Building a Hotel Concierge Bot: an industrial case study

Behzad Golshan George Mihaila Chen Chen Jonathan Engel
Alon Halevy Yoshihiko Suhara Wang-Chiew Tan Michael Matuschek

{behzad,george,chen,jonathan,alon,yoshi,wangchiew}@megagon.ai, michael.matuschek@trustyou.net

ABSTRACT

We describe our experience in developing ConciergeBot, an industrial strength question-answering bot for hotels. The bot automatically suggests answers for information-seeking questions over an input knowledge base of facts about the hotel and its amenities. We demonstrate how ConciergeBot handles unique challenges that arise in our setting. More specifically, we show how our system trains effective models with limited training data, how it can be deployed in different hotels with almost no hotel-specific tuning, and how it manages heterogeneity in questions and data. Our experiments validate that ConciergeBot achieves high precision (78%) and good recall (71%) with as few as 1,300 questions for training purposes.

PVLDB Reference Format:
. Building a Hotel Concierge Bot: an industrial case study. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Our company recently embarked on the task of building a commercial-strength hotel concierge bot (ConciergeBot), a question answering (QA) system whose goal is to accurately answer *information-seeking* inquiries from hotel guests. These are questions that can be answered by experienced hotel staff without having to consult with the hotel reservation system or requiring further action. Questions such as *"how can we get to the hotel from the airport?"* or *"when does the breakfast start?"* are information seeking questions, while questions such as *"can we change our room?"* or *"can we have more towels delivered to our room?"* are not. The initial goal is for ConciergeBot to provide three suggested answers and a member of the hotel staff will either select one of these suggestions or provide her own (modified) answer.

At the time of the project's inception, we believed it would be relatively easy to find off-the-shelf research prototypes or industrial tools and services for finding the right answers. Surprisingly, even though the QA problem has received a great deal of attention in recent years [1, 15, 17, 5], we could not immediately apply any existing solutions or tools (e.g., Google's Dialogflow, Amazon Lex, RasaNLU) to our needs (see Section 4 for a detailed discussion). This is due to two fundamental challenges that ConciergeBot needs to deal with in our setting:

1) Adaptability. Our goal is to build a hotel concierge bot that is adaptable; it can be deployed at a different hotel without tuning models for that specific hotel as long as the information about the hotel and its amenities are provided. This requires a design with no prior commitments to features or amenities specific to individual hotels.

Another important adaptability requirement is support for multiple languages, as hotel guests who do not speak English or the local language spoken in the hotel tend to rely on QA systems more than other guests. ConciergeBot is designed to be as language-agnostic as possible so that it can be deployed in a different language with minimal effort.

2) Limited data. From our experience, the data available to train ConciergeBot is extremely sparse. There are some hotels for which there is no record of past messages between guests and the concierge. For other hotels, the available data is usually skewed toward a few types of inquiries (e.g., check-in and check-out time).

In addition to having limited data, we also face the complication that the data is collected from heterogeneous sources (i.e., different hotels). For instance, the question "*What time the burger place opens?*" would have different answers depending on the hotel. This question may even be irrelevant for hotels with no restaurants serving burgers.

Contributions. We have developed an industrial strength ConciergeBot that overcomes the above challenges. Our experiments demonstrate that ConciergeBot can successfully bootstrap using little data and improve its performance over time. More specifically, ConciergeBot achieves 78% precision and 71% recall with as little training data as 1,300 questions. It also proves to be adaptable to new settings (i.e., a 7% drop in F1-score without re-training the models).

