
BoilerBot: A Reliable Task-Oriented Chatbot Enhanced with Large Language Models

Yifei Hu¹ Jinen Setpal^{2,*} Damin Zhang^{1,*} Jacob Zietek²

Jack Lambert³ Rey (Alex) Gonzalez¹ Julia Rayz¹

¹Computer and Information Technology, ²Computer Science, ³Sociology
{hu381, jsetpal, zhan4060, jzietek, lamberjb, gonza736, jtaylor1}@purdue.edu

*: Equal Contribution

Abstract

This paper outlines the design and deployment of BoilerBot: a task-oriented multimodal conversational agent developed for the Alexa Prize TaskBot 2 competition. BoilerBot features flexible response generation, leveraging Large Language Models (LLMs) to enable adaptable user experiences. We discuss our novel contributions towards advancing state-of-the-art task-oriented conversational agents, highlighting user-facing challenges and proposing fault-tolerant, iterative solutions for carefully guided workflows that enable Alexa users to maximize BoilerBot’s functionality. An open source re-implementation of BoilerBot is available at <https://github.com/jinensetpal/boilerbot/> as a git repository.

1 Introduction

The first iteration of the Alexa Prize TaskBot Challenge [1] laid a strong benchmark for cutting-edge approaches to conversational task resolution while shedding light on factors that impacted user experience. In conducting literature review, we identified significant rigidity in the user experiences designed by previous contestants. Consequently, our approach to the Alexa Prize TaskBot Challenge 2 [2] emphasised user-level adaptability and engagement in dialogue.

Our objective in developing BoilerBot was not only to guide users through a task, but also to enhance their skill in the underlying techniques required to complete the task.

1.1 Overview of the Challenges

Building task-oriented conversational chatbots that interact with users in natural language poses the following challenges:

Intent Classification is the process of mapping a user’s utterance to one of the taskbot’s functions. While this can be considered a classic text classification task, the informal language used by the users and the potential errors from the speech recognition model create various edge cases that necessitate developing novel, noise resilient solutions.

Query Parsing, under the scope of the Taskbot Challenge, entails an accurate interpretation of user utterances in natural language into structured data. The accuracy of parsed data directly impacts the relevance of the retrieved data from external Search APIs.

Fluid User Experiences for multimodal interfaces with a unified end goal present a significant design challenge for conversational chatbots. Communicating complicated visual instruction sets using only

the Voice User Interface (VUI) and structuring visual information using the Graphical User Interface (GUI) in a manner that is instantly interpretable accounts for a large part of this challenge.

Latency Minimization is a direct contributor towards user satisfaction. As shown by [4]: unresponsive, latent applications result in lost users. This is especially applicable to Alexa since it is a conversational agent, and presents a significant challenge owing to the extensive computation required to run the large language models integrated within the inference pipeline.

Sections 3 to 6 provide an in-depth analysis of each challenge and our proposed solutions.

1.2 Large Language Model Usage

The rise of Large Language Model (LLM) research has been remarkable since the advent of Large Language Model Meta AI (LLaMA) [25]. Although Large language models have come to underpin a vast majority of chatbots, they present a distinct challenge when applied to TaskBots. Unlike general-purpose chatbots, TaskBots are intended for specialized domains, introducing out-of-domain inference. Effectively harnessing the strengths of large language models – such as reasoning and generalization abilities – requires careful domain adaptation. Ensuring that BoilerBot’s utilized models retain in-distribution performance while not hallucinating remained our primary objective through this competition.

Auto-regressive LLMs are susceptible to hallucination, generating nonsensical or unfaithful text based on the input [16]. Malformed instructions can be risky: inapplicable substitutions or improper portions during cooking tutorials can sharply and irrevocably ruin the given recipe. Therefore, despite their proficiency as general-purpose responders, we exclusively employ LLMs for non-factual tasks, redirecting factual queries to specialized APIs or knowledge bases similar to [19].

Accordingly, we integrate LLMs into the following downstream tasks: Intent Classification (Section 3), WikiHow and Recipe Query Parsing (Section 4), Factual Question Response (part of Section 5), and General Chit-Chat (part of Section 5).

For each of our LLM-based functions, we also propose alternative solutions as fallback mechanisms due to both: our limited serving capability (Section 6.1), and edge cases triggering unexpected behaviors from LLMs.

2 Architecture

BoilerBot’s architecture is designed with the following guiding principles:

Regression-Tested Deployment: Despite rigorous automated testing, under-performing bot versions are not uncommon. We take a containerized approach towards deployment to ensure regression-free performance, with immediate rollbacks in case of failure or untested edge cases in order to minimize rating impacts and to prevent divergence between local and deployment runtimes during development.

Modularity: Adding, removing, and customizing modules is a constant process throughout the length of the competition. Any modifications to a specific module should not have cascading effects on downstream modules.

Cost Effectiveness: Compute-accelerated remote modules scale linearly by module count. Eliminating unnecessary modules and associated requests enables us to minimize our cost footprint, which can be put towards accomplishing the research goals outlined in our research proposal.

BoilerBot extends the cobot architecture to operate the Alexa endpoint. In addition, we deployed parallel endpoints running different build versions enabling multiple development and testing channels. Our inference pipeline enables modular intent subsetting, which is further discussed in Section 2.1, with further changes to the framework detailed in Section 6.2.

Figure 1 illustrates the inference pipeline used by BoilerBot. We implement a custom selection strategy that encapsulates our graph-based logic module. This is proxied through the state manager, and accumulates utterance metadata used during intent classification. The labelled user utterance is then parsed using service modules. The corresponding user state is accordingly run, and user

responses – conditionally including ASK APL cards – are presented to the end user. All interaction data is recorded to our **Data Studio** web application which allows the team to view, analyze, and annotate chat logs, clean various datasets, test external APIs, and prepare fine-tuning data for the Large Language Model components.

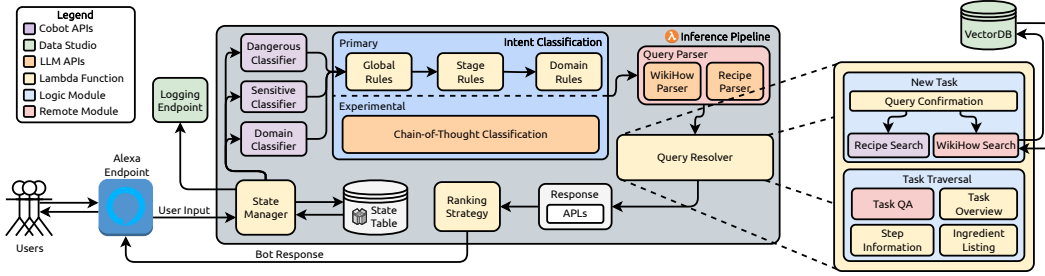


Figure 1: The BoilerBot Inference Execution Pipeline.

2.1 Logic Module

BoilerBot’s logic module encapsulates a bi-leveled objective: a) to subset the potential utterance intents (intent classification), and b) execute step-specific functions based on the selected intents (query resolution).

