Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

How to manage a task-oriented virtual assistant software project: an experience report

Shuyue LI[1], Jiaqi GUO[1], Yan GAO[2], Jianguang LOU[2],
Dejian YANG[2], Yan XIAO[2], Yadong ZHOU[1], Ting LIU[‡1]

[1]*Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China*

[2]*Microsoft Research Asia, Beijing 100080, China*

E-mail: lishuyue1221@stu.xjtu.edu.cn; jasperguo2013@stu.xjtu.edu.cn; yan.gao@microsoft.com; jlou@microsoft.com; dejian.yang@microsoft.com; yan.xiao@microsoft.com; ydzhou@xjtu.edu.cn; tingliu@mail.xjtu.edu.cn

Received Sept. 30, 2021; Revision accepted Jan. 30, 2022; Crosschecked Mar. 22, 2022

Abstract: Task-oriented virtual assistants are software systems that provide users with a natural language interface to complete domain-specific tasks. With the recent technological advances in natural language processing and machine learning, an increasing number of task-oriented virtual assistants have been developed. However, due to the well-known complexity and difficulties of the natural language understanding problem, it is challenging to manage a task-oriented virtual assistant software project. Meanwhile, the management and experience related to the development of virtual assistants are hardly studied or shared in the research community or industry, to the best of our knowledge. To bridge this knowledge gap, in this paper, we share our experience and the lessons that we have learned at managing a task-oriented virtual assistant software project at Microsoft. We believe that our practices and the lessons learned can provide a useful reference for other researchers and practitioners who aim to develop a virtual assistant system. Finally, we have developed a requirement management tool, named SpecSpace, which can facilitate the management of virtual assistant projects.

Key words: Experience report; Software project management; Virtual assistant; Machine learning
https://doi.org/10.1631/FITEE.2100467 **CLC number:** TP311.5

1 Introduction

Task-oriented virtual assistants (VAs) are software systems that provide users with a natural language (NL) interface to complete domain-specific tasks, such as booking flights, ordering foods, and getting insights from business data. Although previous VAs had a long history from the early 1900s (Wikipedia, 2021), they were not mature enough to be widely applied in practice. Owing to the recent new technological advances in natural language processing (NLP) and machine learning (ML), we are now witnessing an increasing number of commercial VAs being developed. As predicted, both consumers and businesses will spend $3.5 billion by the end of 2021 on what are referred to as virtual personal assistants (Bradley, 2020).

It is, however, particularly challenging to manage a VA software project, as observed in our hands-on experience. First, due to the flexibility and diversity of NLs, it is difficult to define the scope of the functionality of a VA. Specifically, it is difficult to formally specify how people talk to a VA, using either texts or speeches, which brings challenges to the tasks of requirement gathering, requirement specifying, and software testing. Second, due to ML's data-driven nature and weak interpretability, it is difficult to estimate the effort needed to meet a requirement using ML techniques, which brings huge uncertainty

‡ Corresponding author
ORCID: Shuyue LI, https://orcid.org/0000-0001-6009-1707; Ting LIU, https://orcid.org/0000-0002-7600-0934

to development management. As we know, state-of-the-art VAs are built mainly based on new deep learning technologies. However, deep learning models are often black-box and not interpretable, which brings new challenges to development management. For example, testing and maintaining ML-based software systems are known to be challenging tasks (Marijan et al., 2019; Zhang TY et al., 2019; Zhang JM et al., 2022). Although many different VAs have been developed in the past few years, e.g., Analyza (Dhamdhere et al., 2017), Tableau Ask Data (Tableau, 2020a), and TaskVirtual (Task Virtual, 2020), their development management approaches and experiences have been hardly studied or shared in the research community or industry.

To bridge this gap, here we share our experience and the lessons learned from our practical VA project named XTalk (anonymized for confidential reason). Specifically, we describe five main concrete software engineering problems that we encountered in our project, including the problems in requirement management, development management, and quality management. Moreover, we share seven practices adopted in the XTalk project to tackle or mitigate the problems and we also discuss the insights behind these practices. Then, we summarize three lessons learned from the development of XTalk and present four open challenges that we encountered during the lifecycle of XTalk, some of which are promising research avenues. We believe that our practices and the lessons learned can be a useful reference for other researchers and practitioners to develop modern VAs.

In summary, we make the following main contributions in this study: (1) We summarize the problems we encountered in managing a task-oriented VA software project, including problems in requirement management, development management, and quality management; (2) We share our practices for addressing the problems and present three lessons learned at managing a task-oriented VA software project; (3) We develop a novel requirement management tool, named SpecSpace, which can improve the management efficiency for task-oriented VA software projects.

2 Overview of XTalk

In this section, we give a brief overview of XTalk, supplying the context for the discussion around management problems and practices in the subsequent sections.

XTalk is a VA that provides users with an NL interface to complete data analytics tasks. XTalk can make data analytics much more accessible to business users and relieves users from the burden of mastering professional tool usage and programming skills. Fig. 1 presents an example use of XTalk, in which a user raises a series of analytic questions concerning the best-selling product against a tabular dataset about product sales. XTalk can understand the user's questions and returns the corresponding analytic answers in various forms (e.g., a resulting table or a chart).

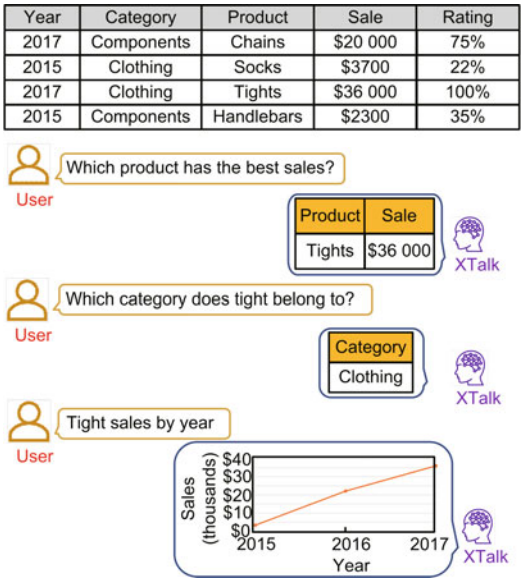

Fig. 1 An example use of XTalk

Like most VAs (Dhamdhere et al., 2017; Campagna et al., 2019), our XTalk can be formulated as a program synthesis problem (or a semantic parsing problem in the field of NLP): Taking a table and an NL question as the inputs, XTalk first translates the NL question to a program (e.g., a structured query language query) and then executes the program against the table to obtain its answer to the question. We solve the program synthesis problem with a hybrid method that integrates a set of rules and a deep neural network (DNN). As we will show in Section 4, using the hybrid method is one of our important practices for effective VA software project management.