2. CONCIERGEBOT'S ARCHITECTURE

Figure 1 illustrates ConciergeBot's architecture. Its input is a natural language question along with a knowledge base (KB) storing information about the hotel. The output is the set of relevant answers, also in natural language. ConciergeBot first consults different scorers, each relying on a different NLU technique, to score each possible response. The obtained scores are then passed to a meta scorer which outputs a final probability for each answer. The most probable answer(s) are then verbalized and returned.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

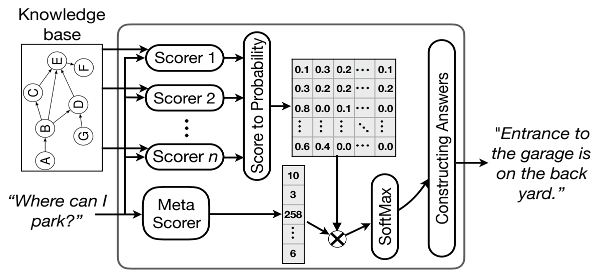


Figure 1: The architecture of ConciergeBot

2.1 Knowledge base

Since information about hotels can be organized hierarchically, we model our knowledge base about hotels accordingly (e.g., hotels have a set of facilities where each facility has an opening and closing time). The KB follows a schema which enforces a small degree of standardization, such as ensuring all information regarding the working hours of facilities is listed under the "*hours*" attribute. We use XPath notation (e.g., `/amenities/reception/hours`) to refer to specific nodes in the KB.

Each node in the tree (except for the root) is a *candidate* entity or attribute that may be inquired upon. Given a question, the goal is to find the node that contains the information the user seeks. As there will always be questions for which the relevant information is not modeled in the KB, we also use a special `none` candidate to deal with these cases. Choosing `none` as the answer is equivalent to refraining from answering the question.

2.2 Scorers

ConciergeBot runs an ensemble of QA systems or *scorers*. Each scorer processes the input question and outputs a score for each node in the KB. The node with the highest score is considered the most relevant answer. Our architecture is extensible to any number of scorers. We deploy three scorers in ConciergeBot, each with unique strengths (see Table 1) that contribute to good overall performance.

Semantic Role Labeling (SRL). This scorer leverages a state-of-the-art semantic role labeling framework called FrameIt [6]. We train SRL models over a hotel messaging corpus and build SRL frames for a set of common intents, such as inquiries about hotel facility locations, hours and charges, room amenities, and check-in/checkout times. The scorer then uses the detected frames to score the nodes in the KB. For instance, for the question *"where are the tennis courts?"*, the SRL model detects the `HotelFacilityLocation` frame and further extracts *"tennis courts"* as a filler for the `FacilityName` slot of the frame. Finally, each node in the KB is assigned a score based on how well it matches the information in the detected frame.

The SRL models are effective at dealing with the nuances in how questions are expressed, and are robust to the changes in the KB. However, the intents for the SRL models have to be known a priori and building those models requires training data, which is largely available only for frequent intents. Our trained models cover 55% of the questions. Hence, we complement the SRL scorer with other scorers that can handle infrequent intents.

Question Similarity (Q-Sim). The Q-Sim scorer compares the input question with previously answered questions for which the associated KB node is known. For each node x, the Q-Sim maintains a set of past questions Q_x which seek the information in node x. Given a question, the score of each node x in the KB is computed as the maximum similarity between the input question and the questions in Q_x. We evaluated several methods for computing question similarity and we settled on cosine similarity of FastText [7] sentence embeddings in our implementation. Note that the performance of Q-Sim naturally improves over time, as more question-path pairs are collected through relevance feedback. This scorer is easily adaptable to other languages as pre-trained FastText models are available for 294 languages.

Lexicon-based Parser (LexParse). LexParse identifies the entity and the attribute (if any) in the question using a domain-specific lexicon. For example, the `Check-in` entity is associated with phrases such as *"check in"* and *"arrive"* in the lexicon. Similarly, phrases *"how late"* and *"what time"* are associated with the `hours` attribute. Using the lexicon, each candidate node in the KB is assigned both an *entity match score* and an *attribute match score*. Additionally, for each hotel, we automatically extend the lexicon to include the names that appear in the KB. For example, for the question *"what time does The Grill open?"*, the node `/amenities/restaurants/[name='The Grill']/hours` receives a high score because "The Grill" matches the `name` predicate. This scorer works well for common inquiries for which the linguistic domain is curated by domain experts. It can support other languages as long as a lexicon in the desired language is provided.