The intent classification module is a three-phase waterfall process that maps a user’s request into either a global, stage-dependent, or domain-dependent intent. If this classification process fails to identify an appropriate intent within the three phases, it defaults to an unknown-intent responder, which leverages chatbot-user transparency to inform the user about potential options based on their current stage, with keywords that may be used to unambiguously communicate with BoilerBot.

Within the three phases, the first handles requests that occur at any stage of conversation, including stage-to-stage navigation (forward and backward), domain classification, history retrieval, and conversation reset. For domain classification, we use a bi-layer mechanism to extend the topics that BoilerBot can cover, including “Chit-chat”, “Factual QA”, and “Unsure”, further discussed in Section 3). The second phase handles stage-dependent requests, such as task selection and QA based on domain and task knowledge. The third and final phase handles domain-dependent requests, such as task-to-module routing.

To prevent dangerous or inappropriate interactions we screen each request through Cobot’s [12] dangerous and sensitive classifiers, forwarded it to the intent classification pipeline. Dangerous topics include those requiring domain expertise, such as: **Medical**, **Financial**, **Legal**, as well as personal welfare: such as **Harmful**, and **Suicide**. Sensitive topics include **Identity** and **Politics**. Safe, appropriate requests are classified into six domains: **Cooking**, **DIY**, **Chit-chat**, **Factual QA**, **Unsupported**, and **Unsure**. Any domain score less than a set probability threshold falls into **Unsure** domain.

3 Intent Classification

BoilerBot aids users in searching and following tutorials from either the Whole Foods or WikiHow datasets. User intents through the execution of a task vaguely follow a specific flowchart. The user begins their conversation with BoilerBot by making a task query. This is then followed by navigations commands, such as “next”, “option three” or clarifying questions. However, since some users are unfamiliar with the taskbot’s scope, they treat the taskbot as a general-purpose chatbot similar to the Social Bot [14]. To assist these users and guide them back on the right track or to gracefully recover from incomplete input, we categorized the “Other” intents into **out-of-scope requests**, such as weather information or tracking orders, **factual questions**, **incomplete inputs** and **general chit-chat**.

The presence of some “Other” intents introduced edge cases to the conversation. For instance, commands like “play music” could be considered out-of-scope, whereas requests like “how to play music” should be interpreted as a WikiHow tutorial search. This ambiguity was evident in queries like “Alexa, I want to play music,” which could be attributed to either intent label. Additionally,

we encountered cases such as "recipe for my husky"; despite the word "recipe", the query should be interpreted as a WikiHow search request, which keyword-based deterministic approaches were unable to consistently account for.

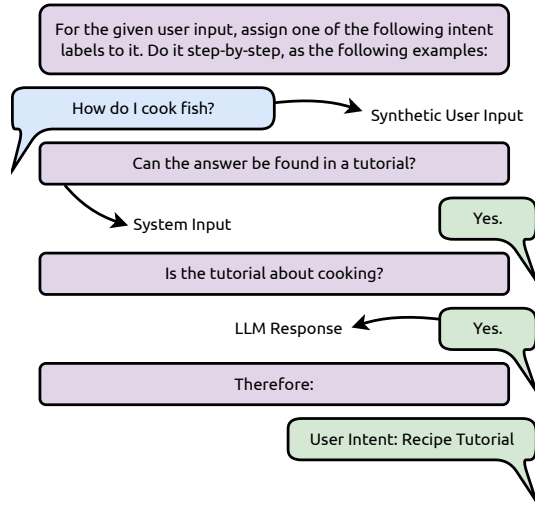
The three-phase waterfall intent classification pipeline, discussed in Section 2.1, was found to lack resilience in handling these edge cases. Hence, we re-implemented the intent classification component using large language models, which will be elaborated on in the following subsection. Although the LLM-based classifier demonstrated superior performance, it is exclusively used in experiment pipeline due to concerns about latency and potential issues with the output format. The original waterfall intent classification pipeline remains the default mechanism for intent classification and also serves as the fallback mechanism for the LLM solution.

3.1 Large Language Model and Chain-of-Thought Intent Classifier

Large Language Models capture linguistic similarity through trillions of trained tokens, enabling them to effectively emulate real-world knowledge. Following [30], we hypothesize that small, well-annotated datasets will enable effective model alignment, preserving generalization performance across niche domains.

We leveraged few-shot learning to establish a rudimentary proof-of-concept to validate our hypothesis. Our original few-shot template describes the TaskBot’s mission and provides examples by class label. This few-shot prompt over WizardLM 30B [27] did not yield ideal classification results. Despite being aided by few-shot examples, the perplexity observed in mapping the input sequences to their corresponding class labels remained considerably high.

Chain-of-Thought [26] is a prompting technique that encourages the LLMs to perform step-by-step reasoning. Using Chain-of-Thought can significantly boost LLMs’ performance on various challenging and complex tasks. We incorporated it into our few-shot prompt template in the following manner:



The Chain-of-Thought process proved highly effective for the specified use case. Our initial assessment indicated its capability to handle edge cases beyond the scope of few-shot examples. It encountered challenges in dealing with chit-chat, factual question answering, and incomplete sentences due to the overlapping nature of user inputs across these categories. Nevertheless, the few-shot prompt classifier demonstrated strong reasoning capability during our overall preliminary assessment.

Unfortunately, using a 30 billion parameter model with Chain-of-Thought classification resulted in slow inference speeds. We will delve into the relevant serving and performance concerns in the subsequent subsection. The insights gleaned from the utilization of Chain-of-Thought and few-shot prompting served as a catalyst for further improvement, leading us to fine-tune LLMs for achieving even better performance. We annotated thousands of user inputs and then augmented them with Chain-of-Thought and Few-Shot prompts similar to the prompt template above. We then fine-tuned a LLaMa2-13B [24] LLM with QLora [7]. The smaller fine-tuned 13B model showed a similar or even

stronger performance than the 30B model with similar prompting techniques. Table 1 below shows our evaluation results on the fine-tuned 13B model and our non-LLM intent classification pipeline described in Section 2.1 used as our fallback mechanism. The “Acceptable” category categorizes situations where the classifier correctly separated “Other” intents from “Search” intents but did not assign the most appropriate “Intent” under the “Other” category. For example, the classifier assigned “Chit Chat” intent to what was supposed to be “out-of-scope” intent.

Table 1: Intent Classifier Benchmark

	Perfect	Acceptable	Poor	Total
LLM Classifier	333 (76.2%)	54 (12.4%)	50 (11.4%)	437
Non-LLM Classifier	238 (54.4%)	47 (10.8%)	153 (35%)	437

3.2 Lightweight Transformer Intent Classifier

The lightweight transformer-based intent classifier was developed in the earlier stages of the competition with the primary goal to classify user intent with minimal latency. This supplements the keyword-based classification of the logic module described in subsection 2.1 when encountering an unknown intent.

After reviewing chat logs and common use cases of BoilerBot, we developed a list of common actions intended by users as captured within our UI. These intents are a portion of all of the actions users can take while interacting with BoilerBot and are described in subsection 10.4.

