PizzaPal: Conversational Pizza Ordering using a High-Density Conversational AI Platform

Antoine Raux, Yi Ma, Paul Yang, and Felicia Wong
b4.ai / botbotbotbot, Inc.
3225 Ash Street
Palo Alto, CA, USA
{antoine,yi,paul,felicia}@b4.ai

Abstract

This paper describes PizzaPal, a voice-only agent for ordering pizza, as well as the Conversational AI architecture built at b4.ai. Based on the principles of high-density conversational AI, it supports natural and flexible interactions through neural conversational language understanding, robust dialog state tracking, and hierarchical task decomposition.

1 High-Density Conversational AI

Following the recent rise to prominence of smart speakers, as well as the continuous improvement of core technologies for speech recognition and natural language understanding, voice-only interactive applications, whether in the home or in car, have attracted increasing attention and investment from the industry. Such applications generally fall under two broad categories: assistants and bots. Voice assistants, pioneered by Siri in 2010, aim at providing a broad range of information and services across many domains, primarily by leveraging natural language's evocative power, i.e. its ability to summon any intent, concept or entity at any point in a conversation. On the other hand, bots (also known as skills on Alexa and action on Google Assistant) are much narrower in scope, often providing a voice interface to a single brand, service, or API.

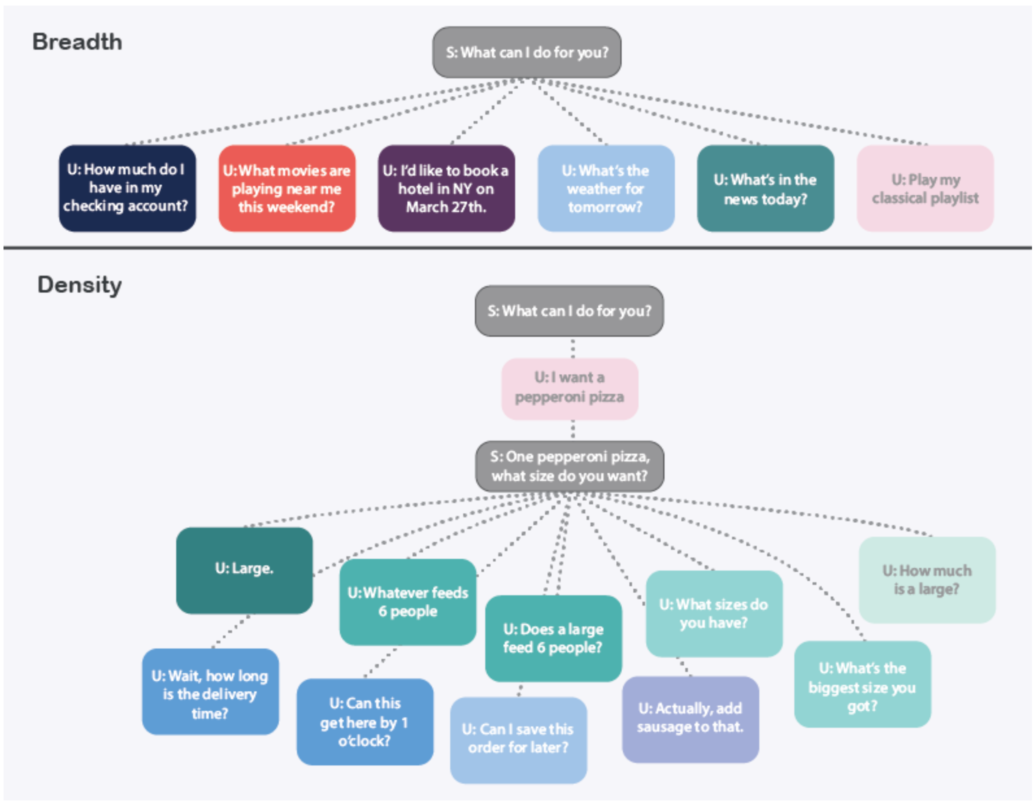
While technological progresses are undeniable, these applications have only met limited success[1], and largely fail to sustain even simple task-oriented conversations with humans. We believe this relatively poor user experience stems from the fact that neither assistants nor bots are able to cover the space of possible (or even reasonable) conversations with enough density. In other words, while a given set of user intents are recognized and supported, even small variations over those are not properly handled. There are several root causes to these limitations. While platforms such as Google's DialogFlow or Facebook's wit.ai provide a simple way of building a relatively small set of distinct intents to a large developer community, these alone cannot support truly natural, sustained, interaction. Therefore, companies (typically startups) that develop bots might not have the necessary resources or knowledge to build truly compelling conversational experiences. On the other hand, some of the largest tech companies are behind most voice assistants (Apple, Google, Amazon) have plenty of resources, financial, human, and intellectual, but they are typically focused on expanding the *breadth* of their applications, to the expense of its *density*. Figure 1 gives an example of the contrast between breadth and density.