Finally, we discuss how the special node `none` is scored by the above scorers. With the exception of Q-Sim, all scorers score `none` with value 1 if no significant score is assigned to other nodes in the KB and 0 otherwise. Therefore, the `none` score is not comparable with other nodes' scores. The Q-Sim scorer, however, can directly score `none` in the same way as the other nodes by maintaining a list of past questions with `none` as their label.

2.3 Score to Probability

Since the scorers use different approaches, the scores they calculate are not comparable. The *score to probability* (S2P) component alleviates this problem by converting the scores produced by each scorer into probabilities. As we expect ConciergeBot to be adaptable to new KBs, the method for converting scores to probabilities should be independent of the actual candidate nodes in the input KB. To achieve this, the S2P computes the probability of each node x in the KB using the scorer S_i as:

$$P_i(x) = \frac{e^{\alpha_i S_i(x)}}{e^{\beta_i S_i(\texttt{none})} + \sum_{x' \in \text{KB}} e^{\alpha_i S_i(x')}} \tag{1}$$

Note that none of the parameters depend on node x. In fact, these parameters only determine how the results of each scorer should be normalized regardless of what and how many nodes are being scored. This enables us to use the same set of parameters over different KBs. Note that the score for node `none` is scaled using a different parameter (β_i) as its score does not follow the same distribution as the other nodes in the KB. During the training phase, the S2P component learns the α_i and β_i parameters for each scorer by optimizing the logarithmic loss.

2.4 Meta Scorer

The *Meta Scorer* is the module that orchestrates the ensemble. Given the input question, it decides how to combine

	Strengths	Example that works well
SRL	robust to linguistic variations; highly portable	*Where are the tennis courts?*
Q-Sim	utilizes past data; improves over time; adaptable to other languages	*Where can we leave our stuff?*
LexParse	utilizes domain knowledge; language agnostic	*What time does The Grill open?*

Table 1: An overview of scorers in ConciergeBot.

the probabilities obtained from each scorer. The meta scorer needs to be KB agnostic to ensure that the system is adaptable. To achieve this, the meta scorer computes a weight w_i for each of the scorers S_i based on the input question which we describe shortly. The final probability for each node x is then computed as follows:

$$P(x) = \text{softmax}(\sum_{i=1}^{n} P_i(x) * w_i) \qquad (2)$$

Note that n refers to the number of scorers (i.e., 3 in our case). The weights w_i are computed using a dense feed-forward neural network with one hidden layer. The network takes as input the embedding of the input question (using FastText) which is a vector of size 700. The hidden layer consists of 100 nodes and the output layer consists of 3 nodes corresponding to the three available scorers. All activation functions are ReLU. The objective is to minimize the logarithmic loss of the probabilities obtained from Equation 2.

As mentioned earlier, many input questions may not be information-seeking. To deal with this data imbalance problem we assign a different sample weight η to questions with `none` label during the training phase.

2.5 Constructing Answers

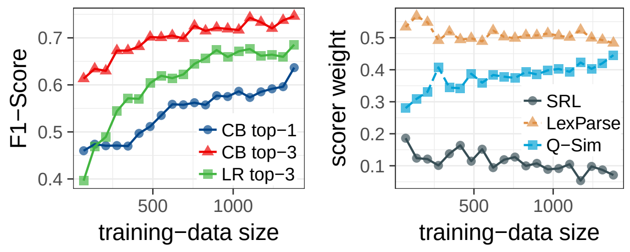
After the final probabilities P for each node in the KB (as well as `none`) are computed, this module sorts and selects the top-k probable nodes. We design it so that any node that has a probability smaller than `none` is ignored. In other words, the system prefers not answering over suggesting a probably wrong answer. Due to this pruning, our system often returns one candidate node even when we allow for 3 candidates to be returned. Finally, we assume each node in the KB also stores a message that verbalizes the information stored therein, which is usually provided by the hotel. While Natural Language Generation would be an option here, our clients preferred having carefully curated answers for the guests. Hence, by design, a returned node has an associated answer in natural language.

3. EXPERIMENTS

Here, we show that ConciergeBot achieves a good performance (F1-score of 0.75) using as little as 1,300 training examples over a KB with more than 300 nodes. Moreover, we show that changes in the KB only drop the overall F1-score by 7%. We describe our experimental setup next.