We chose DistilBERT to handle intent classification due to its small size, which is important for real-time inference, and due to the success of transformer-based models at classifying user intent in the first TaskBot challenge [8] [10]. Our final classifier achieved 72.4% accuracy on a testing set.

After deploying the lightweight transformer-based intent classifier, we identified a significant overlap between ‘unknown’ classifications in both the keyword and transformer based methods, suggesting that the transformer based approach weighted keywords heavily. Consequently, the approach added latency with insignificant benefit to the user’s experience, and prompted us to explore sophisticated solutions at the cost of latency, as described in subsection 3.1.

4 Query Parsing

The query parsing task has two sub-categories: recipe query parsing and WikiHow query parsing. BoilerBot retrieves recipe and WikiHow data from two external APIs.

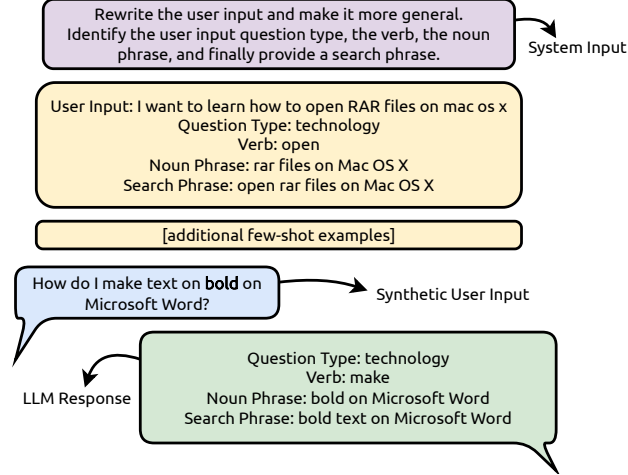
The WikiHow Search utilized a vector database, employing Sentence-BERT [20] to encode article titles and summaries as embeddings. This enabled natural language searches by finding the nearest neighbors to the query text in the embedding space. Hosting the WikiHow Search API facilitated content moderation (discussed in Section 7.3) on the local database, thereby minimizing the risk of recommending unsuitable content.

For the recipe search, we used the Cobot Recipe Search API. The API accepts search parameters like ingredients, recipe title, dietary type, etc., which requires the taskbot to parse the user request to structured data. This process is also known as “slot filling”.

4.1 Large Language Model for Parsing

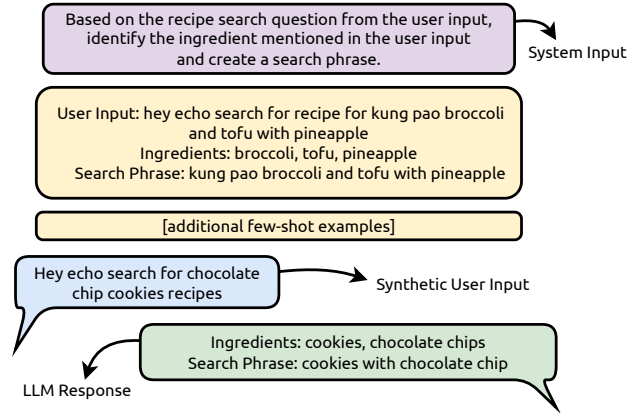
The vector database can improve search results with a more effective search phrase, even though it can process raw user input. For example, converting the user input “my grandpa wants to build a fence for cows and how should he do that?” to the search phrase “build a fence for cows” yields more relevant results.

To address the challenge of WikiHow query parsing, a proof-of-concept experiment was conducted using few-shot prompting. However, as discussed in the previous section, a standard few-shot prompt proved inadequate in generating reliable results due to the distance between the user input and the anticipated search phrase. Therefore, we used the chain-of-thought technique to generate intermediate steps to help smooth the reasoning process. Below is the prompt template we used for WikiHow search parsing (user input is synthetic):



This method worked extremely well on the WizardLM 30b model. The model not only removed non-essential elements from the user input but also can rephrase certain elements. For the “my grandpa wants to build a fence for cows and how should he do that” example, the model even substituted “cows” with “cattle”.

Similar prompting techniques were applied to analyze user inputs for recipe search. We prompted the LLM to extract ingredients and a shortened search phrase from the user input. Below is the prompt template we used for recipe search parsing:



Similar to WikiHow search parsing, LLMs (without fine-tuning on domain-specific data) can also generate reliable outputs for recipe search parsing with few-shot prompting.

In a production environment, opting for an even smaller, fine-tuned LLM would be considered an ideal choice (due to faster inference time and cheaper deployment costs). We collected 1,600 WikiHow and 1,500 recipe parsing examples (annotated) and fine-tuned one LLaMa-2 13B model with QLoRA to do both tasks. For WikiHow parsing, LLM correctly shortened all 131 (100%) test cases, and 26% of the outputs were considered “perfect” (for search) based on our human evaluator.

For the recipe parsing, we compared the performance between the LLM parser and our fallback recipe parsing mechanism. Table 2 below shows the evaluation results:

Table 2: Recipe Parsing Benchmark

	Perfect	Acceptable	Poor	Total
LLM Recipe Parser	38 (28.6%)	88 (66.2%)	6 (4.5%)	133
Non-LLM Recipe Parser	44 (33.1%)	74 (55.6%)	15 (11.3%)	133

4.2 Fallback Mechanism for Recipe Parsing

As a fallback mechanism for LLM-based recipe parser, we used a rule-based approach to parse recipe query into structure data: **dishName**, **drinkName**, **ingredients**, **dietaryFilters**, and **mealType**. This approach can be completed with nearly zero latency. We also ensured flexibility by allowing the insertion/removal of attribute parsers to handle different use cases. As **dietaryFilters** and **mealType** have finite values, we used a simple keyword-matching approach to parse these two attributes. The rest three attributes were connected with AND/NOT logic, for example, a complex request '*I want have red wine along with stew cooked with mushroom but no garlic*' contains three attribute information: "*soup*" as dish name, "*mushroom*" and "*garlic*" as ingredients, and "*red wine*" as drink name. Additionally, a user wants to include "*mushroom*" and exclude "*garlic*". Upon converting the request into attributes, we obtain the following:

*drinkName : red wine AND (dishName : stew AND ingredient : mushroom NOT
ingredient : garlic)*

4.2.1 Semantic classification of cooking-related items

To ensure effective parsing of user queries, the bot must distinguish between cooking instruments, recipe ingredients, beverages, and more. SVM was harnessed in tandem with Sentence-BERT [20] embeddings to efficiently classify cooking-related items within user inputs. We leveraged the datasets from Whole Foods and Serious Eats, which encompassed a comprehensive array of ingredients. These datasets were grouped with data from widely recognized cooking instruments and beverages. Sentence-BERT was then employed to encode these items for classification training. Even though advancements in LLMs have birthed models capable of similar classification, this approach served as a robust interim solution until such models are seamlessly integrated.