Fig. 2 presents the overall architecture of XTalk with a running example. Given a table and an NL

question, XTalk takes three major steps to obtain the analytics results. First, XTalk's preprocessor module pre-processes the table and the question, including recognizing columns' metadata (e.g., data type, named entity) and extracting the lexical features for the question (e.g., part-of-speech tag and lemmas). Next, the program synthesizer module searches for the optimal programs for the question. For different types of questions, the program synthesizer module performs the program search with a guidance either from the set of rules or from the DNN model. We will discuss the trade-off between the rules and a DNN model in Section 4. Finally, the visualization module executes the top-ranked program against the table to obtain the analytics results. The module also visualizes the results according to the user's intents and the types of results. Interested readers can refer to our technical paper (Gao et al., 2019) for details about XTalk.

As a typical ML-incorporated software system, our experience in XTalk can be generalized to other ML systems in terms of the development-and-maintenance process. Besides, we tackle many NLP challenges with a hybrid solution that incorporates rule- and DNN-based approaches. This particularity can provide some insights for practitioners on decision-making.

3 Software requirement management

Before we start our development of XTalk, we need to know what kinds of functionalities XTalk needs to support and subsequently make an actionable and concrete plan, which is a typical software requirement analysis process. Although software requirement management has been well studied for traditional software projects, the new characteristics of DNN-based VA have brought new difficulties to the requirement analysis. In this section, we share our practices in XTalk.

3.1 Problems in requirement management

During the lifecycle of XTalk, we identify two major problems in managing the requirements of VAs. The first problem concerns how to specify clear and actionable requirements; i.e., what kinds of NL questions our VA needs to support. The second one is about defining reasonable and achievable acceptance criteria for the requirements.

Problem 1 Articulate actionable requirement specifications for VAs.

Ideally, a VA should correctly respond to all NL inputs related to its domain-specific tasks; for instance, XTalk is expected to accept all NL inputs related to data analytics. However, it is challenging

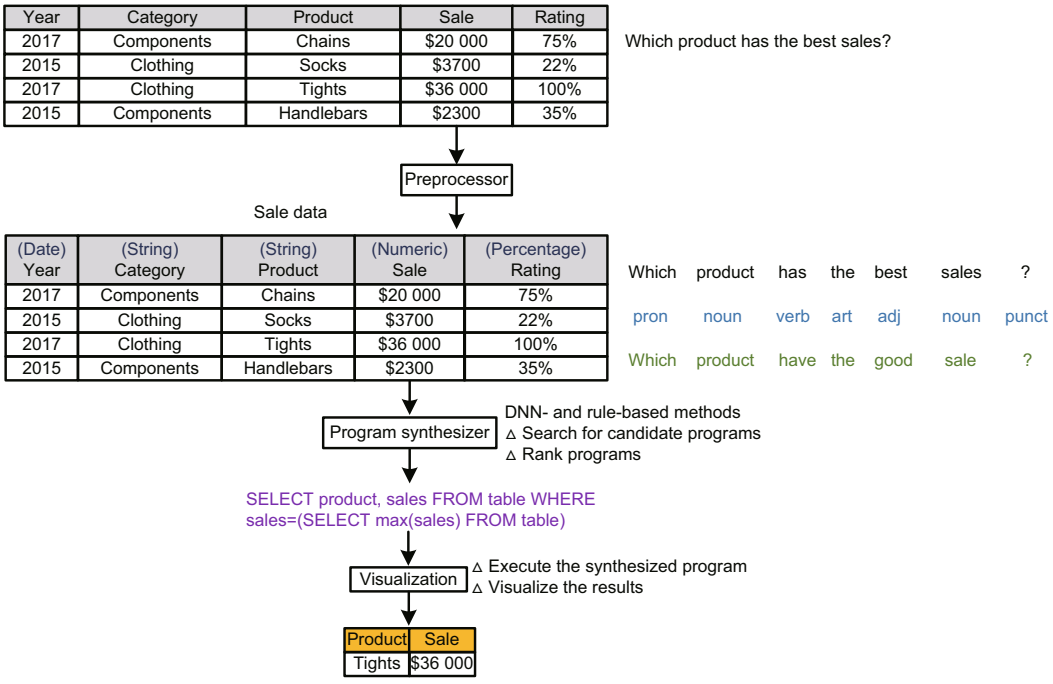

Fig. 2 Overall architecture of XTalk

to translate this ambitious goal into a series of actionable requirements that include clear and unambiguous specifications. Because of the inherent complexity and flexibility of NL, it is a non-trivial task to describe the scope of NL that our VA needs to support. For example, considering the three utterances below, "Which products have the best sales?" "Which goods sell the best?" and "best-selling product," they express the same meaning but exhibit dramatically different linguistic patterns. This huge and diverse NL input space makes it difficult to (1) figure out how people talk in the context of a VA's aimed domain and (2) specify which NL inputs need to be understood by a VA case by case. Without a clear scope, we cannot break down our goal for the VA into a series of actionable requirements for developers. The lack of actionable requirements inevitably leads to miscommunications between developers and product managers. Furthermore, product managers cannot effectively assess how far the VAs are from practical use, and consequently, many problems cannot be detected until the VA is released, making it less competitive in the market.

Problem 2 Define acceptance criteria for VAs.

Acceptance criteria are metrics that must be achieved to mark a requirement as met. Typically, defining an acceptance criterion involves three tasks: (1) metric selection; (2) metric measurement; (3) metric threshold determination. In metric selection, we need to choose a metric to quantify the development progress, such as the passing ratio of test case and accuracy. In metric measurement, we need to determine how to compute the metrics; e.g., determine which test cases are used to compute the passing ratio. However, due to NL's huge input space, it is impossible to enumerate all possible inputs to compute the metrics for a VA. As a result, selecting an adequate set of representative NL inputs for metric measurement is a critical task. However, this is challenging due to the flexibility and long tail distribution of linguistic characteristics. Finally, a metric threshold is used to indicate that a requirement is met; for example, a system should achieve a 100% passing ratio of test cases. However, defining a reasonable and achievable metric threshold is particularly challenging for VAs, because understanding the meaning of any NL remains an open artificial intelligence (AI) problem (Bender and Koller, 2020). Hence, it requires managers to make a careful trade-off between development efficiency and user satisfaction.

3.2 Practices for requirement management

In what follows, we share our practices adopted in XTalk to handle the problems discussed above.

Practice 1 Organize a VA's requirements according to its aimed domain-specific task and the linguistic patterns of NL.

Considering that a VA is expected to accept all NL inputs related to its domain-specific tasks and that these tasks' requirements are relatively easy to specify, it is natural to derive and organize the VA's requirements according to the tasks. Besides, we observe that while the NL input space is huge, users tend to express their intents for each domain-specific task in NL with a set of linguistic patterns. We can further organize the VA's requirements according to the linguistic patterns. In the following paragraphs, we present how we carry out this practice in XTalk.