At b4.ai, we believe that only *high-density conversational AI* can deliver the seamless, natural experience that matches users' expectations of Artificial Intelligence and leads to truly successful conversational consumer products. The following sections describe how we are tackling this challenge by first focusing on domain-specific applications in the form of Alexa skills and Google Actions.

2 The PizzaPal Conversational Ordering System

The PizzaPal system is a voice-only conversational agent that supports ordering pizza, drinks and side dishes for delivery or pickup. It runs either as an Alexa skill or a Google Assistant action and is connected to the Domino's Pizza API. While there exist other conversational agents (assistants, skills or actions) that support pizza or-

[1]According to Smith (2017), the retention rate after two weeks for Alexa skills was only 6% in September 2017.



Breadth
S: What can I do for you?
U: How much do I have in my checking account?
U: What movies are playing near me this weekend?
U: I'd like to book a hotel in NY on March 27th.
U: What's the weather for tomorrow?
U: What's in the news today?
U: Play my classical playlist
Density
S: What can I do for you?
U: I want a pepperoni pizza
S: One pepperoni pizza, what size do you want?
U: Large.
U: Whatever feeds 6 people
U: Does a large feed 6 people?
U: What sizes do you have?
U: How much is a large?
U: Wait, how long is the delivery time?
U: Can this get here by 1 o'clock?
U: Can I save this order for later?
U: Actually, add sausage to that.
U: What's the biggest size you got?

Figure 1: Leveraging Natural Conversation for Breadth vs Density

dering or similar e-commerce tasks, PizzaPal is unique in the naturalness and flexibility of its interactions, as shown in the example of Figure 2.

P: (...) What will you be having?
U: Can I get a large spinach pizza?
P: One large spinach pizza. Can I get you anything else?
U: I'll also take a pepperoni pizza.
P: One pepperoni pizza. *Do you want that in large as well?**
U: How much was the spinach pizza?
P: A large spinach pizza is \$18.49. *Let's continue with the pepperoni pizza.*** Should I make that a large as well?
U: *Do you have anything smaller?****
P: Medium is smaller. It is 12 inches. Shall I order it for you?
U: Sure.
P: Sure. Can I get you anything else?

Figure 2: Example dialogue with PizzaPal.[2]

The italicized sentences show instances of natural conversational behavior exhibited by PizzaPal:

$\ast$ PizzaPal proactively suggests sizes and items based on the dialog history when possible.

$\ast\ast$ While the system can drive the conversation to completion by asking the user direct questions, it also allows the user to take the initiative at any time.

$\ast\ast\ast$ The user can express constraints and intents freely, including using contextual expressions.

These are a few examples of features characteristic of high-density AI, which are implemented in PizzaPal. In the remainder of the paper, we will describe the b4.ai framework architecture and components that support the PizzaPal system.

3 Architecture

3.1 Overview

At a high level, the b4.ai framework is composed of three main services that run on top of a front end platform such as Amazon Alexa or Google Assistant. As illustrated in Figure 3, the front end controls application launch, voice recognition