4.3 Syntactic Re-Ranking

Ranking results based on the salience of terms in the query is crucial in ensuring the return of relevant tasks to the user. To achieve this, we rely on the syntax to approximate the salient terms within the user's input. A key component of this approach involves determining the head of a noun phrase or a verb phrase and its associated modifiers within a user's request. For instance, if it is discerned that the head noun pertains to food, a search is initiated for recipes related to that specific dish type, with the modifiers serving as the potential ingredients. For example, a user may say "let's make a banana smoothie", with a root verb (make), head noun (smoothie), as well as head noun modifiers (banana). The syntactic search would then search for all smoothie recipes with ingredients including banana.

Improving results for WikiHow search requires attention to more than just the head and its modifiers, since both the root verb and head-modifier combinations are varied in the DIY-Hobby space, and user requirements will likely be very specific. For example, instructions for "painting a brick wall" will unlikely fulfill a user's original request to "build a brick wall". Although these tasks may share some related elements or items, their required actions might demand significantly different methods and equipment to achieve successful task completion. Cooking tasks on the other hand have implied cooking root verb, and have noun heads that specifically pertain to food as far as Alexa is concerned. In this case, results need to be filtered to make sure root verbs and head nouns match closely to the user's literal request.

Parsing syntactic features allows for a more targeted and accurate search, ensuring that the results align closely with the user's request. Once intermediate results have been obtained, and not pertaining to the original head noun are systematically excluded, thereby increasing the relevance of the output. Results are then ranked by their user rating score, in order to assure quality of the recipe results.

Baseline recipe and syntactic re-ranked searches were compared on a sample of 30 real user utterances. Out of 30 scenarios, baseline recipe search failed to return 8 relevant results, while syntactic re-ranking failed on only 2. While this seems promising, the failures between the models didn't have overlapping scenarios, indicating that baseline results were still successful where the syntactic re-ranking was not. From the cases that both models succeeded (total of 20), they retrieved the same number of relevant results in 14 scenarios. Examples of the results are listed in Appendix 10.5.

5 User Experience

The taskbot caters to a diverse user base across different devices, including Smart TV, Smart Speakers, Echo Shows, among others. Ensuring a smooth and efficient user interaction, while reducing the learning curve for proficient engagement, is of utmost importance. Consequently, we have tailored both the Voice User Interface (VUI) and Graphical User Interface (GUI) to enhance the overall user experience.

5.1 Voice User Interface

5.1.1 Large Language Models for Flexible Response

Our taskbot's mission is to provide user assistance by offering tutorials and guiding users through them. In cases where users are unsure about the taskbot's limitations, the response should be both contextually appropriate and functional. This requires the response to first acknowledge user requests, even when it detects out-of-scope queries, and then provides clarifying information.

When the taskbot identifies a "factual question answering" intent, it will forward the question to the Cobot Neural Response Generator API. The taskbot will then utilize the API's response when the confidence level is high. Otherwise, the user will be advised to seek more reliable sources for the question.

For chit-chats, we are currently planning on using the Conversationally Pre-Trained Alexa Teacher Model [9] to provide smooth, safe, polite, flexible, and humorous responses.

5.1.2 Standard Voice Responses

Conveying the right amount of information is crucial for a pleasant user experience through voice channels. We tailored voice responses based on speech acts [3]: assistance and task.

The variation in assistance information depends on the user's conversation state. Instead of overwhelming the user with instructions, a gradual approach is adopted, wherein minimal yet sufficient instructions, accompanied by supportive prompts through the voice channel, are offered. In situations where the user faces confusion with the given instructions, BoilerBot offers separate information to assist the user in comprehending how to proceed. In contrast to assistance information, the communication of task aims for concise instructions that minimize distractions while conveying what needs to be done. Table 4 in Appendix 10.3 shows a sample voice conversation.

5.1.3 Flexible Selection and Navigation

BoilerBot ensures natural interaction to improve the user experience at selection and navigation. It is unnatural to ask the user to give specific rules during interaction with the system, for example, a user must say "*option 3*" or "*step 3*" to select a result or go to a specific step. Instead of looking for specific commands, we allow a user to use causal language to select a task or navigation through a task, for example, "*I don't know, maybe the coffee one*" or "*show me the fifth step*". First, we enable cardinal and ordinal recognition to extract the number from utterance, so that the user can select a task or navigate a task by saying "*let's see the fifth one*" or "*go to the twenty-first step*". In addition to cardinal and ordinal selection, we enable flexible selection by leveraging TheFuzz [21] library to calculate the partial matching ratio between sequences using Levenshtein Distance. For cases like "*the coffee one*" when a user tries to select from a list of cakes, we remove the common identifier token "*cake*" from the result titles and compute string similarities only with the unique identifier tokens. Additionally, when the similarity ratio is below the threshold, the system interprets the request as a new task search, for example, "*how to make coffee*" even when the list of cakes contains a coffee flavor.

5.2 Graphical User Interface

Our team conducted a small-scale user study at the beginning of the competition to identify user preferences on a Graphical User Interface (GUI) design. The study revealed three key findings:

1. Experienced cooks prefer a single-page format that includes all ingredients and steps. They then adapt the information to suit their comfort and style.

2. New users can be confused about navigating through different pages.
3. Users with some technical experience are interested in how the bot interprets their requests.

To cater to diverse user preferences, we implemented a preview page presenting comprehensive information in a scrollable format. Additionally, users have the option to switch to a step-by-step view. This simple strategy ensures a personalized user experience for distinct user groups.

For the other findings, we revisited both the GUI design and the issues users and our taskbot encountered on each page. The following subsections will identify the challenges and present our solutions.

5.2.1 Communication through GUI

The launch page plays a critical role in acquainting users with the taskbot's capabilities and scope. Its primary functions encompass: 1) aiding users in formulating search requests that yield more precise and relevant results, and 2) communicating the taskbot's limitations.

Significant modifications were undertaken to transform the default launch page, which featured three buttons leading to default search results in each category. Our initial redesign (V1) removed all the buttons, thus maximizing the utilization of screen space to exhibit sample questions that users can ask. The objective of this redesign was to mitigate instances where users express dissatisfaction with the search results and subsequently submit new search requests on the search result page. In such cases, distinguishing between a user's intent to select a specific item from the search result list or initiate a fresh search can pose considerable challenges.

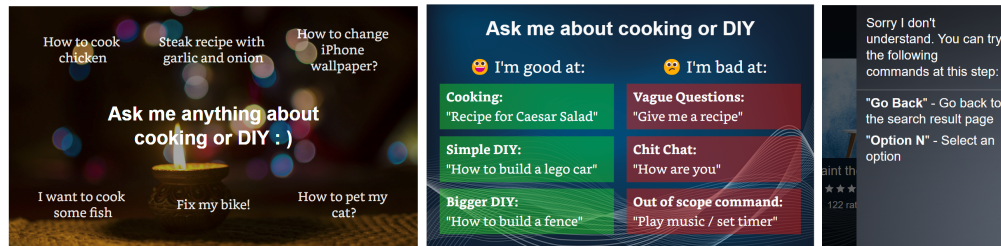


Figure 2: Launch Page V1 (left), Launch Page V2 (mid), and Unknown Responder (right) Design

The enhanced version, "Launch Page V1," can be conveniently adapted to promote special events. During the event period, a distinct button dedicated to the special event is visible on the screen, leading users to a curated compilation of tutorials specifically curated by the taskbot team for the event. Additionally, the dynamically displayed floating questions will align with the theme of the special event.