The requirements of data analytics tasks are easy to derive and specify, because a lot of professional data analytics tools have been developed in the market, such as Tableau (Tableau, 2020b) and Power BI (Microsoft, 2020). To gather the requirements of data analytics, we first learn the functionalities provided by professional data analytics tools. Then, we conduct a user study with experienced data analysts to understand their commonly used functionalities; for instance, we can learn that basic analysis functions such as filter, percentage, and aggregation are frequently used in data analytics.

As a VA for data analytics, XTalk is expected to accept diverse kinds of NL inputs that express the use of these analysis functions and their combinations. Therefore, to specify XTalk's requirements, we conduct a formative study to understand how users express their intentions for using these analysis functions. Concretely, in the formative study, we recruit experienced data analysts to express their intentions in NL for a given analysis function or a combination of analysis functions. Then, we summarize the frequently used linguistic patterns from the collected NL inputs, and group them according to the linguistic patterns. Table 1 presents a sub-set of illustrative examples for linguistic patterns that we have summarized. In practice, voice shift and morphological changes also need to be considered. In this way, we translate a data analytics task's requirement

Table 1 Frequently used linguistic patterns for expressing data analytics intentions

Linguistic pattern	Explanation	Example
Keyword-based expression	Phrase-level expression	Brand_A sales
Imperative expression	Utterances that express commands	Show me the sales of Brand_A
"Wh" interrogatives	Utterances that use "Wh" interrogatives	What are the sales of Brand_A?
"Yes/No" interrogatives	Utterances that expect "Yes/No" answer	Are the sales of Brand_A higher than those of Brand_B?
Formula expression	Utterances that contain symbolic conditional expressions	Show me brands with sales>$3000

into multiple clear and actionable requirements for XTalk. Each requirement is clearly specified with its corresponding analysis functions and linguistic pattern. In addition, each requirement's priority can be jointly determined by the importance of its corresponding analysis functions and the occurrence frequency of its linguistic pattern. To facilitate the requirement management, we develop a tool which will be illustrated in detail in Section 3.3.

Practice 2 Define acceptance criteria for each requirement based on its representative NL inputs.

By carrying out Practice 1, we manage to specify clear and actionable requirements for VAs. Moreover, for each requirement item, we have a set of corresponding typical NL inputs. Then, it is natural to use these NL inputs as the measurement for acceptance because they are written by experienced data analysts and are hopefully representative. In terms of the threshold, we determine it primarily considering (1) the complexity of the NL inputs and (2) the technique's feasibility. For example, in XTalk, we set up a very high threshold for those requirements that involve only a single analysis function, since their NL inputs are relatively simple and are frequently asked by users. However, for those requirements that involve complex combinations of analysis functions, we set up a relatively low threshold because even the state-of-the-art NLP techniques are still weak in reasoning over complex compositional NL inputs, and moreover, these inputs are not frequently asked by users. Finally, note that both the set of representative NL inputs and the threshold for each requirement should be continuously maintained.

3.3 SpecSpace: a requirement management tool for VAs

To facilitate requirement management for XTalk, we additionally develop a checklist, called SpecSpace. SpecSpace organizes XTalk's requirements into a two-dimensional matrix, as illustrated in Fig. 3, in which the rows are analysis functions (or their combinations) that we aim to support in XTalk and the columns are linguistic patterns (or their combinations) that we have summarized from the collected NL inputs. Each cell in SpecSpace, such as Ⓐ in Fig. 3, is a basic requirement of XTalk, and it includes some essential contents: the requirement's description, representative NL inputs, acceptance criteria, priority, development progress, and estimated cost. Additionally, multiple cells can be combined to specify a complex, compositional requirement; for example, requirement Ⓒ consists of three basic analysis functions: filtering with cell values, aggregation, and comparison. Using SpecSpace, we can keep track of XTalk's requirements on time, such as current progress and the requirements to be scheduled. Moreover, SpecSpace records the actual cost, e.g., time and computation resource, on each requirement, which is a valuable reference for cost estimation in the future. In addition to managing requirements, SpecSpace plays an essential role in the lifecycle of XTalk; for example, the evaluation of system quality is also based on SpecSpace with careful checking of the acceptance criteria for each cell.

4 Software development management

In this section, we focus on software development management, which is the process of making decisions on how to implement requirements and ensure that requirements are implemented on time and in high quality. Recently, much research effort has been devoted to studying the best practices for developing ML-based systems (Zhang TY et al., 2019), most of which are practices that address specific coding problems. By contrast, we discuss here two development management problems that we encountered in the XTalk. The first problem concerns how to

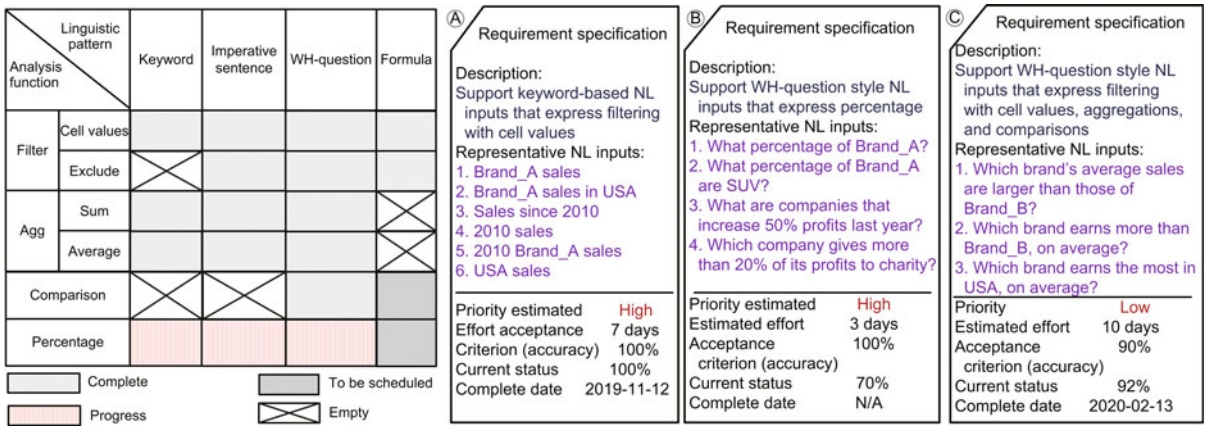

Fig. 3 An overview of SpecSpace

The matrix on the left gives an overview of XTalk's requirements, from which we can clearly learn how many requirements are under development and how many of them are waiting for scheduling (agg: aggregation). Three detailed specifications of requirements Ⓐ–Ⓒ are presented on the right. It is worth mentioning that requirement Ⓒ is a compositional requirement that involves filtering with cell values (using the average aggregation) and comparisons, as illustrated by its representative natural language (NL) inputs (SUV: sport-utility vehicle)

enjoy the benefit of ML while effectively managing the development process. The second one is about how to efficiently collect high-quality data for ML algorithms. We also share our practices for tackling the problems.