[2]The video of this dialog is available at https://youtu.be/BQrzgJk4-yI

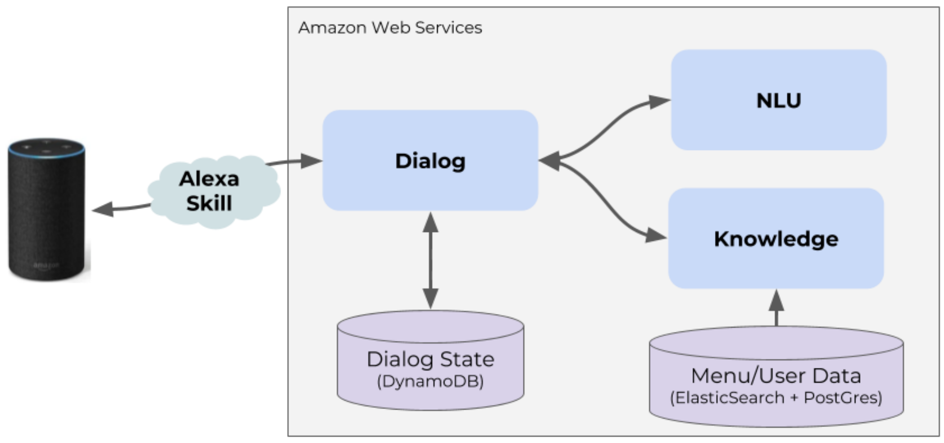


Figure 3: Overview of the b4.ai dialog framework.

and voice synthesis. The front end sends the transcription of each user input to the core dialog service, which first calls the NLU service to extract non-contextual semantic information, and interacts with the knowledge backend, before returning the bot response to the front end as a string to be spoken back to the user. All three services are implemented as REST APIs hosted on AWS EC2 instances.

3.2 Knowledge Service

The role of the Knowledge Service is to retrieve domain knowledge (e.g. menus, restaurant addresses, as well as user prefererences), help resolve references (e.g. when the user says ”add chicken to my pizza”, ”chicken” might match to different possible toppings like ”BBQ Chicken”, ”Garlic Chicken” or ”Teriyaki Chicken”), and execute transactions with 3rd party services (e.g. placing the order for delivery). Even for a single domain application such as pizza ordering, this requires access to a variety of databases, APIs and services. The Knowledge Service abstracts away these different source schemas and allows unified access from the dialog service. In general, the Knowledge Service supports a variety of queries, both structured (e.g. getting the menu of a given store given its store ID) and unstructured (e.g. finding menu items matching certain keywords and key phrases), which are performed via an API defined in terms of questions about entities and properties (e.g. ”get the available toppings for menu item X”, ”resolve ingredient named N for dish type T”). The initial implementation of the Knowledge Service relies on 3rd party REST APIs, and our own PostGres and ElasticSearch databases to access restaurant and menu information.

3.3 Natural Language Generation

The NLG service takes the semantic output generated by the Dialog service and converts it to natural language. We have implemented a simple, scalable template based approach to NLG, that allows to control the language used by the system with some amount of variation. The templates incorporate some conditionals so that entities such as menu items or ingredients can be rendered differently based on entity properties and context.

4 Natural Language Understanding

The Natural Language Understanding (NLU) service of PizzaPal converts the surface text of a user's request (taken from Alexa or Google Assistant's APIs) into a structured semantic representation that serves as the input for the Dialog Manager. The output of the NLU follows a food ordering schema that defines what a `MenuItem` consists of.

Conventional task-oriented dialog systems use intent detection and slot filling to identify user's intention and extract semantic constituents from the natural language query. This intent-slot configuration might suffice when the backend task is outlined as a database lookup operation where the extracted slots are used as constraints and the retrieved information is presented to the user by populating fixed language templates. However, it is not sufficient for building a genuinely natural conversational system that requires frequent elaborate high-density actions. Consider the following user input:

"*I want **three large pizzas**, the **Honolulu Hawaiian**, and **two with cheese and chicken**.*"

The NLU should identify three separate entities (in bold) in order to resolve the ambiguity that the user requested two kinds of pizzas with different quantities instead of three *Honolulu Hawaiian* with extra *chicken*, which is inferred by the entity model. The Subsection 4.1.1 and 4.1.2 describe the intent and slot model; with Subsection 4.1.3 explains the entity model.

4.1 Model Description

All the neural networks for our NLU service are trained and run using the Python deep learning library Keras (Chollet et al., 2015), with a TensorFlow backend (Abadi et al., 2015).