After analyzing user inputs on the launch page, it was observed that a significant number of users asked questions recommended on the page. However, some users had difficulty understanding the taskbot's scope despite the presence of floating questions. They tried to use the taskbot for purposes beyond its intended functionalities, such as controlling smart home devices, playing music, news, weather information, or engaging in chit-chat. The original redesign, 'Launch Page V1,' didn't effectively address these limitations. To rectify this, the 'V2 Redesign' was proposed to provide explicit and unambiguous guidance to users about the taskbot's capabilities and constraints.

Throughout the conversation, users frequently submitted out-of-scope requests, requiring a holistic approach. To address this, we introduced the "unknown responder," a slide-in window from the right side of the screen. This feature displays acceptable request types and examples relevant to the ongoing interaction, occupying only 40% of the screen. It keeps users oriented and minimizes confusion.

5.2.2 Mitigating Parsing Challenges with Query Confirmation

In the previous section, extracting structured recipe search parameters from user input proved challenging. To address this issue from a GUI design perspective, we added an additional stage, the “Confirmation Stage,” between the launch stage and the search result stage.

The recipe confirmation stage consists of two pages (Figure 3): “Recipe Query Confirmation” and “Recipe Query Edit.” The confirmation page displays the user’s utterance alongside its parsing into various search parameters. If a user wishes to adjust the search parameters, they can proceed to the edit page where they have the option to add or remove ingredients, dietary types, or meal types.

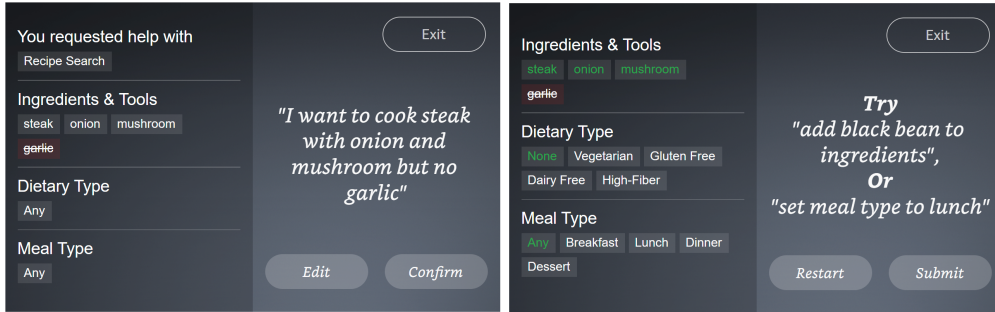


Figure 3: Recipe Request Confirmation (left) and Edit (right) Page Design

6 Engineering Challenges

The taskbot exhibits a diverse array of functionalities, presenting a considerable engineering challenge. The primary objectives include reducing the latency, increasing the throughput rate, and optimization of cost-efficiency. We used various techniques to overcome the engineering challenges.

6.1 LLM Quantization, Fine-tuning, and Serving

The majority of the development and experiment were completed on servers with A10 (24GB) or T4 (16GB) GPUs in order to comply with the budget. Quantization techniques [11][6] allowed us to load large models on hardware without enough VRAM or even without a GPU. Quantization is also helpful in one of the work-in-progress projects (Section 8.2).

Full-parameter fine-tuning of a 7B LLM requires at least 160GB VRAM which was not easily accessible by our team. As a solution, we adopted the parameter-efficient fine-tuning (PEFT) techniques like Lora [15] and QLoRA [7] (which combined quantization and Lora to further save memory). On a GPU server with 4*A10 GPUs, we were able to comfortably load a 13B model and fine-tune it with at least 1024 max sequence length.

Serving the LLMs in a production environment requires high throughput and low latency. We currently deployed a 13B model using vLLM [23] on a server with 4*A10 GPUs. The latency for intent classification ranges between 0.7 to 1.7 seconds and latency for parsing remains below 1.2 seconds. With even smaller models and shorter prompts, the latency is likely to be further reduced. However, under the current serving capability and inference settings, we are only using the LLM-based components in the B testing.

6.2 Changes to the Cobot Framework

The Cobot Framework establishes a strong foundation for our conversational agent. We extend this functionality in the following ways:

ASK-Cobot APL Bridge The Cobot SDK provides access to a set of APL templates that may be used to develop multimodal interfaces for Alexa users. However, these are limited and lack

customizability, making it hard to develop interfaces that utilize the full range of APL features towards the competition. We independently developed APL interfaces, taking input files used by a custom class extending `cobot_core.path.to.BaseApI.Document`. This class overrides `build_document`, and is rendered by the Alexa front-end as an equivalent to a Cobot template, enabling us to extensively utilize the ASK stack for our taskbot designs.

Restrictions on Lambda Environments AWS Lambda’s provided environments enable the use of up to Python v3.10. Additionally, AWS S3 artifacts do not allow us to roll back without rebuilding. By containerizing the Lambda function, obtaining fine-grained control over the runtime environment, we made it possible by using ECR to enable rollbacks in case of failure. This also enabled us to set up parallel TaskBot deployments for quicker prototyping and testing.

Microservice to Monolithic Accelerated-Compute Cluster The Cobot framework uses a microservice-based architecture that processes NLU/NLP modules asynchronously on user input. The primary concerns with this approach are: a) microservices run inference even with incompatible input, which requires integrating redundancy, and b) creating microservers for every individual NLU functionality on the default instances results in significant cost overheads that scale linearly.

An effective solution for both of these was using a monolithic remote module server, using an instance with GPU acceleration through the NVIDIA T4 that runs accelerated functions on a query, not by user utterance. We discuss specific costs and compute savings from this approach in Appendix 10.2.1.

7 Data Analysis and Annotation

7.1 BoilerBot Data Studio

The Data Studio allows the entire team access to key data and tools necessary for the competition. It is a website developed in-house that visualizes all conversation ratings (Figure 5), allows the team to annotate the conversation with notes or labels (Fig 4), access debugging logs quickly, and test our in-house APIs. The Data Analysis Studio centralizes all of our data and annotation processes.

7.2 Annotate with Large Language Models

The Alpaca project [22] proved that by using outputs from larger models, smaller models could significantly improve their overall capability through the process of fine-tuning.

For the recipe and WikiHow parsing dataset, we developed a semi-automated pipeline to annotate the data. The first step is to collect diverse search phrases. Our data came from the following sources (all stored in the data studio):

1. Real user inputs from the chat logs labeled by our team members
2. Recipe and WikiHow titles from the offline datasets augmented with irregular language elements (vocal fillers, prefix like “how to make” and “how do i cook”, suffix like “please” and “thank you”).