4.1 Problems in development management

Problem 3 The emergent properties of ML, e.g., data-driven and statistical nature, hinder efficient development.

It is non-trivial to meet the requirements both on time and in high quality, due to the huge uncertainty caused by ML techniques. First, given ML's weak interpretability and data-driven nature, (1) whether we can implement a requirement within the budget is uncertain and (2) deciding which ML algorithm is the best is challenging. In most times, we cannot predict how much time and data an ML algorithm will need to meet the requirement's acceptance criteria. Most things can be settled only after rounds of experiments, i.e., sometimes days or weeks of model training. Even if a model can achieve 95% or higher accuracy in an offline testing, it still makes unexpected mistakes in an online environment after release, which we may not be able to completely avoid before hand. Given that training large-size modern DNN models often costs days or even weeks, this process may make the development plan out of control and risky. For product managers, it becomes a big problem.

Second, in practice, the requirements are even data-driven. In XTalk, e.g., by saying "sales of Brand_A," the user actually means "the whole sales of Brand_A." Before user study, we are not aware of that. This kind of requirement change constantly happens in our development and forces us to re-conduct the development (e.g., collect training data again, fine-tune hyper-parameters again, and re-design features for ML algorithms). This uncertainty can make the development process highly iterative and repetitive.

Problem 4 Efficiently collect data for VA projects under the constraints of domain knowledge and data privacy regulations.

It is well known that high-quality data are essential for ML algorithms. However, collecting high-quality NL data for developing VAs is rather expensive, especially in those scenarios that require expertise. Take XTalk as an example. Due to the program synthesis formulation in XTalk, labeling the data requires expertise in data analytics and programming. Specifically, labelers are required to raise reasonable data analysis questions against tables and to write corresponding programs that express the same intentions as the questions, making it a high-cost proposition to collect data on a large scale. What is worse, constrained by privacy policies such as general data protection regulation (GDPR) (Voigt and von dem Bussche, 2017), the real user inputs are inaccessible at all.

After data collection, it is crucial to validate whether the labeled data are of high quality and free of bugs; for instance, in XTalk, we need to validate whether all the programs are written in the style that is required and whether the programs fully express the meaning of their corresponding NL questions. Such a validation process, however, can be tedious and time-consuming because it relies mainly on manual inspection.

4.2 Practices for development management

To tackle the problems, we explore three practices which have been adopted in the development process of XTalk.

Practice 3 Develop VAs using a hybrid of DNN- and rule-based methods.

To enjoy the benefit of advanced DNN techniques and make the development process manageable, we find it important to develop VAs using a hybrid of DNN- and rule-based methods. As mentioned above, the intrinsic reason for why DNN makes the development process hard to manage is its weak interpretability and data-driven nature. Fortunately, these weak points of DNN can be partially compensated for by the rule-based methods, since rules are deterministic and interpretable. Moreover, compared to DNN techniques, the efforts on refactoring rules to implement potentially changed requirements are often more predictable, which makes the development management more controllable. Hence, we choose to develop XTalk with a hybrid of DNN- and rule-based methods, as illustrated in Section 2. Each requirement can be implemented using either DNN techniques or rules. Specifically, to implement a requirement using DNN techniques, we typically collect additional training and test data based on the requirement's specification to re-train DNN algorithms and modify the algorithms (e.g., tuning hyper-parameters and adding features), if necessary. Alternatively, to implement a requirement using rules, we modify the set of rules in XTalk according to the specification. Based on our hands-on experience, using a hybrid method does make the development process much more manageable. For example, for an urgent but small requirement update, we can quickly implement it by refactoring the rules in XTalk's program synthesizer module, while updating a DNN model may be very costly.

However, there is no free lunch. Developing VAs using a hybrid method requires careful trade-offs between DNN techniques and rules. Otherwise, it is highly likely that the hybrid method degrades to a pure rule-based method eventually, especially under the pressure of fast delivery. To help other practitioners make the trade-off, we share our practice adopted in XTalk. First of all, when a requirement is clearly specified and we have enough time on our hands, using DNN techniques to implement the requirement is our first choice. However, for a small or urgent requirement update, we often do not take the risk to implement it with DNN techniques. Instead, we implement it using rules. Moreover, when we do not have a systematic understanding of a requirement (e.g., we cannot enumerate how the users express their needs), we prefer to implement this requirement with rules. Later, as more NL inputs are collected for the requirement, we will deepen the understanding for it, and its implementation is migrated from rules to DNN techniques. Last but not the least, we regularly migrate rule-based implementations to DNN techniques, especially for those urgent requirements implemented with rules.

Practice 4 Paraphrase the automatically generated NL questions with synthesized oracles.

Initially, we recruit labelers and ask them to raise reasonable data analytics questions against a table freely and to write the corresponding programs (i.e., SQL-like queries). However, this approach is ineffective since we find that it is challenging for labelers to raise diverse questions about data analytics on a table. Based on our observation, most labelers can raise only several NL questions on one table and often stick to a small set of linguistic forms. Meanwhile, hiring these labelers who are proficient in programming is very expensive.

To this end, we propose a paraphrasing approach for data collection, which does not require labelers to be experts in either data analytics or programming. Specifically, given a requirement of XTalk and a table, we automatically synthesize a lot of programs according to the requirement's corresponding analysis functions. Consider requirement Ⓐ shown in Fig. 3 which requires us to support keyword-based NL inputs that express the use of filters with cell values. To collect data for this requirement, we first synthesize thousands of programs containing the filter function, e.g., "SELECT sales WHERE brand = Brand_A." Then, we prune

unreasonable programs and translate each remaining program to an NL question using off-the-shelf tools (Yao et al., 2019). Next, labelers are requested to rewrite the generated NL questions according to the requirement's corresponding linguistic pattern. For the program above, we generate a corresponding NL question "Show me sales of Brand_A," and ask labelers to rewrite the question into a keyword-based expression such as "Brand_A sales." With this approach, we can collect a large amount of data with a high linguistic diversity at a low cost. Note that the second approach is not free, because the space of program is unbounded, and it requires some manual effort to prune unreasonable programs.

Practice 5 Validate data with a semi-automated, human-computer interactive approach.

To relieve the burden of data validation, we develop a novel human-computer interactive approach, inspired by the active learning technique (Bonwell and Eison, 1991). The intuition of the approach is that we can bootstrap an ML algorithm with a small amount of training data and use the resulting ML model to help validate the remaining data.