4.1.1 Intent Model

The intent model learns a Convolutional Neural Networks (CNN) with multiple filters and fixed kernel size from an $n \times k$ representation of sentence using GloVe word embeddings (Pennington et al., 2014). Random word vectors were generated for Out-of-vocabulary (OOV) words before training. In addition to the word embedding features, we are adding a few more dimensions to the word representation with additional one-hot encoding categorical features based on whether the word appears in different sets of collections of phrases that specify *ingredients*, *dish names*, or *portion sizes* etc. At the time of writing, there are 26 intents in the current model. A few example intents include `RequestItem`, `GivePortionSize`, and `AddToppings` etc.

4.1.2 Slot Model

Slot model is independent of the intent model, which allows different intents to share the same set of slots. The input for learning the slot model is identical to that for learning the intent model. The slot model learns a Bidirectional Long Short-Term Memory (LSTM) Networks with dropout and emits a slot label for each token in the word sequence. Currently, the slot model predicts a label from a total of 21 slots. Example slots include `dish_name`, `portion_size`, and `ingredient` etc.

4.1.3 Entity Model

Similar to the slot model, the entity model learns a Bidirectional LSTM Networks with dropout that emits one of the `B`, `I`, `O` labels for each token in the word sequence to indicate the boundaries of separate entities. Usually an entity corresponds to a `MenuItem` in the schema. As shown in the example user request below, each underlined chunk of text represents an entity.

"*I'd like <u>**two large hand tossed Hawaiian**</u> and a <u>**medium cheese pizza with double pepperoni**</u>.*"

In addition to the word embedding and categorical features as used for learning the intent and slot model, the input for learning the entity model additionally contains the one-hot encoding of the slot label information. The ground truth slot labels are used for training the entity model, while during prediction the output of the slot model is used as features.

4.2 Data Collection

In order to bootstrap our NLU models before we obtain real user data from the released product, we have been leveraging crowdsourcing to generate reasonable sentences for the various intents, slots and entities. We have collected both freeform sentences by giving crowdworkers a general scenario and asking them what they would say to PizzaPal, as well as paraphrases, for which we provide a target sentence with known annotation and ask workers to provide variations with the same meaning. The first approach has proven useful to uncover intents, slots and ways to formulate queries, while the paraphrase approach allows us to rapidly collect data for specific intents and scenarios.

5 Dialog Management

5.1 Overview

Once the NLU service has extracted intents, slots and entities from a user utterance, the Dialog service first updates the persistent state of the dialog based on the new input, and second decides what response to give to the user. Recent work on dialog management has focused on Deep Learning based approaches (Liu et al., 2018), showing great promise when large amounts of training dialog data are available. We also believe that such data-driven approaches are part of the solution to scale high-density AI. However, in order to bootstrap an initial system that displays our target flexibility and naturalness, we opted for an engineered solution based on data structures and algorithms inspired by computational linguistics research. Specifically, the two core components of

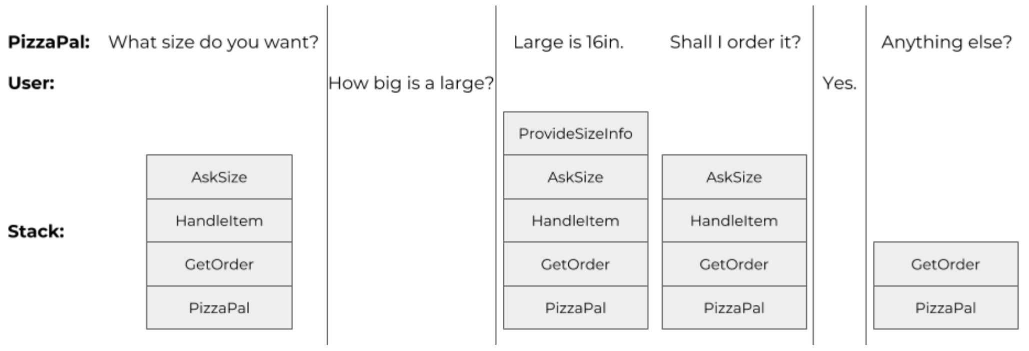


Figure 4: Example of Dialog Flow

our dialog state are:

- A *topic stack* which tracks the hierarchical topic structure of the conversation and allows switching to new topics and coming back to previous topics. This is inspired by the RavenClaw architecture (Bohus and Rudnicky, 2009). Practically speaking, the stack contains modules, each in charge of a particular subdialog.
- An *entity reference set* that can be retrieved by content as well as recency to match referring expressions provided by the user such as ”my pizza”, ”the large one”, ”the pepperoni”, etc.