Training data and KB construction. Our training data consists of past conversations between guests and hotel concierge staff that are obtained from partner hotels which have been using our messaging platform for communication. Since many guest messages such as arrival updates or thank you notes are irrelevant, we applied a set of heuristics[1] to filter for actual questions. To further increase data quality we also removed malformed sentences or questions that are part of an ongoing conversation (e.g., *what else?*). We were left with about 1,700 questions after this step.

[1]For instance, selecting messages with a ? mark or sentences with `BARQ` or `SQ` tags in their constituency parse trees.



(a) ConciergeBot's F1-score (b) Average scorers' weight

Figure 2: ConciergeBot's behavior with different amounts of training data

Next, we developed a unified KB by adding the requested information for each question and labeling it accordingly. For questions that are too detailed or rare, we omit them from the KB and label them as `none` (e.g., *"does the door get locked automatically?"*). Through refining and extending the KB while annotating, we have roughly 300 unique nodes in the KB, which now constitutes an exhaustive and detailed model of the services and amenities offered by most hotels.

Evaluation metrics. We randomly selected 20% of our labeled data (i.e., 345 questions) as our test set for evaluation. We report precision, recall and the F1-score at top-k results achieved by ConciergeBot in our experiments.

Performance with limited data. Figure 2a shows the performance of ConciergeBot (CB for short) given different amounts of training data. By tuning parameter η (i.e., the weight associated with the `none` label), we can adjust the trade-off between precision and recall. We use $\eta = 0.35$ in our experiments as it yields the best F1-score. Figure 2a shows that the meta scorer notably boosts the performance with just a few hundred training examples. In fact, CB's top-3 performance is already at 0.67 F1-score given only 250 examples and it improves rapidly with more training data. For instance, we observe more than 34% improvement in the F1-score for CB's top-1 answer using only 1,000 more data points. Finally, Figure 2a also shows the top-3 performance of a *Logistic Regression* (LR) baseline trained over the FastText embeddings of questions as features. We omitted the LR top-1 results as they were not competitive. To obtain strong LR performance, we simplified its task by limiting its KB to only 77 nodes for which training examples were available (CB handles 300 nodes in comparison). Furthermore, LR ignores the adaptability requirement. Despite these simplifications, CB still outperforms LR by achieving 10% improvement in F1-score.

Bootstrapping. Recall that each scorer has its own strengths. In particular, Q-Sim is a data-driven approach which improves over time as more answers are provided via ConciergeBot to questions from guests. Figure 2b confirms this trend where the meta scorer relies more and more on Q-Sim over time. The x-axis shows how much data is provided to the system, and the y-axis demonstrates the average weight that the meta scorer assigns to each of the three components for

	F1@3 (@1)	**P@3** (@1)	**R@3** (@1)
No data	0.65 (*0.57*)	0.64 (*0.56*)	0.67 (*0.58*)
Past data	0.70 (*0.53*)	0.69 (*0.53*)	0.70 (*0.54*)

Table 2: Performance over unseen nodes in the KB.

the question in the test set. Note that the weights are normalized so that they sum up to 1.

Measuring adaptability. We now evaluate how well the ConciergeBot performs on inquiries that were not observed as part of the training data. This experiment will inform us on how well we can adapt to a different client hotel where the KB is likely to be different.

We randomly omit 20% of nodes in the KB along with their associated questions from the training set and train ConciergeBot on the remaining data. We evaluate the system using the questions that were omitted. We repeated the experiment for 10 trials. The average size of the evaluation set (per trial) was roughly 110 questions. Table 2 shows the average performance achieved by ConciergeBot over 10 trials. Note that @1 and @3 notations refer to the performance by selecting the top-1 and top-3 candidate answers respectively. The first row corresponds to the case where no data is available over the nodes in the new KB, while the second row shows the results when Q-sim has access to set of past question over the new KB[2]. We can observe that we obtain a F1-score of 0.65 with access to no hotel-specific data and no adjustments to the model. The obtained F1-score is 0.70 when past data is available which is only a 7% drop compared to the 0.75 F1-score achieved in Figure 2a.