Algorithm 1 outlines the approach's workflow. Specifically, given a set of collected examples $\mathcal{D} = \{e_0, e_1 \cdots, e_M\}$ including M collected examples, we first randomly partition them into N different sets $\{\mathcal{D}_0, \mathcal{D}_1, \cdots, \mathcal{D}_{N-1}\}$ (line 1). We then manually validate all examples in the first set $\mathcal{D}_0$ and use the validated examples to train an ML model (lines 3–4). The resulting ML model is then evaluated on examples in the second set $\mathcal{D}_1$. We consider an example potentially invalid if the ML model fails to generate the same outputs on it as its original label. Afterward, we manually validate all the potential invalid examples and train another ML model on $\{\mathcal{D}_0, \mathcal{D}_1\}$ (lines 7–9). The last two steps iterate until all the examples in $\mathcal{D}$ are validated. Intuitively, as more examples are validated and used in training, the ML model can generate correct outputs for more examples, and the number of examples that require manual validation decreases. In XTalk, with this approach, we manage to reduce the number of examples that require manual validation by more than 50%.

Algorithm 1 Human-computer interactive data validation approach

Input: a set of labeled examples to be validated $\mathcal{D} = \{e_0, e_1, \cdots, e_M\}$ and the number of partitions N
Output: a set of validated examples $\mathcal{D}_\mathrm{V}$
1: S = partition($\mathcal{D}$, N), where $S = \{\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{N-1}\}$;
2: $\mathcal{D}_0$ = pop(S);
3: $\mathcal{D}_\mathrm{V}$ = validate($\mathcal{D}_0$);
4: Model = train($\mathcal{D}_\mathrm{V}$);
5: **while** S is not empty **do**
6: $\mathcal{D}_i$ = pop(S);
7: $\mathcal{D}_\mathrm{e}$ = evaluate(Model, $\mathcal{D}_i$);
8: $\mathcal{D}_\mathrm{V}$ = $\mathcal{D}_\mathrm{V}$|ManualValidate($\mathcal{D}_\mathrm{e}$), where "|" represents the intersection operation;
9: Model = train($\mathcal{D}_\mathrm{V}$);
10: **end while**
11: **return** $\mathcal{D}_\mathrm{V}$;

5 Software quality management

Software quality management is a process to manage the quality of a software program in such a way to best meet the quality standards expected by both users and developers. This process involves mainly system testing and maintenance. The statistical nature and weak interpretability of ML raise additional problems for software testing (Zhang JM et al., 2022), but most of them remain unsolved. As a type of ML-based system, VAs pose all the testing problems of ML-based systems. Since the testing problems of ML-based systems have been widely studied in the literature (Marijan et al., 2019), in the following subsections, we briefly introduce them and primarily focus on our practices for managing XTalk's quality.

5.1 Problems in quality management

Problem 5 Ensure the product readiness of a VA, including evaluation, tracking, and improvement of its quality, in the absence of test oracle and with limited interpretability of the system.

Test case generation is difficult when evaluating the system accuracy and detecting bugs in the system. Because of the complexity of NL, test case generation for VAs is usually conducted intuitively and is thus much inefficient. Meanwhile, since the behaviors of VAs are mostly learned from data rather than specified in code, it becomes extremely hard to identify the root cause of a bug. Regression bug, such as the new version fails on a set of previous successful cases, is another headache for quality management. These regression bugs are often uninterpretable, which really threatens the effective management of system quality.

5.2 Practices for quality management

Practice 6 Rigorously test the system on diverse data.

Since white-box testing techniques can be costly due to the missing test oracles (Marijan et al., 2019), we use mainly black-box testing techniques in XTalk. Specifically, whenever requirements are implemented, we perform four black-box testing activities to ensure that the required level of XTalk's quality is achieved, as depicted in Fig. 4.

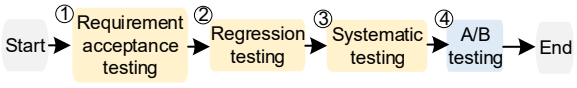

Fig. 4 Testing workflow of XTalk
Testing activities ①–③ are conducted offline, while activity ④ is conducted online

First of all, we conduct the requirement acceptance testing, aiming to measure whether XTalk meets the acceptance criteria defined for each newly implemented requirement. As we have discussed in Practice 1, the representative NL inputs we have collected for each requirement play an essential role in specifying the requirement and defining acceptance criteria. Hence, we think that these inputs are adequate to test a requirement's implementation. Then, we conduct regression testing to ensure that the previously implemented requirements still perform after a change. Specifically, we test whether their acceptance criteria are still met by XTalk. Due to the statistical nature of ML, we observe that regression frequently occurs in practice, and hence, it is important to conduct regression testing. With SpecSpace, both the requirement acceptance testing and regression testing are conducted automatically, and based on SpecSpace, we can generate bug reports for those requirements for which XTalk fails to meet the acceptance criteria.

In the previous two testing activities, we primarily test XTalk according to its requirements. However, doing so is by no means enough because real users do not have a clear scope for XTalk and they can raise arbitrary questions. Therefore, to expose as many bugs as possible before release, we recruit some data analysts to test XTalk in systematic testing. Unlike the data collection discussed in Practice 4, we do not provide data analysts a specification of what data analytics tasks they need to test. Instead, we ask them to explore XTalk freely and evaluate whether XTalk can help them fulfill their daily data analytics tasks. In this process, data analysts are required to record all their inputs, including NL questions and tables, as well as their expected answers for their questions. However, for those inputs that XTalk fails to return correct answers, data analysts need to provide correct programs and write bug reports. Only when XTalk achieves higher accuracy than its previous version in systematic testing, will XTalk be allowed to be tested online. In practice, the data collected in systematic testing are not only to test XTalk, but also to refine XTalk's requirements; for example, we will add some cases to a requirement's representative NL inputs if the cases are important and even add a new requirement. At this point, we finish the offline testing.

To detect bugs after XTalk is deployed online, we further conduct A/B testing (Young, 2014). Users are split into two groups using the new and old versions of XTalk separately. Compared to systematic testing, XTalk is tested by more users in real scenarios. Moreover, since we have no access to any of the inputs of users due to the privacy policies, not to mention their expected outputs, it is impossible to evaluate XTalk's online performance as in the offline scenario. Therefore, to assess XTalk's online performance, we design an evaluation metric according to users' behaviors; for instance, if a user saves the results returned by XTalk, we consider that XTalk meets the user's requirement. When XTalk is improved over its previous version on the online evaluation metric, we increase the percentage of sample users who use the new version.

Note that since systematic testing and A/B testing are more expensive and risky, we conduct them only before release. As for requirement acceptance testing and regression testing, we frequently conduct them with SpecSpace when a requirement is implemented.

Practice 7 Design an effective bug management process (i.e., bug report generation, bug triage, and bug fixing) for VAs.