We then carefully created chain-of-thought few-shot prompt templates and tasked a 30B LLM to annotate each user request. Some of the outputs did not follow the format we expected. Therefore, we used rule-based scripts to validate the outputs and filter out the bad responses. Finally, a team member inspected the filtered annotations and manually removed the low-quality ones.

With the semi-automated data annotation pipeline, we managed to quickly create the dataset for fine-tuning WikiHow and recipe parsing LLMs.

7.3 Content Moderation

The WikiHow dataset poses challenges due to the inclusion of tutorials that may be unsuitable for a younger audience or may be dangerous if followed by untrained individuals. To address this concern, we initially developed a neural-net classifier to identify hazardous user requests. However, this approach unintentionally blocked some legitimate and safe queries, negatively impacting the overall user experience.

To resolve these issues, we implemented a comprehensive content moderation process at the data source. Initially, we used keyword-based filtering to remove inappropriate content, resulting in a dataset of 30,000 articles. Subsequently, human inspection was conducted on this filtered dataset, with annotations performed by our team members through the Data Studio website. Despite these efforts, certain edge cases, such as requests involving activities like "play shotgun game" or "build a bomb in Minecraft," triggered content violation alerts.

These challenges emphasized that content moderation extends beyond a purely technical problem and necessitates interdisciplinary solutions. Striking a balance between ensuring absolute safety and providing a seamless user experience proved to be a complex undertaking, requiring insights from diverse domains beyond technical considerations.

8 Work in progress

8.1 Streamlining the Data Annotation, LLM finetuning, and Deployment Pipeline

As we learned from projects like Orca [18], WizardLM[27], LIMA [30], and phi-1[13], having the correct data is the key to train or fine-tune better LLMs. We are currently building a semi-automated pipeline to streamline the process from data collection to LLM deployment. As we received chat logs from the users every day, some team members will identify the cases where our LLM fails and then annotate the cases with the correct response. The data studio will fine-tune our current LLM checkpoints periodically with the newly annotated data and then deploy the new checkpoints for A/B testing. Thanks to the quantization and parameter-efficient finetuning techniques, our LLMs will be able to improve continuously with minimum human intervention.

8.2 Automated Testing with LLMs

Identifying problematic user inputs and taskbot responses typically requires human effort. Since we can now deploy a 4-bit quantized LLaMa-2 70B model [24] with less than 48GB VRAM (latency is acceptable for testing and larger models show less degradation after quantization [29] [5]), fully automated testing has become possible with larger models and more sophisticated prompting techniques like Tree of Thoughts [28]. Instead of having human testers interact with the taskbot and identify issues from the taskbot outputs, LLM could potentially fulfill the testing role, interacting with the taskbot through the testing API. Using larger models to evaluate outputs from smaller models has been used widely in LLM benchmarks [17]. Our team is working on adapting the methodology to the taskbot domain.

9 Conclusion

In this paper, we presented BoilerBot, a conversational agent designed for the Alexa Prize TaskBot 2 competition. Through fine-tuning large language models on curated domain-specific data, we demonstrated the efficacy of this approach for specialized downstream tasks requiring complex linguistic understanding and inference. Techniques such as chain-of-thought prompting and few-shot learning enabled accurate parsing and classification for challenging user requests.

Furthermore, insights from user studies informed a redesigned interface aimed at providing a smooth user experience. Features such as confirmation stage and unknown responder serve to orient users and minimize confusion, and also mitigated certain technical challenges with graphical user interface elements.

Moving forward, priorities include continuing to streamline integration of large language models with lower latency, higher throughput rate, and better reliability. Automating data annotation, training, and deployment will enable continuous enhancements as more conversational data is gathered. Additionally, exploring larger models and advances in prompting methodology will further augment capabilities for automated testing. By combining cutting-edge natural language processing technology with user-centered design, we aim to develop an increasingly natural and useful task-oriented conversational agent.

References

- [1] Eugene Agichtein, Yoelle Maarek, and Oleg Rokhlenko. “Alexa Prize TaskBot Challenge”. In: *Alexa Prize TaskBot Challenge Proceedings*. 2022. URL: <https://www.amazon.science/alexaprize/proceedings/alexaprize-taskbot-challenge>.
- [2] Eugene Agichtein et al. “Alexa, Let’s work together: Introducing the Second Alexa Prize TaskBot Challenge”. In: *Alexa Prize TaskBot Challenge 2 Proceedings*. 2023. URL: <https://www.amazon.science/alexaprize/proceedings/alexaprize-lets-work-together-introducing-the-second-alexaprize-taskbot-challenge>.
- [3] James F Allen and C Raymond Perrault. “Analyzing intention in utterances”. In: *Artificial intelligence* 15.3 (1980), pp. 143–178.
- [4] Christiane Attig et al. “System latency guidelines then and now—is zero latency really considered necessary?” In: *Engineering Psychology and Cognitive Ergonomics: Cognition and Design: 14th International Conference, EPCE 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9-14, 2017, Proceedings, Part II 14*. Springer. 2017, pp. 3–14.
- [5] Tim Dettmers and Luke Zettlemoyer. “The case for 4-bit precision: k-bit inference scaling laws”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 7750–7774.
- [6] Tim Dettmers et al. “LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale”. In: *arXiv preprint arXiv:2208.07339* (2022).
- [7] Tim Dettmers et al. “Qlora: Efficient finetuning of quantized llms”. In: *arXiv preprint arXiv:2305.14314* (2023).
- [8] Rafael Ferreira et al. “TWIZ: A conversational Task Wizard with multimodal curiosity-exploration”. In: *Alexa Prize TaskBot Challenge Proceedings*. <https://www.amazon.science/alexaprize/proceedings/twiz-a-conversational-task-wizard-with-multimodal-curiosity-exploration> (2022).
- [9] Jack FitzGerald et al. “Alexa teacher model: Pretraining and distilling multi-billion-parameter encoders for natural language understanding systems”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 2893–2902.
- [10] Carlos Gemmell, Iain Mackie, and Jeffrey Dalton. “GRILLBot: A flexible conversational agent for solving complex real-world tasks”. In: 2022. URL: <https://api.semanticscholar.org/CorpusID:249261462>.
- [11] Georgi Gerganov. *llama.cpp: Port of Facebook’s LLaMA model in C/C++*. <https://github.com/ggerganov/llama.cpp>. 2023.
- [12] Anna Gottardi et al. *Alexa, Let’s Work Together: Introducing the First Alexa Prize TaskBot Challenge on Conversational Task Assistance*. 2022. arXiv: 2209.06321 [cs.CL].
- [13] Suriya Gunasekar et al. “Textbooks Are All You Need”. In: *arXiv preprint arXiv:2306.11644* (2023).
- [14] Dilek Hakkani-Tür. “Alexa Prize Socialbot Grand Challenge Year IV”. In: *Alexa Prize Social-Bot Grand Challenge 4 Proceedings*. 2021. URL: <https://www.amazon.science/alexaprize/proceedings/alexaprize-socialbot-grand-challenge-year-iv>.
- [15] Edward J Hu et al. “Lora: Low-rank adaptation of large language models”. In: *arXiv preprint arXiv:2106.09685* (2021).
- [16] Ziwei Ji et al. “Survey of hallucination in natural language generation”. In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38.
- [17] Yang Liu et al. “Gpteval: Nlg evaluation using gpt-4 with better human alignment”. In: *arXiv preprint arXiv:2303.16634* (2023).
- [18] Subhabrata Mukherjee et al. *Orca: Progressive Learning from Complex Explanation Traces of GPT-4*. 2023. arXiv: 2306.02707 [cs.CL].
- [19] Shishir G Patil et al. “Gorilla: Large language model connected with massive apis”. In: *arXiv preprint arXiv:2305.15334* (2023).
- [20] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *arXiv preprint arXiv:1908.10084* (2019).
- [21] Seatgeek. *TheFuzz*. <https://github.com/seatgeek/thefuzz>. 2021.
- [22] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