4. RELATED WORK

QA systems can be grouped based on the information sources they rely on [11]. Most QA systems over knowledge-bases process the input questions at two levels: the *lexical* and the *compositional* level [8]. At the lexical level, the systems find a mapping between phrases in the question and the entities in the knowledge base. At the compositional level, the systems interpret the semantics of the question using the discovered entities and relations.

At the lexical level, most restricted-domain QA systems either rely on manually curated lexicons for their specific domain [16, 2], use fuzzy string matching techniques [14, 13, 4], or consult lexical databases such as WordNet or collections of known named entities [15, 10]. Although the scorers in ConciergeBot use similar techniques, our setup has its own unique challenges since the solution needs to be adaptable and deal with heterogeneous data. Open-domain QA systems, on the other hand, rely on the large amount of text available on the web, and build a lexicon by analyzing the alignments between the KB entities and phrases in the text [1, 17]. Obviously, we cannot use such techniques as the amount of data available in our setting is too small. Finally, existing industrial NLU tools including Dialogflow, Amazon Lex, and RasaNLU are designed to detect and serve answers for a set of predefined intents (which is not possible to enumerate in our setting). Dialogflow has recently released a beta feature to serve a KB. However, the KB is expected to be an information-dense text document similar to encyclopedia articles.

[2]The test questions are excluded from the set of past questions in a leave-one-out-cross-validation (LOOCV) manner.

There is a large body of work on analyzing questions at the compositional level [12, 3, 9]. However, we observed that there is almost no need for compositional analysis of questions in our setting. This is because almost all questions involve only a single entity in the KB, and thus a successful lexical analysis is sufficient to generate a proper response.

5. CONCLUSIONS AND FUTURE WORK

We have described the challenges we face in building an industrial strength concierge bot. We have shown that one can develop a bot that achieves good performance with a very limited amount of training data while maintaining adaptability through a powerful ensemble classification approach.

There are several directions for future work. One challenge is to go beyond information seeking questions. Another is to handle follow-up questions and multi-message threads which requires reasoning based on the context.

6. REFERENCES

[1] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[2] S. Bloehdorn, P. Cimiano, A. Duke, P. Haase, J. Heizmann, I. Thurlow, and J. Völker. Ontology-based question answering for digital libraries. ECDL'07, 2007.

[3] J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth. Driving semantic parsing from the world's response. CoNLL '10, 2010.

[4] D. Damljanovic, M. Agatonovic, and H. Cunningham. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. ESWC, 2010.

[5] D. Diefenbach, V. Lopez, K. Singh, and P. Maret. Core techniques of question answering systems over knowledge bases: A survey. *Knowl. Inf. Syst.*, 55(3):529–569, 2018.

[6] D. Iter, A. Halevy, and W.-C. Tan. Frameit: Ontology discovery for noisy user-generated text. In *EMNLP Workshop W-NUT*, pages 173–183, 2018.

[7] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. In *ECACL*, 2017.

[8] C. Lei, F. Özcan, A. Quamar, A. R. Mittal, J. Sen, D. Saha, and K. Sankaranarayanan. Ontology-based natural language query interfaces for data exploration. *IEEE Data Eng. Bull.*, 41(3):52–63, 2018.

[9] P. Liang. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408, 2013.

[10] V. Lopez, M. Pasin, and E. Motta. Aqualog: An ontology-portable question answering system for the semantic web. ESWC, 2005.

[11] A. Mishra and S. K. Jain. A survey on question answering systems with classification. *Journal of King Saud University - Computer and Information Sciences*, 28(3):345 – 361, 2016.

[12] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. IUI, 2003.

[13] V. Tablan, D. Damljanovic, and K. Bontcheva. A natural language query interface to structured information. ESWC, 2008.

[14] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based interpretation of keywords for semantic search. ISWC'07/ASWC'07, 2007.

[15] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In *WWW*, 2012.

[16] C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. NLDB, 2011.

[17] X. Yao. Lean question answering over freebase from scratch. In *ACL Demonstrations*, 2015.