To relieve the burden of bug management for XTalk, we standardize our bug management process as follows:

1. Bug report generation. A good bug report should be easily understood and reproduced. In XTalk, we require that a bug report should contain at least five elements, namely, an NL question, a

table, the NL question's correct answer, XTalk's returned answer, and system version. Based on these elements, we can quickly reproduce a bug. Fig. 5 shows an example of a typical bug report.

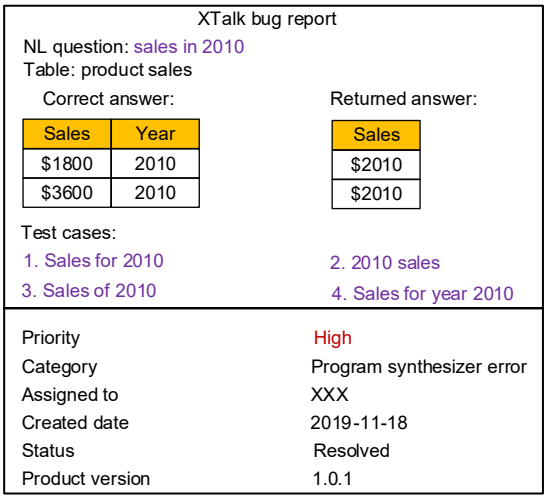

Fig. 5 An example bug report of XTalk

In XTalk 1.0.1, for natural language (NL) question "sales in 2010," given the table "product sales," the expected result is filtering the sales in the year 2010, while XTalk wrongly filtered sales that were equal to 2010; it was caused by the program synthesizer module

2. Bug triage. Once a bug is reported, we try to reproduce it and debug each module of XTalk step by step to find out the root cause. Take the debugging of XTalk's parsing module as an example. We first check whether the correct program exists in the candidate results. If it exists, we then examine the input features and intermediate results of the ML model to understand the reason for not ranking the correct program first. After finding the root cause, we quantify the impact of a bug by testing XTalk with some similar variants of the original NL input. As shown in Fig. 5, three variants along with the original input serve as the test cases for bug fixing. Based on a bug's root cause and its quantified impact, we can classify the bug into a specific category, assign the severity level, and notify the corresponding developer to fix the bug. Currently, we have four levels of bug priority: P0 (bug should be immediately fixed); P1 (bug should be fixed within one week); P2 (bug should be fixed in the next version); P3 (bug is hard to explain).

3. Bug fixing. For P0 bugs, developers use rules to fix them quickly. Bugs in other levels of priority are encouraged to be fixed with ML techniques, e.g., adding more training data or new features. Once a bug is fixed, we conduct regression testing to ensure that the implemented functions still work after a change.

6 Discussions

6.1 Lessons learned

In this subsection, we summarize the overarching lessons that we have learned at XTalk.

6.1.1 Data

It is well known that data are important for developing an ML model. However, the value of data has been rather underestimated in the software management process, except for the development phase, for a long time. We argue that data also play an essential role in project management throughout the lifecycle of VA software. Making full use of the available data will tremendously facilitate solving the problems occurring while managing a VA software project, including problems in requirement management, development management, and quality management.

6.1.2 Requirement management

We organize a VA's requirements according to its aimed domain-specific task and the linguistic patterns of NL. The number of utterances in NL is infinite, which causes difficulties in organizing clean and actionable requirements for task-oriented VAs. Through our formative study, we find that the scope of NL could be organized through its domain-specific tasks and linguistic patterns. For one thing, a VA is expected to understand all NL inputs related to its domain-specific tasks. For another, there exist a set of different linguistic patterns in NL that are able to express the tasks. Moreover, compared with NL, domain-specific tasks are relatively easy to specify. With these insights, we start from organizing NL according to the domain-specific tasks first, and further group the NL system according to its linguistic patterns for each task. In this way, we figure out the scope of NL, and thus specify clear and actionable requirements for the VA. In addition, introducing novel management tools such as SpecSpace will further help specify, prioritize, and track the requirements.

6.1.3 A good trad-off between DNN and rules

In the age of DNN, both industry and the research field tend to pursue DNN-based methods blindly. However, from our lessons, there still exist many problems in managing DNN-based software projects. To enjoy the benefit of advanced DNN techniques and make the development process manageable, we argue that it is necessary to find a good balance between DNN- and rule-based methods. By deeply understanding pros and cons of the DNN- and rule-based methods, we have confidence that we can make the best choice between DNN and rules according to the characteristics of the project requirements.

6.2 Open challenges

When managing our VA project, we encounter a host of problems that we have presented in previous sections. In the following, we summarize some open challenges remaining in management based on our experience.

6.2.1 Acceptance criteria for release

In a software project, we need to have not only the release date but also release criteria to ensure user satisfaction. Designing the acceptance criteria for release is difficult for ML-based systems. First, since incorrect model prediction is inevitable in these systems, it is important to determine what error rates are acceptable for user satisfaction. However, current evaluation metrics for ML models are not a direct reflection of user experience. For example, even though we achieve a $\geq 95\%$ accuracy with the offline test data, we are still uncertain about its behaviors in an online environment after release. Second, the incorporation of ML introduces some new factors to consider before release, such as interpretability and fairness, but how to effectively evaluate them is still an open question. To this end, some fine-grained and diverse metrics for ML models can be a great help for release and also serve as a research opportunity.

6.2.2 Planning and scheduling

ML shifts software development from a deductive process to an inductive one. As mentioned, the behavior of ML-based systems strongly depends on data and models that cannot be clearly specified a priori, so either time or cost of implementing a requirement is not easy to estimate. Under this situation, planning and scheduling of the development process and keeping the development controllable and effective to avoid work delay or unexpected model behaviors become challenges. In a way, close collaboration and constant communication become extremely important for the teams developing VAs.

6.2.3 Circumvent regression errors

ML models are expected to learn some knowledge from provided data, and more importantly, we expect the models not to forget old knowledge when learning the new ones. However, regression on model performance is currently a severe problem when iteratively developing VAs. It can happen whenever we make some changes on the ML models and it is usually uninterpretable, which makes the development less efficient. There are some existing attempts in the literature to tackle this problem, such as lifelong learning and boosting technology (Schapire, 1990; Thrun, 1998; Mason et al., 1999). However, it is still an open problem. In the software community, regression testing on ML-based systems is also a worthy research opportunity to help expose the model regression.

6.2.4 Robustness testing

Robustness measures the system resilience in the presence of perturbations, and is extremely important for user satisfaction with ML-based systems. Although there are various research works in this field, it is still hard to automatically generate a large number of test cases for practice.

For systems processing NL inputs such as VA, generating test cases for robustness is more challenging due to the boundless and discrete NL space, and it remains an open question. Thus, robustness testing needs manual effort, which is obviously insufficient. Since the current research on test case generation is primarily centered on image classification, more research efforts on NL related tasks are needed.