In the current implementation, the domain logic within each module is implemented in Python code. We are also exploring ways of implementing modules as Deep Networks trained on example dialogs for both state tracking and decision making.

5.2 Example Runtime Flow

Initially, the stack contains the root module. After each user input, the stack is traversed from top to bottom and each module attempts to interpret the NLU according to its own context to update the dialog state. As soon as a module does so, the iteration stops and the decision making phase starts, where the module at the top of the stack can either output a prompt to the user, push another module, or pop itself from the stack, indicating that it has completed its intended subtask. For example, one module might be in charge of handling the specification of one item from the menu (`HandleItem`). When executing this module, it might decide (e.g. based on information from the backend about the given item) that it needs to enquire about the size of the item, and push another module specialized in this subtask `AskSize`. If, instead of answering the question by providing a size, the user asks a question (e.g. ”How big is a large?”), the corresponding module is pushed on top of the stack and handles the question (`ProvideSizeInfo`). Once it is done (presumably by providing the answer), `ProvideSizeInfo` pops itself from the stack and `AskSize` is back on top and tries again to obtain a size from the user. Figure 4 shows a simplified example of a dialog flow and the evolution of the stack.

Modules operate as asynchronous functions that perform a task. Once a module is completed (which could involve several turns of interactions with the user), it returns the information that it was able to obtain from the user to its calling module via callback function. This asynchronous mechanism allows the system to both lead the conversation to complete the task, while also leaving the option of the user to switch topics and ask questions, without losing track of the main task.

6 Challenges

The ambitious goals of high-density AI outlined in section 1 raise significant challenges on all components of the system. First, of course, are speech recognition errors, which, even given the high quality of ASR provided by today's voice platforms, are still prominent for certain idiosyncratic words or phrases. Since we do not control the ASR module, there is little we can do here, though we have observed that training the NLU on properly annotated ASR output help resolves the most common issues (e.g. ”I want two pizzas” misrecognized as ”I want to pizzas”).

Even when ASR faithfully transcribes the utterances, some nuances that are essential to understanding the user intent are sometimes minute such as the difference between ”I want a pepperoni pizza” and ”I just want a pepperoni pizza” which, in some contexts, means that the user wants to remove all other items from the order. This type of small but crucial distinctions, characteristic of high-density AI, point to the limitations of the traditional intent/slot approach where every nuance must be captured by a different intent. In addition, while it is more practical to implement NLU as independent of context (and leave the contextual interpretation to the Dialog service), the strong influence of context on interpretation makes NLU labeling (both by human annotators, particularly when crowdsourcing the task, and by the trained DNNs) a challenging task. For these reasons, we are exploring new approaches to conversational NLU that avoid these pitfalls by integrating Dialog and NLU more tightly.

Finally, we found that project and product management tasks for high-density AI, while critical to the development of a robust and useful product, present some significant challenges too. Because dialog flows are never rigidly defined and the user can always say anything at any point of the conversation, representing different features of the agent (e.g. ”supporting crust customization”) for purposes of communication between product, engineering and QA teams is a non trivial problem. Similarly, traditional metrics used for tracking progress toward milestones and product releases often fail to capture the seemingly infinite number of ways users can interact with each feature. We believe that new metrics, tools and processes appropriate to high-density AI systems are a critical requirement toward the development of large scale successful conversational products and we are actively working on building them.

7 Conclusion

In this paper, we present PizzaPal, a pizza ordering bot that customers can interact with through Amazon Alexa and Google Assistant using multi-turn dialogs with natural language. It is based on a proprietary dialog framework developed by b4.ai and is the first implementation of High-Density Conversational AI (#highdensityai) as a commercially viable product.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems.

Dan Bohus and Alexander I. Rudnicky. 2009. The ravenclaw dialog management framework: Architecture and systems. *Comput. Speech Lang.*, 23(3):332–361.

François Chollet et al. 2015. Keras. `https://keras.io`.

Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. 2018. Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems. In *NAACL*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Jessica Smith. 2017. The Voice Apps Report. Technical report, BI Intelligence.