- [23] vLLM Team. “vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention”. In: *GitHub repository* (2023).
- [24] Hugo Touvron et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *arXiv preprint arXiv:2307.09288* (2023).
- [25] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).
- [26] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.
- [27] Can Xu et al. “Wizardlm: Empowering large language models to follow complex instructions”. In: *arXiv preprint arXiv:2304.12244* (2023).
- [28] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *arXiv preprint arXiv:2305.10601* (2023).
- [29] Zhewei Yao et al. “A comprehensive study on post-training quantization for large language models”. In: *arXiv preprint arXiv:2303.08302* (2023).
- [30] Chunting Zhou et al. “Lima: Less is more for alignment”. In: *arXiv preprint arXiv:2305.11206* (2023).

10 Appendix

10.1 Data Studio Screenshots

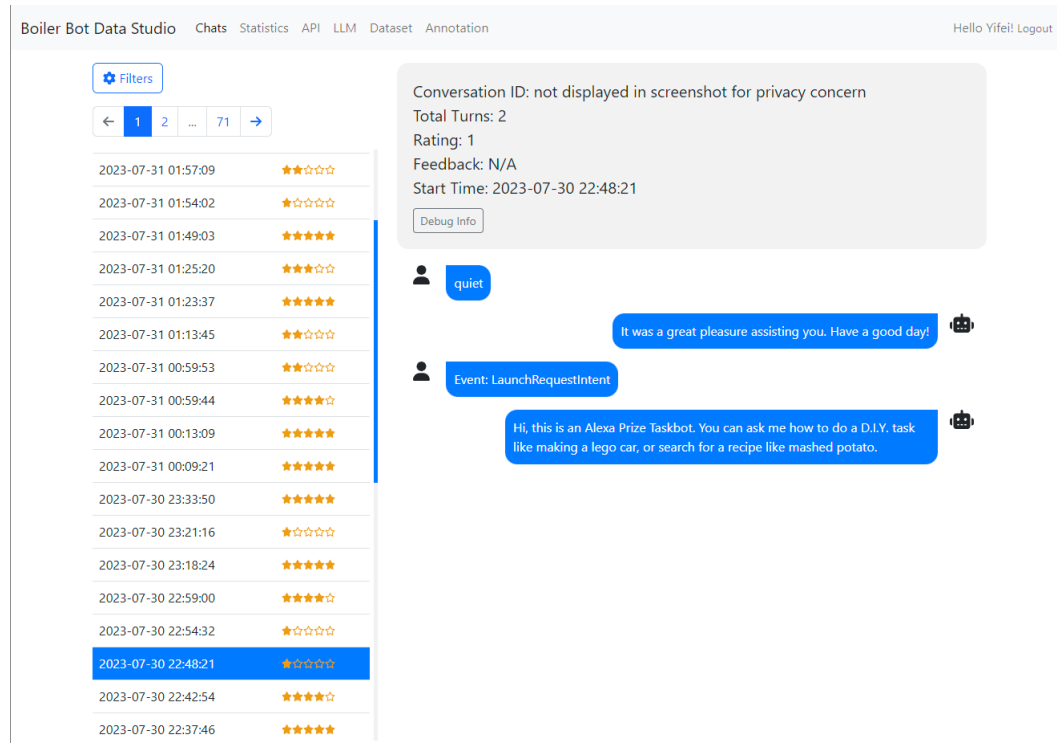
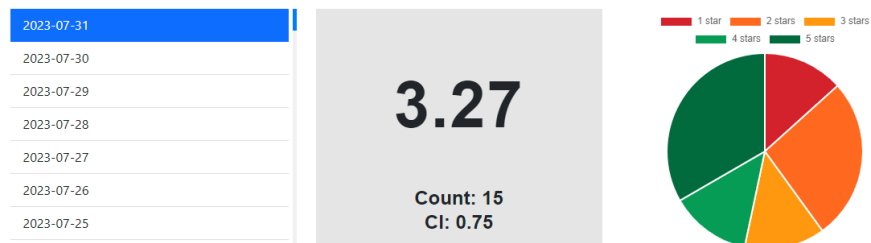


Figure 4: The BoilerBot Data Studio - Chat Logs

Daily Statistics



Trends

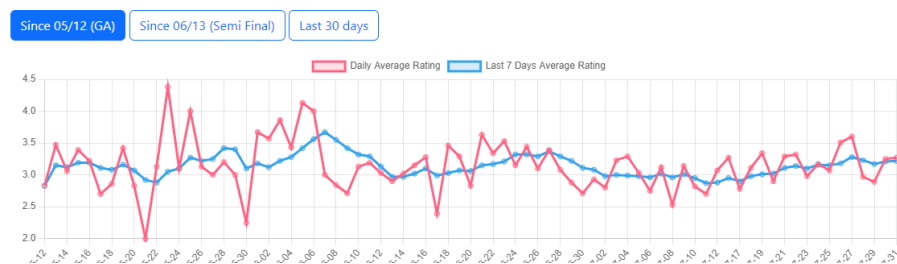


Figure 5: The BoilerBot Data Studio - Data Visualization Dashboard

10.2 Infrastructure

As part of the changes made to the cobot infrastructure, BoilerBot makes significant changes to the build format, illustrated in the following diagram:

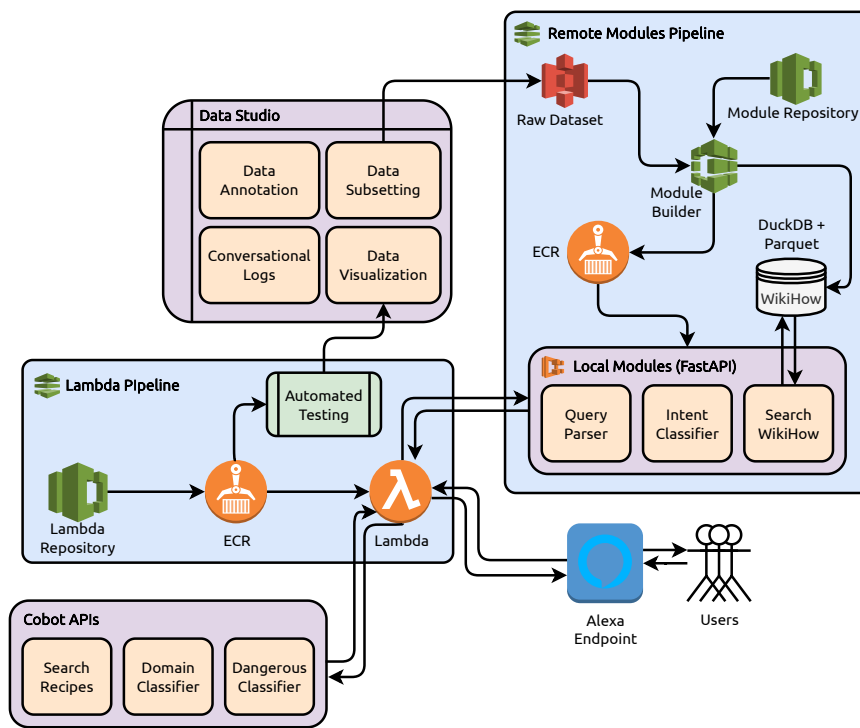


Figure 6: The BoilerBot Infrastructure diagram.

10.2.1 Cost Savings on Monolithic Computer Cluster

Currently, we host 11 remote functions in our monolithic server. Even with GPU acceleration, the monolithic server has proven significantly cheaper than hosting 4 microservices without GPU acceleration, while serving a larger number of users. For a fine-grained comparison, find below cost estimates that highlight service usage to costs, with 11 microservices instead of 4:

Architecture	Monthly Cost Estimate	GPU Acceleration	Workload
Microservices	\$3567.23	CPU only	Baseline: 11, Peak: 22 @ 4h/day
Monolithic	\$1123.00	NVIDIA T4	1 instance, consistent: 55%/month

Table 3: On-Demand costs for asynchronous remote modules. Costs and utilizations are representative to observed metrics.

10.3 Voice Response Examples

Taskbot (A)	Hi, this is an Alexa Prize Taskbot. You can ask me how to do a DIY task like making a lego car, or search for a recipe like mashed potato
User	I don't know
Taskbot (A)	If you need help with cooking or home improvement tasks, I can help! Ask me how to make pasta or build a fence
User	Okay, make pasta
Taskbot (T)	Here is what I found in Whole Foods Market. Say option number to select. Option 1 is...
User	Alright
Taskbot (A)	I found these great options! You can tell me the option number, or the option title
User	The pasta with peas
Taskbot (A)	Great! Before we get started, please be careful when using any tools or equipment... Once you are ready, just say, start cooking
User	What should I do
Taskbot (A)	You can ask me about the ingredients, or say "start" to begin the recipe. You can also jump to a step by telling me the step number
User	What are the ingredients
Taskbot (T)	You will need 2 tablespoons olive oil ...
User	Okay I got everything. Let's go to the last step
Taskbot (T)	Step 12. You can serve with ...
User	All set
Taskbot (T)	Task complete! You can say "stop" to exit the taskbot mode. Or ask anything to start a new search

Table 4: Examples of Voice UI. A stands for assistance information and T is task information. All user inputs are synthetic.

10.4 Intents

Intent	Description
BoilerBot.NextIntent	Go to the next page, proceed with the conversation, or confirm something.
BoilerBot.StopIntent	End the conversation.
BoilerBot.CompleteIntent	Finished with the current task.
BoilerBot.PreviousIntent	Go to the previous page or undo an action.
BoilerBot.SelectionIntent	Selecting a task or a task step.
BoilerBot.MoreTasksIntent	Would like more tasks after an initial search.
BoilerBot.QAIntent	Asking a general question BoilerBot can reasonably answer.
BoilerBot.SearchTaskIntent	Wants to start a new task.
BoilerBot.UnknownIntent	Unknown intent.

Figure 7: BoilerBot Intents Classified by DistilBERT

10.5 Syntactic Re-Ranking Recipes

The following is a sample of results of baseline recipe search pipeline compared to a syntactic-based search. The bold items are irrelevant returned from corresponding utterances. While the pipelines agree in many cases, or have multiple recipes in common and in order, the baseline pipeline returned nothing during several interactions. On the other hand, the syntactic parsing and re-ranking currently tended not to return enough relevant recipes when the baseline pipeline could.

User Utterances	Baseline Recipe Results	Syntactic Re-ranking Results
'how do i get smoothie'	1. 'Sorry, I didn't find anything related. Please ask me another question, hopefully about cooking or d.i.y.'	1. 'Green Smoothie' 2. 'Coffee Smoothie' 3. 'Superfood Smoothie' 4. 'Pomegranate Smoothie' 5. 'Blueberry smoothie'
'how do i cook an egg'	1. 'Eggs In Clouds' 2. 'Cloud Eggs' 3. 'Cloud Eggs (Egg Nests)' 4. 'Avocado Toast with Egg' 5. 'Eggs Florentine'	1. 'Egg Bhurji' 2. 'Sausage-wrapped Soft Boiled Egg (Scotch Egg)'
'how to make croutons'	1. 'I probably misheard. Let's try again. I can help you with cooking tasks or D.I.Y. tasks.'	1. 'Garlic Croutons' 2. 'Homemade Croutons' 3. 'Croutons' 4. 'Croutons' 5. 'Croutons'
'how to make a doughnut'	1. 'African Doughnut (Drop Doughnut)' 2. 'Vegan Doughnuts' 3. 'Italian Doughnuts' 4. 'Italian Doughnuts' 5. 'Galaxy doughnuts'	1. 'African Doughnut (Drop Doughnut)'
'make banana'	1. 'Blueberry Banana Smoothie' 2. 'Banana Cream Pie' 3. 'Fluffy Banana Pancakes' 4. 'Banana Crunch Muffins' 5. 'Banana Banana Bread'	1. NONE
'alexa how to make smoothies'	1. 'Raspberry molten chocolate cupcakes' 2. 'Air Fryer Salmon' 3. 'Easy Overnight Oats' 4. 'Air Fryer Sweet Potato Fries' 5. 'Lebanese Garlic Sauce (Toum)'	1. 'Ombre Smoothies'
'ask her how to make a salad'	1. 'Sorry, I didn't find anything related. Please ask me another question, hopefully about cooking or d.i.y.'	1. 'Shirazi Salad' 2. 'Broccoli Salad' 3. 'Ratatouille Salad' 4. 'Zucchini Salad' 5. 'Harvest Salad'
'barbecue rub'	1. 'I probably misheard. Let's try again. I can help you with cooking tasks or D.I.Y. tasks.'	1. 'Korean Pork Barbecue' 2. 'Pulled Pork Barbecue'
'how do you make a taco'	1. 'Sorry, I couldn't find any relevant results. Let's try another question.'	1. 'Fish Taco' 2. 'Taco Hotdish' 3. 'Taco Salad'