7 Related works

In this section, we present some related works on task-oriented VAs and software project management of ML-based systems.

7.1 Task-oriented VAs

In recent years, the capabilities and usage of VAs have expanded rapidly with new products entering the market, helping people with some domain-specific tasks. For example, many database management systems are deployed with VA, providing an NL interface for data analytics (Dhamdhere et al., 2017; Zhong et al., 2017; Sun et al., 2020), including XTalk. Two kinds of techniques are used to develop a VA system: rule-based methods driven by handcrafted logic (Oram, 2019; Facebook, 2020) and ML-based methods driven by data (Dhamdhere et al., 2017; Zhong et al., 2017; Sun et al., 2020). Rule-based VAs are relatively straightforward, but writing rules for different scenarios is very time-consuming and it is impossible to cover every possible scenario. ML-based VAs rely on ML techniques such as NLP and information retrieval (IR), which makes them more efficient than rule-based VAs. Given the popularity of VAs, the market has become fiercely competitive, but the development is still challenging due to some bottlenecks in understanding NL commands. In this study, we make a very first attempt to elicit the problems in VA project management based on our experience in XTalk.

7.2 Empirical studies on management for ML-based software projects

7.2.1 Requirement management

There are some research reports that aim to define the characteristics and challenges unique to requirement engineering (RE) for ML-based systems. Vogelsang and Borg (2019) conducted comprehensive interviews among practitioners, and found that some new requirements introduced by ML, e.g., explainability, are difficult to manage in practice. Another work (Horkoff, 2019) focused on managing non-functional requirements in ML-based systems, such as privacy, security, and testability. Despite the aforementioned works, how requirements are managed in real-world ML-based products is hardly known. To fill the gap, in this study, we share our experience on both requirement management in XTalk at Microsoft and a corresponding tool implemented by ourselves.

7.2.2 System development

Many researchers have focused on identifying the challenges of developing ML-based applications by conducting empirical studies among practitioners or open forums such as StackOverflow as well as by case studies (Arpteg et al., 2018; Schelter et al., 2018a; Amershi et al., 2019; Islam et al., 2019; Zhang TY et al., 2019). Besides, there are some works sharing good or best practices on developing ML-based systems based on practitioners' real-world experience. For example, some practitioners from Google shared their valuable insights and advice on maintaining data, code, and models in ML-based systems (Sculley et al., 2015; Breck et al., 2017). They highlighted some technical debts hidden in ML-based systems, such as boundary erosion, entanglement, and data dependencies, and presented their practices to avoid them. Another focus of research is the testing process in ML-based systems, also referred to as ML testing (Hains et al., 2018; Huang et al., 2018; Masuda et al., 2018; Zhang JM et al., 2022). A recent work (Zhang JM et al., 2022) presented a comprehensive survey of the techniques of ML testing through a literature review covering 144 papers. The survey provides a landscape of ML testing, including testing properties, testing components, testing workflow, and application scenarios.

In general, these research works provide valuable insights on developing ML-based systems effectively and efficiently, which is still an open question. In this study, we summarize some problems especially on developing VA systems and the lessons learned through our experience in XTalk.

7.2.3 Data management

There are various works and tools aimed for management of data or related sub-tasks. Polyzotis et al. (2018) explored challenges in three main areas, i.e., data understanding, data validation and cleaning, and data preparation, drawn from their experience in developing data-centric infrastructure at Google. Breck et al. (2017) designed a data validation system to continuously monitor and detect data anomalies. There are also many frameworks for sub-tasks, such as data cleaning and bug detection in data (Krishnan et al., 2016, 2017; Schelter et al., 2018b). However, many of these frameworks still require humans in the loop or heavy human labors,

which limits their effectiveness in the industry. Other trivial aspects such as data collection are also significant in practice and were hardly discussed in research. In this study, we have presented the main activities of data management in XTalk which could also be applied to other VA projects.

8 Conclusions

In this study, we described five critical problems in managing a task-oriented VA software project, including problems in requirement management, development management, and quality management. To help researchers and practitioners address the problems, we shared seven practices adopted in our project on XTalk, which is a VA for data analytics. These practices are not necessarily the best solutions to the problems, but we found them effective and useful in our project. Besides, we summarized three lessons learned at managing XTalk and four open challenges that require more research efforts.

In future work, we plan to make our requirement management tool, SpecSpace, open sourced. To improve the testing efficiency of XTalk, we plan to explore advanced natural language input generation techniques such as generative pre-trained transformer (GPT) (Radford et al., 2019) and metamorphic testing techniques (Lee et al., 2020). Besides, we would like to collaborate with research teams in the community to figure out solutions for the open challenges.

Contributors

Shuyue LI, Jiaqi GUO, Jianguang LOU, and Ting LIU designed the research. Yan GAO, Dejian YANG, and Yan XIAO developed XTalk and provided the data. Shuyue LI, Jiaqi GUO, Yan GAO, and Ting LIU drafted the paper. Yadong ZHOU helped organize the paper. Jianguang LOU and Ting LIU revised and finalized the paper.

Acknowledgements

This work is supported by Microsoft Research Asia and is done during the internships of Shuyue LI and Jiaqi GUO.

Compliance with ethics guidelines

Shuyue LI, Jiaqi GUO, Yan GAO, Jianguang LOU, Dejian YANG, Yan XIAO, Yadong ZHOU, and Ting LIU declare that they have no conflict of interest.

References

Amershi S, Begel A, Bird C, et al., 2019. Software engineering for machine learning: a case study. Proc IEEE/ACM 41st Int Conf on Software Engineering: Software Engineering in Practice, p.291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Arpteg A, Brinne B, Crnkovic-Friis L, et al., 2018. Software engineering challenges of deep learning. Proc 44th Euromicro Conf on Software Engineering and Advanced Applications, p.50-59. https://doi.org/10.1109/SEAA.2018.00018

Bender EM, Koller A, 2020. Climbing towards NLU: on meaning, form, and understanding in the age of data. Proc 58th Annual Meeting of the Association for Computational Linguistics, p.5185-5198. https://doi.org/10.18653/v1/2020.acl-main.463

Bonwell CC, Eison JA, 1991. Active Learning: Creating Excitement in the Classroom. ERIC Number ED336049. The George Washington University, Washington, USA.

Bradley AJ, 2020. Brace Yourself for an Explosion of Virtual Assistants. https://blogs.gartner.com/anthony_bradley/2020/08/10/brace-yourself-for-an-explosion-of-virtual-assistants/ [Accessed on Aug. 10, 2020].

