half benchmark data. Both the mixed and benchmark NLU engines achieve similar performance for different sizes of training set on the costly crowdsourced ATIS dataset. This implies that we can reduce nearly half human workload for developing a skill, given the negligible pruning effort (Section 5.2).

Real-World Setting Evaluation: We further evaluate CRUISE in a simulated real-world scenario when a software developer starts to develop a new skill. In order to do so, we create two custom datasets: (a) *Food* and (b) *Hotel*. Food data has three intents and hotel data has only one intent. Each intent is associated with six to eight different attributes/tags selected from InfoBox template or provided by internal developers. For each intent, we ask two developers to generate a list of tagged utterances manually and using our CRUISE system respectively. The total sizes of human and CRUISE generated utterances are 5,352 and 21,429 in food and hotel datasets respectively. For fairness, we randomly select a subset from human dataset as a standard test data to test both NLU engines. Table 1 shows that CRUISE NLU outperforms human NLU in most cases. This is because CRUISE dataset has a larger number and varieties of high quality utterances than human dataset.

5.1.2 Subjective Human Evaluation

We further evaluate the CRUISE dataset subjectively by soliciting judgments from Amazon Mechanical Turkers. Each turker was presented a task of rating utterances sampled from mixed CRUISE and human generated datasets. Turkers rate each question on a 5 point Likert scale (Likert, 1932) as to whether the utterance is *natural* and *grammatically correct*. Ratings range from 1 (worst) to 5 (best). Thus, our evaluation provides more detailed rating than what automatic metrics such as BLEU can provide (Papineni et al., 2002). In order to control the evaluation quality, we further judge the *trustworthiness* of each turker by scoring their performance on 20-30 gold-standard utterances that were internally rated by experts. Based on this trustworthiness score, we establish a group of *trusted turkers*. Then, we collect 10 ratings for each utterance from these trusted turkers. Finally, we compute the average score over all trusted ratings on 300-500 randomly sampled utterances in each dataset.

Table 2 reports human evaluation results between CRUISE and human generated data. We observe that CRUISE generated dataset achieves close performance in terms of both metrics in

Table 2: Human Evaluation Results

Dataset		Naturalness	Grammar	Overall
ATIS	CRUISE	3.32	3.37	3.35
	Human	3.74	3.78	3.76
Custom	CRUISE	**3.60**	**3.41**	**3.50**
	Human	3.35	3.08	3.21

ATIS, which is collected via costly crowdsourcing. More importantly, for human data generated by only a single developer in custom datasets, the results show that CRUISE data has better quality than human data in terms of both metrics.

5.2 Human Workload Analysis

We analyze the cognitive load via preliminary qualitative analysis from internal developers. Specifically, we interview the participated developers regarding different types of cognitive load (Sweller et al., 1998). In terms of *intrinsic cognitive load* about the inherent difficulty level to use CRUISE system, the developers concluded CRUISE as a more easy-to-use system than both existing industrial tools and academic solutions. *Extraneous cognitive load* is also largely reduced since our design enables batch processing of human pruning by one-click marking of all utterances in each page. At last, the developers are also satisfied with the reduced *germane cognitive load* due to the iterative pruning design in CRUISE which dramatically minimize the whole pruning effort by generating more utterances only based on the correct ones in previous iteration.

Next, we report the time consumption of human pruning to evaluate the workload quantitatively. As shown in Figure 5, we observe that it takes less than 0.15s on average to prune each utterance, and as low as 0.01s for some intents. This is because iteration design in our CRUISE system enables the capability that each picked correct utterance can generate many utterances at one time. In comparison, we observe that human developers take around 30 seconds on average to generate and annotate an utterance in our custom dataset. In the example of ATIS dataset, it takes around 0.05s on average to prune each utterance. Thus, a developer only needs to spend less than 5 mins on average to prune incorrect utterances in order to find 5,000 correct utterance for training a competitive NLU engine. For frequent intents, it takes less time to prune as CRUISE intends to generate more correct utterances determined by a better language model.

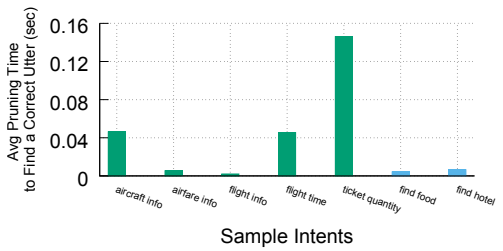


Figure 5: Time Consumption for Human Pruning (Green:ATIS, Blue:custom)

References

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *NAACL-HLT*.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The atis spoken language systems pilot corpus. In *HLT*.

InfoBox. 2017. https://en.wikipedia.org/wiki/Infobox.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *ACL*.

David Kauchak and Regina Barzilay. 2006. Paraphrasing for automatic evaluation. In *HLT–NAACL*.

Likert. 1932. https://en.wikipedia.org/wiki/Likert_scale.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. In *INTERSPEECH*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *ACL*.

Aaditya Prakash, Sadid A. Hasan, Kathy Lee, Vivek V. Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. 2016. Neural paraphrase generation with stacked residual LSTM networks. In *COLING*.

Chris Quirk, Chris Brockett, and William Dolan. 2004. Monolingual machine translation for paraphrase generation. In *EMNLP*.

RASA. 2017. https://rasa.ai/.

John Sweller, Jeroen J. G. Van Merrienboer, and Fred G. W. C. Paas. 1998. Cognitive architecture and instructional design. *Educational Psychology Review*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *ACL*.

Zhongyuan Wang and Haixun Wang. 2016. Understanding short texts. In *ACL (Tutorial)*.

Shiqi Zhao, Xiang Lan, Ting Liu, and Sheng Li. 2009. Application-driven statistical paraphrase generation. In *ACL*.