Breck E, Cai SQ, Nielsen E, et al., 2017. The ML test score: a rubric for ML production readiness and technical debt reduction. Proc IEEE Int Conf on Big Data, p.1123-1132. https://doi.org/10.1109/BigData.2017.8258038

Campagna G, Xu SL, Moradshahi M, et al., 2019. Genie: a generator of natural language semantic parsers for virtual assistant commands. Proc 40th ACM SIGPLAN Conf on Programming Language Design and Implementation, p.394-410. https://doi.org/10.1145/3314221.3314594

Dhamdhere K, McCurley KS, Nahmias R, et al., 2017. Analyza: exploring data with conversation. Proc 22nd Int Conf on Intelligent User Interfaces, p.493-504. https://doi.org/10.1145/3025171.3025227

Facebook, 2020. Surveybot. https://surveybot.io/ [Accessed on Aug. 10, 2020].

Gao Y, Lou JG, Zhang DM, 2019. A hybrid semantic parsing approach for tabular data analysis. https://arxiv.org/abs/1910.10363

Hains G, Jakobsson A, Khmelevsky Y, 2018. Towards formal methods and software engineering for deep learning: security, safety and productivity for DL systems development. Proc Annual IEEE Int Systems Conf, p.1-5. https://doi.org/10.1109/SYSCON.2018.8369576

Horkoff J, 2019. Non-functional requirements for machine learning: challenges and new directions. Proc 27th Int Requirements Engineering Conf, p.386-391. https://doi.org/ 10.1109/RE.2019.00050

Huang XW, Kroening D, Kwiatkowska M, et al., 2018. Safety and trustworthiness of deep neural networks: a survey. https://arxiv.org/abs/1812.08342v1

Islam J, Nguyen HA, Pan R, et al., 2019. What do developers ask about ML libraries? A large-scale study using stack overflow. https://arxiv.org/abs/1906.11940

Krishnan S, Wang JN, Wu E, et al., 2016. ActiveClean: interactive data cleaning for statistical modeling. *Proc VLDB Endow*, 9(12):948-959. https://doi.org/10.14778/2994509.2994514

Krishnan S, Franklin MJ, Goldberg K, et al., 2017. BoostClean: automated error detection and repair for machine learning. https://arxiv.org/abs/1711.01299

Lee DTS, Zhou ZQ, Tse TH, 2020. Metamorphic robustness testing of Google Translate. Proc 42nd Int Conf on Software Engineering Workshops, p.388-395. https://doi.org/10.1145/3387940.3391484

Marijan D, Gotlieb A, Ahuja MK, 2019. Challenges of testing machine learning based systems. Proc IEEE Int Conf on Artificial Intelligence Testing, p.101-102. https://doi.org/10.1109/AITest.2019.00010

Mason L, Baxter J, Bartlett PL, et al., 1999. Boosting algorithms as gradient descent. Proc 12th Int Conf on Neural Information Processing Systems, p.512-518.

Masuda S, Ono K, Yasue T, et al., 2018. A survey of software quality for machine learning applications. IEEE Int Conf on Software Testing, Verification and Validation Workshops, p.279-284. https://doi.org/10.1109/ICSTW.2018.00061

Microsoft, 2020. Power BI. https://powerbi.microsoft.com/ [Accessed on Aug. 10, 2020].

Oram R, 2019. Meeting Edward: Chatbots and the Changing Face of the Hotel Guest Experience. https://blogs.oracle.com/hospitality/chatbots-and-the-changing-the-face-of-the-hotel-guest-experience [Accessed on Aug. 10, 2020].

Polyzotis N, Roy S, Whang SE, et al., 2018. Data lifecycle challenges in production machine learning: a survey. *ACM SIGMOD Rec*, 47(2):17-28. https://doi.org/10.1145/3299887.3299891

Radford A, Wu J, Child R, et al., 2019. Language Models are Unsupervised Multitask Learners. https://openai.com/blog/ [Accessed on Jan. 1, 2020].

Schapire RE, 1990. The strength of weak learnability. *Mach Learn*, 5(2):197-227. https://doi.org/10.1007/BF00116037

Schelter S, Lange D, Schmidt P, et al., 2018a. Automating large-scale data quality verification. *Proc VLDB Endow*, 11(12):1781-1794. https://doi.org/10.14778/3229863.3229867

Schelter S, Biessmann F, Januschowski T, et al., 2018b. On challenges in machine learning model management. *IEEE Data Eng Bull*, 41(4):5-15.

Sculley D, Holt G, Golovin D, et al., 2015. Hidden technical debt in machine learning systems. Proc 28th Int Conf on Neural Information Processing Systems, p.2503-2511.

Sun NY, Yang XF, Liu YF, 2020. TableQA: a large-scale Chinese text-to-SQL dataset for table-aware SQL generation. https://arxiv.org/abs/2006.06434v1

Tableau, 2020a. Ask Data. https://www.tableau.com/products/new-features/ask-data/ [Accessed on Aug. 10, 2020].

Tableau, 2020b. Tableau. https://www.tableau.com/ [Accessed on Aug. 10, 2020].

Task Virtual, 2020. TaskVirtual. https://taskvirtual.com/ [Accessed on Aug. 10, 2020].

Thrun S, 1998. Lifelong learning algorithms. In: Thrun S, Pratt L (Eds.), Learning to Learn. Kluwer Academic Publishers, Norwell, USA, p.181-209.

Vogelsang A, Borg M, 2019. Requirements engineering for machine learning: perspectives from data scientists. Proc 27th Int Requirements Engineering Conf Workshops, p.245-251. https://doi.org/10.1109/REW.2019.00050

Voigt P, von dem Bussche A, 2017. The EU General Data Protection Regulation (GDPR): a Practical Guide. Springer, Cham, Germany. https://doi.org/10.1007/978-3-319-57959-7

Wikipedia, 2021. Virtual Assistant. https://en.wikipedia.org/wiki/Virtual_assistant [Accessed on Aug. 10, 2020].

Yao ZY, Su Y, Sun H, et al., 2019. Model-based interactive semantic parsing: a unified framework and a text-to-SQL case study. Proc Conf on Empirical Methods in Natural Language Processing and the 9th Int Joint Conf on Natural Language Processing, p.5447-5458. https://doi.org/10.18653/v1/D19-1547

Young SWH, 2014. Improving library user experience with A/B testing: principles and process. *Weave J Libr User Exper*. https://doi.org/10.3998/weave.12535642.0001.101

Zhang JM, Harman M, Ma L, et al., 2022. Machine learning testing: survey, landscapes and horizons. *IEEE Trans Softw Eng*, 48(1):1-36. https://doi.org/10.1109/TSE.2019.2962027

Zhang TY, Gao CY, Ma L, et al., 2019. An empirical study of common challenges in developing deep learning applications. Proc 30th Int Symp on Software Reliability Engineering, p.104-115. https://doi.org/10.1109/ISSRE.2019.00020

Zhong V, Xiong CM, Socher R, 2017. Seq2SQL: generating structured queries from natural language using reinforcement learning. https://arxiv.org/abs/1709.00103v5